УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ ТЕОРИЯ ВЕРОЯТНОСТЕЙ И СТАТИСТИЧЕСКИЙ АНАЛИЗ ДАННЫХ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

Лабораторная работа №1

Описательная статистика пакете MS Excel при изучении случайных функций. Пакет Анализ данных.

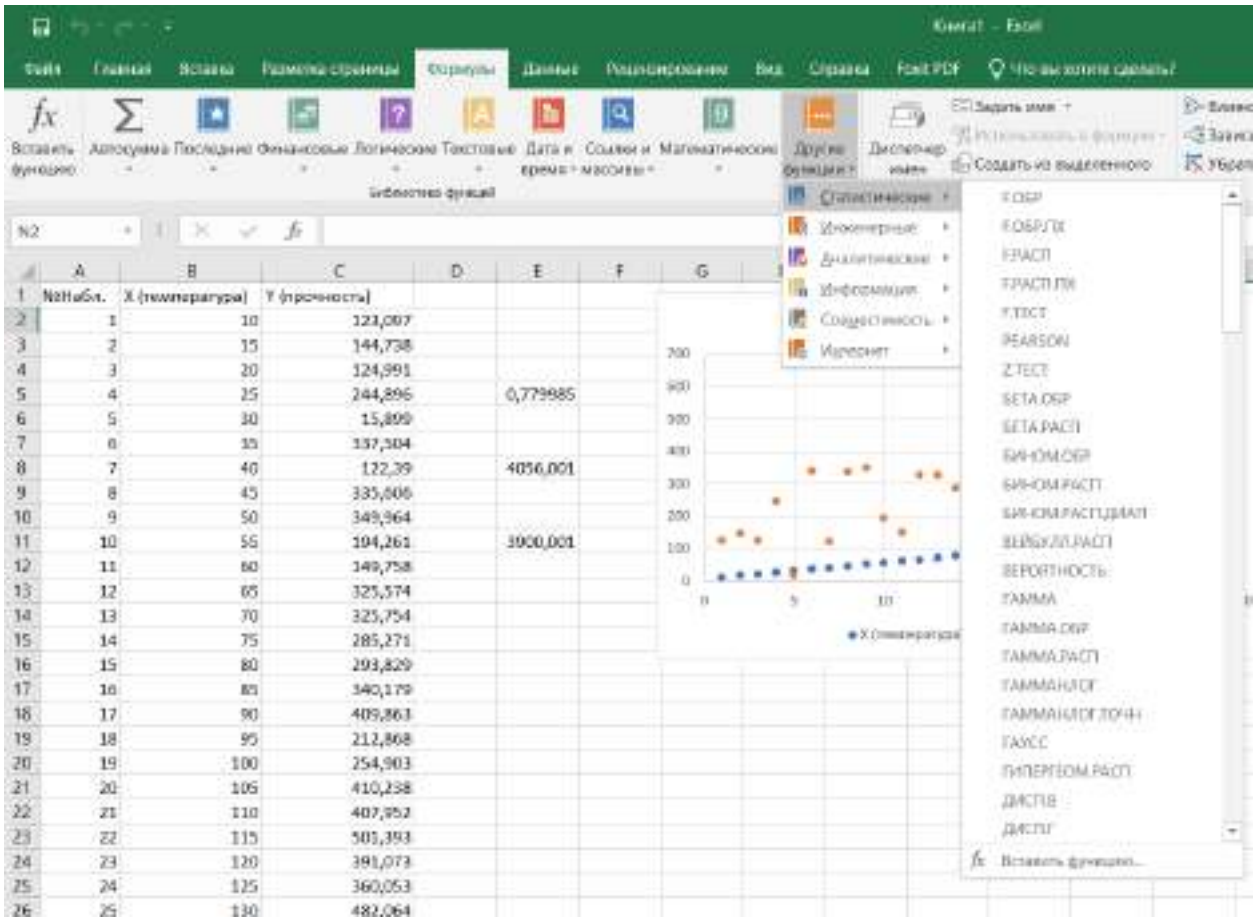
Цель работы. Научиться использовать «Пакет анализа» MS Excel для генерации случайных чисел, вычисления описательных статистик. Изучить статистические функции предложенные в данном пакете.

Изучить настройку и использование функции Анализа данных.

1. Запустить MS Excel и создать файл.
2. Загрузить данные (не менее 50 элементов каждого массива) из любой базы (по желанию).

   Например, https://data.mos.ru/opendata, https://data.humdata.org/ , http://archive.ics.uci.edu/ml/index.php.

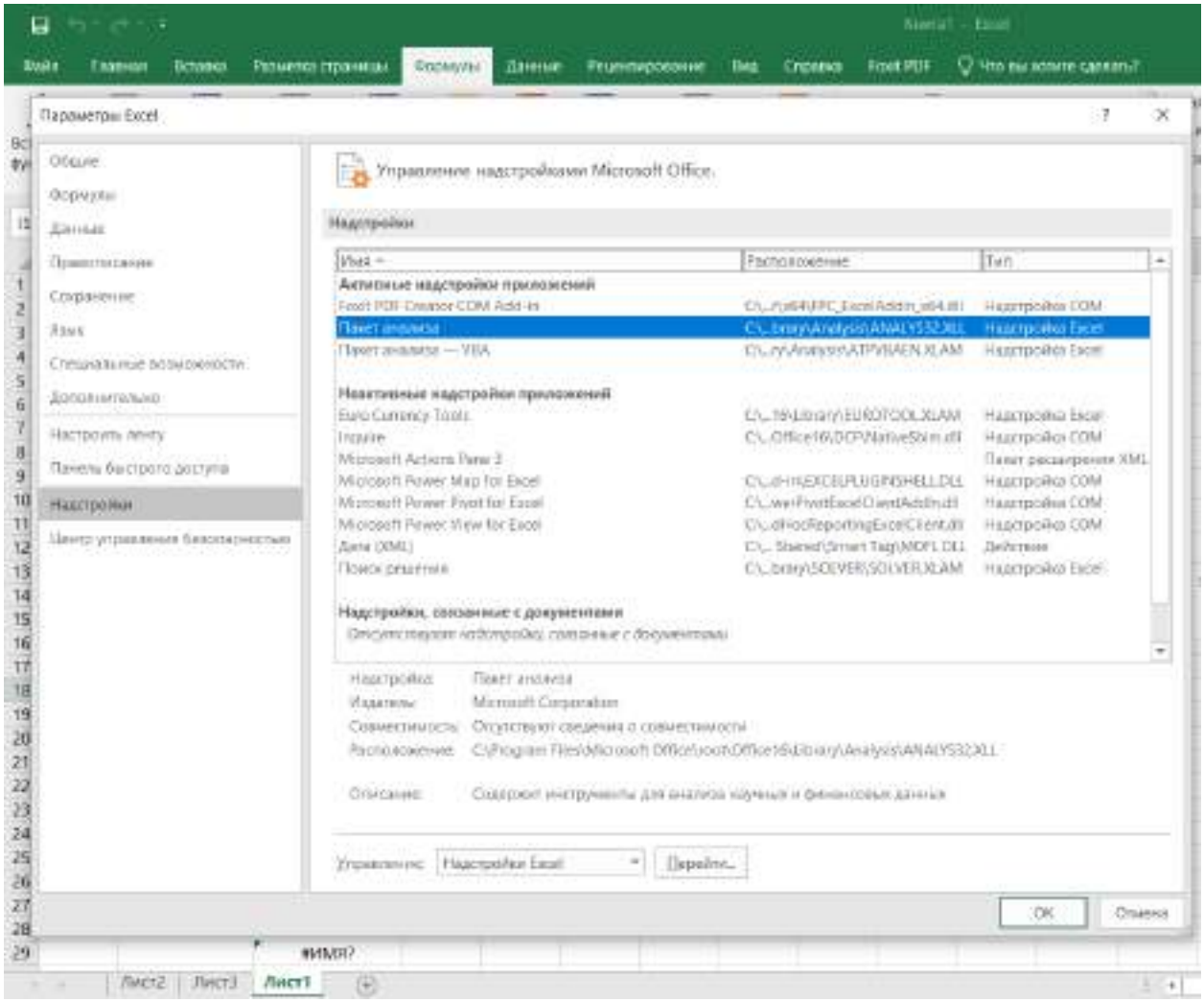
3. Выбрать независимую X и зависимую Y переменные.
4. Выбрать Формулы-> Другие функции -> Статистические. В открывшейся панели изучить представленные функции.



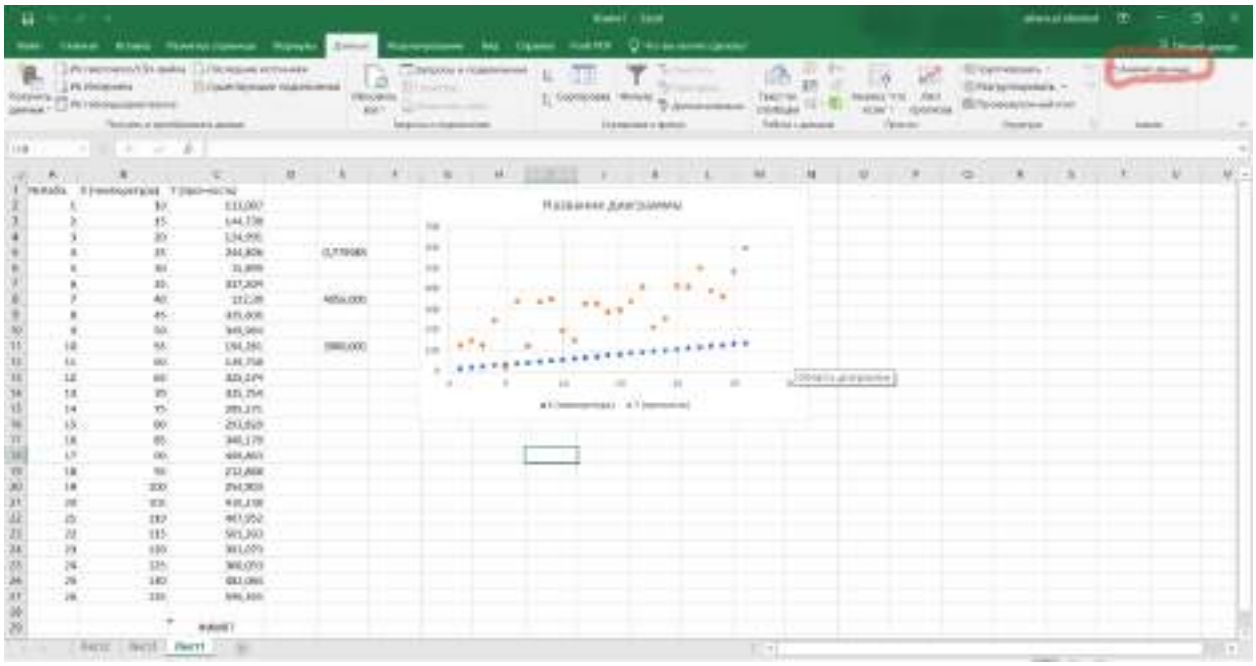
5. Настроить функцию «Анализ данных».

Для этого в левом меню перейти в:

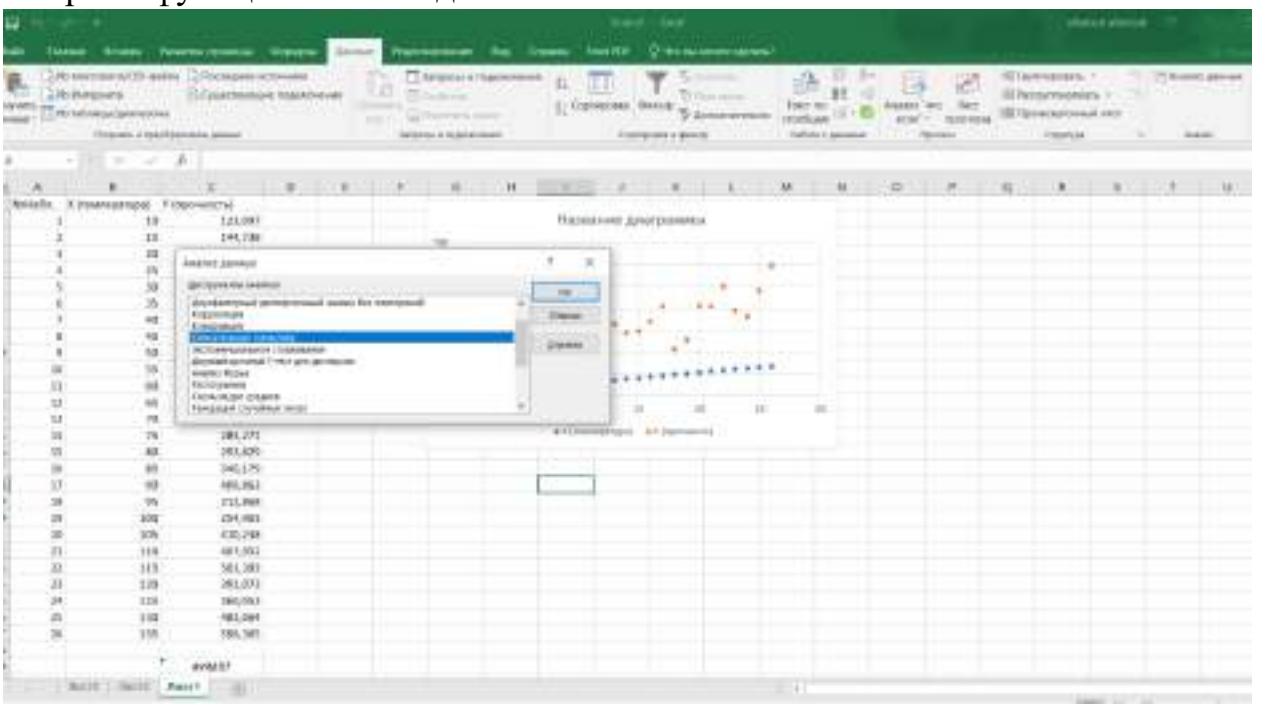
Файл->Параметры->Настройки->выбрать Пакет анализа



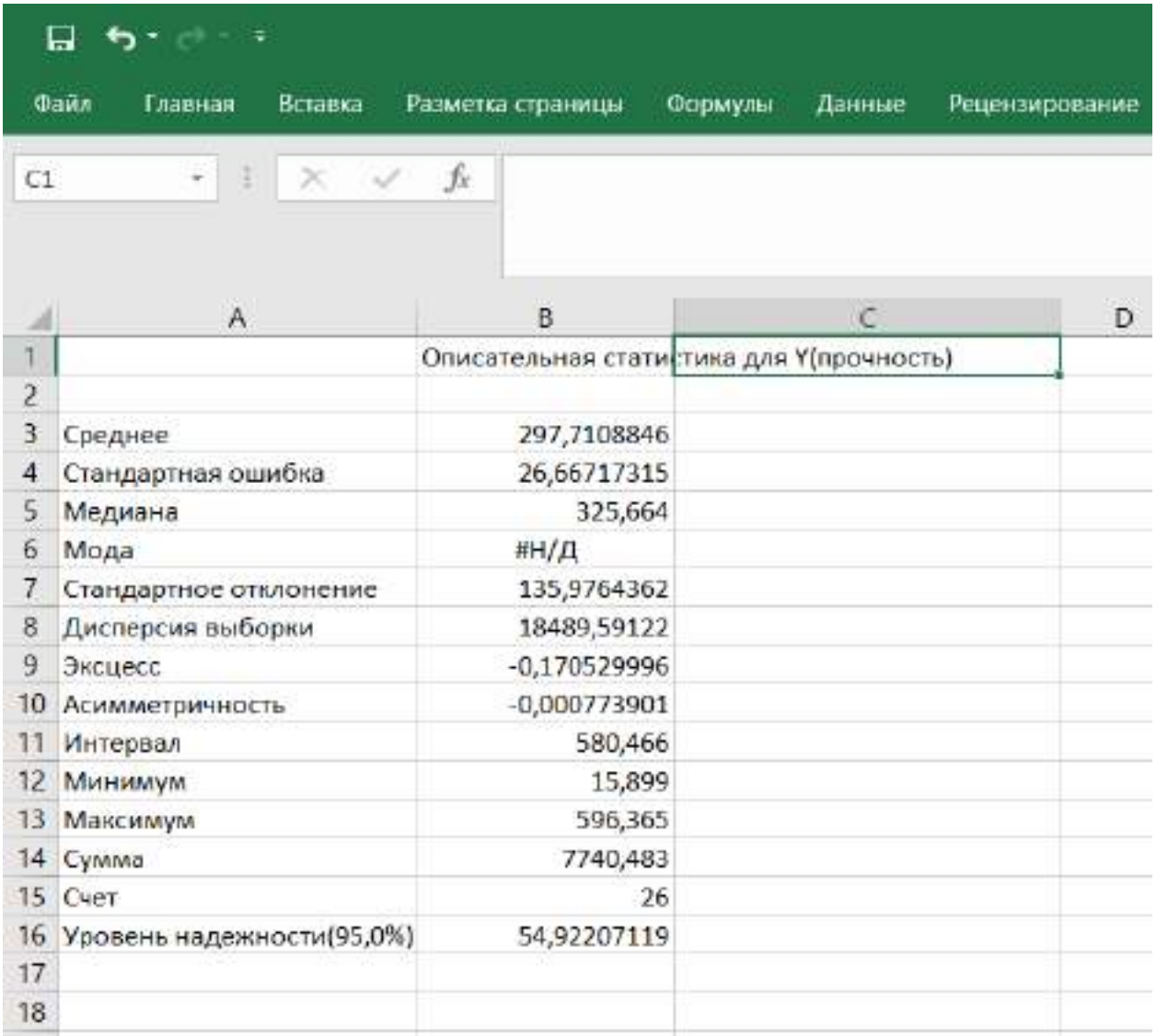
После этого меню появится функция Анализ данных



6. Открыть функцию Анализ данных

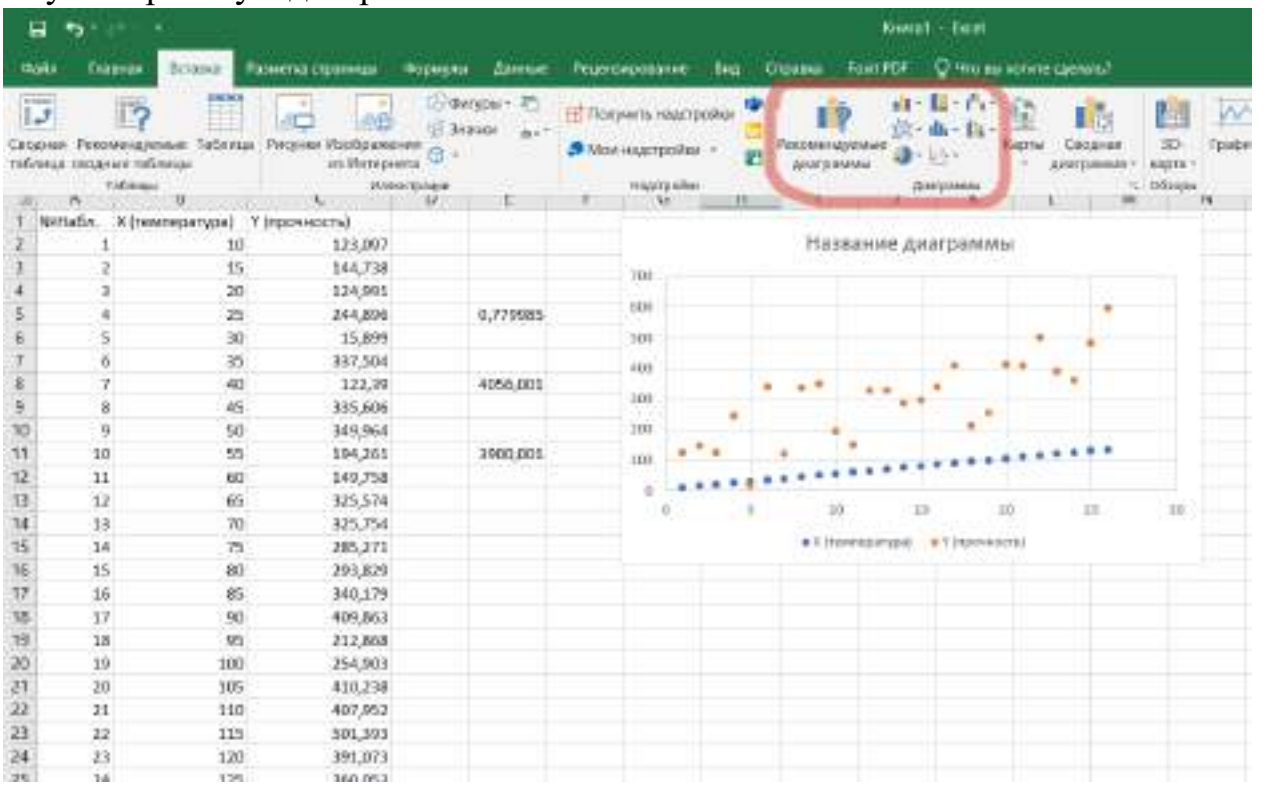


7. Изучить опции этой функции.
8. Использовать опцию Описательная статистика к выбранными Вами данным. В итоге открывается окно с набором статистик.



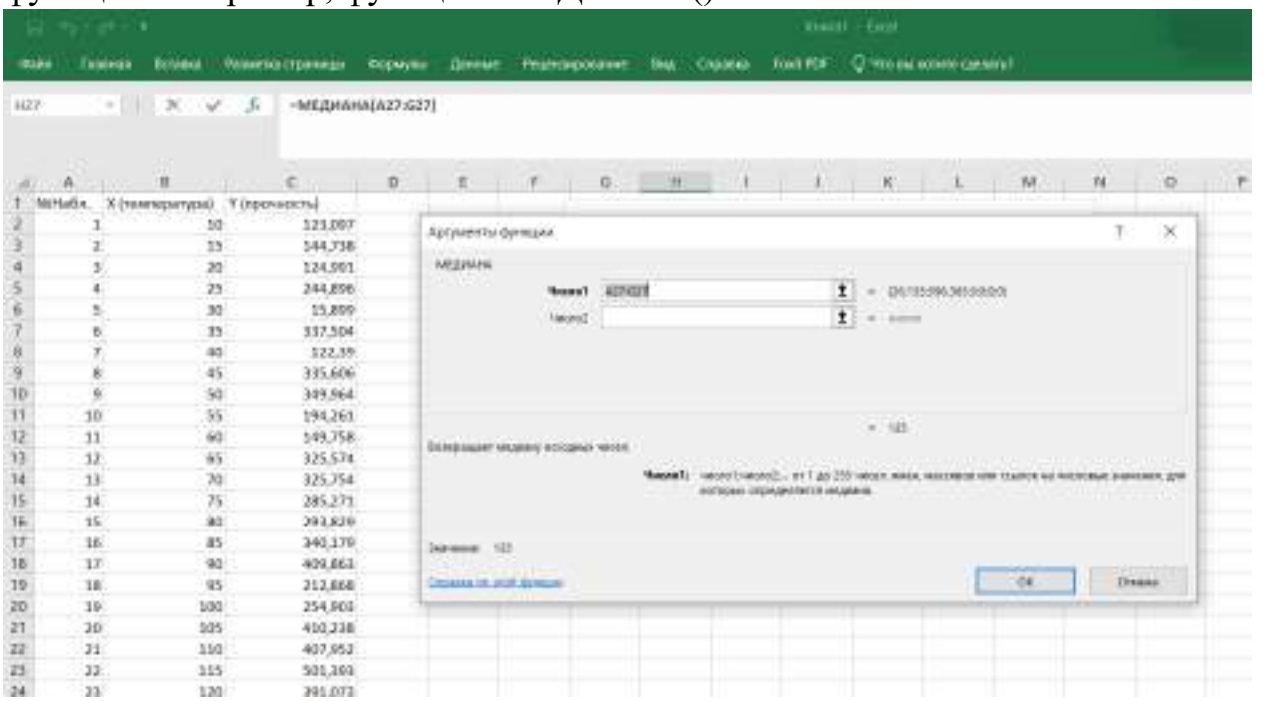
	A	B	C	D
1		Описательная стати	тика для Y(прочность)	
2				
3	Среднее	297,7108846		
4	Стандартная ошибка	26,66717315		
5	Медиана	325,664		
6	Мода	#Н/Д		
7	Стандартное отклонение	135,9764362		
8	Дисперсия выборки	18489,59122		
9	Эксцесс	-0,170529996		
10	Асимметричность	-0,000773901		
11	Интервал	580,466		
12	Минимум	15,899		
13	Максимум	596,365		
14	Сумма	7740,483		
15	Счет	26		
16	Уровень надежности(95,0%)	54,92207119		
17				
18				

9. Изучить работу с диаграммами



Задание.

1. Дать интерпретацию полученной описательной статистике для своих данных.
2. Сравнить возможности функции Анализ данных и Статистические функции. Например, функцию МЕДИАНА()



3. Построить диаграмму для своих данных.
4. Дать статистическую интерпретацию полученных результатов.

Лабораторная работа №2

Расчеты коэффициента корреляции и уравнения регрессии в пакете MS Excel

Линейная регрессия - модель зависимости одной (объясняемой, зависимой) переменной y от другой или нескольких других переменных x (факторов) с линейной функцией зависимости**.**

Простая линейная регрессия:

$$Y_i = \beta_0 + \beta_1 X_i + \varepsilon_i$$

где β_0 — сдвиг , β_1 — наклон прямой Y, ε_i — случайная ошибка переменной Y в i-м наблюдении, $M(\varepsilon_i)=0$.

В качестве оценки параметров генеральной совокупности (β_0 и β_1) можно использовать сдвиг b_0 и наклон b_1 прямой Y. Таким образом, уравнение простой линейной регрессии принимает следующий вид:

$$\hat{Y}_i = b_0 + b_1 X_i$$

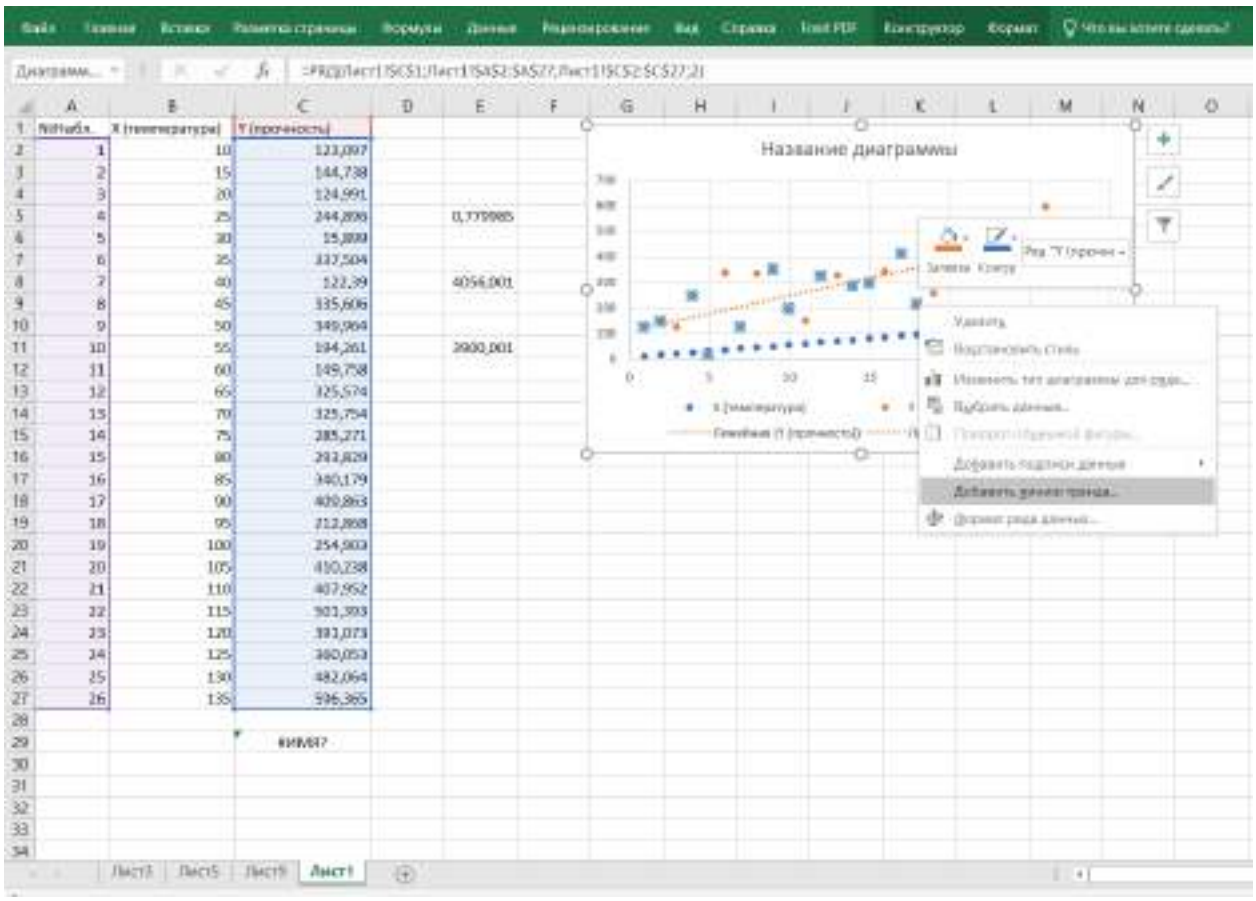
где $\hat{Y}_i$ — предсказанное значение (предиктор) переменной Y для i-го наблюдения, X_i — значение переменной X в i-м наблюдении.

Для того, чтобы определить два коэффициента регрессии — сдвиг b_0 и наклон b_1

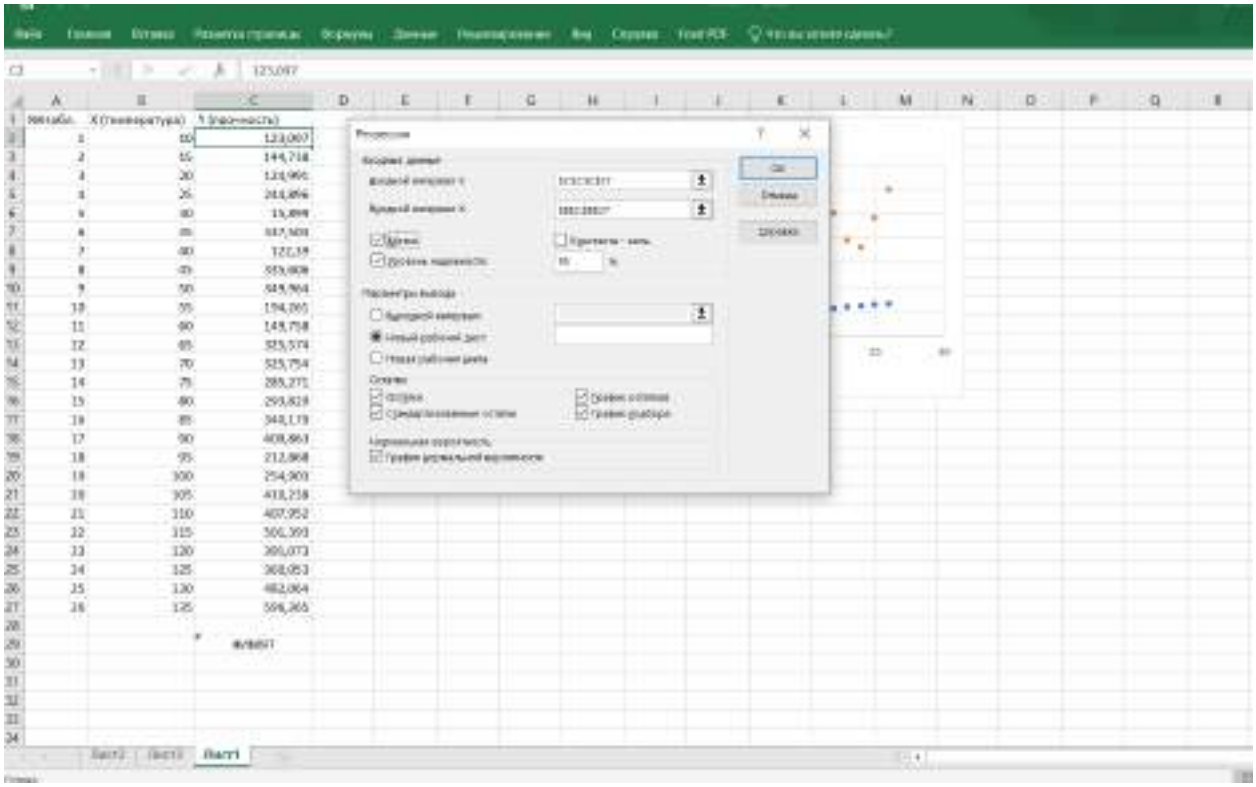
используем метод наименьших квадратов

$$\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 = \sum_{i=1}^{n}(Y_i - (b_0 + b_1 X_i))^2 \to min.$$

Для того, чтобы быстро представить аппроксимацию, достаточно кликнуть правой кнопкой мыши и выбрать Добавить линию тренда и далее Линейная.



Процедуру регрессивного анализа в MS Excel удобно выполнять, используя функцию Анализ данных.



В появившемся диалоговом окне задаем следующие параметры:

1. *Входной интервал Y* - это диапазон данных по результативному признаку. Он должен состоять из одного столбца.

2. *Входной интервал X* - это диапазон ячеек, содержащих значения факторов (независимых переменных). Число входных диапазонов (столбцов) должно быть не больше 16.

3. Флажок *Метки*, устанавливается втом случае, если в первой строке диапазона стоит заголовок.

4. Флажок *Уровень надежности* активизируется, если в поле, находящееся рядом с ним необходимо ввести уровень надежности, отличный от установленного по умолчанию. Используется для проверки значимости коэффициента детерминации R^2 и коэффициентов регрессии.

5. *Константа ноль.* Данный флажок необходимо установить, если линия регрессии должна пройти через начало координат ($a_0=0$).

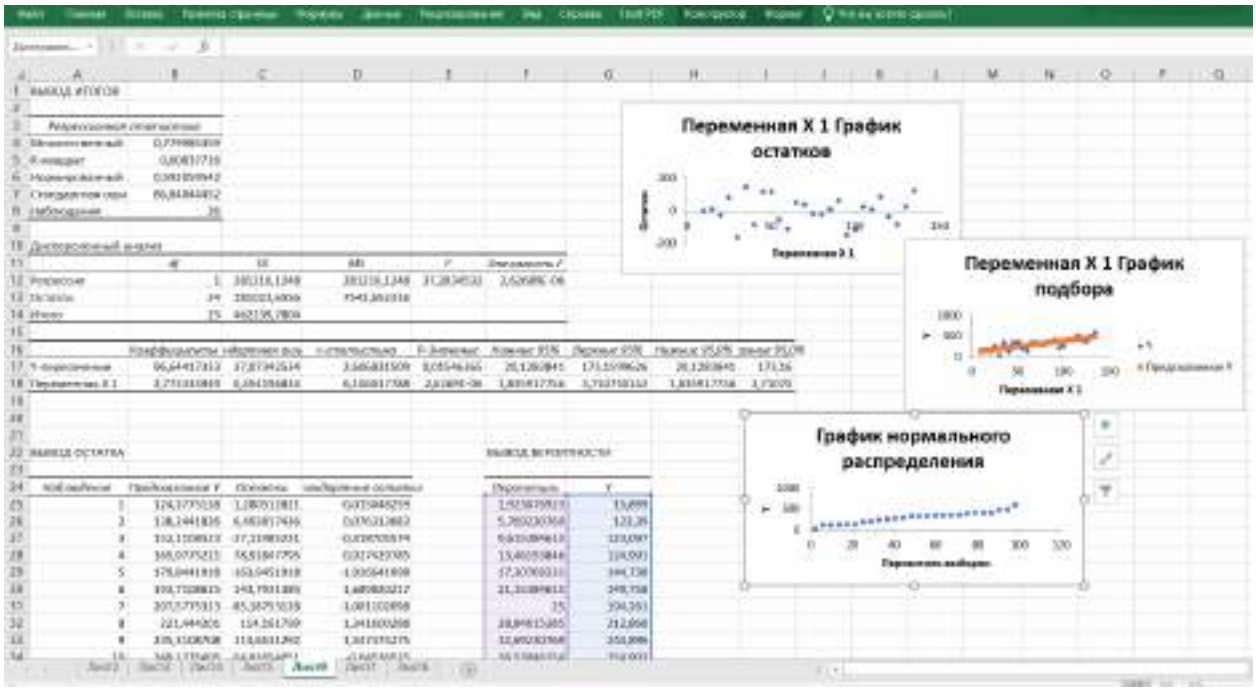
6. *Выходной интервал/ Новый рабочий лист/ Новая рабочая книга* – указать адрес верхней левой ячейки выходного диапазона.

7. Флажки в группе *Остатки* устанавливаются, если необходимо включить в выходной диапазон соответствующие столбцы или графики.

8. Флажок График нормальной вероятности необходимо сделать активным, если требуется вывести на лист точечный график зависимости наблюдаемых значений Y от автоматически формируемых интервалов персентилей.

После нажатия кнопки ОК в выходном диапазоне получаем отчет.

В результате имеем



функцию регрессии

$$Y = 96.64 + 2.77\ X.$$

Регрессионная статистика включает в себя:

- Множественный R - коэффициент множественной корреляции R - выражает степень зависимости независимых переменных X и зависимой переменной (Y). Множественный R равен квадратному корню из коэффициента детерминации, эта величина принимает значения в интервале от нуля до единицы.
- R-квадрат - это коэффициент детерминации.
- Нормированный R-квадрат . скорректированный (адаптированный, поправленный) коэффициент детерминации.
- Стандартная ошибка регрессии.
- Наблюдения - число значений наблюдаемой величины Y.

Величина *коэффициента детерминации* есть

$$R^2 = \frac{\sum_{i=1}^{n}\left(\hat{Y}_i - \overline{Y}\right)^2}{\sum_{i=1}^{n}\left(Y_i - \overline{Y}\right)^2},$$

которая показывает, насколько хорошо найденная функция регрессии описывает связь между исходными значениями факторов X и Y. Если значение R^2 близко к единице, то построенная модель хорошо аппроксимирует данные, если R^2 близкое к нулю, то качество построенной модели плохое и возможно рассмотреть один из видов нелинейной регрессии.

Недостатком коэффициента детерминации R^2 является то, что он увеличивается при добавлении новых объясняющих переменных, хотя это и не обязательно означает улучшение качества регрессионной модели. В этом смысле предпочтительнее использовать нормированный, который в отличие от R^2 может уменьшаться при введении в модель новых объясняющих переменных, не оказывающих существенное влияние на зависимую переменную.

Стандартная ошибка регрессии $S_e = \sqrt{\frac{(e_i - \bar{e})^2}{n-2}}$

дает оценку стандартного отклонения ненаблюдаемых эффектов, которые влияют на результат (или, что то же самое, оценку стандартного отклонения ненаблюдаемые эффекты, которые влияют на результат после того, как эффекты объясняющих переменных были удалены. Чем меньше стандартное отклонение ошибок e_i, тем лучше построенная модель.

Дисперсионный анализ включает в себя

дисперсионный анализ сумм квадратов SSR, SSE, SST.

Значения коэффициентов регрессии находятся в столбце Коэффициенты и соответствуют:

- Y-пересечение – это результат вычисления коэффициента b_0 , Переменная X1 - b_1, далее приведены статистики для коэффициентов, $0 \leq R^2 \leq 1$.

- t-статистика соответствующего коэффициента.
- P-Значение – вероятность определяющая значимость коэффициента регрессии. Если P-Значение<0,05, расчет коэффициента удовлетворяет

условиям модели. При Р-Значение>0,05, данный коэффициент на модель не влияет.

- Нижние 95% – Верхние 95% – доверительный интервал для параметра , т.е. с надежностью 0.95 этот коэффициент лежит в данном интервале.

Вывод остатка :

- Предсказанное Y - теоретические (расчетные) значения результативного признака.
- Остатки – остатки по модели регрессии.

Коэффициент корреляции

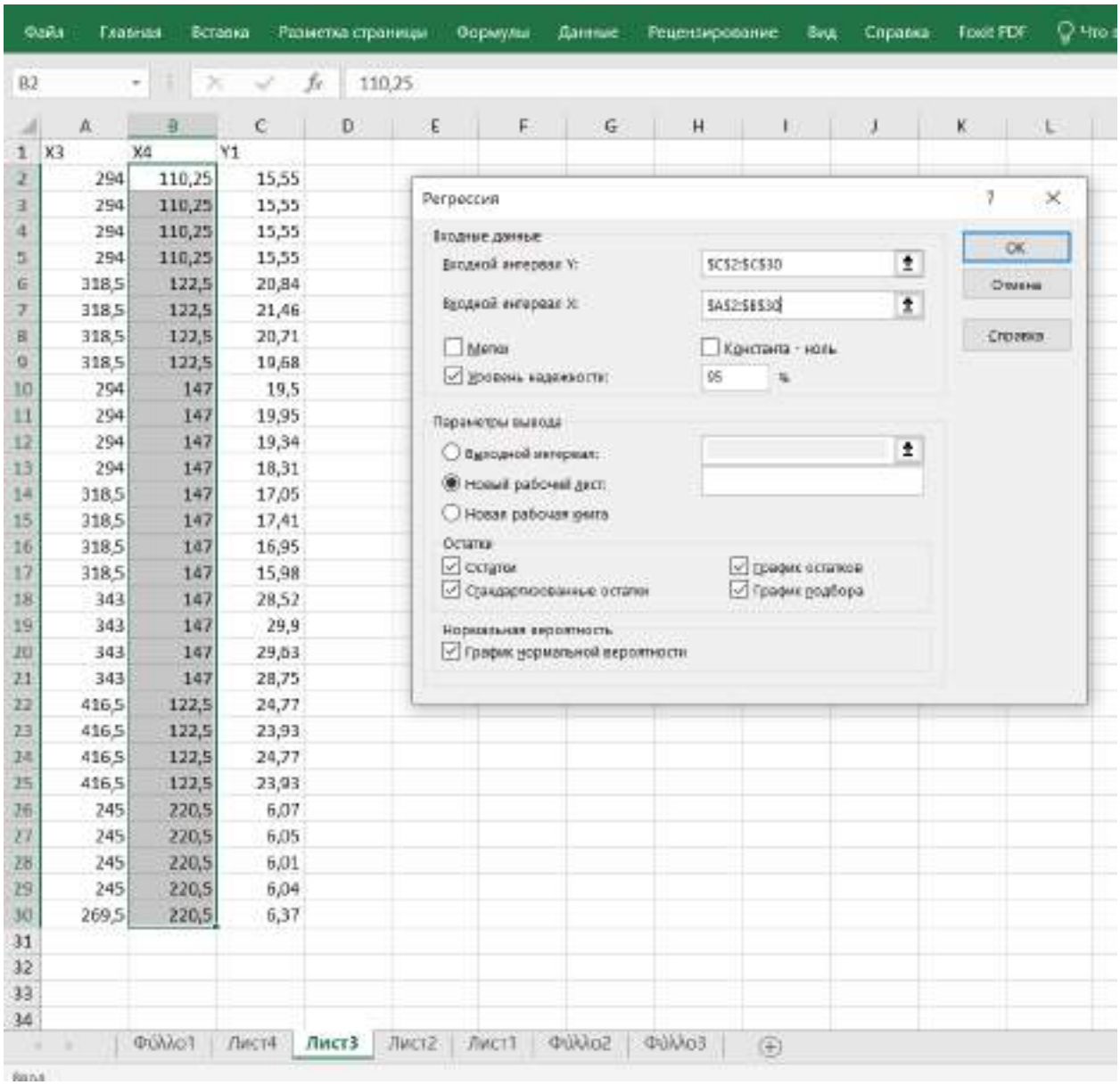
Задание.

1. Выбрать две группы данных.
2. Вычислить коэффициент корреляции между величинами X и Y. Сделать вывод о зависимости случайных величин.
3. Определить коэффициенты a, b уравнения регрессии $y = ax + b$.
4. Построить график функции. Сделать выводы.
5. Сделайте прогноз о поведении функции Y**.**

Лабораторная работа №3

Проведения многофакторного регрессионного анализа.

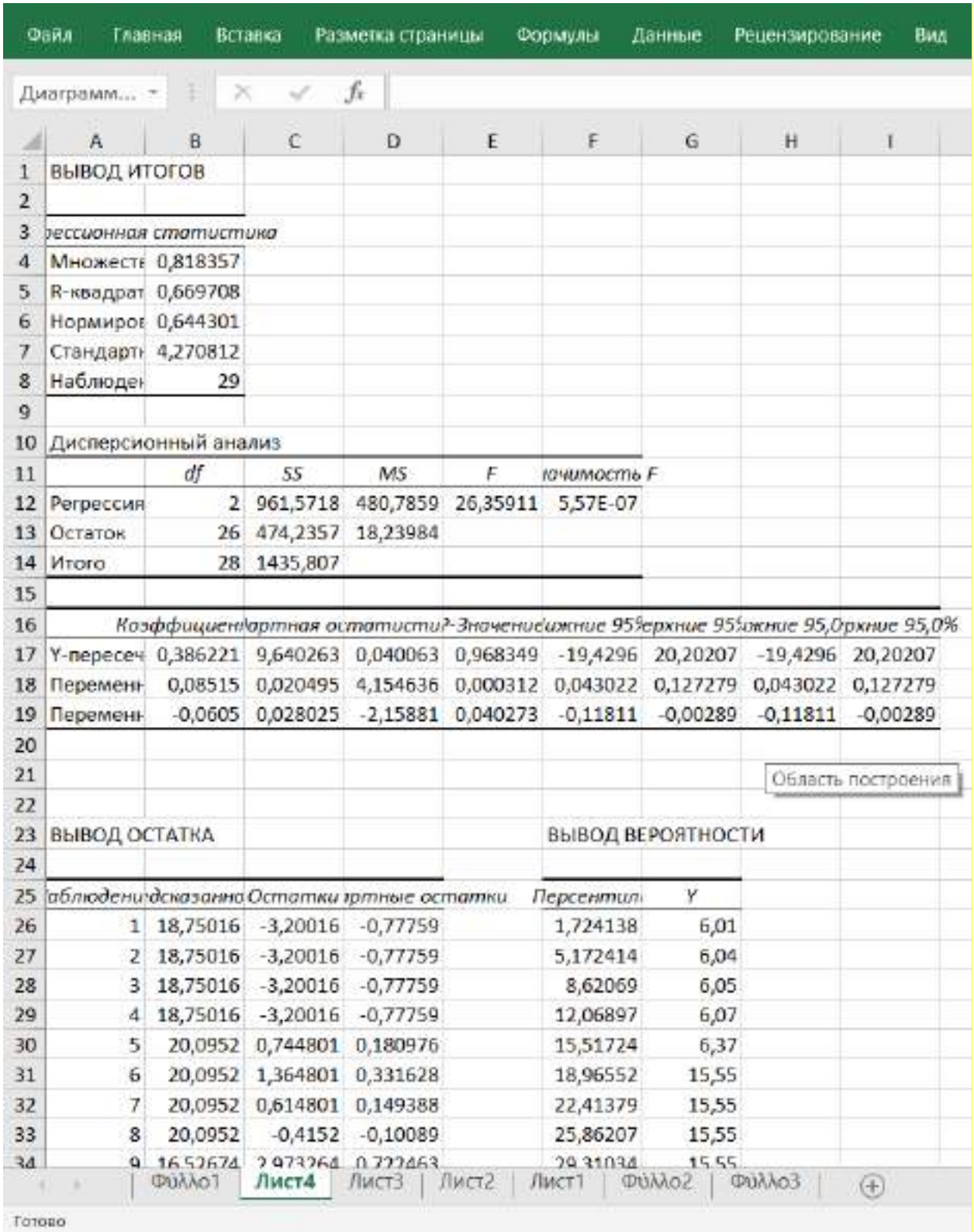
Пользуясь функцией Анализ данных имеем



	A	B	C
1	X3	X4	Y1
2	294	110,25	15,55
3	294	110,25	15,55
4	294	110,25	15,55
5	294	110,25	15,55
6	318,5	122,5	20,84
7	318,5	122,5	21,46
8	318,5	122,5	20,71
9	318,5	122,5	19,68
10	294	147	19,5
11	294	147	19,95
12	294	147	19,34
13	294	147	18,31
14	318,5	147	17,05
15	318,5	147	17,41
16	318,5	147	16,95
17	318,5	147	15,98
18	343	147	28,52
19	343	147	29,9
20	343	147	29,63
21	343	147	28,75
22	416,5	122,5	24,77
23	416,5	122,5	23,93
24	416,5	122,5	24,77
25	416,5	122,5	23,93
26	245	220,5	6,07
27	245	220,5	6,05
28	245	220,5	6,01
29	245	220,5	6,04
30	269,5	220,5	6,37

Следует обратить внимание, что при моделировании множественной регрессии Входной интервал данных охватывает все столбцы независимых переменных (в данном примере X3 и X4).

Вывод результатов



Файл Главная Вставка Разметка страницы Формулы Данные Рецензирование Вид

Диаграмм...

	A	B	C	D	E	F	G	H	I
1	ВЫВОД ИТОГОВ								
2									
3	рессионная статистика								
4	Множеств	0,818357							
5	R-квадрат	0,669708							
6	Нормиров	0,644301							
7	Стандарт	4,270812							
8	Наблюден	29							
9									
10	Дисперсионный анализ								
11		*df*	*SS*	*MS*	*F*	*ючимость F*			
12	Регрессия	2	961,5718	480,7859	26,35911	5,57E-07			
13	Остаток	26	474,2357	18,23984					
14	Итого	28	1435,807						
15									
16		*Коэффициенлартная остатисти*			*?-Значение*	*ижние 95%*	*ерхние 95%*	*ижние 95,0*	*рхние 95,0%*
17	Y-пересеч	0,386221	9,640263	0,040063	0,968349	-19,4296	20,20207	-19,4296	20,20207
18	Перемен	0,08515	0,020495	4,154636	0,000312	0,043022	0,127279	0,043022	0,127279
19	Перемен	-0,0605	0,028025	-2,15881	0,040273	-0,11811	-0,00289	-0,11811	-0,00289
20									
21									
22									
23	ВЫВОД ОСТАТКА					ВЫВОД ВЕРОЯТНОСТИ			
24									
25	*аблюдени*	*дсказанно*	*Остатки*	*ртные остатки*		*Персентил*	*Y*		
26	1	18,75016	-3,20016	-0,77759		1,724138	6,01		
27	2	18,75016	-3,20016	-0,77759		5,172414	6,04		
28	3	18,75016	-3,20016	-0,77759		8,62069	6,05		
29	4	18,75016	-3,20016	-0,77759		12,06897	6,07		
30	5	20,0952	0,744801	0,180976		15,51724	6,37		
31	6	20,0952	1,364801	0,331628		18,96552	15,55		
32	7	20,0952	0,614801	0,149388		22,41379	15,55		
33	8	20,0952	-0,4152	-0,10089		25,86207	15,55		
34	9	16,52674	2,973264	0,722463		29,31034	15,55		

Область построения

Фύλλο1 | Лист4 | Лист3 | Лист2 | Лист1 | Фύλλο2 | Фύλλο3

Готово

R-квадрат 0.67, что говорит о среднем качества подгонки модели, при этом дисперсионный анализ показывает P-значимость модели по тесту Фишера <0.05.

Очевидно, что данная модель включает три параметра и уравнение регрессии имеет вид

$Y = 0.386 + 0.085\ X1 - 0.06X2.$

Задание

1. Получить описательные статистики по каждому признаку.
2. Составить уравнение множественной регрессии, оценить его параметры пояснить их экономический смысл.
3. Проанализировать линейные коэффициенты парной и частной корреляции.
4. Оценить значения линейных коэффициентов множественной корреляции.
5. С помощью коэффициента детерминации R-квадрат и F-критерия Фишера оценить статистическую надежность уравнения регрессии в целом.

Лабораторная работа №4

Однофакторный дисперсионный анализ

Ставится задача исследования влияния некоторого фактора на изменения значений случайной величины X.

Различные значения фактора называют уровнями фактора.

Данные для задач однофакторного дисперсионного анализа обычно записывают в виде прямоугольной таблицы, каждая строка которой содержит значения x_{ij} , в столбцах – значения, полученные в разных экспериментах, т.е. – значение случайной величины, полученное при i-м уровне фактора в j-м эксперименте. Количество экспериментов на разных уровнях фактора может быть разным.

Например, таблица значений фактора может иметь Место для уравнения. вид

Э	1	2	3	4
Ф1	x_{11}	x_{12}	x_{13}	x_{14}
Ф2	x_{21}	x_{22}	x_{23}	x_{24}
Ф3	x_{31}	x_{32}	x_{33}	x_{34}

Формулируется нулевая гипотеза о том, что математические ожидания случайных величин для разных уровней фактора одинаковы.

Если гипотеза принимается, то фактор не влияет на изменчивость. В противном случае – влияет.

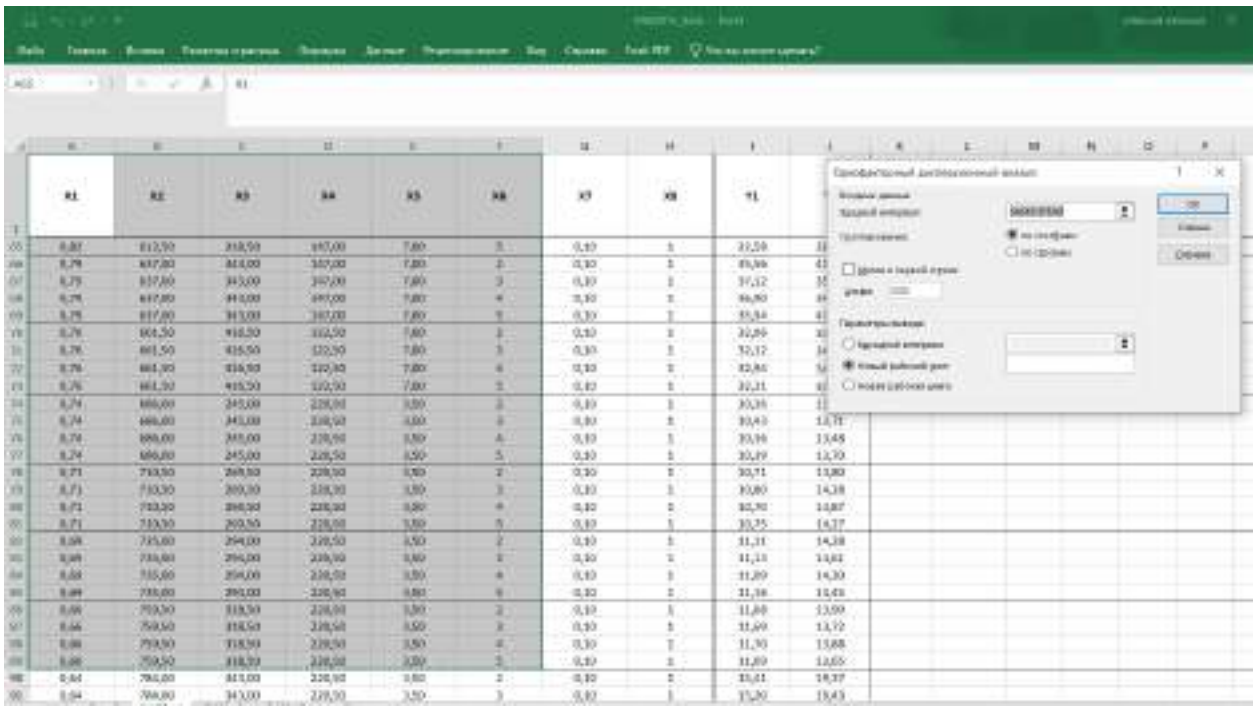
Результаты работы процедуры «Однофакторный дисперсионный анализ» выводятся в двух таблицах».

Первая из них содержит статистики для разных уровней фактора (по строкам).

Вычисляется количество значений в группе, точечные оценки математических ожиданий и дисперсий.

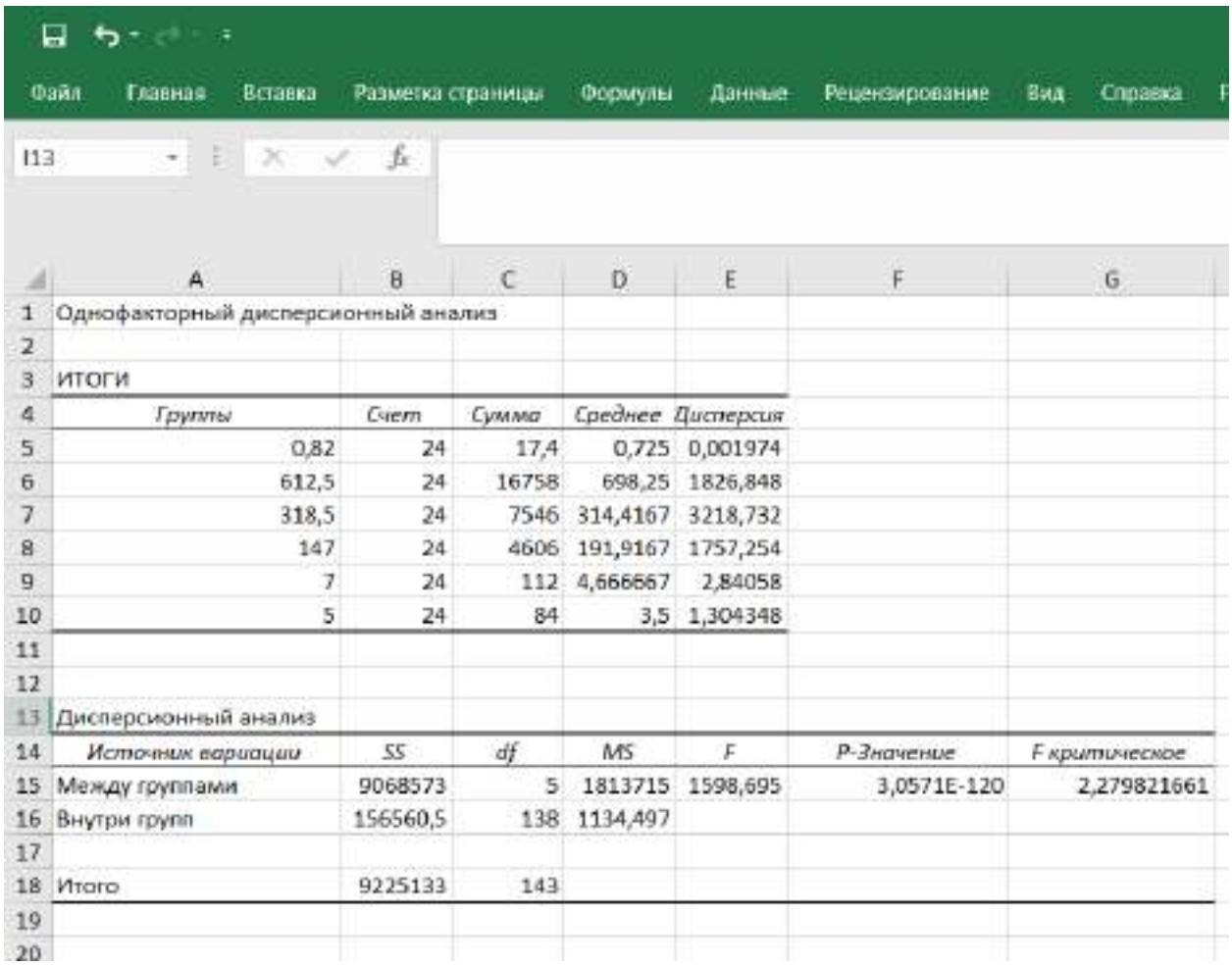
Во второй таблице, она так и называется – Дисперсионный анализ, выведены значения суммы квадратов, число степеней свободы, среднее суммы квадратов, выборочное значение критерия Фишера, вероятность этого значения, критическая точка распределения Фишера для заданного значения a.

Процедура не вычисляет коэффициент детерминации.



Данные для анализа сгруппированы по столбцам.

В таблице ниже приведены результаты расчетов.



	A	B	C	D	E	F	G
1	Однофакторный дисперсионный анализ						
2							
3	ИТОГИ						
4	*Группы*	*Счет*	*Сумма*	*Среднее*	*Дисперсия*		
5	0,82	24	17,4	0,725	0,001974		
6	612,5	24	16758	698,25	1826,848		
7	318,5	24	7546	314,4167	3218,732		
8	147	24	4606	191,9167	1757,254		
9	7	24	112	4,666667	2,84058		
10	5	24	84	3,5	1,304348		
11							
12							
13	Дисперсионный анализ						
14	*Источник вариации*	*SS*	*df*	*MS*	*F*	*P-Значение*	*F критическое*
15	Между группами	9068573	5	1813715	1598,695	3,0571E-120	2,279821661
16	Внутри групп	156560,5	138	1134,497			
17							
18	Итого	9225133	143				
19							
20							

Итак, наблюдаемое значение критерия Фишера, F = 1598.695 при этом критическое значение, Fкр= 2.279.

Нулевая гипотеза о равенстве математических ожиданий для разных уровней фактора отклоняется в пользу альтернативной.

Заключаем, что с доверительной вероятностью 0.95 математические ожидания различны.

Следовательно изменчивость случайной величины объясняется влиянием фактора.

Задание.

1. Поставить задачу для анализа. Для этого определить фактор и его уровни. Сформулировать гипотезы.
2. Сформировать данные по различным уровням фактора. Сгруппировать по столбцам.
3. Провести однофакторный дисперсионный анализ с помощью функции Анализ данных.
4. Интерпретировать полученные результаты.

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ ТЕОРИЯ СИСТЕМ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

УДК 658.01

Структуризация и оценка систем: Методические указания к лабораторным работам по дисциплине «Теория систем» (редакция 2021 г.) //

Рассмотрены особенности анализа и синтеза структур с применением методов теории графов, а также аналитические методы оценки многокритериальных систем, основанные на регрессионном анализе и использовании интегральных критериев. Приведены примеры практического применения описанных методов.

Пособие предназначено для студентов магистратуры, обучающихся по направлению «Информатика и вычислительная техника», направленность (магистерская программа) подготовки «Прикладной искусственный интеллект», а также может быть использовано для других направлений, связанных с изучением и проектированием сложных технико-экономических систем.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Изучение и проектирование системы – сложный и трудоемкий процесс. Он включает много различных работ, в том числе определение структуры системы. Для определения структуры используют формализованные и неформализованные методы. Формализованные методы основаны на математическом аппарате, а неформализованные – на опыте специалистов. Формализованные методы имеют два важных достоинства: во-первых, они объективны, то есть их результат не зависит от личности и знаний конкретного человека, а во-вторых, они могут быть реализованы средствами вычислительной техники.

Для организационных систем формализованные методы могут быть применены не всегда, но если это возможно, то следует использовать именно их. К числу таких методов, в частности, относятся методы, основанные на теории графов. Все элементы системы и связи между ними представляются в виде формализованной схемы – графа. Подсистемы, как части системы, будут в этом графе соответствовать подграфам. Рациональность выделения подграфов может быть проверена путем исследования их свойств, для этого есть соответствующие приемы. Аппарат теории графов хорош тем, что применим к системам самой различной природы. Первые две работы лабораторного цикла по дисциплине «Теория систем» посвящены изучению именно этого аппарата. В лабораторных работах он используется для изучения свойств системы и ее структуризации.

После проектирования и пуска системы в эксплуатацию важно правильно оценить ее работу. Для этого также применяют формализованные и неформализованные методы. Состояние системы оценивают по специальным правилам – критериям. При использовании формализованных методов критерии строятся на основе математических и логических приемов, при использовании неформализованных – на основе мнений специалистов.

Для принятия правильных решений важно определить, как будет вести себя система в будущем, то есть решить задачу прогнозирования. Для этого также созданы формализованные методы, основанные на математике. Две последние работы лабораторного цикла направлены на изучение методов прогнозирования и оценки состояния систем: метода наименьших квадратов, весового, минимаксного и паретовского критериев. Для выполнения всех лабораторных работ понадобится знание программирования, коробочных программных продуктов, облачных информационных сервисов.

ЛАБОРАТОРНАЯ РАБОТА №1.
ИССЛЕДОВАНИЕ СИСТЕМЫ ГРАФОАНАЛИТИЧЕСКИМ МЕТОДОМ

Цель работы: изучить графоаналитический метод исследования графов, практически использовать его для анализа свойств системы. В качестве системы рассматривается поток информации, представленный ориентированным графом.

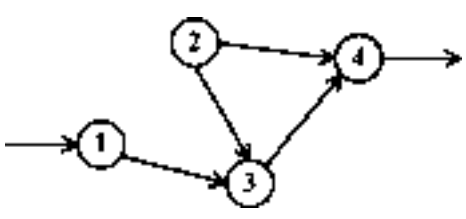
Методические указания

Информационный поток, или поток данных — это последовательность данных, передаваемых в системе управления от источника к приемнику. Источниками и приемниками данных могут быть как технические средства, так и люди. Исследование потоков информации проводят с целью их совершенствования. Оно заключается в улучшении форм бумажных или электронных документов, в сокращении их числа, оптимизации маршрутов движения документов и алгоритмов их формирования.

Распространенным методом, позволяющим применять для анализа вычислительную технику, является графоаналитический метод. Он основан на построении информационного графа и анализе его матрицы смежности, результатов возведения ее в степень, матрицы достижимости.

Информационный граф строится следующим образом. Элементы потока информации (документы, показатели) сопоставляются вершинам графа, каждая пара вершин соединяется дугой или ребром в том случае, если переход между ними осуществляется без каких-либо промежуточных результатов. Информационный граф является ориентированным, так как информационные потоки всегда имеют направление.

Пользуясь известными свойствами графов, можно выявить ряд важных характеристик системы. Для этого используется матрица смежности, т. е. матрица, отражающая наличие связей между вершинами графа. Элемент матрицы a_{ij}, стоящей на пересечении i-ой строки и j-го столбца равен единице, если из вершины x_i в вершину x_j проведена дуга (или ребро), и равен нулю в противоположном случае. На рисунке 1,а,б для примера показаны ориентированный граф и его матрица смежности $\boldsymbol{A}$.



а

$\boldsymbol{A}$

i \ j	1	2	3	4
1			1	
2			1	1
3				1
4				

$\boldsymbol{A^2}$

i \ j	1	2	3	4
1				1
2				1
3				
4				

$\boldsymbol{A}^3=0$

б

Рисунок 1 – Граф системы (а) и его матрицы A^L (б)

Для удобства матрица представлена в форме таблицы. В ней проставлены только единицы, пустые клетки предполагают нули.Для выявления основных свойств графов, кроме матрицы $\boldsymbol{A}$, используются матрицы $\boldsymbol{A^L}$, полученные возведением $\boldsymbol{A}$ в степень L, а также матрица достижимости $\boldsymbol{D}$, образованная их суммированием:

$$D = \sum_{L=1}^{n} A^L ,$$

где n — размерность матрицы смежности.

Элемент $a_{ij}^{(2)}$ матрицы $\boldsymbol{A^2}$, стоящий на пересечении i–ой строки и j–го столбца, вычисляется как произведение i–ой строки матрицы смежности на её j–ый столбец: $a_{ij}^{(2)} = \begin{pmatrix} a_{i1} & a_{i2} & \cdots & a_{in} \end{pmatrix} \begin{pmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{nj} \end{pmatrix}$

или:

$$a_{ij}^{(2)} = a_{i1}a_{1j} + a_{i2}a_{2j} + \ldots + a_{in}a_{nj} .$$

Аналогично выполняется дальнейшее возведение в степень. Матрица $\boldsymbol{A^L}$ показывает, сколько путей длиной в L тактов (шагов продвижения от вершины к вершине графа) имеется между двумя любыми вершинами x_i, x_j. Количество этих путей определяется цифрой $a_{ij}^{(L)}$ на пересечении i–ой строки и j–го столбца $\boldsymbol{A^L}$. Матрица $\boldsymbol{D}$ определяет, сколько всего путей, независимо от их длины, соединяет две любые вершины графа.

Для ориентированного графа количество матриц $\boldsymbol{A^L}$ определяется его порядком, который зависит от порядка элементов графа. Порядок p_j элемента x_j формально определяется соотношением:

$$p_j = L \quad \text{при} \quad \sigma_j(L) > 0 \quad \text{и} \quad \sigma_j(L+1) = 0 ,$$

где $\sigma_j(L)$ — сумма элементов j–го столбца матрицы $\boldsymbol{A^L}$.

Физический смысл порядка элемента таков: это номер такта, на котором элемент готов к своему формированию, т.е. поступили все нужные для этого компоненты. Входные (исходные) элементы графа имеют нулевой порядок. На первом такте из них образуются элементы первого порядка. На втором такте из элементов нулевого и первого порядка образуются элементы второго порядка и т.д.

Порядок информационного графа P определяет самый длинный путь в графе и совпадает с максимальным порядком его элементов: $P = \max_j p_j$.

Для P справедливо соотношение:

$$A^P \neq 0, \quad A^{P+1} = 0,$$

например, для графа, изображенного на рисунке 1,а, $P = 2$.

Анализ графа проводится обычно в следующей последовательности.

- **Выявление ошибок построения графа.** При построении граф-схемы потока данных в неё ошибочно могут быть включены элементы, не имеющие к ней отношения (не связанные с другими элементами графа). Таким элементам в матрице $\boldsymbol{A}$ соответствуют строка и столбец, заполненные нулями. Для информационного графа ошибкой является наличие контуров.

- **Определение входных и выходных элементов.** Правила для нахождения контуров, входных и выходных элементов приведены в методических указаниях к лабораторной работе №1.

- **Определение диаметра графа.** Если d_{ij} — длина минимального пути между входным элементом x_i и выходным x_j, выраженная числом дуг, составляющих этот путь, то диаметр графа d определяется выражением d=max d_{ij}. Таким образом, диаметр — это максимальный из минимальных путей, соединяющих во всех возможных сочетаниях входные и выходные вершины графа. Для информационных потоков он характеризует их временную задержку (в тактах), т.е. время, которое пройдёт от момента подачи исходных данных до начала формирования результата (документа, показателя, выходного потока). Диаметр определяется по матрицам $\boldsymbol{A^L}$. Чтобы не ошибиться в определении диаметра графа, нужно придерживаться следующей методики. Организуется перебор выходных вершин графа, и для каждой выходной вершины анализируются пути, ведущие к ней из входных вершин. В результате анализа для каждой выходной вершины находится минимальный путь. Из минимальных путей, найденных для выходных вершин, выбирается максимальный, это и есть диаметр графа.

- **Определение порядка элементов графа.** По матрицам $\boldsymbol{A^L}$ можно определить порядок каждого элемента графа. Для этого последовательно просматриваются столбцы с одинаковыми номерами во всех матрицах $\boldsymbol{A^L}$, начиная с первой (L = 1). При обнаружении столбца с нулевым содержимым элементу присваивается порядок, на единицу меньший степени последней просмотренной матрицы. Когда порядки всех элементов определены, можно построить информационный граф, упорядоченный по тактам. Для этого вершины с одинаковыми порядками располагаются в отдельных зонах, а затем соединяются дугами так же, как в исходном графе. Вид графа, изображенного на рисунке 1, после упорядочения по тактам показан на рисунке 2. Упорядоченный граф позволяет более наглядно

представить схему потока, визуально оценить очерёдность формирования его элементов.

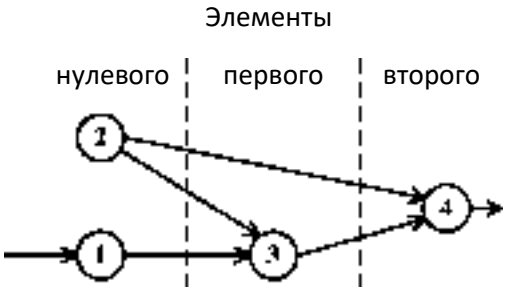


Рисунок 2 – Информационный граф, упорядоченный по тактам

- **Определение относительных длительностей формирования элементов.** Для определения времени, необходимо для формирования каждого элемента (в тактах) относительно формирования любого другого элемента, пользуются матрицами $\boldsymbol{A^L}$. Это время зависит от числа шагов (от числа дуг), соединяющих два выбранных элемента. Например, для графа на рисунке 2 можно видеть, что вершины 2 и 4 соединены двумя путями: длиной в один и в два такта (единицы на пересечении второй строки и четвертого столбца в матрицах $\boldsymbol{A}$ и $\boldsymbol{A^2}$ на рисунке 1). Значит, элемент 4 будет сформирован не через один, а через два такта после элемента 2. Формально для определения времени формирования последующего элемента x_k относительно некоторого предыдущего элемента x_m нужно найти разность их порядков ($p_k - p_m$).

- **Определение использования элементов потока.** При анализе потоков данных важно знать, какие показатели или документы используются для формирования других показателей или документов, и каких именно. Об этом говорит матрица $\boldsymbol{D}$: отличные от нуля элементы j-го столбца этой матрицы указывают все элементы, участвующие в формировании элемента x_j, а ненулевые элементы i-ой строки $\boldsymbol{D}$ указывают все элементы, при формировании которых используется элемент x_i. Так, чтобы сформировать элемент 3 графа на рисунке 2, нужно воспользоваться результатами формирования элементов 1 и 2, а элемент 2 участвует в формировании элементов 3 и 4. Об этом говорит матрица достижимости графа, приведенная на рисунке 3.

$\boldsymbol{D}$

i	j			
1			1	1
2			1	2
3				1
4				

Рисунок 3 – Матрица достижимости графа по рисунку 2

Программа работы и задания

При выполнении работы студенты получают задания у преподавателя. Информационный поток для исследования задается граф-схемой. В качестве примера информационного графа можно использовать граф, описанный таблицей А.1 Приложения А (без данных, помеченных жирным шрифтом). Для заданного варианта графа студенты должны

- построить матрицу смежности,
- с помощью указанной преподавателем вспомогательной программы ввести матрицу смежности в компьютер. Для этого, следуя подсказкам программы, ввести номер варианта и размерность матрицы смежности, а затем ввести саму матрицу, пользуясь только клавишами “0”, “1” и “Backspace” (“Забой”). Задачей программы является возведение матрицы смежности в степень, получение матрицы достижимости и определение диаметра графа. Результаты выводятся в файл, который затем распечатывается известными средствами. Местоположение и имя файла задаются в диалоге с программой. Возведение матрицы смежности в степень идёт до тех пор, пока очередная матрица не станет нулевой. Если матрица смежности введена в программу с ошибками или если исходный граф имеет контуры, программа выдаёт сообщение об ошибке. В этом случае нужно исправить ошибки и повторить ввод данных до получения безошибочного результата. Для устранения контуров их следует отыскать и разорвать, устранив ошибочную, нелогичную (с точки зрения студента) дугу,
- пользуясь полученными данными, произвести анализ:

– выявить и перечислить ошибки составления схемы потока данных;

– найти и указать входные и выходные элементы потока;

– определить порядок элементов, найти порядок информационного графа, упорядочить граф по тактам и изобразить его в таком виде;

– определить время задержки начала формирования и время окончания формирования выходного потока данных;

– определить, какие элементы используются для формирования каждого элемента потока и для формирования каких элементов используется каждый элемент потока, результат представить в табличной форме;

– определить (с пояснениями) диаметр графа по его рисунку, проверить по распечатке правильность определения величины диаметра вручную.

Примеры выполнения некоторых пунктов задания приведены в Приложении А.

Контрольные вопросы

1. Как найти контур в схеме информационного потока? Предложите алгоритм.

2. Предложите алгоритм нахождения входных (выходных) элементов информационного потока на ЭВМ.

3. Как определить порядок заданного элемента графа? Разработайте алгоритм.

4. Как определить время, необходимое для формирования любого элемента информационного потока относительно любого другого? Предложите алгоритм.

5. При реализации некоторого элемента информационного потока была зафиксирована ошибка. Как определить, какие элементы могли стать ее причиной, на какие элементы ошибка могла повлиять? Предложите алгоритмы.

6. Каков физический смысл порядка информационного графа?

7. Почему контур в граф-схеме информационного потока является ошибкой?

8. Каков физический смысл матрицы достижимости? Докажите правильность Вашего понимания.

9. Покажите аналитически, что элемент a_{ij} матрицы A^2 действительно должен показывать число путей длиной в 2 такта из вершины x_i в вершину x_j информационного графа.

10. Определите относительную длительность формирования одного элемента информационного потока относительно другого (по заданию преподавателя).

11. В чём достоинства графоаналитического метода исследования информационных потоков?

ЛАБОРАТОРНАЯ РАБОТА №2. СТРУКТУРИЗАЦИЯ СИСТЕМЫ

Цель работы: изучить подход к структуризации системы на основе принципа агрегирования, получить навыки экспертного и формализованного разбиения системы на подсистемы, оценить эффективность разбиения по критерию структуризации.

Методические указания

Структуризация предполагает выделение подсистем в составе системы и установление связей между ними. Согласно принципу агрегирования за подсистему принимается группа таких элементов системы,

которые связаны между собой гораздо теснее, чем с элементами других таких групп. Названный признак подсистемы характерен для реальных экономических систем. Например, работники одного подразделения любой организации взаимодействуют между собой гораздо интенсивнее, чем с работниками соседнего подразделения.

Выделение подсистем может выполняться на основе мнений специалистов (неформализованный подход) и математических методов (формализованный подход). Одним из формализованных методов структуризации является метод выделения сильно связных подграфов. Элементы системы и их связи представляются в виде ориентированного графа. Далее он разбивается на подграфы, в которых между двумя любыми вершинами есть путь. Это и есть сильно связные подграфы (ССП).

Методика структуризации на основе выделения ССП такова:

- выбирается некоторая начальная вершина графа x_j,
- находится множество вершин графа $\boldsymbol{R}_j$, достижимое из x_j (достижимое множество). Достижимое множество для вершины x_j может быть установлено по ненулевым элементам j-ой строки матрицы достижимости графа,
- находится множество вершин графа $\boldsymbol{Q}_j$, из которых может быть достигнута вершина x_j (контрдостижимое множество). Контрдостижимое множество для вершины x_j может быть установлено по ненулевым элементам j-го столбца матрицы достижимости графа,
- находится пересечение множеств $\boldsymbol{R}_j \cap \boldsymbol{Q}_j$ (взаимодостижимое множество), которое принимается за ССП. На вершины взаимодостижимого множества указывают ненулевые элементы j-ой строки и j-го столбца матрицы достижимости, стоящие в одинаковых позициях. Найденному ССП ставится в соответствие подсистема,
- ССП удаляется из графа системы. На практике удаление означает запрет на использование номеров вершин, вошедших в ССП,
- выбирается одна из оставшихся вершин графа, которая принимается за новую начальную вершину. После этого процесс повторяется сначала и идет до тех пор, пока множество вершин графа не станет пустым.

В результате применения методики возникает вариант разбиения графа на ССП.

При формализованном или неформальном выделении подсистем нужно соблюдать два правила:

1) количество элементов в подсистеме должно быть ограничено снизу и сверху,
2) должны быть рассмотрены все возможные варианты разбиения графа.

После нахождения альтернативных вариантов структуризации системы необходимо выбрать оптимальный вариант. Для этого используют правило для проверки структуры на оптимальность – критерий структуризации. В качестве критерия может выступать минимизация числа связей или суммарной мощности связей между подсистемами. Альтернатива, прошедшая по критерию, выбирается в качестве окончательного результата структуризации.

Программа работы и задания

При выполнении работы студенты получают задания у преподавателя. В качестве примера задания на структуризацию можно использовать описание системы из таблицы А.1 Приложения А. ***Используются все данные*****, в том числе отмеченные жирным шрифтом и подчеркиванием.** В процессе выполнения работы студенты должны

- по описанию системы получить ее ориентированный граф, используя результаты лабораторной работы №1;
- написать программу выделения различных вариантов ССП на основе обработки матрицы ***D***. Для ввода данных в компьютер и получения ***D*** используется программа, подготовленная в лабораторной работе №1. Результаты выделения альтернативных вариантов ССП вывести на печать;
- выступая в роли эксперта, выделить подсистемы в графе вручную. Чтобы облегчить выделение подсистем, имеет смысл перерисовать граф, сгруппировав элементы, тесно связанные между собой. Изобразить предполагаемые границы подсистем при экспертной структуризации и при выделении ССП (на разных рисунках);
- определить оптимальный вариант структуризации для неформализованного и формализованного подхода по критерию. Для заданий с нечетным номером использовать минимизацию количества связей между подсистемами, для заданий с четным номером – минимизацию суммарной мощности связей между подсистемами. Сравнить результаты экспертной и формализованной структуризации;
- сделать выводы по работе.

Результаты выполнения некоторых пунктов задания на структуризацию системы приведены в Приложении А.

Контрольные вопросы

1. Что такое структуризация? Выполнение каких работ предполагает решение этой задачи?
2. В чем суть принципа агрегирования?
3. Дайте определение сильно связного подграфа.
4. Какова методика экспертной структуризации?

5. Какова последовательность выделения подсистем на основе сильно связных подграфов?

6. На чем основана возможность применения ССП для выделения подсистем?

7. Назовите разновидности критериев структуризации.

8. Выделите подсистемы в структуре, предложенной преподавателем.

ЛАБОРАТОРНАЯ РАБОТА №3.
ПРОГНОЗИРОВАНИЕ СОСТОЯНИЯ СИСТЕМЫ

Цель работы: изучить способ выявления тренда системы путем сглаживания методом наименьших квадратов, определить состояние экономической системы на заданный момент времени в будущем.

Методические указания

Чтобы спрогнозировать поведение экономической системы в будущем, используют знания о ее поведении в прошлом. Зная значения некоторого параметра системы в определенные моменты времени, можно выявить тенденцию изменения этого параметра – тренд. Далее можно предположить, что в недалеком будущем эта тенденция не изменится. Тогда точки, взятые на линии тренда в будущем, с некоторой погрешностью дадут ожидаемое состояние параметра. Тренд описывается в аналитической форме (уравнением регрессии), поэтому задача прогнозирования превращается в задачу экстраполяции, то есть нахождение значений функции в точках, лежащих вне отрезка, на котором поведение функции известно.

Главная задача прогнозирования – объективное нахождение тренда. Выявлять тренд можно по различному числу отсчетов: можно по всем имеющимся, а можно – по последним отсчетам. Для решения этой задачи используют различные методы сглаживания известных отсчетов функции. Одним из распространенных методов сглаживания является метод наименьших квадратов.

Пусть есть экспериментальная зависимость $y = f(x)$. Тренд отыскивается в форме полинома $\bar{y} = f(x)$. Его график должен как можно ближе проходить к точкам экспериментальной зависимости (x_i ,y_i). В качестве критерия близости берется квадрат отклонения линии тренда от известных значений на всем интервале сглаживания. Этот квадрат отклонения *w* минимизируют:

$$w = \sum_{i=1}^{N} (y_i - \underline{y}_i)_2 \to \min .$$

Тренд может быть линейным, квадратическим или иным, в общем виде его можно описать так:

$$\bar{y} = a_0 x^n + a_1 x^{n-1} + \ldots + a_{n-1} x + a_n .$$

Обычно используют полиномы степени 1 и 2.

В приведенном уравнении нужно найти коэффициенты $a_0,a_1,\ldots,a_n$. Подход к определению всех коэффициентов один и тот же: дифференцируя w по $a_0,a_1,\ldots,a_n$ и приравнивая производные нулю (условия экстремума), получают $(n+1)$ уравнение с $(n+1)$ неизвестным. Например, для тренда в виде квадратического полинома:

$$\bar{y} = a_1 x^2 + a_2 x + a_3,$$

$$w = \sum_{i=1}^{N} (y_i - a_1 x_i^2 - a_2 x_i - a_3)^2,$$

$$\frac{\partial w}{\partial a_1} = 0, \qquad \frac{\partial w}{\partial a_2} = 0, \qquad \frac{\partial w}{\partial a_3} = 0.$$

Члены перегруппировываются, и получается система уравнений:

$$\begin{cases} a_1 \sum_{i=1}^{N} x_i^4 + a_2 \sum_{i=1}^{N} x_i^3 + a_3 \sum_{i=1}^{N} x_i^2 = \sum_{i=1}^{N} y_i x_i^2, \\ a_1 \sum_{i=1}^{N} x_i^3 + a_2 \sum_{i=1}^{N} x_i^2 + a_3 \sum_{i=1}^{N} x_i = \sum_{i=1}^{N} y_i x_i, \\ a_1 \sum_{i=1}^{N} x_i^2 + a_2 \sum_{i=1}^{N} x_i + a_3 N = \sum_{i=1}^{N} y_i . \end{cases}$$

где N – число отсчетов.

В системе уравнений коэффициенты при a_1,a_2,a_3 получаются из известных экспериментальных данных и могут быть рассчитаны компьютером. Тогда система может быть решена хотя бы методом подстановки. Получится уравнение тренда. В него можно подставлять любые значения аргумента и находить прогнозные данные. Кстати, в случае линейного тренда можно использовать ту же систему уравнений, приняв в ней $a_1=0$:

$$\begin{cases} a_2 \sum_{i=1}^{N} x_i^2 + a_3 \sum_{i=1}^{N} x_i = \sum_{i=1}^{N} y_i x_i, \\ a_2 \sum_{i=1}^{N} x_i + a_3 N = \sum_{i=1}^{N} y_i . \end{cases}$$

Процесс отыскания коэффициентов $a_0,a_1,\ldots,a_n$ легко алгоритмизируется и автоматизируется. При тестировании программы нахождения

коэффициентов можно рекомендовать следующую последовательность действий. В качестве экспериментальных данных можно взять координаты точек, лежащих на кривой известной формы, например, на параболе. Для этого в качестве отсчетов аргумента имеет смысл брать числа, облегчающие расчеты, например, x=0,1,2…, и подставлять их в уравнение параболы. Полученные значения функции можно, при желании, несколько изменить, чтобы придать им "более случайный" вид. Ясно, что коэффициенты уравнения тренда, полученные по таким исходным данным, должны практически совпасть с коэффициентами параболы, что и будет признаком верной программы.

Программа работы и задания

Поведение экономической системы задается набором отсчетов ее параметров. Вариант задания приведен в таблице А.2 Приложения А. При выполнении работы студенты получают варианты заданий у преподавателя. Необходимо спрогнозировать состояние всех параметров каждой системы "на один шаг" вперед, то есть на один шаг дискретизации аргумента за пределами известных отсчетов. Для вычисления значений тренда нужно написать программу на выбранном языке программирования или средствами математического пакета прикладных программ.

Последовательность выполнения работы следующая:

- используя приведенную в методических указаниях систему уравнений, получить выражения для расчета коэффициентов тренда для линейного и параболического сглаживания. Вывод выражений описать в отчете по лабораторной работе;
- написать процедуру сглаживания отсчетов методом наименьших квадратов, процедуру протестировать. Тренд получить в форме прямой и кривой второго порядка (параболы);
- написать процедуру, вычисляющую значения параметров системы для заданного интервала аргумента по выражениям тренда. Процедуру протестировать;
- рассчитать сглаженные значения параметров всех систем, описанных в задании, внутри известного интервала аргумента и "на один шаг" вперед. Использовать как линейное, так и параболическое сглаживание;
- изобразить графики тренда и известные отсчеты для всех параметров одной системы. Сделать вывод о сглаживающей способности метода наименьших квадратов в случае линейного и параболического сглаживания.

Прогнозирование одного из параметров системы на основе использования параболического тренда показано в Приложении А.

Контрольные вопросы

1. В чем суть задачи прогнозирования? На каком предположении основываются формализованные методы прогнозирования?

2. Что такое тренд и как его получить?

3. В чем состоит сглаживание методом наименьших квадратов?

4. Предложите оценку погрешности прогнозирования на основе сглаживающих функций. Определите погрешность прогнозирования по Вашему заданию для заданных преподавателем параметров.

5. Какой характер примет сглаживание, если в системе уравнений, служащих для нахождения коэффициентов тренда, положить $a_1 = a_2 = 0$?

6. В работе использовано линейное и параболическое сглаживание. Предложите другую форму сглаживающей линии.

7. Как Вы думаете, на что повлияет уменьшение числа отсчетов, принимаемых во внимание при составлении уравнения регрессии?

8. Найдите прогнозное значение заданного преподавателем параметра, использовав линейное сглаживание по двум последним известным отсчетам этого параметра (можно графически). Сравните результат со значениями, полученными в работе.

9. Предложите методику прогнозирования параметра, известные значения которого имеют 1 – 2 резких выброса. Разработайте алгоритм.

10. Спрогнозируйте среднюю температуру первой недели апреля, используя средние температуры четырех недель марта. Данные приведены в таблице 1. Вид сглаживания выберите самостоятельно. Оцените погрешность прогноза.

Таблица 1 – Средние температуры марта

Номер недели	1	2	3	4
Средняя температура, ^oC	-2	-3	0	+1

ЛАБОРАТОРНАЯ РАБОТА №4.
ОЦЕНКА СИСТЕМ С ПОМОЩЬЮ ФОРМАЛИЗОВАННЫХ КРИТЕРИЕВ

Цель работы: изучить минимаксный, весовой и паретовский критерии эффективности системы, оценить эффективность функционирования экономической системы с помощью формализованного критерия.

Методические указания

Чтобы оценить эффективность экономической системы, нужно выделить наиболее важные для нее параметры и проанализировать, насколько они близки к желаемым значениям. Сложные системы

характеризуются многими параметрами, причем зачастую у одних систем близки к оптимуму одни параметры, у других – другие. В таких условиях непросто оценить системы без специального формализованного правила – критерия. Экономические системы, как правило, являются многокритериальными. Это означает, что оценить их эффективность по одному параметру не удается. Для таких систем используют интегральные (составные) критерии, объединяющие несколько простых (частных) критериев. Различных критериев много, и различаются они своей задачей. Одни критерии помогают выявить системы, наилучшие по всем параметрам, другие – системы, характеризующиеся минимальными потерями, третьи – системы, которые не хуже других систем и т.д. В лабораторной работе используются минимаксный, весовой и паретовский критерии.

Максимизация или минимизация отдельного *i*-го параметра системы – это частный критерий k_i. Для него существует некоторое наилучшее (оптимальное) значение k_i^o, к которому нужно стремиться. Оптимальным полагается минимально достижимое значение для минимизируемого частного критерия и максимально достижимое для максимизируемого. Минимаксный критерий помогает выбрать наилучшую систему s^* таким образом: для s^* максимальное отклонение частных критериев от их оптимальных значений минимально, то есть

$$k(s^*) = \min_{s \in S} \left(\max_i \left| k_i^o - k_i \right| \right),$$

где $k(s^*)$ – значение критерия для s^*;

$\boldsymbol{S}$ – множество сравниваемых систем.

Поскольку абсолютные значения частных критериев могут очень сильно различаться, для приведения к одному диапазону осуществляется деление их на граничные значения:

$$\bar{k}_i = \frac{k_i}{k_{i\max}}, \qquad \bar{k}_i^o = \frac{k_i^o}{k_{i\max}}.$$

Если этого не сделать, параметр с большим значением поглотит параметр с малым значением (эффект компенсации).

Описанный способ построения критерия не учитывает значимость (ранг, вес) частных критериев. Их учитывает весовой критерий:

$$k = \sum_{i=1}^{n} \alpha_i \cdot k_i ,$$

где n – число частных критериев, k_i – их значения для оцениваемой системы, а α_i – их весовые коэффициенты (веса). Для устранения компенсации одного

частного критерия другим их значения используются в приведенной к максимуму форме. Наилучшая система выбирается по максимальному значению критерия:

$$k(s^*) = \max_{s \in S} \sum_{i=1}^{n} \alpha_i \frac{k_i}{k_{i\max}}.$$

Весовые коэффициенты назначаются экспертами и берутся положительными для максимизируемых и отрицательными для минимизируемых частных критериев. Применяют два подхода к выбору величин этих коэффициентов:

1. в качестве коэффициентов используют дробные числа, сумма которых для каждой системы равна единице,
2. используют целые коэффициенты, причем для наименее важного частного критерия берут единичный коэффициент, а для остальных берут коэффициенты, кратные единице.

В любом случае более важному частному критерию соответствует больший по абсолютному значению коэффициент.

Паретовский критерий назван по имени итальянского экономиста Вильфредо Парето. Этот критерий служит для нахождения множества систем (паретовского множества), каждая из которых не превосходится по всем параметрам ни одной другой из этого множества.

Каждая альтернативная система s_i характеризуется своими значениями выходных параметров:

$$y_1(s_i), y_2(s_i), \ldots, y_n(s_i).$$

Если частные критерии заключаются в минимизации выходных параметров, то эффективной по Парето будет система s^* $(s^* \in S)$, если не существует системы $\tilde{s}$ $(\tilde{s} \in S)$ такой, что

$$y_j(\tilde{s}) \le y_j(s^*)$$ для всех j.

Если частные критерии заключаются в максимизации выходных параметров, то в последнем соотношении следует заменить знак "меньше" на знак "больше".

Методика отыскания паретовского множества

1. Организуется перебор альтернативных систем.

2. Для каждой очередной системы решается вопрос: оставлять ее в паретовском множестве или нет.

3. Для этого очередная система сравнивается с остальными. Если среди остальных систем найдется хотя бы одна, превосходящая рассматриваемую

систему по ВСЕМ параметрам, то рассматриваемая система НЕ включается в паретовское множество, иначе – включается.

4. Если часть частных критериев заключается в минимизации выходных параметров системы, а другая часть – в их максимизации, то у одной части выходных параметров знак меняется на противоположный, после чего идет либо максимизация, либо минимизация всех частных критериев.

Программа работы и задания

В лабораторной работе №4 студенты определили прогнозные значения параметров для различных экономических систем. В лабораторной работе №5 студенты должны оценить, какая из альтернативных систем их задания действует наиболее эффективно. Для этого в разных вариантах заданий применяются различные формализованные критерии, указанные в таблице 2.

Таблица 2 – Критерии эффективности систем

Вариант задания	Критерий
1, 4, 7, 10	Минимаксный
2, 5, 8, 11	Весовой
3, 6, 9, 12	Паретовский

Последовательность выполнения работы следующая:

- разработать алгоритм оценки систем по их выходным параметрам в соответствии с заданным критерием. Учесть эффект компенсации, обратить внимание на максимизацию и минимизацию частных критериев;
- написать программу оценки систем по разработанному алгоритму;
- протестировать программу. Для тестирования использовать прогнозные значения параметров, полученные в лабораторной работе №4. Тестирование описать в отчете;
- провести оценку эффективности систем, описанных в задании,

1) по последним известным отсчетам выходных параметров,

2) по прогнозным значениям выходных параметров,

3) по приращениям, то есть разностям прогнозных значений и последних известных отсчетов.

Весовые коэффициенты задать самостоятельно (обосновать);

- сделать выводы о эффективности альтернативных систем.

Минимаксная оценка трех систем на примере задания по таблице А.2 приведена в Приложении А.

Контрольные вопросы

1. Для чего нужен критерий эффективности системы? Как Вы понимаете термин "многокритериальная система"?

2. В чем суть минимаксного критерия? Запишите его с учетом эффекта компенсации. Пользуясь этим критерием, выявите наилучшую систему из альтернативных систем, описанных в вопросе 5.

3. В чем суть весового критерия? Запишите его с учетом эффекта компенсации. Как выбрать весовые коэффициенты для этого критерия?

4. Охарактеризуйте паретовский критерий. Сформулируйте методику его применения.

5. Каждая из трех альтернативных систем $s1,s2,s3$ (например, производственных фирм) характеризуется тремя параметрами: количеством работающих $P1$, объемом прибыли $P2$ и энергопотреблением $P3$. Их значения в некоторых единицах даны в таблице 3.

Таблица 3 – Параметры экономических систем

Система	$P1$	$P2$	$P3$
$s1$	1700	3.5	0.6
$s2$	1200	2.8	0.8
$s3$	1500	3.1	0.5

Оцените эффективность действий систем $s1,s2,s3$ с помощью предложенного преподавателем критерия. Недостающие данные определите самостоятельно.

ТРЕБОВАНИЯ К СОДЕРЖАНИЮ И ОФОРМЛЕНИЮ ОТЧЁТА

Отчёт должен отвечать гостам ЕСКД и ЕСПД. Он выполняется один на бригаду и включает в себя общую и индивидуальную части. Общая часть содержит необходимые для выполнения работы теоретические положения (определения, формулы, методики) и выводы по работе. Индивидуальные части являются результатом выполнения вариантов заданий членами бригады и должны отвечать программе работы. В них включаются результаты обработки варианта задания, а также все распечатки и комментарии к ним. Результат выполнения каждого пункта программы должен быть снабжён необходимыми пояснениями и заключениями. Выводы по работе должны включать конкретные результаты её выполнения, отражать полученные студентами знания и умения, а не повторять пункты программы работы.

Оформление отчёта должно соответствовать ГОСТ 2.105-95 ЕСКД и ГОСТ 19.701-90 ЕСПД. Допускается размещать текст на обеих сторонах листов. Отчёт подписывается всеми исполнителями.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Кориков А.М., Павлов С.Н. Теория систем и системный анализ. Учебное пособие. – М.: ИНФРА-М, 2017. – 288 с. [Электронный ресурс] – Режим доступа: http://znanium.com/catalog.php?bookinfo=752468

2. Клименко, И. С. Системный анализ в управлении: учебное пособие для вузов / И. С. Клименко. — Санкт-Петербург: Лань, 2020. — 272 с. — ЭБС «Лань». Режим доступа: https://e.lanbook.com/book/147336 .

3. Козлов В.Н. Системный анализ, оптимизация и принятие решений [Текст]: учебное пособие / В.Н.Козлов. – М.: Проспект, 2014. – 176 с. – 10 экз.

4. Антонов А.В. Системный анализ: учебник. – 4-е изд., перераб. и доп. — М.: ИНФРА-М, 2018. — 366 с. + Доп. материалы [Электронный ресурс] . – Режим доступа: http://znanium.com/catalog/product/973927.

5. Булыгина О.В. Системный анализ в управлении : учеб. пособие [Электронный ресурс] / О.В. Булыгина, А.А. Емельянов, Н.З. Емельянова [и др.] ; под ред. д-ра экон. наук, проф. А.А. Емельянова. — 2-е изд., перераб. и доп. — М. : ФОРУМ : ИНФРА-М, 2017. — 450 с. — Режим доступа: http://znanium.com/bookread2.php?book=900361

ПРИЛОЖЕНИЕ А. ПРИМЕРЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНЫХ ЗАДАНИЙ

Вариант задания к лабораторным работам 1,2

В таблице А.1 приведено описание структуры системы или информационного потока. Показаны связи между элементами системы, потока и мощности этих связей (в скобках). Данные, выделенные жирным шрифтом и подчеркиванием, используются при структуризации системы. В ходе структуризации система должна быть разделена на две подсистемы.

Таблица А.1 – Описание системы, информационного потока

Элемент	Связь (мощность)
1	2 (1) 3 (4)
2	**5 (2)** 6 (6)
3	4 (3) 7 (2)
4	8 (4)
5	2 (2)
6	**1 (2)** 7 (4) Внешняя среда (5)
7	8 (2)
8	3 (5)
9	Внешняя среда (5)
Внешняя среда	1 (2) 9 (4)

Выявление ошибок информационного графа

Данные таблицы А.1 (без выделенных жирным шрифтом) описывают информационный граф. Ошибки его составления находятся по по матрицам A^L. Матрицы A и A^3, полученные средствами математического пакета Mathcad, приведены на рисунке А.1.

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad A^3 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Рисунок А.1 – Матрицы исходного информационного графа

Матрица $\boldsymbol{A}$ показывает, что вершина x_9 не связана с другими вершинами графа и включена в него ошибочно. Отличные от нуля диагональные элементы матрицы $\boldsymbol{A}^3$ говорят о наличии в графе контуров, образуемых вершинами x_3, x_4, x_7 и x_8.

Анализ скорректированного графа

Коррекция графа заключается в удалении лишней вершины и неверной связи (например, x_8–x_3). Матрицы графа после коррекции приобретают вид, показанный на рисунке А.2.

$$\boldsymbol{A}=\begin{pmatrix}0&1&1&0&0&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&1&0&0&1&0\\0&0&0&0&0&0&0&1\\0&1&0&0&0&0&0&0\\0&0&0&0&0&0&1&0\\0&0&0&0&0&0&0&1\\0&0&0&0&0&0&0&0\end{pmatrix}\qquad \boldsymbol{A}^3=\begin{pmatrix}0&0&0&0&0&0&1&2\\0&0&0&0&0&0&0&1\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&1&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\end{pmatrix}$$

Рисунок А.2 – Матрицы информационного графа после коррекции

По нулевым столбцам матрицы $\boldsymbol{A}$ находятся входные вершины графа x_1, x_5, а по ее нулевым строкам – выходная вершина x_8. По матрице $\boldsymbol{A}^3$ можно найти, например, ранги элементов потока:

$$r_1=\frac{3}{5},\quad r_2=\frac{1}{5},\quad r_5=\frac{1}{5},\quad r_7=\frac{2}{5},\quad r_8=\frac{3}{5},\quad r_3=r_4=r_6=0.$$

Скорректированный информационный граф, упорядоченный по тактам, приведен на рисунке А.3.

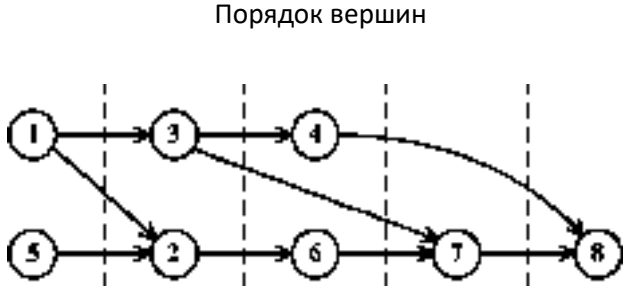


Рисунок А.3 – Скорректированный информационный граф, упорядоченный по тактам

Порядок скорректированного графа равен максимальному порядку его элементов: P=4. Для нахождения диаметра графа нужно рассмотреть пути, ведущие от входных вершин к выходной. В вершину x_8 из входных вершин x_1, x_5 ведут пути длиной 3 и 4 такта. Минимальный из них имеет длину 3

такта. Он же является максимальным из минимальных путей, так как других выходных вершин исследуемый граф не имеет. В итоге d=3.

Структуризация системы

Таблица А.1 с учетом данных, выделенных жирным шрифтом, описывает структуру системы. Матрица достижимости, вычисленная для графа этой структуры средствами пакета Mathcad, приведена на рисунке А.4.

$$\boldsymbol{D} = \begin{pmatrix} 6 & 11 & 18 & 14 & 8 & 8 & 20 & 16 \\ 8 & 14 & 16 & 11 & 11 & 11 & 19 & 20 \\ 0 & 0 & 6 & 7 & 0 & 0 & 7 & 14 \\ 0 & 0 & 7 & 3 & 0 & 0 & 3 & 7 \\ 6 & 11 & 11 & 7 & 8 & 8 & 13 & 10 \\ 5 & 8 & 17 & 9 & 6 & 6 & 14 & 20 \\ 0 & 0 & 7 & 3 & 0 & 0 & 3 & 7 \\ 0 & 0 & 7 & 7 & 0 & 0 & 7 & 6 \end{pmatrix}$$

Рисунок А.4 – Матрица достижимости графа системы

Сравнение одноименных строк и столбцов матрицы показывает, что в состав графа входят два сильно связных подграфа. В состав первого входят вершины x_1, x_2, x_5, x_6, в состав второго – x_3, x_4, x_7, x_8. Это результат применения формализованного метода структуризации. Применение принципа агрегирования экспертами дает тот или иной результат в зависимости от выбранного критерия структуризации. На рисунке А.5 показано разбиение системы на подсистемы по двум критериям: минимизации количества связей между подсистемами (граница 1) и минимизации суммарной мощности связей между подсистемами (граница 2).

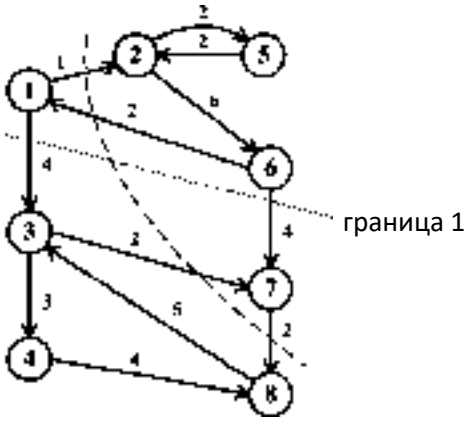


Рисунок А.5 – Разбиение системы на подсистемы на основе принципа агрегирования и различных критериев

Вариант задания к лабораторным работам 3,4

В таблице А.2 отражены результаты хозяйствования трех регионов Р1, Р2, Р3. Они оцениваются четырьмя показателями: объемом промышленного производства в денежном выражении ОП, млн. руб.; объемом сельскохозяйственного производства в денежном выражении ОС, млн. руб.; объемом дотаций из федерального бюджета ДФ, млн. руб.; объемом отчислений в федеральный бюджет ОФ, млн. руб.

Таблица А.2 – Результаты хозяйствования трех регионов

Год	Р1				Р2				Р3			
	ОП	ОС	ДФ	ОФ	ОП	ОС	ДФ	ОФ	ОП	ОС	ДФ	ОФ
2016	100	64	20	34	473	172	54	173	97	49	12	12
2017	84	62	24	28	421	154	50	121	95	48	8	6
2018	85	61	23	27	394	160	47	101	99	50	2	10
2019	81	61	20	32	410	180	40	118	110	59	0	16
2020	89	65	18	39	415	212	42	92	124	64	0	23

Прогнозирование параметра ОС региона Р1 на 2021 год

Для простоты годам присвоены обозначения от 1 до 5. Применение метода наименьших квадратов дает исходную систему уравнений

$$979 \cdot a_1 + 225 \cdot a_2 + 55 \cdot a_3 = 3462,$$
$$225 \cdot a_1 + 55 \cdot a_2 + 15 \cdot a_3 = 940,$$
$$55 \cdot a_1 + 15 \cdot a_2 + 5 \cdot a_3 = 313.$$

Для нахождения коэффициентов квадратического тренда a_1, a_2, a_3 можно применить встроенную функцию *lsolve* пакета Mathcad. Ее входными параметрами являются матрица ***M***1 числовых коэффициентов исходной системы уравнений и матрица ***M***2 свободных членов этой системы. Функция возвращает матрицу ***X*** коэффициентов тренда, как это показано на рисунке А.6.

$$M1 := \begin{pmatrix} 979 & 225 & 55 \\ 225 & 55 & 15 \\ 55 & 15 & 5 \end{pmatrix} \quad M2 := \begin{pmatrix} 3462 \\ 940 \\ 313 \end{pmatrix} \quad X := \text{lsolve}(M1, M2) \quad X = \begin{pmatrix} 0.929 \\ -5.471 \\ 68.8 \end{pmatrix}$$

Рисунок А.6 – Вычисление коэффициентов тренда средствами математического пакета Mathcad

Прогнозное значение параметра ОС находится подстановкой в уравнение тренда значения аргумента, равного 6. Результат прогнозирования показан на рисунке А.7.

$$x := 6 \qquad y := 0.929 \cdot x^2 - 5.471 \cdot x + 68.8 \qquad y = 69.418$$

Рисунок А.7 – Вычисление прогнозного значения параметра ОС

Оценка эффективности хозяйствования регионов

Для оценки работы регионов в 2021 году применяется критерий минимакса. При оценке предполагается максимизация параметров ОП, ОС, ОФ и минимизация параметра ДФ. Вычисление максимального отклонения параметров от их оптимальных значений для каждого региона показано ниже

$$k(\text{Р1}) = \max\left|\frac{89-415}{415};\frac{65-212}{212};\frac{18-0}{42};\frac{39-92}{92}\right| = \max\left|-0.79;-0.69;0.43;-0.58\right| = 0.79,$$

$$k(\text{Р2}) = \max\left|\frac{415-415}{415};\frac{212-212}{212};\frac{42-0}{42};\frac{92-92}{92}\right| = \max\left|0.0;\ 0.0;\ 1.0;\ 0.0\right| = 1.0,$$

$$k(\text{Р3}) = \max\left|\frac{124-415}{415};\frac{64-212}{212};\frac{0-0}{42};\frac{23-92}{92}\right| = \max\left|-0.70;-0.70;0.0;-0.75\right| = 0.75.$$

Минимальное значение из найденных максимальных

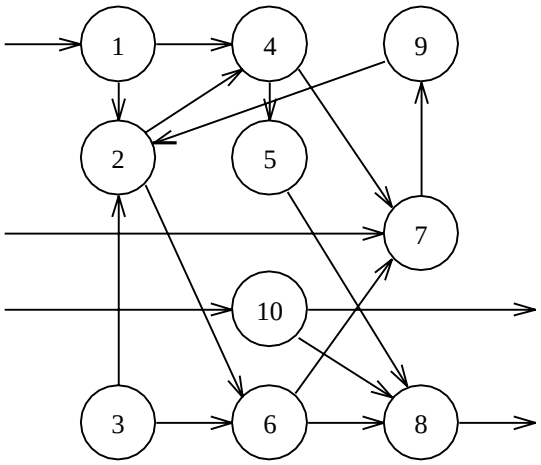
$$k\left(\text{Р}^*\right) = \min\left(0.79;1.0;0.75\right) = 0.75$$

позволяет утверждать, что наиболее эффективно в 2021 году может действовать регион Р3: Р^*=Р3.

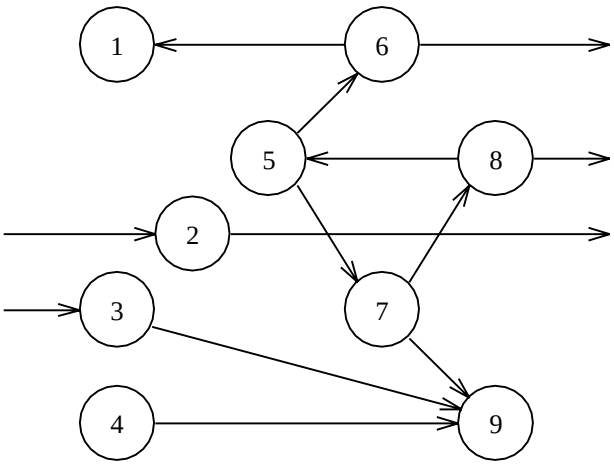
ПРИЛОЖЕНИЕ Б. ВАРИАНТЫ ЛАБОРАТОРНЫХ ЗАДАНИЙ

Б.1. Задания к лабораторной работе 1. Варианты информационных графов

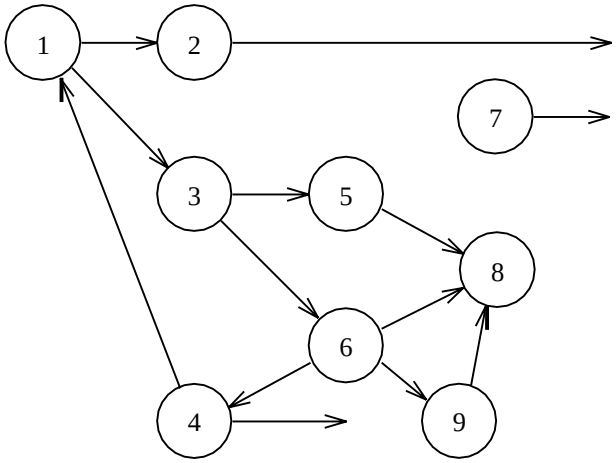
1.



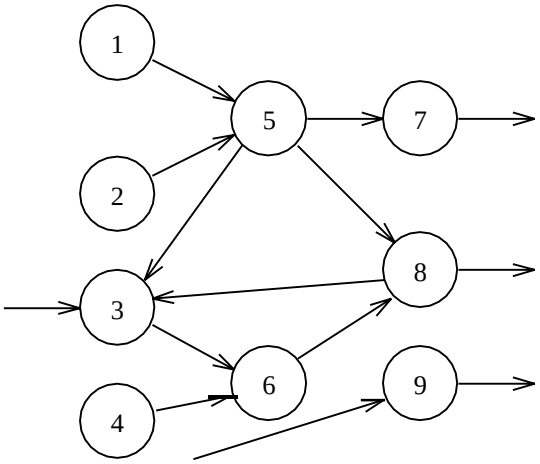
2.



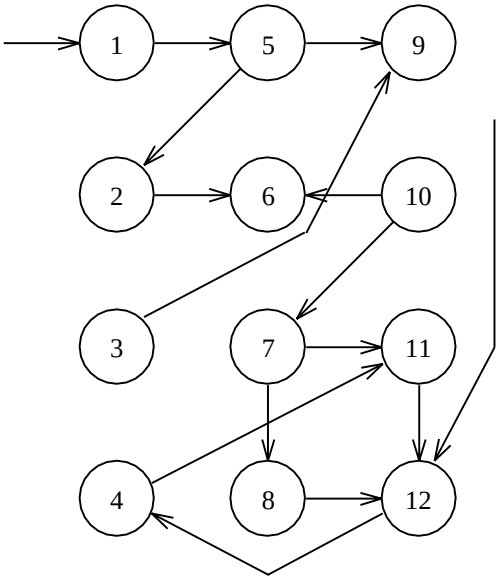
3.



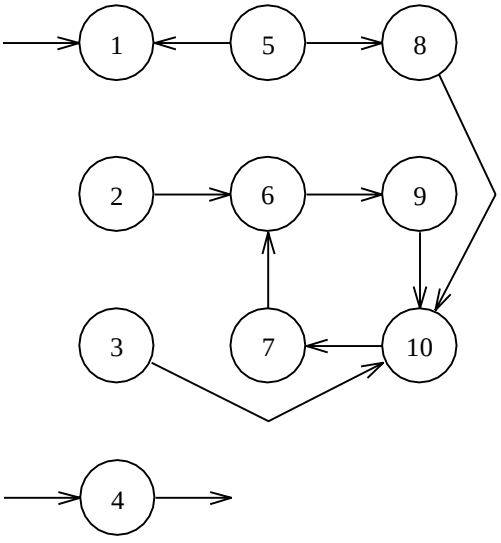
4.



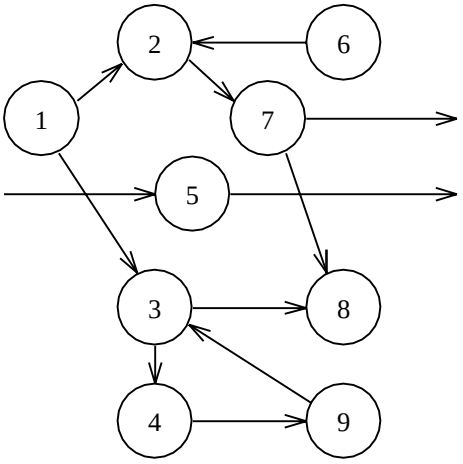
5.



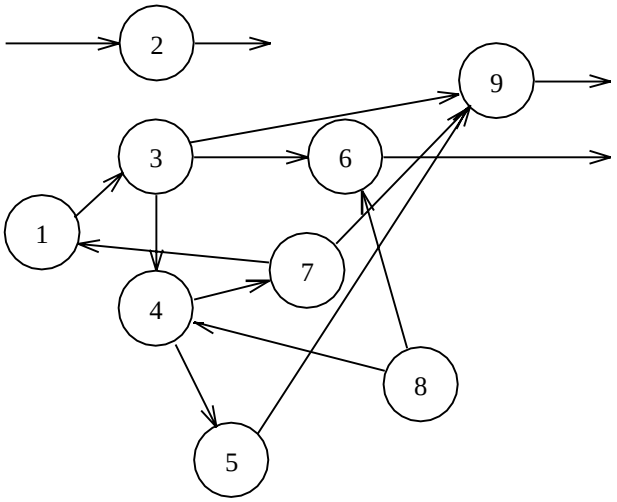
6.



7.



8.



Б.2. Задания к лабораторной работе 2. Варианты описания систем

Вариант 1

Разделить систему на три подсистемы.

Элемент	Связь (мощность)
1	5 (2) 6 (2) 7 (1) 11 (1)
2	8 (2) 9 (2) 10 (2)
3	13 (4) 14 (4)
4	3 (1) 13 (2) 14 (2)
5	15 (3)
6	**1 (1)** 15 (4)
7	6 (4) 11 (4)
8	7 (5) 12 (3)
9	16 (3)
10	3 (6) 16 (3) 17 (4)
11	15 (6)
12	11 (4) 16 (4)
13	**4 (2)** **17(6)** Внешняя среда (8)
14	**4 (3)** 13 (4) Внешняя среда (8)
15	7 (3) Внешняя среда (14)
16	**2 (2)** 17 (2)
17	**10 (2)** Внешняя среда (12)
Внешняя среда	1 (2) 2 (3) 3 (1) 4 (1) 7 (1)

Вариант 2

Разделить систему на две подсистемы.

Элемент	Связь (мощность)
1	2 (1) 7 (2)
2	3 (2) 7 (1)
3	**4 (3)** 8 (2)
4	5 (4) 8 (2)
5	9 (3) 10 (3)
6	10 (2) 11 (1)
7	**1 (1)** 12 (4) Внешняя среда (4)
8	12 (3)
9	13 (4) 14 (4)
10	5 (2) 6 (1) 14 (4)
11	15 (2)
12	**7 (2)** Внешняя среда (10)
13	12(5) 15 (6)
14	10 (3) 15 (6)
15	**14 (2)** Внешняя среда (12)
Внешняя среда	1 (2) 2 (1) 3 (1) 4 (1) 5 (1) 6 (2)

Вариант 3

Разделить систему на две подсистемы.

Элемент	Связь (мощность)
1	6 (2) 10 (1)
2	6 (2) 7 (2)
3	8 (1) 9 (2)
4	8 (2)
5	12 (3)
6	**2 (2)** 11 (3)
7	3 (2) 4 (1) 10 (3) 11 (3)
8	12 (4)
9	13 (3) 14 (3)
10	**1 (3)** 11 (4)
11	**7 (2)** Внешняя среда (6)
12	11(3) 13 (3) 14 (3)
13	**14 (2)** Внешняя среда (8)
14	3 (1) 4 (2) **5 (2)** 13 (4) Внешняя среда (10)
Внешняя среда	1 (1) 2(2) 3 (2) 4 (2) 5 (1)

Вариант 4

Разделить систему на три подсистемы.

Элемент	Связь (мощность)
1	5 (1) 6 (2)
2	6 (2) 7 (1)
3	8 (2) 9 (2) 10 (2)
4	11 (3)
5	12 (2)
6	**1 (1)** **2 (1)** 12 (4)
7	12(2)
8	9 (2) 13 (1)
9	**3 (3)** 4 (4) 13 (4)
10	9 (2) 13 (1)
11	4 (2) 14 (4)
12	**6(3)** Внешняя среда (9)
13	6 (6) 9(2) Внешняя среда (8)
14	13(1) 15 (5)
15	**11(2)** Внешняя среда (7)
Внешняя среда	1(1) 2 (1) 3 (3) 4 (2)

Вариант 5

Разделить систему на три подсистемы.

Элемент	Связь (мощность)
1	4 (2)
2	5 (2)
3	6 (3)
4	**1 (1)** 2 (2) 7(3)
5	2 (2) 3 (3) 8(3) 13 (2)
6	3 (2) 9 (3)
7	10 (4)
8	11 (3)
9	12 (2) 14 (2)
10	4 (2) Внешняя среда (6)
11	**5 (2)** Внешняя среда (6)
12	**3 (3)** Внешняя среда (5)
13	10(1) 11 (1)
14	**9 (1)** Внешняя среда (5)
Внешняя среда	1 (1) 2 (1) 3 (2)

Вариант 6

Разделить систему на три подсистемы.

Элемент	Связь (мощность)
1	2 (4) 4 (2) 6 (2)
2	**3 (2)** 7 (3)
3	8 (3) 11 (4)
4	9 (2)
5	**1 (4)** 9 (4)
6	5 (2) 10 (5)
7	2 (4) 14 (4)
8	11 (2)
9	10 (3)
10	5 (2) 6 (1) 13 (8) 15 (11) 14(10) Внешняя среда (8)
11	14 (3)
12	**16 (2)**

13	9(4)	12 (4)		
14	**7 (7)**	**11 (6)**	Внешняя среда (6)	
15	16 (5)			
16	13 (4)	15 (2)	Внешняя среда (8)	
Внешняя среда	1 (1)	2 (1)	5 (2)	10 (1)

Вариант 7

Разделить систему на две подсистемы.

Элемент	Связь (мощность)			
1	2(6)	4(1)	5(2)	6(2)
2	7(2)	8(2)		
3	8(1)	9(2)	10(3)	
4	13(3)			
5	11(4)			
6	7(4)	11(5)		
7	2(2)			
8	**2(4)**	12(5)		
9	**3(4)**	8(4)		
10	12(2)			
11	**1(2)**	8(5)	12(6)	Внешняя среда (4)
12	**9(5)**			Внешняя среда (6)
13	6(2)	11(4)		
Внешняя среда	1(2)	2(1)	3(1)	

Вариант 8

Разделить систему на две подсистемы.

Элемент	Связь (мощность)			
1	6(3)			
2	11(2)			
3	**2(4)**	5(4)	8(3)	
4	7(2)			
5	8(2)			
6	7(3)	9(4)	10(4)	
7	**4(2)**	11(6)	12(4)	
8	11(3)	14(4)		
9	12(3)			
10	12(2)			
11	3(2)	13(5)	14(4)	
12	**1(2)**	Внешняя среда (7)		
13	**11(6)**	**12(2)**	Внешняя среда (8)	
14	13(6)			
Внешняя среда	1(2)	2(1)	4(1)	5(1)

Б.3. Задания к лабораторным работам 3,4. Варианты описания выходных параметров экономических систем

Вариант 1

Четыре предприятия одного профиля – государственное предприятие П1, акционерное общество П2, частная фирма П3 и совместное предприятие П4 – применяют разные формы хозяйствования. Их деятельность оценивается следующими параметрами: объем основных фондов ФН, млн. руб.; месячный объем производства ПР, млн. руб.; объем месячных затрат ЗТ, млн. руб.; процент отношения прибыли к затратам ПБ, %. Значения названных параметров в течение двух первых кварталов года приведены в таблице.

	П1				П2				П3				П4			
Месяц	ФН	ПР	ЗТ	ПБ	ФН	ПР	ЗТ	ПБ	ФН	ПР	ЗТ	ПБ	ФН	ПР	ЗТ	ПБ
Январь	90	18	10	30	60	10	8	24	20	5	4	25	70	12	7	30
Февраль	88	17	10	26	55	10	7	26	30	6	4	28	70	12	7	30
Март	86	18	10	30	60	12	8	27	30	6	4	28	70	13	6	35
Апрель	84	18	11	30	60	12	8	27	30	6	4	28	70	12.5	6	30
Май	82	15	11	24	60	12	9	25	35	7	5	26	70	12	6	28
Июнь	80	16	13	25	60	12	9	25	40	8	5	30	68	13	7	30

Вариант 2

Три фирмы Ф1, Ф2, Ф3 в течение ряда лет изготавливали радиоаппаратуру. Она оценивается следующими параметрами: частотный диапазон ЧД; коэффициент искажения КИ, %; энергопотребление ЭП, вт; цена ЦН, тыс. руб.. Динамика названных параметров в изделиях фирм за последние 4 года показана в таблице.

	Ф1				Ф2				Ф3			
Год	ЧД	КИ	ЭП	ЦН	ЧД	КИ	ЭП	ЦН	ЧД	КИ	ЭП	ЦН
2016	18	0.4	8	1	12	0.6	2	0.6	14	0.5	3	0.8
2017	19	0.4	6	1.2	13	0.5	2.5	0.8	14	0.5	4	0.9
2018	19.5	0.3	6	1.3	13	0.5	2	0.8	15	0.4	4	0.8
2019	19.5	0.2	5.5	1.2	13.5	0.4	1.8	0.9	15	0.3	3.5	0.7

Вариант 3

Четыре региона Р1,…,Р4 соревнуются в проведении социальной политики. Её результаты оцениваются пятью параметрами: прожиточный минимум ПМ, руб.; средняя зарплата СЗ, руб.; средняя продолжительность жизни мужчин ЖМ, лет; средняя продолжительность жизни женщин ЖЖ, лет; количество зарегистрированных преступлений на 10000 населения ЗП, ед. Значения параметров за шесть лет приведены в таблице.

Год	Р1					Р2				
	ПМ	СЗ	ЖМ	ЖЖ	ЗП	ПМ	СЗ	ЖМ	ЖЖ	ЗП
2010	800	1500	60	64	8,5	700	1500	61	65	10
2011	1200	2000	59	64	11	1000	1800	61	66	12

2012	1500	2500	59,5	65	12	1200	2000	62	66	14
2013	1700	3000	59	65	10,5	1500	2500	62	67	14
2014	1900	3400	60	66	10	1700	2800	61	66	14
2015	2100	3600	61	67	9	1800	3100	60	65	13

Год	Р3					Р4				
	ПМ	СЗ	ЖМ	ЖЖ	ЗП	ПМ	СЗ	ЖМ	ЖЖ	ЗП
2010	900	2000	58	63	12	600	1400	57	61	8,5
2011	1300	2400	57	63	15	900	1900	57	62	10
2012	1700	2700	57	64	16	1200	2300	58	62	13
2013	2000	3000	58	65	17	1400	2600	59	63	14
2014	2200	3400	58	65	16	1700	3000	61	65	14
2015	2300	3800	59	65	14	2100	3500	63	67	12

Вариант 4

В таблице приведена поквартальная динамика показателей, описывающих занятость трудоспособного населения в пяти городах Г1,…,Г5 области за два года. К этим показателям относятся: количество безработных КБ, тыс. чел; средний возраст работающих СВ, лет; средняя зарплата работающих СЗ, тыс. руб.; удельный вес работающих, получающих доход ниже прожиточного минимума УВ, %.

Квартал	Г1				Г2				Г3			
	КБ	СВ	СЗ	УВ	КБ	СВ	СЗ	УВ	КБ	СВ	СЗ	УВ
I	16	47	2.4	30	12	45	2.1	32	10	42	2.3	29
II	15.5	48	2.5	31	12	45	2.2	30	10.5	42	2.3	27
III	15	48	2.6	31	13	46	2.4	29	11.5	43	2.5	27
IV	14.5	46	2.8	29	14	46	2.4	26	12	42	2.8	26
V	14.5	45	3.0	27	14.5	46	2.6	23	12	43	3.1	26
VI	15.5	44	3.2	26	13	47	2.7	19	11.5	43	3.1	24
VII	15.5	44	3.5	24	12.5	48	3.0	20	11.5	44	3.3	21
VIII	16	43	3.8	20	12	48	3.4	21	12	45	3.6	22

Квартал	Г4				Г5			
	КБ	СВ	СЗ	УВ	КБ	СВ	СЗ	УВ
I	18	46	1.9	31	14	50	1.8	34
II	19	45	2.1	27	13.5	51	2.0	31
III	19	45	2.4	24	14	51	2.1	29
IV	21	44	2.7	24	13	50	2.2	30
V	22	44	2.7	22	12	49	2.5	29
VI	21	45	2.9	23	11	47	2.6	27
VII	22	44	3.2	20	9	45	2.9	26
VIII	20	43	3.5	18	10	42	3.1	24

Вариант 5

Санитарная обстановка в четырех областях 01,…,04 характеризуется следующими показателями: обеспеченность жилплощадью ОЖ, кв.м.; средняя продолжительность жизни ПЖ, лет; количество больных социально опасными болезнями (туберкулез, наркомания, СПИД и т.д.) КБ, тыс.чел.; количество детей, охваченных диспансеризацией ДД, тыс.чел. Значения показателей за десять лет показаны в таблице.

Год	01				02				03				04			
	ОЖ	ПЖ	КБ	ДД	ОЖ	ПЖ	КБ	ДД	ОЖ	ПЖ	КБ	ДД	ОЖ	ПЖ	КБ	ДД
2005	16	62	1	12	18	64	0,8	9	16	65	3,1	21	15	64	8	40
2006	16	62	1,2	11	20	63	0,7	10	18	65	3,2	22	16	64	8,5	42
2007	12	61	1,3	10	20	63	0,7	8	18	64	3,1	22	16	62	8,9	42
2008	12	61	1,5	8	20	63	0,8	7	18	64	2,9	23	18	61	9,1	38
2009	12	60	1,4	9	20	62	1,0	7	16	63	2,8	22	18	61	9,2	35
2010	14	60	1,3	12	18	63	1,2	8	16	64	2,6	20	20	60	9,0	31
2011	14	60	1,4	15	16	62	1,4	6	16	63	2,3	18	20	60	8,7	30
2012	12	59	1,6	10	14	60	1,7	5	12	62	2,4	15	20	61	9,5	24
2013	12	59	1,8	9	14	59	1,6	4	12	62	2,3	14	16	62	10,5	22

2014	14	60	2,1	11	18	59	1,5	6	14	61	2,1	14	16	63	10,5	25

Вариант 6

Три исследовательские организации И1, И2, И3 на конкурсной основе выполнили разработку технологического процесса изготовления некоторого устройства. Каждая организация прошла через шесть модификаций технологического процесса. Процесс характеризуется следующими показателями: производительность ПР, ед/мес., погрешность изготовления ПИ,%; стоимость единицы изделия СИ, тыс. руб.; стоимость технологического оборудования СО, тыс.руб.; энергопотребление ЭП, квт/час. Значения показателей для разработанных модификаций технологического процесса сведены в таблице.

Номер модификации	И1					И2					И3				
	ПР	ПИ	СИ	СО	ЭП	ПР	ПИ	СИ	СО	ЭП	ПР	ПИ	СИ	СО	ЭП
1	37	0,35	2,4	150	0,6	40	0,40	2,0	120	0,9	35	0,30	2,1	160	0,8
2	37	0,35	2,0	150	0,5	44	0,42	1,8	130	0,7	38	0,30	2,0	165	0,7
3	42	0,3	2,1	150	0,45	47	0,42	1,8	130	0,5	43	0,32	1,9	175	0,8
4	46	0,3	2,1	150	0,42	48	0,38	2,0	145	0,4	46	0,34	1,7	190	0,6
5	48	0,2	2,0	165	0,40	48	0,30	1,8	155	0,4	50	0,30	1,5	210	0,4
6	49	0,1	2,0	180	0,38	50	0,20	1,5	170	0,35	54	0,25	1,5	210	0,4

Вариант 7

Три завода З1, З2, З3 получили госзаказ на освоение выпуска новой продукции. Их деятельность оценивается следующими показателями: объем производства в натуральном выражении ОН, тыс.шт./полугодие; объем производства в денежном выражении ОД, млн. руб./полугодие; количество работающих КР, тыс.чел.; себестоимость продукции СП, руб. Значения показателей за четыре года работы (по полугодиям) приведены в таблице.

	Полугодие	З1				З2				З3			
		ОН	ОД	КР	СП	ОН	ОД	КР	СП	ОН	ОД	КР	СП
2011	1	8.2	1.0	232	57.5	26.3	2.6	1200	48.3	15.2	1.5	350	53.0
	2	8.3	1.2	245	58.0	27.4	2.7	1100	48.7	15.6	1.6	350	53.0
2012	1	8.5	1.3	260	58.6	28.9	2.9	1020	49.2	17.1	2.0	372	53.3
	2	8.8	1.6	260	59.4	32.4	3.4	900	50.0	18.1	2.6	380	53.4
2013	1	10.0	2.0	210	59.3	36.1	3.8	800	51.0	19.4	2.8	380	53.9
	2	10.4	2.0	200	59.0	39.1	4.6	750	51.5	20.2	2.9	370	54.6

2014	1	12.1	1.9	184	59.0	40.5	4.9	700	51.8	20.8	3.1	360	54.5
	2	12.7	2.4	190	58.7	40.9	4.9	700	51.9	21.6	3.2	340	54.1

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ ТЕХНОЛОГИИ ОБРАБОТКИ БОЛЬШИХ ДАННЫХ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

Лабораторная работа №1. Обработка данных с использованием специализированной библиотеки PANDAS и языка Python

ЦЕЛЬ РАБОТЫ

Изучить основные подходы к обработке больших данных с использованием программной библиотеки PANDAS и языка программирования Python в среде Anaconda

ХОД РАБОТЫ

1. Настройка рабочего места. Самостоятельная установка на персональный компьютер среды Anaconda и подключение программной библиотеки PANDAS для языка Python. Настройка рабочего места осуществляется в соответствии с прилагаемыми инструкциями в файле *Инструкция по установке среды Anaconda.doc*.
2. Изучение базового синтаксиса языка Python. Описание основных конструкций языка, необходимых для дальнейшей работы с большими данными приводится в файле *Быстрое введение в синтаксис языка Python.doc*.
3. Изучение и выполнение основных функций библиотеки PANDAS в соответствии с общей частью данных методических указаний.
4. Получение у преподавателя индивидуального задания.
5. Написание и отладка программы.
6. Подготовка отчета
7. Сдача программы, отчета и теоретическое собеседование с учетом перечня контрольных вопросов.

ОБЩАЯ ЧАСТЬ

Используемые названия и термины

Pandas (сокр. от англ. **Pan**el **Da**ta – панельные, то есть, табличные данные) – это программная библиотека для языка Python, предназначенная для сбора, очистки и анализа (в основном, статистического) данных, представленных в табличных структурах.

DataFrame (датафрейм) – индексированный массив двухмерных данных (пример на рис. 1).

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
5	Amir Johnson	Boston Celtics	90.0	PF	29.0	6-9	240.0	NaN	12000000.0
6	Jordan Mickey	Boston Celtics	55.0	PF	21.0	6-8	235.0	LSU	1170960.0
7	Kelly Olynyk	Boston Celtics	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0
8	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
9	Marcus Smart	Boston Celtics	36.0	PG	22.0	6-4	220.0	Oklahoma State	3431040.0
10	Jared Sullinger	Boston Celtics	7.0	C	24.0	6-9	260.0	Ohio State	2569260.0
11	Isaiah Thomas	Boston Celtics	4.0	PG	27.0	5-9	185.0	Washington	6912869.0
12	Evan Turner	Boston Celtics	11.0	SG	27.0	6-7	220.0	Ohio State	3425510.0
13	James Young	Boston Celtics	13.0	SG	20.0	6-6	215.0	Kentucky	1749840.0

Рисунок 1 – Пример датафрейма

csv (сокр. от англ. Comma-Separated Values — значения, разделённые запятыми) – текстовый формат файла предназначенный для представления табличных данных. Каждая строка этого файла является отдельной строкой таблицы, а столбцы отделены один от другого специальными символами - разделителями. В качестве таких символов могут выступать: запятая, точка с запятой, символ табуляции, пробел и др. Пример csv-файла показан на рис.

2., а соответствующий ему вид таблицы – на рис. 3. Простейшая программа, которая способна создавать и считывать файлы csv – MS Excel.

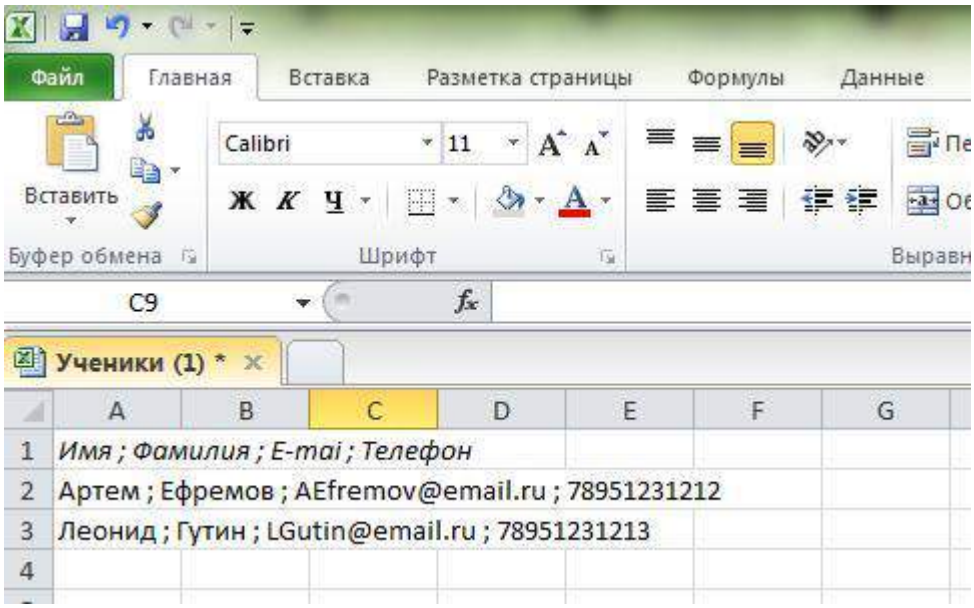


Рисунок 2 – Пример файла csv

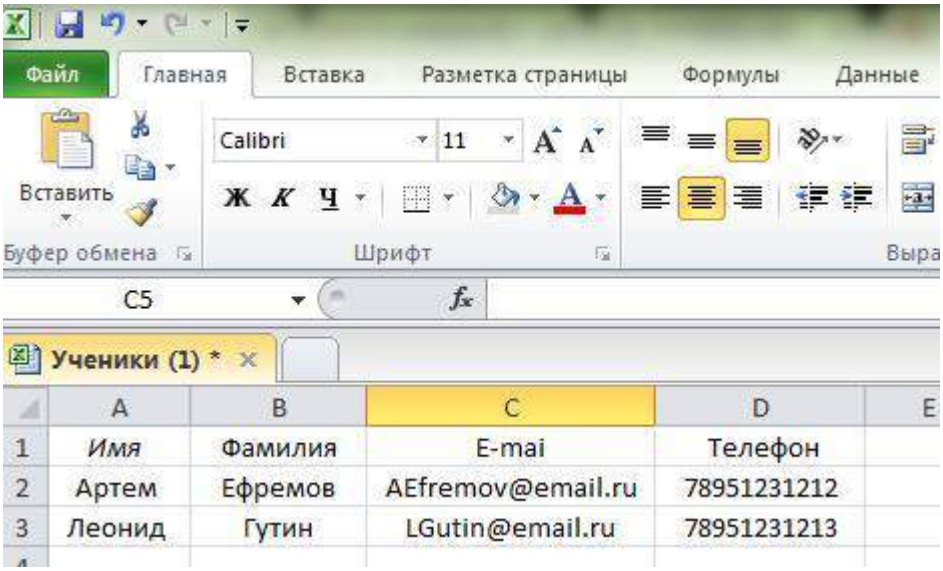


Рисунок 3 – Табличный вид файла csv, показанного на рис. 2

Импорт библиотеки Pandas

По умолчанию, внешние библиотеки, такие как Pandas, недоступны при работе с языком Python. Необходимо предварительно выполнить операцию импорта при помощи ключевого слова `import`:

```
import pandas as pd
```

где:

– `import` – ключевое слово языка Python, предназначенная для импорта сторонних библиотек;
– `pandas` – название импортируемой библиотеки;
– `as` – ключевое слово языка Python, позволяющее задать библиотеки сокращенное имя (псевдоним), по которому в дальнейшем к ней будет происходить обращение;
– `pd` – псевдоним библиотеки (это может быть произвольное имя). Среди data-аналитиков, в качестве негласного правила, для библиотеки Pandas принято использовать псевдоним pd. Однако, при желании, можно задать и другое имя.

Более простой вариант импорта (без указания псевдонима):

```
import pandas
```

В этом случае, обращение к библиотеке будет происходить через имя pandas.

Считывание данных в датафрейм из csv-файла

Для того, чтобы считать данные из csv-файла используется метод `read_csv` из библиотеки Pandas:

```
df = pd.read_csv ('path_to_your.csv')
```

где:

– `df` – имя датафрейма, куда будут считаны данные файла;
– `pd` – псевдоним библиотеки, где хранится метод `read_csv`;
– `read_csv` – имя метода;
– `'path_to_your.csv'` – имя файла и путь к нему. В ОС Windows пути к файлам содержат символ \, который является специальным символом в строках во многих языках программирования, включая и Python. Поэтому

необходимо либо удвоить все \ в строке с путём, либо поставить r перед строкой.

Например, если файл находится в папке:

```
C:\mydir\mycsv\mydata.csv
```

то этот путь должен быть указан параметром метода read_csv одним из двух способов:

```
pd.read_csv ('C:\\mydir\\mycsv\\mydata.csv')
pd.read_csv (r'C:\mydir\mycsv\mydata.csv')
```

Рассмотрим некоторые дополнительные параметры метода read_csv:

encoding – параметр, отвечающий за кодировку текста, которая может быть различной. Самые распространённые – utf-8 (для ОС Linux) и Windows-1251 (для ОС Windows). Пример указания кодировки:

```
df = pd.read_csv ('C:\\mydir\\mycsv\\mydata.csv',
                  encoding = 'Windows-1251')
```

sep – разделитель между столбцами в строке (по умолчанию ‘,’). Пример одновременного указания и кодировки и разделителя:

```
pd.read_csv (r'C:\mydir\mycsv\mydata.csv',
             encoding='Windows-1251', sep=';')
```

parse_data – параметр, указывающий как воспринимать даты – как обычные строки или как специальные значения, типа data (по умолчанию воспринимаются как строки). Значения параметра могут быть следующие:

– True – пытаться перевести в тип data первую колонку;
– список колонок, значения которых нужно переводить в тип data.

Например,

```
df = pd.read_csv ('mydata.csv',
     parse_dates=['birth_data', 'death_data'])
```

Примеры ошибок несовпадения кодировки и разделителя, возникающих при чтении csv-файла показаны на рис. 4, 5.

Рисунок 4 – Ошибка чтения csv-файла по причине несовпадения выбранной кодировки текста (файл в кодировке windows-1251, а функция read_csv по умолчанию пытается читать тест в кодировке utf-8)

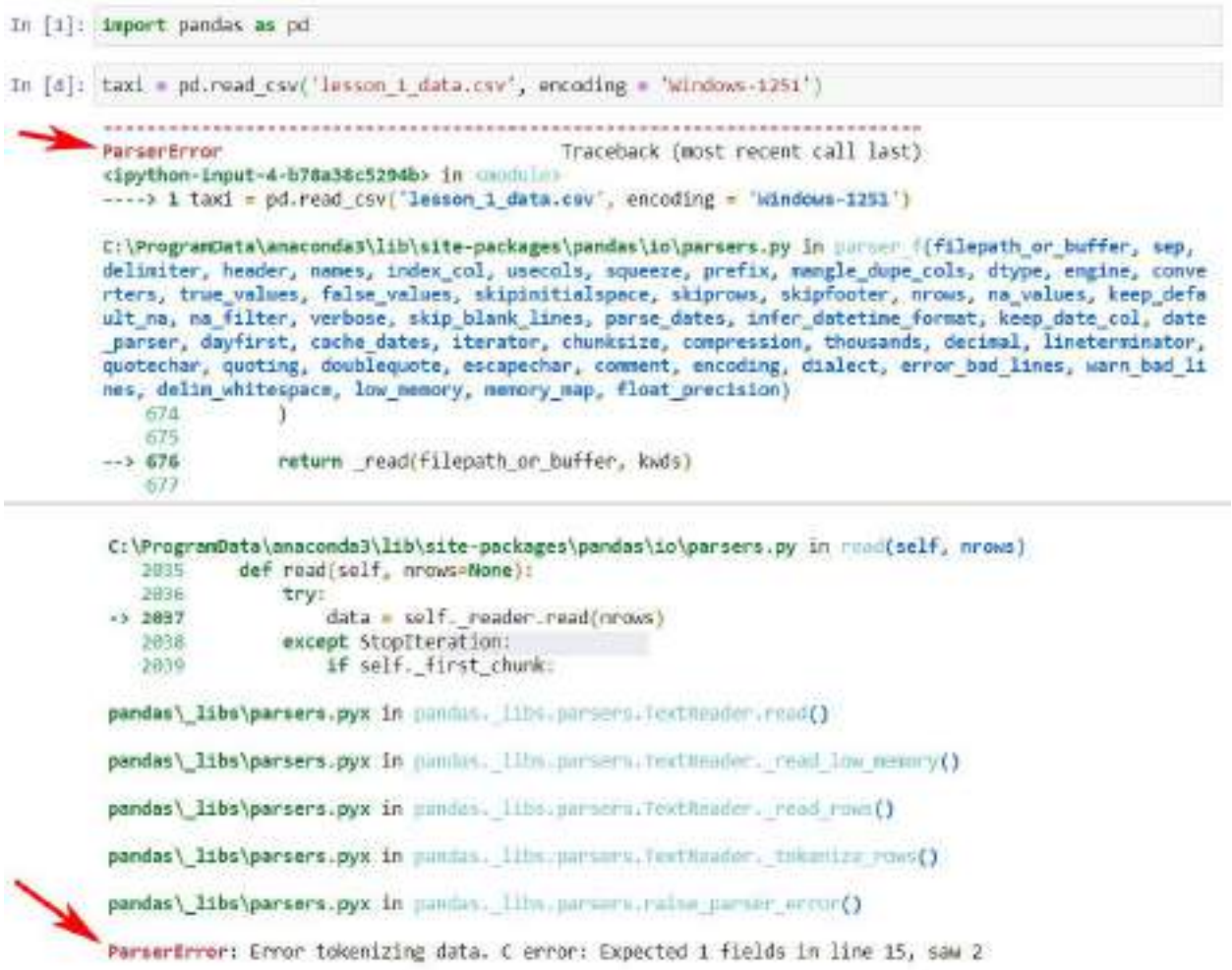


Рисунок 5 - Ошибка чтения csv-файла по причине неправильно указанного разделителя (в файле разделителем является ‘;’, а метод `read_csv` по умолчанию ищет разделители в виде ‘,’).

Подобного рода ошибки устраняются исправлением параметров кодировки и разделителя при вызове `read_csv`.

Датафреймы. Методы и атрибуты

В предыдущих примерах, мы создали переменную `df` и считали в нее данные из файла формата csv. Переменная `df` получает тип DataFrame. Датафрейм – это специальная структура данных. Если выражаться более специальным языком, то это объект класса DataFrame, который содержится в

библиотеке Pandas. Теперь можно посмотреть содержимое датафрейма – см. рис. 6

```
In [1]: import pandas as pd

In [6]: df = pd.read_csv('lesson_1_data.csv', encoding = 'windows-1251', sep = ';')

In [10]: df
```

Out[10]:

	Номер	Дата создания	Дата оплаты	Title	Статус	Заработано	Город	Платежная система
0	1062823	01.12.2019 10:50	01.12.2019 10:52	Курс обучения «Эксперт»	Завершен	29597.50	Чита	Сбербанк эквайринг
1	1062866	01.12.2019 20:53	01.12.2019 21:27	Курс обучения «Эксперт»	Завершен	17460.30	Краснодар	Яндекс.Касса
2	1062886	01.12.2019 21:43	NaN	Курс обучения «Специалист»	Отменен	0.00	NaN	NaN
3	1062880	03.12.2019 0:18	NaN	Курс обучения «Консультант»	Отменен	0.00	г.Москва и Московская область	NaN
4	1062889	03.12.2019 21:43	NaN	Курс обучения «Эксперт»	Отменен	0.00	г.Москва и Московская	NaN

Рисунок 6 – Вывод содержимого датафрейма `df`

Концепция объектно-ориентированного программирования подразумевает, что каждый объект класса может иметь атрибуты и методы.

Атрибуты – это набор переменных, описывающих состояние объекта.

Методы – это функции, которые могут быть применены к объекту.

Визуально, основным их отличием является то, что после метода следуют скобки, в которых могут указываться параметры, а после атрибута скобок нет.

Рассмотрим некоторые полезные атрибуты датафрейма:

– `shape` – содержит кортеж из двух чисел, в которых указан размер датафрейма (количество строк и столбцов). Пример на рис. 7.
– `dtypes` – содержит серию (массив) с описанием типа каждой колонки датафрейма (сложные типы, отличающиеся от простейших типов Python обозначаются как тип `object`). Пример на рис. 8
– `columns` – содержит названия столбцов датафрейма.

Рисунок 7 – Вывод содержимого атрибута `shape`

Рисунок 8 - Вывод содержимого атрибута `dtypes`

Перечислим несколько полезных методов датафрейма:

– `head()` – возвращает первые 5 строк датафрейма (рис. 9).
– `tail()` – возвращает последние 5 строк датафрейма.
– `describe ()` – выводит статистическое описания числовых колонок в датафрейме (рис. 10): число строк, среднее и стандартное отклонение, минимум, максимум, значения по 25-ому, 50-ому и 75-ому квартилям.
– `rename ()` – метод переименования (можно переименовывать столбцы, индексы строк и т.п.).

На примере метода переименования `rename()` можно рассмотреть работу с его параметрами. Наиболее часто, имеет смысл переименовывать столбцы и индексы строк, поскольку названия столбцов несут основную семантическую нагрузку датафрейма, а индексы строк по умолчанию имеют значения 0, 1, … и это не всегда удобно. Поэтому в параметрах метода `rename()` указывается, какие элементы подлежат переименованию и их новые имена.

```
In [1]: import pandas as pd

In [6]: df = pd.read_csv('lesson_1_data.csv', encoding = 'windows-1251', sep = ';')

In [14]: df.head()
```

Out[14]:

	Номер	Дата создания	Дата оплаты	Title	Статус	Заработано	Город	Платежная система
0	1062823	01.12.2019 10:50	01.12.2019 10:52	Курс обучения «Эксперт»	Завершен	29597.5	Чита	Сбербанк эквайринг
1	1062855	01.12.2019 20:53	01.12.2019 21:27	Курс обучения «Эксперт»	Завершен	17450.3	Краснодар	Яндекс Касса
2	1062856	01.12.2019 21:43	NaN	Курс обучения «Специалист»	Отменен	0.0	NaN	NaN
3	1062880	03.12.2019 0:18	NaN	Курс обучения «Консультант»	Отменен	0.0	г.Москва и Московская область	NaN
4	1062800	03.12.2019 21:43	NaN	Курс обучения «Эксперт»	Отменен	0.0	г.Москва и Московская область	NaN

Рисунок 9 – Вызов метода `head()`

На рисунке 10 показан пример переименования столбцов (columns) ‘Номер’ и ‘Статус’ в англоязычный вид ‘Number’ и ‘Status’. На рисунке 11 переименовываются индексы двух первых строк от 0, 1 к ‘Иванов’ и ‘Петров’.

```
In [1]: import pandas as pd

In [6]: df = pd.read_csv('lesson_1_data.csv', encoding = 'windows-1251', sep = ';')

In [18]: df = df.rename(columns={'Номер':'Number', 'статус':'status'})

In [19]: df
```

Out[19]:

	Number	Дата создания	Дата оплаты	Title	Status	Заработано	Город	Платежная система
0	1062823	01.12.2019 10:50	01.12.2019 10:52	Курс обучения «Эксперт»	Завершен	29597.50	Чита	Сбербанк эквайринг
1	1062855	01.12.2019 20:53	01.12.2019 21:27	Курс обучения «Эксперт»	Завершен	17450.30	Краснодар	Яндекс Касса
2	1062856	01.12.2019 21:43	NaN	Курс обучения «Специалист»	Отменен	0.00	NaN	NaN
3	1062880	03.12.2019 0:18	NaN	Курс обучения «Консультант»	Отменен	0.00	г.Москва и Московская область	NaN

Рисунок 10 – Пример переименования двух столбцов датафрейма

```
In [1]: import pandas as pd

In [6]: df = pd.read_csv('lesson_1_data.csv', encoding = 'windows-1251', sep = ';')

In [20]: df = df.rename(index={0:'Иванов', 1:'Петров'})

In [21]: df

Out[21]:
```

	Number	Дата создания	Дата оплаты	Title	Status	Заработано	Город	Платежная система
Иванов	1062823	01.12.2019 10:50	01.12.2019 10:52	Курс обучения «Эксперт»	Завершен	29597.50	Чита	Сбербанк эквайринг
Петров	1062865	01.12.2019 20:53	01.12.2019 21:27	Курс обучения «Эксперт»	Завершен	17450.30	Краснодар	Яндекс.Касса
2	1062866	01.12.2019 21:43	NaN	Курс обучения «Специалист»	Отменен	0.00	NaN	NaN
3	1062880	03.12.2019 0:10	NaN	Курс обучения «Консультант»	Отменен	0.00	г.Москва и Московская область	NaN

Рисунок 11 – Пример переименования двух индексов строк датафрейма

Таким образом, большинство задач анализа данных средствами Pandas, сводится к комбинированным вызовам методов и атрибутов датафрейма, загруженного из файла.

Обращение к столбцам датафрейма

Самым удобным способом обращения к столбцам датафрейма является следующая форма:

`df.column_name`

где:

– `df` – имя датафрейма;

– `column_name` – имя столбца.

Такой способ работает особенно хорошо, если имя столбца состоит из одного слова и не совпадает с именами имеющихся атрибутов и методов. Например: `df.Status` – приведет к выводу всех значений столбца Status, а `df.Status['Иванов']` – выдаст первое значение этого столбца. Напомним, что после команды, показанной на рис. 11, `'Иванов'` является индексом первой строки. Если бы такого переименования не было, то первое значение столбца выдавалось командой `df.Status[0]`.

Если столбец состоит из нескольких слов, то обратиться к нему можно так:

```
df['Дата создания']
```

Для получения значений сразу нескольких столбцов, их имена передаются списком в квадратных скобках. Например,

```
df['Дата создания', 'Дата оплаты', 'Платежная система']
```

Вычислительные методы

Перечислим набор основных методов, которые позволяют производить простейшие расчеты по числовым столбцам датафрейма:

– суммирование всех числовых значений столбца (метод `sum`);
– нахождение среднего значения (метод `mean`);
– перемножение (метод `product`);
– нахождение среднеквадратичного отклонения (метод `std`);
– вычисление дисперсии числовых значений столбца (метод `var`).

Так, к примеру, можно рассчитать, сколько всего денег заработано от продажи обучающих курсов (столбец ‘Заработано’) – рис. 12

Рисунок 12 – Пример использования метода `sum`

Аналогичным образом можно посчитать среднее значение продаж, вызвав метод `mean()`.

Сортировка

Для сортировки значений выбранного столбца используется метод `sort_values`. Например, отсортируем строки нашего датафрейма по возрастанию значения уплаченной суммы– рис. 13, и по его убыванию – рис. 14. Обратите внимание, что для указания порядка сортировки используется специальный параметр `ascending` - по умолчанию он равен логическому True, что указывает на сортировку по возрастанию. Чтобы сортировать по убыванию, следовательно, нужно задать для этого значения логическое значение False. Возможна также сортировка по тестовым столбцам (рис. 15).

```
In [1]: import pandas as pd

In [2]: df = pd.read_csv('lesson_1_data.csv', encoding = 'windows-1251', sep = ';')

In [8]: df.sort_values('заработано')
```

Out[8]:

	Номер	Дата создания	Дата оплаты	Title	Статус	Заработано	Город	Платежная система
145	1063697	19.12.2019 21:09	NaN	Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автос...	Отменен	0.0	Ростовская область	NaN
153	1063707	19.12.2019 21:09	NaN	Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автос...	Отменен	0.0	Астрахань	NaN
154	1063708	19.12.2019 21:09	NaN	Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автос...	Отменен	0.0	Амурская область	NaN
155	1063710	19.12.2019 21:09	NaN	Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автос...	Отменен	0.0	Ростовская область	NaN

Рисунок 13 – Сортировка строк по возрастанию значений столбца ‘Заработано’

```
In [1]: import pandas as pd

In [2]: df = pd.read_csv('lesson_1_data.csv', encoding = 'windows-1251', sep = ';')

In [9]: df.sort_values('Заработано', ascending=False)
```

Out[9]:

	Номер	Дата создания	Дата оплаты	Title	Статус	Заработано	Город	Платежная система
37	1063251	14.12.2019 14:34	18.12.2019 13:11	Курс обучения «Эксперт»	Завершен	42750.0	Балаш	Системы быстрых переводов
46	1063330	16.12.2019 11:11	17.12.2019 13:42	Курс обучения «Специалист»	Завершен	29695.7	Московская область, г. Подольск	Сбербанк эквайринг
270	1064553	27.12.2019 18:08	27.12.2019 18:09	Курс обучения «Специалист»	Завершен	29695.7	Красноярский край	Сбербанк эквайринг
234	1064053	23.12.2019 0:38	23.12.2019 0:42	Курс обучения «Специалист»	Завершен	29695.7	Магаданская область	Сбербанк эквайринг
50	1063391	17.12.2019	17.12.2019	Курс обучения «Специалист»	Завершен	29695.7	Краснодар	Сбербанк

Рисунок 14 - Сортировка строк по убыванию значений столбца ‘Заработано’

```
In [1]: import pandas as pd

In [2]: df = pd.read_csv('lesson_1_data.csv', encoding = 'windows-1251', sep = ';')

In [10]: df.sort_values('Город')
```

Out[10]:

	Номер	Дата создания	Дата оплаты	Title	Статус	Заработано	Город	Платежная система
14	1062947	05.12.2019 21:39	07.12.2019 13:35	Курс обучения «Консультант»	Завершен	5044.05	Массандра	PayPal
238	1064114	23.12.2019 19:10	23.12.2019 19:16	Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автос...	Завершен	2935.44	в.в.	Яндекс Касса
77	1063615	18.12.2019 20:58	19.12.2019 21:05	Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автос...	Завершен	2914.30	Алматы	CloudPayments
154	1063708	19.12.2019 21:09	NaN	Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автос...	Отменен	0.00	Амурская область	NaN
134	1063686	19.12.2019	NaN	Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным	Отменен	0.00	Архангельск	NaN

Рисунок 15 - Сортировка строк по возрастанию значений столбца ‘Город’

Подсчет количества уникальных значений

С помощью метода `value_counts` можно подсчитать, сколько раз встречается каждое уникальное значение столбца, а также оценить некоторые их статистические свойства.

Например, рассчитаем сколько всего продано различных курсов (будем считать количество уникальных значений по столбцу Title) – рис. 16. На выходе получаем две колонки – в первой, название курса, во второй –

сколько раз это название встречается в датафрейме (по сути, сколько раз этот курс был продан).

```
In [1]: import pandas as pd

In [2]: df = pd.read_csv('lesson_1_data.csv', encoding = 'Windows-1251', sep = ';')

In [11]: df.Title.value_counts()

Out[11]: Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автосписанием    182
         Курс обучения «Консультант»                               52
         Курс обучения «Эксперт»                                   22
         Курс от Школы Диетологов. Повышение квалификации.         15
         Курс обучения «Специалист»                                12
         Курс от Школы Диетологов. Бизнес                           9
         Name: Title, dtype: int64
```

Рисунок 16 – Статистика продажи курсов

Метод `value_counts` имеет несколько параметров, наиболее важные из которых:

– `normalize` - показать относительные частоты уникальных значений (по умолчанию равен False). Присвоив этому параметру значение True, можно оценить не только количественное, но и процентное соотношение значений (рис. 17);

– `dropna` - не включать количество NaN (по умолчанию равен True). Значение NaN – это, по сути, пропущенные или пустые значения. Поэтому, по умолчанию, они не учитываются в расчете. Однако эту ситуацию можно изменить, присвоив этому параметру False.

– `bins` – позволяет выполнить группировку количественной переменной (например, разбить возраст на возрастные группы); для использования данного параметра нужно указать, на сколько групп разбить переменную. Например, на рис. 18 показано разбиение сумм, уплаченных за курсы, на 4 группы.

```
In [1]: import pandas as pd

In [2]: df = pd.read_csv('lesson_1_data.csv', encoding = 'windows-1251', sep = ';')

In [13]: df.Title.value_counts(normalize=True)

Out[13]: Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автосписанием    0.623288
         Курс обучения «Консультант»                             0.178082
         Курс обучения «Эксперт»                                 0.075342
         Курс от Школы Диетологов. Повышение квалификации.       0.051370
         Курс обучения «Специалист»                              0.041096
         Курс от Школы Диетологов. Бизнес                        0.030822
         Name: Title, dtype: float64
```

Рисунок 17 - Статистика продажи курсов в относительных частотах

```
In [1]: import pandas as pd

In [2]: df = pd.read_csv('lesson_1_data.csv', encoding = 'Windows-1251', sep = ';')

In [15]: df['Заработано'].value_counts(bins=4)

Out[15]: (-42.751, 10687.5]    280
         (21375.0, 32062.5]      7
         (10687.5, 21375.0]      4
         (32062.5, 42750.0]      1
         Name: Заработано, dtype: int64
```

Рисунок 18 – Результат разбиение сумм, уплаченных за курсы, на 4 группы

Запросы

Механизм запросов позволяет делать выборку строк из датафрейма по определенным условиям (одному или сразу нескольким). Для этих целей служит метод `query`.

Например, извлечем из нашего датафрейма все записи, для которых статус оплаты завершен. То есть, условием выборки будет наличие в столбце ‘Статус’ значения ‘Завершен’ – рис. 19. Далее, можно усложнить запрос и выдать записи, у которых статус завершен и уплаченная сумма больше 20 000 руб. – рис. 20. Или, сделать выборку данных, касательно только отдельно взятых городов (например, Чита или Балхаш) – рис. 21.

```
In [1]: import pandas as pd

In [2]: df = pd.read_csv('lesson_1_data.csv', encoding = 'Windows-1251', sep = ';')

In [18]: df.query("Статус=='Завершен'")
```

Out[18]:

	Номер	Дата создания	Дата оплаты	Title	Статус	Заработано	Город	Платежная система
0	1062823	01.12.2019 10:50	01.12.2019 10:52	Курс обучения «Эксперт»	Завершен	29597.50	Чита	Сбербанк эквайринг
1	1062955	01.12.2019 20:53	01.12.2019 21:27	Курс обучения «Эксперт»	Завершен	17450.30	Краснодар	Яндекс.Касса
12	1062938	05.12.2019 12:07	22.12.2019 12:29	Курс обучения «Консультант»	Завершен	8910.00	Пермский край	Яндекс.Касса
13	1062940	05.12.2019 15:35	05.12.2019 15:40	Курс от Школы Диетологов Бизнес	Завершен	9394.56	Воронежская область	Бонусный счет,Сбербанк эквайринг

Рисунок 19 – Выборка данных со статусом ‘Завершен’

```
In [1]: import pandas as pd

In [2]: df = pd.read_csv('lesson_1_data.csv', encoding = 'Windows-1251', sep = ';')

In [19]: df.query("Статус == 'Завершен' and Заработано > 20000")
```

Out[19]:

	Номер	Дата создания	Дата оплаты	Title	Статус	Заработано	Город	Платежная система
0	1062823	01.12.2019 10:50	01.12.2019 10:52	Курс обучения «Эксперт»	Завершен	29597.5	Чита	Сбербанк эквайринг
16	1062949	05.12.2019 21:57	06.12.2019 9:16	Курс обучения «Эксперт»	Завершен	29597.5	г.Санкт-Петербург и Ленинградская область	Сбербанк эквайринг
21	1063014	08.12.2019 21:56	08.12.2019 23:51	Курс обучения «Эксперт»	Завершен	29597.5	г.Санкт-Петербург и Ленинградская область	Сбербанк эквайринг
37	1063251	14.12.2019 14:34	16.12.2019 13:11	Курс обучения «Эксперт»	Завершен	42750.0	Балхаш	Системы быстрых переводов
46	1063333	16.12.2019 11:11	17.12.2019 13:42	Курс обучения «Специалист»	Завершен	29695.7	Московская область, г. Подольск	Сбербанк эквайринг

Рисунок 20 – Выборка данных со статусом ‘Завершен’ и уплаченной суммой больше 20 000 руб

```
In [1]: import pandas as pd

In [2]: df = pd.read_csv('lesson_1_data.csv', encoding = 'Windows-1251', sep = ';')

In [20]: df.query("Город == 'Чита' or Город == 'Балхаш'")
```

Out[20]:

	Номер	Дата создания	Дата оплаты	Title	Статус	Заработано	Город	Платежная система
0	1062823	01.12.2019 10:50	01.12.2019 10:52	Курс обучения «Эксперт»	Завершен	29597.50	Чита	Сбербанк эквайринг
37	1063251	14.12.2019 14:34	16.12.2019 13:11	Курс обучения «Эксперт»	Завершен	42750.00	Балхаш	Системы быстрых переводов
210	1063792	19.12.2019 22:00	19.12.2019 22:13	Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автос...	Завершен	2935.44	Балхаш	Яндекс.Касса

Рисунок 21 - Выборка данных, относящихся к городам Чита и Балхаш

Таким образом, условие запроса является параметром метода `query` и оформляется в кавычках. При этом, можно использовать логические операции and и or для построения сколь угодно сложной логики условия.

Последовательное выполнение методов (цепочки методов)

Существует простой и эффективный способ для объединения нескольких действий в одно. Зачастую методы датафреймов возвращают результат, который тоже является датафреймом (например так, как это делает метод `query`). Следовательно, от него тоже можно вызвать метод.

Например, стоит задача узнать – сколько всего денег было заработано в городах Чита и Балхаш. Очевидно, для этого нам потребуется выполнить два действия: сначала выполнить запрос `query` (как на рис. 21), а затем использовать функцию `sum` для суммирования значений в столбце ‘Заработано’:

```
df = df.query("Город == 'Чита' or Город == 'Балхаш'")
df['Заработано'].sum()
```

Этот код выдаст правильный результат, но он громоздкий. Можно сократить запись следующим образом:

```
df.query("Город == 'Чита' or Город ==
'Балхаш'")['Заработано'].sum()
```

Или другой пример: отсортируем данные для городов Чита и Балхаш по названию курса (столбец Title). В этом случае нам потребуется цепочка из двух методов `query` и `sort_values` (рис. 22). Для удобства понимания таких длинных цепочек допускается их разбиение на блоки, где каждый следующий метод записывается с новой строки. При этом весь блок заключается в скобки (рис. 23).

```
In [1]: import pandas as pd

In [2]: df = pd.read_csv('lesson_1_data.csv', encoding = 'Windows-1251', sep = ';')

In [25]: df.query("Город == 'Чита' or Город == 'Балхаш'").sort_values('Title')
```

Out[25]:

	Номер	Дата создания	Дата оплаты	Title	Статус	Заработано	Город	Платежная система
0	1052823	01.12.2019 10:50	01.12.2019 10:52	Курс обучения «Эксперт»	Завершен	29507.50	Чита	Сбербанк эквайринг
37	1053251	14.12.2019 14:34	16.12.2019 13:11	Курс обучения «Эксперт»	Завершен	42750.00	Балхаш	Системы быстрых переводов
210	1053792	19.12.2019 22:00	19.12.2019 22:13	Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автос...	Завершен	2935.44	Балхаш	Яндекс.Касса

Рисунок 22 – Цепочка методов (обычная версия)

```
In [1]: import pandas as pd

In [2]: df = pd.read_csv('lesson_1_data.csv', encoding = 'Windows-1251', sep = ';')

In [30]: (df.query("Город == 'Чита' or Город == 'Балхаш'")
           .sort_values('Title'))
```

Out[30]:

	Номер	Дата создания	Дата оплаты	Title	Статус	Заработано	Город	Платежная система
0	1052823	01.12.2019 10:50	01.12.2019 10:52	Курс обучения «Эксперт»	Завершен	29507.50	Чита	Сбербанк эквайринг
37	1053251	14.12.2019 14:34	16.12.2019 13:11	Курс обучения «Эксперт»	Завершен	42750.00	Балхаш	Системы быстрых переводов
210	1053792	19.12.2019 22:00	19.12.2019 22:13	Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автос...	Завершен	2935.44	Балхаш	Яндекс.Касса

Рисунок 23 – Цепочка методов (с разбиением на блоки)

Блочный способ оформления цепочек позволяет существенно повысить наглядность и читабельность программного кода.

Группировка и агрегация

Для упрощения дальнейшего анализа данных, часто бывает полезным сгруппировать данные датафрейма по определенным столбцам. Выполнить это можно с помощью функции `groupby`. Например, группировка данных по названию курса (столбец Title) будет выглядеть так:

```
df.groupby('Title')
```

Однако, после выполнения такой группировки, функция ничего не возвращает и мы не увидим результат на экране. Поэтому группировка используется как вспомогательное действие для некоторых других действий – в частности, агрегации.

Функция агрегации – `agg` позволяет произвести определенные действия над сгруппированными данными. Например, можно просуммировать сгруппированные данные по столбцу ‘Заработано’. Тем самым, мы можем подсчитать, сколько всего денег заработано на продаже каждого курса – рис. 24.

Существуют разные способы передать в `agg` что и как нужно агрегировать. Самый простой и полный – использовать словарь, где ключи это названия столбцов, а значения – применяемые к ним функции. Чтобы применить несколько функций, используйте список функций. Функции могут быть переданы сами (`sum`), либо строки, их обозначающие (`'sum'`).

В результате агрегации из массива значений (столбец) получается одно значение на каждую агрегирующую функцию.

```
In [1]: import pandas as pd

In [43]: df = pd.read_csv('lesson_1_data.csv', encoding = 'windows-1251', sep = ';')

In [45]: df.groupby('Title').agg({'Заработано':'sum'})
Out[45]:
```

Title	Заработано
Курс обучения «Консультант»	208163.40
Курс обучения «Специалист»	160862.64
Курс обучения «Эксперт»	148992.80
Курс от Школы Диетологов. Бизнес	18752.54
Курс от Школы Диетологов. Повышение квалификации.	80384.92
Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автосписанием	386947.20

Рисунок 24 – Группировка данных по столбцу ‘Title’ и агрегация (суммирование) по столбцу ‘Заработано’

Можно немного усложнить задачу и после агрегации произвести сортировку по возрастанию заработанных денег (рис. 25).

```
In [1]: import pandas as pd

In [43]: df = pd.read_csv('lesson_1_data.csv', encoding = 'windows-1251', sep = ';')

In [49]: df.groupby('Title').agg({'Заработано':'sum'}).sort_values('Заработано')
```

Out[49]:

Title	Заработано
Курс от Школы Диетологов. Бизнес	18752.54
Курс от Школы Диетологов. Повышение квалификации.	88384.92
Курс обучения «Эксперт»	148662.00
Курс обучения «Специалист»	160862.64
Курс обучения «Консультант»	208183.49
Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автосписанием	388947.20

Рисунок 25 – Группировка данных по столбцу ‘Title’, агрегация (суммирование) по столбцу ‘Заработано’ и последующая сортировка

Ну и наконец, можно проделать все эти же действия только для городов Чита и Балхаш (рис. 26), используя предварительный запрос с рис. 21.

```
In [1]: import pandas as pd

In [43]: df = pd.read_csv('lesson_1_data.csv', encoding = 'windows-1251', sep = ';')

In [50]: (df.query ("Город == 'Чита' or Город == 'Балхаш'")
           .groupby('Title')
           .agg({'Заработано':'sum'})
           .sort_values('Заработано'))
```

Out[50]:

Title	Заработано
Подписка «ОНЛАЙН ДИЕТОЛОГ» с ежемесячным автосписанием	2595.44
Курс обучения «Эксперт»	72347.60

Рисунок 26 - Группировка данных по столбцу ‘Title’, агрегация (суммирование) по столбцу ‘Заработано’ и последующая сортировка: только для городов Чита и Балхаш

Поскольку цепочка функций получается достаточно длинной, то для удобства ее чтения она была разбита на блоки.

Запись датафрейма в файл формата csv

Датафрейм можно записывать в различные форматы. Рассмотрим ситуацию с уже знакомым нам форматом csv. Для этого нужно применить к датафрэйму метод `to_csv` и передать в него путь, по которому необходимо создать файл.

```
df.to_csv('path_to_your_output.csv')
```

Существует несколько дополнительных полезных параметров:

– `index` – в качестве первой колонки в файл будет записан индекс датафрейма (по умолчанию True);
– `sep` – задает разделитель, который будет использован при записи в файл (по умолчанию – ‘,’);
– `enconding` - задает кодировку текста, которая будет использована при записи в файл (по умолчанию – ‘utf-8’).

Например, так будет выглядеть (рис. 27) запись полученного на рис. 26 результата в файл output.csv в папку D:\csv_files\ - индексная колонка не записывается, разделитель – ‘;’, кодировка ‘Windows-1251’. После записи файла сделаем проверку и считаем сохраненные туда данные в новый датафрейм test.

Рисунок 27 – Запись результата анализа данных в файл с последующей проверкой

Обратим внимание, что в функции группировки использован дополнительный параметр `as_index=False`, для того, чтобы группировка была произведена не виртуально (на основе индексации), а физически (чтобы результаты могли быть записаны в файл).

Некоторые полезные функции

Часто возникает задача автоматической замены в текстовых данных одного символа на другой. Для этого удобно использовать метод `replace`:

```
replace ('исходный символ', 'символ замены')
```

Например,

```
test_string = "Это строка с пробелами"
test_string.replace(' ','_')
#Все пробелы заменяются символом '_'
#Результат: "Это_строка_с_пробелами"
```

Данный метод очень удобно использовать при переименовании столбцов или строковых значений в датафрейме.

Другая, часто встречающаяся задача, – удаление лишних пробелов в начале и конце строки. Для ее решения применяется метод `strip`. Например,

```
test_string = "    Это строка с пробелами   "
test_string.strip()
#Результат: "Это_строка_с_пробелами"
```

При работе с числовыми данными также встречаются типовые задачи, решаемые встроенными методами. Одна из таких задач – округление дробных чисел. Решается с помощью метода `round`. В качестве параметра, ему передается целое число, указывающее точность округления (сколько позиций после запятой останется после округления). Например,

```
round (3.123456, 2)
#Результат: 3.12
```

Векторизация

Векторизация – это специальная техника, позволяющая в Pandas быстро перебирать элементы массивов (коллекций), которые в чистом Python требуют как минимум одного цикла.

Благодаря векторизации, можно делать различные операции с группами столбцов, не отвлекаясь на написание циклов. Например, для циклического перебора всех элементов одного столбца, достаточно просто дать команду с его названием (рис. 28).

Рисунок 28 – Векторизация элементов в столбце ‘Заработано’

При этом, на рисунке показан не только перебор элементов, но и примеры перебора с одновременным выполнением над ними определенных действий – выполняется перебор с одновременными умножением значений на 2, а затем, перебор с проверкой условия >5000.

Создание собственных функций

Для того, чтобы создать собственную функцию, необходимо использовать ключевую конструкцию `def`, имеющую следующий синтаксис:

```
def <имя функции> (<параметры>):
    <тело функции>
    return <результат>
```

Чтобы вернуть результат работы функции, последней строкой тела функции должна идти команда `return`.

В качестве примера, напишем свою функцию умножения на 2 и используем ее для удвоения сумм в столбце ‘Заработано’ – рис. 29.

Рисунок 29 – Создание собственной функции и использование ее в процессе векторизации элементов в столбце ‘Заработано’

Этот пример показывает удивительную гибкость Pandas при сочетании векторизации и функций.

СПИСОК КОНТРОЛЬНЫХ ВОПРОСОВ

1. В чем основное назначение библиотеки Pandas?
2. Что понимается под термином «датафрейм»?
3. В чем особенность формата файлов csv?
4. Назовите известный Вам способ подключения (импорта) библиотеки Pandas и используемые при этом параметры.
5. С помощью какого метода выполняется чтение данных из файла csv?

6. На какие параметры важно обратить внимание при чтении csv-файла? Какие ошибки могут возникнуть в процессе чтения и как их устранить?

7. В чем отличие между методами и атрибутами датафрейма? Приведите примеры известных Вам методов и атрибутов.

8. Возникла задача переименования некоторых строк и столбцов датафрейма. Методы или атрибуты Вы будете использовать для ее решения? Какие?

9. Необходимо определить тип каждого столбца датафрейма. Методы или атрибуты Вы будете использовать для решения этой задачи? Какие?

10. Какие данные содержит атрибут датафрейма `shape` ?

11. К какому результату приведет вызов метода `describe`?

12. Каким образом можно обратиться к столбцу датафрейма? А к отдельному элементу столбца?

13. Какие основные вычислительные методы Pandas могут быть применены к числовым столбцам датафрейма?

14. Как произвести сортировку элементов числового столбца по убыванию и по возрастанию?

15. Для чего используется метод `value_counts`? Какие параметры этого метода Вы знаете?

16. Опишите механизм запросов к датафрейму. Приведите примеры запросов. Что возвращает метод `query`?

17. Что понимается под термином «цепочка методов»? Каким образом оформляются такие цепочки и с какой целью?

18. Опишите процедуру группировки и агрегации данных средствами библиотеки Pandas. Приведите примеры.

19. В чем назначение метода `to_csv`? Какие основные параметры этого метода Вы знаете?

20. Что понимается под векторизацией в Pandas?

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ ТЕХНОЛОГИИ СЕМАНТИЧЕСКОГО ВЕБ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

Лабораторная работа № 1

Изучение языков XML-технологии

(на примере языка разметки функциональных блоков IEC 61499)

Цель работы: изучение языка разметки базисных и составных функциональных блоков (ФБ) IEC 61499 на основе XML.

Пример базисного ФБ:

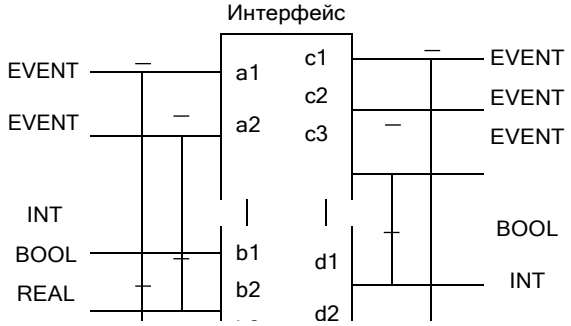


Рис. 1, интерфейс функционального блока.

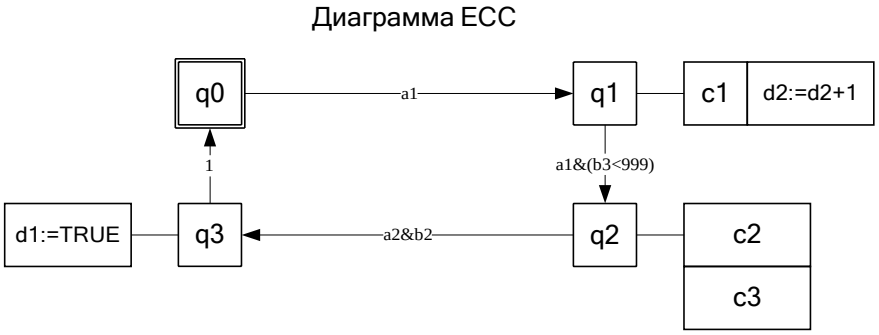


Рис. 2, диаграмма ECC.

Методические указания по представлению базисного ФБ на языке XML

1) Каждому классу, с которым связана некоторая диаграмма состояний, ставится в соответствие базисный ФБ. Этот класс UML назовем БФБ-классом. Класс в диаграмме классов, не имеющий диаграммы состояний, к БФБ-классу не относятся. Примечание: для представления базисного ФБ на языке XML используется пара тегов <BasicFB>…</BasicFB>

2) Связь БФБ-класса и диаграммы состояний может производиться по их именам. Если некоторый БФБ-класс и некоторая диаграмма состояний имеют одно и то же имя, то считается, что она связана с данным БФБ-классом.

3) Диаграмме состояний UML, связанной с некоторым БФБ-классом, соответствует ECC-диаграмма соответствующего базисного ФБ. Примечание: диаграмма ECC описывается на XML внутри пары тегов <ECC>…</ECC>

4) Операции (методу) БФБ-класса, имеющему стереотип EVENT_INPUT, ставится в соответствие событийный вход базисного ФБ. Имя данного событийного входа совпадает с именем соответствующего метода БФБ-класса. Примечание: событийные входы ФБ описываются на XML внутри пары тегов <EventInputs>…</EventInputs>

5) Операции (методу) БФБ-класса, имеющему стереотип EVENT_OUTPUT, ставится в соответствие событийный выход базисного ФБ. Имя данного событийного выхода совпадает с именем соответствующего метода БФБ-класса. Примечание: событийные выходы ФБ описываются на XML внутри пары тегов <EventOutputs>…</EventOutputs>

6) Атрибуту БФБ-класса, имеющему стереотип VAR_INPUT, ставится в соответствие входная переменная базисного ФБ. Имя данной входной переменной совпадает с именем соответствующего атрибута БФБ-класса. Примечание: входные переменные ФБ описываются на XML внутри пары тегов <InputVars>…</InputVars>

7) Атрибуту БФБ-класса, имеющему стереотип VAR_OUTPUT, ставится в соответствие выходная переменная базисного ФБ. Имя данной выходной переменной совпадает с именем соответствующего атрибута БФБ-класса. Примечание: выходные переменные ФБ описываются на XML внутри пары тегов <OutputVars>…</OutputVars>

8) Атрибуту БФБ-класса, имеющему стереотип VAR, ставится в соответствие внутренняя переменная базисного ФБ. Имя данной внутренней переменной совпадает с именем соответствующего атрибута БФБ-класса. Примечание: внутренние переменные ФБ описываются на XML внутри пары тегов <InternalVars></InternalVars>

9) Методу БФБ-класса, имеющему параметры, в базисном ФБ ставится в соответствие событийная линия, связанная с информационными линиями, соответствующими данным параметрам, с помощью квалификатора WITH. Примечание: для задания WITH-связи используется тег <With … />

10) Каждому параметру метода БФБ-класса должен быть назначен тип (Integer, Boolean, String и т.д.). Данный тип используется для генерации типа соответствующей переменной в базисном ФБ. Некоторые правила для преобразования типа параметра метода класса в тип переменной базисного ФБ: Integer®INT, Boolean®BOOL, String®STRING.

11) Параметр метода БФБ-класса может иметь начальные значения. Данные значения используются для определения начальных значений соответствующих переменных. Примечание: для описания переменной (входной, выходной, внутренней) вместе с ее типом и начальными значениями используется тег <VarDeclaration Name="имя_переменной" Type="имя_типа" InitialValue="начальное_значение" />

12) Каждому состоянию диаграммы состояний (за исключение начального состояния, обозначаемого черным кружком) соответствует состояние ECC. Имя состояния ECC совпадает с именем состояния UML диаграммы состояний. Примечание: для представления списка состояний ECC на XML используется пара тегов <ECState></ECState>. Отдельное состояние определяется с помощью пары тегов <ECState></ECState>

13) Начальному состоянию диаграммы состояний не соответствует никакое состояние ECC. Из начального состояния диаграммы состояний должна вести стрелка в состояние, соответствующее начальному состоянию ECC. Данная стрелка помечается условием

[true]. Примечание: описание начального состояния на XML идет первым (впереди других состояний).

14) Конечное состояние диаграммы состояний не используется для представления ECC, поскольку в ECC отсутствует понятие конечного состояния.

15) С состоянием диаграммы состояний могут быть связаны два вида действий: “Action” (выполнение какой-то работы) и “Send Event” (посылка сообщения). Действие первого и второго вида может возникнуть только при входе в состояние (On Entry). Первое действие в базисном ФБ соответствует выполнению алгоритма, связанному с состоянием ECC, а второе действие – выдаче выходных сигналов. Примечание: для описания действий ECC в XML используется тег <ECAction Algorithm="имя_алгоритма" Output="имя_событийного_выхода" />

16) Имя действия типа “Action” в UML диаграмме состояний соответствует имени алгоритма, связанного с состоянием ECC. Сам текст алгоритма на каком-либо языке программирования (например, ST, Java) определяется текстуально в виде комментария, присоединенному к соответствующему состоянию диаграммы состояний. Текст алгоритма не анализируется и включается в описание базисного ФБ в исходной форме. Примечание: для определения алгоритма на XML используется тег <Algorithm Name="имя_алгоритма" Comment="комментарий">

17) Имя действия типа “Send Event” в диаграмме состояний соответствует (в базисном ФБ) имени событийного выхода, на которое будет выдано событие при переходе ECC в соответствующее состояние. При этом (необязательно) возможно задание списка аргументов (Send arguments), которые пересылаются с данным событием. Этот список можно использовать для контроля выходного интерфейса базисного ФБ. Например, если в списке Send arguments у выходного события OUT1 имеется аргумент A1, а в выходном интерфейсе соответствующей выходной переменной нет, то это говорит об ошибке.

18) Переходу из одного состояния в другое в UML диаграмме состояний соответствует (в базисном ФБ) переход состояний ECC-диаграммы. Переход из начального состояния диаграммы состояний в состояние, соответствующее начальному состоянию ECC, в описание базисного ФБ не включается. . Примечание: для описания переходов состояний ECC на XML используется тег <ECTransition Source="состояние-источник" Destination="состояние-приемник" Condition="булево_выражение" />

19) Для описания переходов состояний в UML диаграмме состояний используются: a) имя события, вызывающее переход; b) логическое условие перехода (Guard Condition), например, CU[CV<65535]. Подобное описание преобразуется в условие перехода в ECC базисного ФБ. Для вышеприведенного случая это будет логическое условие вида CU&(CV<65535). Условие Guard Condition не анализируется и включается в условие перехода ECC практически без изменений. Условию Guard Condition со значением [true] соответствует ECC-условие, представляемое как 1. (На XML это выражается как Condition="1")

Приложение 1. XML-представление базисного ФБ

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE FBType SYSTEM "http://www.holobloc.com/xml/LibraryElement.dtd" >
<FBType Name="BasicFB" Comment="Event-driven bistable" >
  <Identification Standard="61499-1-A.1" Classification="Event Processing" />
  <VersionInfo Organization="Rockwell Automation" Version="0.2" Author="JHC"
Date="2003-12-30" Remarks="Updated for 61499-1 CDV." />
  <VersionInfo Organization="Rockwell Automation" Version="0.1" Author="JHC"
Date="2001-08-16" Remarks="XML version" />
  <VersionInfo Organization="Rockwell Automation" Version="0.0" Author="JHC"
Date="1998-04-11" Remarks="SRC version" />
  <CompilerInfo header="package fb.rt.events;" >
    <Compiler Language="Java" Vendor="IBM" Product="VisualAge" Version="3.0" />
  </CompilerInfo>
  <InterfaceList>
    <EventInputs>
      <Event Name="A1" Comment="Set Event A1" >
                  <With Var="B3" />
      </Event>
      <Event Name="A2" Comment="Set Event A2" >
                  <With Var="B1" />
                  <With Var="B2" />
      </Event>
    </EventInputs>
    <EventOutputs>
      <Event Name="C1" Comment="Output Event C1" >
            <With Var="D2" />
          </Event>
          <Event Name="C2" Comment="Output Event C1" >
          </Event>
          <Event Name="C3" Comment="Output Event C1" >
       <With Var="D1" />
      </Event>
    </EventOutputs>
        <InputVars>
      <VarDeclaration Name="B1" Type="UINT" Comment="Input event qualifier" />
          <VarDeclaration Name="B2" Type="BOOL" Comment="Input event qualifier" />
          <VarDeclaration Name="B3" Type="REAL" Comment="Input event qualifier" />
    </InputVars>
    <OutputVars>
      <VarDeclaration Name="D1" Type="BOOL" Comment="Current Output State" />
          <VarDeclaration Name="D2" Type="UINT" Comment="Current Output State" />
    </OutputVars>
  </InterfaceList>
  <BasicFB>
    <ECC >
      <ECState Name="Q0" Comment="Do nothing" >
      </ECState>
      <ECState Name="Q1" Comment="Increase D2 for 1 and issue C1" >
        <ECAction Algorithm="Q1" Output="C1" />
      </ECState>
```

```
        <ECState Name="Q2" Comment="Issue C2 and C3" >
      <ECAction Output="C2" />
              <ECAction Output="C3" />
    </ECState>
        <ECState Name="Q3" Comment="Set D1" >
     <ECAction Algorithm="Q3" Output="" />
    </ECState>
    <ECTransition Source="Q0" Destination="Q1" Condition="A1" />
    <ECTransition Source="Q1" Destination="Q2" Condition="A1&#38;(B3&#60;999)" />
    <ECTransition Source="Q2" Destination="Q3" Condition="A2&#38;(B2=TRUE)" />
        <ECTransition Source="Q3" Destination="Q0" Condition="1" />
  </ECC>
 <Algorithm Name="Q1" Comment="Inc D2 and issue C1" >
  <ST Text="D2:=D2+1;&#10;" />
 </Algorithm>
 <Algorithm Name="Q3" Comment="Set D1" >
  <ST Text="D1:=TRUE;&#10;" />
 </Algorithm>
 </BasicFB>
</FBType>
```

Приложение 2. Результаты визуализации XML-представления в системе FBDK

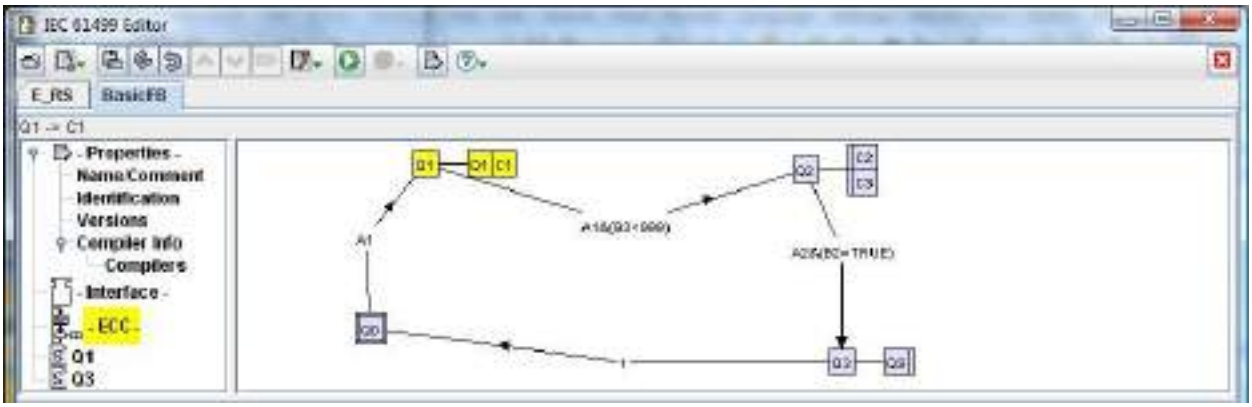


Рис. 3 Диаграмма ECC

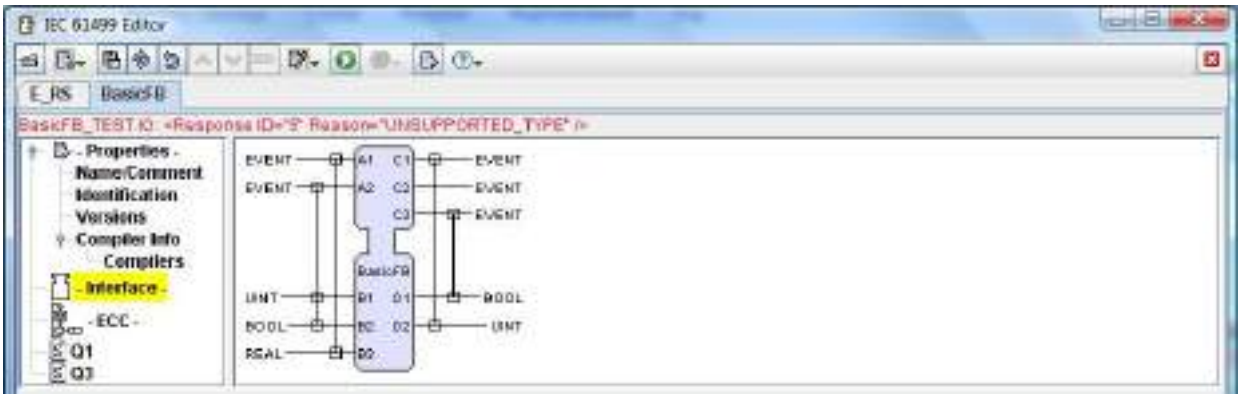


Рис. 4 Basic FB Интерфейс

Лабораторная работа № 1.

Создание OWL-онтологии семейных отношений в системе Protege

Версия 1.0

Цель работы: изучение основных элементов OWL-онтологий, освоение навыков работы в системе Protégé.

Порядок выполнения работы:

0) Изучить онтологию семейных отношений, заданную формулами дескриптивной логики. Данная онтология задается на уровне классов (*TBox*, терминологический словарь).

$$
\begin{aligned}
\text{Woman} &\equiv \text{Person} \sqcap \text{Female} \\
\text{Man} &\equiv \text{Person} \sqcap \neg\text{Woman} \\
\text{Mother} &\equiv \text{Woman} \sqcap \exists\text{hasChild.Person} \\
\text{Father} &\equiv \text{Man} \sqcap \exists\text{hasChild.Person} \\
\text{Parent} &\equiv \text{Father} \sqcup \text{Mother} \\
\text{Grandmother} &\equiv \text{Mother} \sqcap \exists\text{hasChild.Parent} \\
\text{MotherWithManyChildren} &\equiv \text{Mother} \sqcap \geqslant 3\,\text{hasChild} \\
\text{MotherWithoutDaughter} &\equiv \text{Mother} \sqcap \forall\text{hasChild.}\neg\text{Woman} \\
\text{Wife} &\equiv \text{Woman} \sqcap \exists\text{hasHusband.Man}
\end{aligned}
$$

Fig. A terminology (TBox) with concepts about family relationships.

1) Создать классы онтологии (*Classes*). Следует заметить, что базовыми классами будут классы *Person* и *Female*. Остальные классы будут вычисляемыми.

2) Создать отношения онтологии (*Object Properties*, объектные свойства).

3) Создать аксиомы онтологии, используя встроенные редактор аксиом. Для этого использовать цепочку: кнопка *Equivalent to* -> вкладка *Class Expression Editor*.

4) Разработать простое генеалогическое дерево.

5) В соответствии с этим деревом, создать экземпляры классов и отношений (*individuals, class assertion, object property assertion*).

6) Протестировать созданную онтологию, используя встроенный ризонер (*reasoner*) *Hermit*. При тестировании решается задача классификации, когда экземпляры автоматически распределяются по классам. Сопоставить полученную классификацию с исходным генеалогическим деревом.

7) Просмотреть и изучить сгенерированный код *OWL*-онтологии в различных форматах (*OWL/XML, RDF/XML, OWL functional syntax*), используя цепочку меню: *Window->Views->Ontology views.*

8) Сгенерировать визуальное представление онтологии, используя плагин *OntoGraf*. Для активизации данного плагина использовать цепочку меню: *Window-*

>*Tab->OntoGraf*. Построить несколько визуальных представлений: а) только на уровне классов; б) только на уровне экземпляров; в) смешанное представление, использующее классы и экземпляры.

9) Сохранить разработанную онтологию на диске в формате *RDF/XML*.

10) Составить отчет о проделанной работе в Word.

Методические указания

Работа выполняется в системе Protégé путем нажатия нужных кнопок и пунктов меню. Код OWL генерируется автоматически.

Вариативное задание. Добавить в онтологию описание следующего родственника:

Вариант	**Тип родственника**
1	дядя
2	тетя
3	племянник
4	племянница
5	двоюродный брат
6	двоюродная сестра
7	правнук
8	прабабушка
9	свояк
10	свояченица
11	тесть
12	свекор
13	потомок (любого уровня)
14	предок (любого уровня)

Примечание: Для определения родственников №№ 1-8 необходимо ввести и использовать объектное свойство hasParent, которое является обратным по отношению к свойству hasChild. При определении характеристик свойства hasParent в Protégé в поле Inverse of необходимо ввести hasChild.

Лабораторная работа № 2.

Создание SWRL-онтологии семейных отношений в системе Protege

Цель работы: изучение основных элементов SWRL-онтологий, освоение навыков работы в системе Protégé.

Порядок выполнения работы:

0) Изучить онтологию семейных отношений, заданную формулами дескриптивной логики. Данная онтология задается на уровне классов (*TBox*, терминологический словарь).

$$
\begin{aligned}
\text{Woman} &\equiv \text{Person} \sqcap \text{Female} \\
\text{Man} &\equiv \text{Person} \sqcap \neg\text{Woman} \\
\text{Mother} &\equiv \text{Woman} \sqcap \exists\text{hasChild.Person} \\
\text{Father} &\equiv \text{Man} \sqcap \exists\text{hasChild.Person} \\
\text{Parent} &\equiv \text{Father} \sqcup \text{Mother} \\
\text{Grandmother} &\equiv \text{Mother} \sqcap \exists\text{hasChild.Parent} \\
\text{MotherWithManyChildren} &\equiv \text{Mother} \sqcap \geqslant 3\,\text{hasChild} \\
\text{MotherWithoutDaughter} &\equiv \text{Mother} \sqcap \forall\text{hasChild.}\neg\text{Woman} \\
\text{Wife} &\equiv \text{Woman} \sqcap \exists\text{hasHusband.Man}
\end{aligned}
$$

Fig. A terminology (TBox) with concepts about family relationships.

1) Создать классы онтологии (*Classes*). Следует заметить, что базовыми классами будут классы *Person* и *Female*. Остальные классы будут вычисляемыми.

2) Создать отношения онтологии (*Object Properties*, объектные свойства).

3) Создать *SWRL*-правила для каждого вычисляемого класса, используя встроенный редактор правил. Чтобы добраться до редактора правил, следует пройтись по цепочке меню *Window->Views->Ontology views->Rules*.

4) Разработать простое генеалогическое дерево.

5) В соответствии с этим деревом, создать экземпляры классов и отношений (*individuals, class assertion, object property assertion*).

6) Протестировать созданную онтологию, используя встроенный ризонер (*reasoner*) *Hermit*. При тестировании решается задача классификации, когда экземпляры автоматически распределяются по классам. Сопоставить полученную классификацию с исходным генеалогическим деревом.

7) Просмотреть и изучить сгенерированный код *OWL*/*SWRL*-онтологии в различных форматах (*OWL*/*XML*, *RDF*/*XML*, *OWL functional syntax*), используя цепочку меню: *Window->Views->Ontology views.*

8) Сгенерировать визуальное представление онтологии, используя плагин *OntoGraf*. Для активизации данного плагина использовать цепочку меню: *Window->Tab->OntoGraf*. Построить несколько визуальных представлений: а) только на уровне классов; б) только на уровне экземпляров; в) смешанное представление, использующее классы и экземпляры.

9) Сохранить разработанную онтологию на диске в формате *RDF*/*XML*.

10) Составить отчет о проделанной работе в Word.

Методические указания

Большинство шагов в обеих лаб. работах совпадают, за исключением шага 3. Для ускорения выполнения работы № 2 рекомендуется использовать онтологию из работы № 1, из которой удалены только аксиомы, а остальные элементы оставлены без изменения.

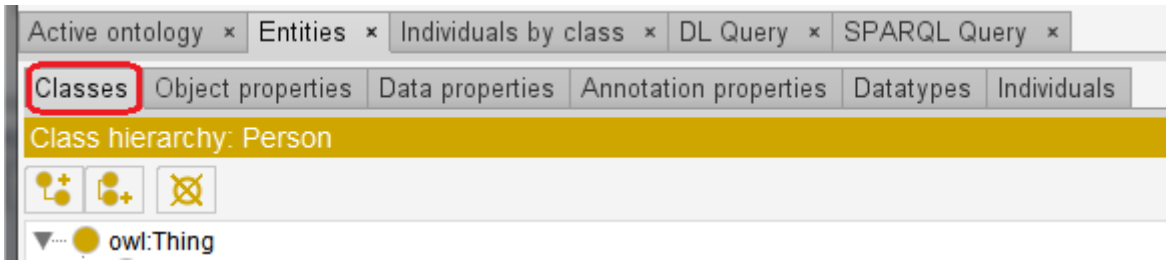
ПРИЛОЖЕНИЕ 1.

Основные сведения по работе в системе Protege

Создание классов (Classes)

Запускаем Protégé

Входим во вкладку Entities->Classes



Первоначально в системе только один общий класс owl:Think. Он включает в себя абсолютно все существующие (и несуществующие) в мире классы.

Для создания нового класса надо навести курсор на класс owl: Think и нажать кнопку  . Появится окно для ввода имени класса.

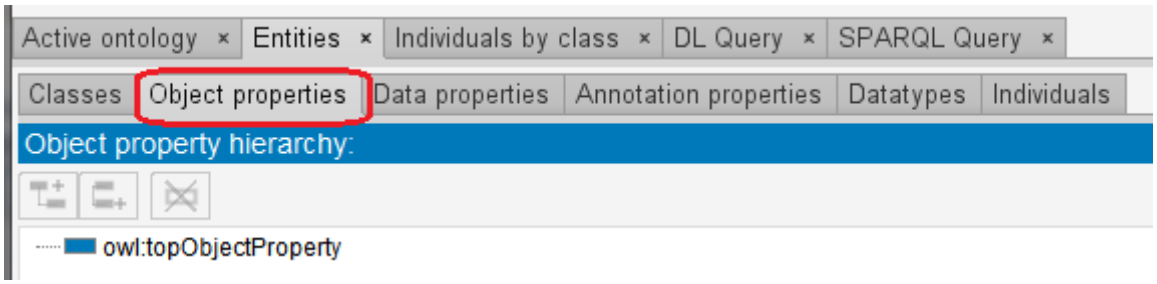
Рекомендуется использовать только латинские буквы, а первую букву лучше делать заглавной (это общепринятые правила хорошего тона).

Для удаления класса надо навести курсор на удаляемый класс и нажать кнопку 

Создание объектных свойств (Object properties)

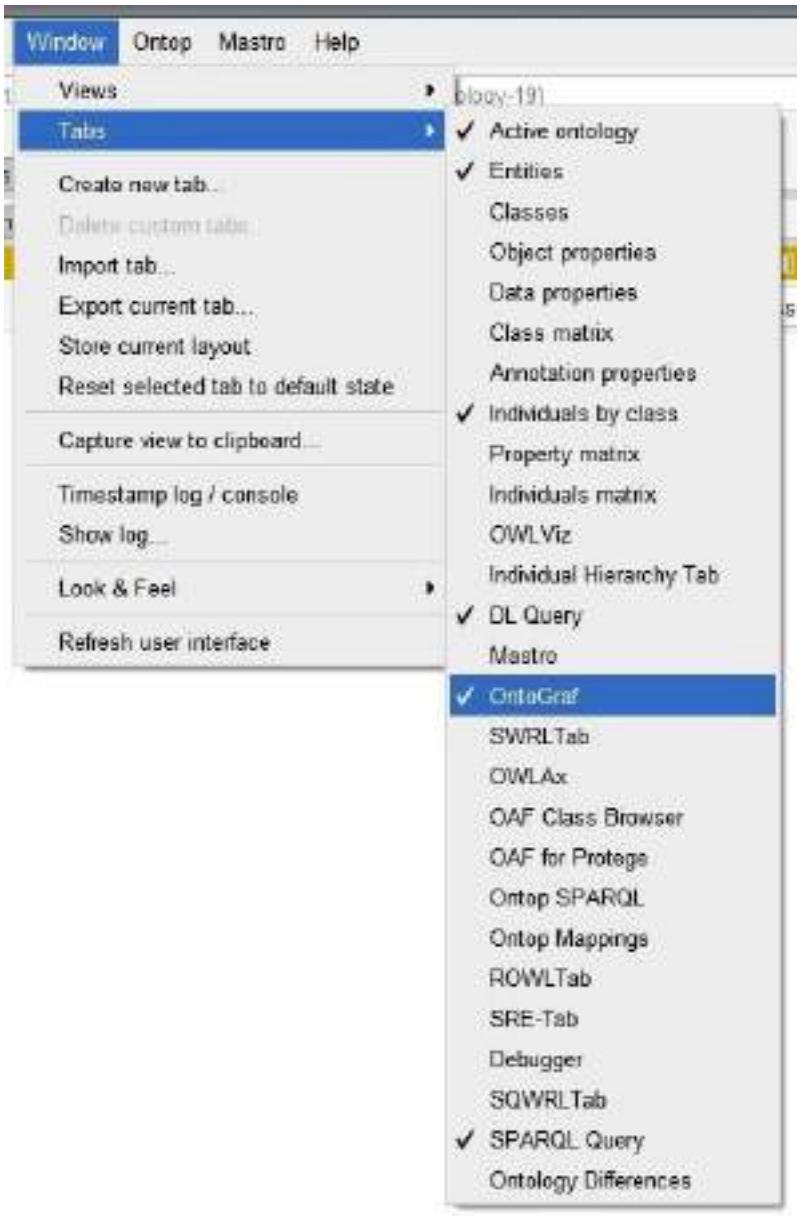
Объектные свойства определяют *отношения* между классами.

Входим во вкладку Entities-> Object properties

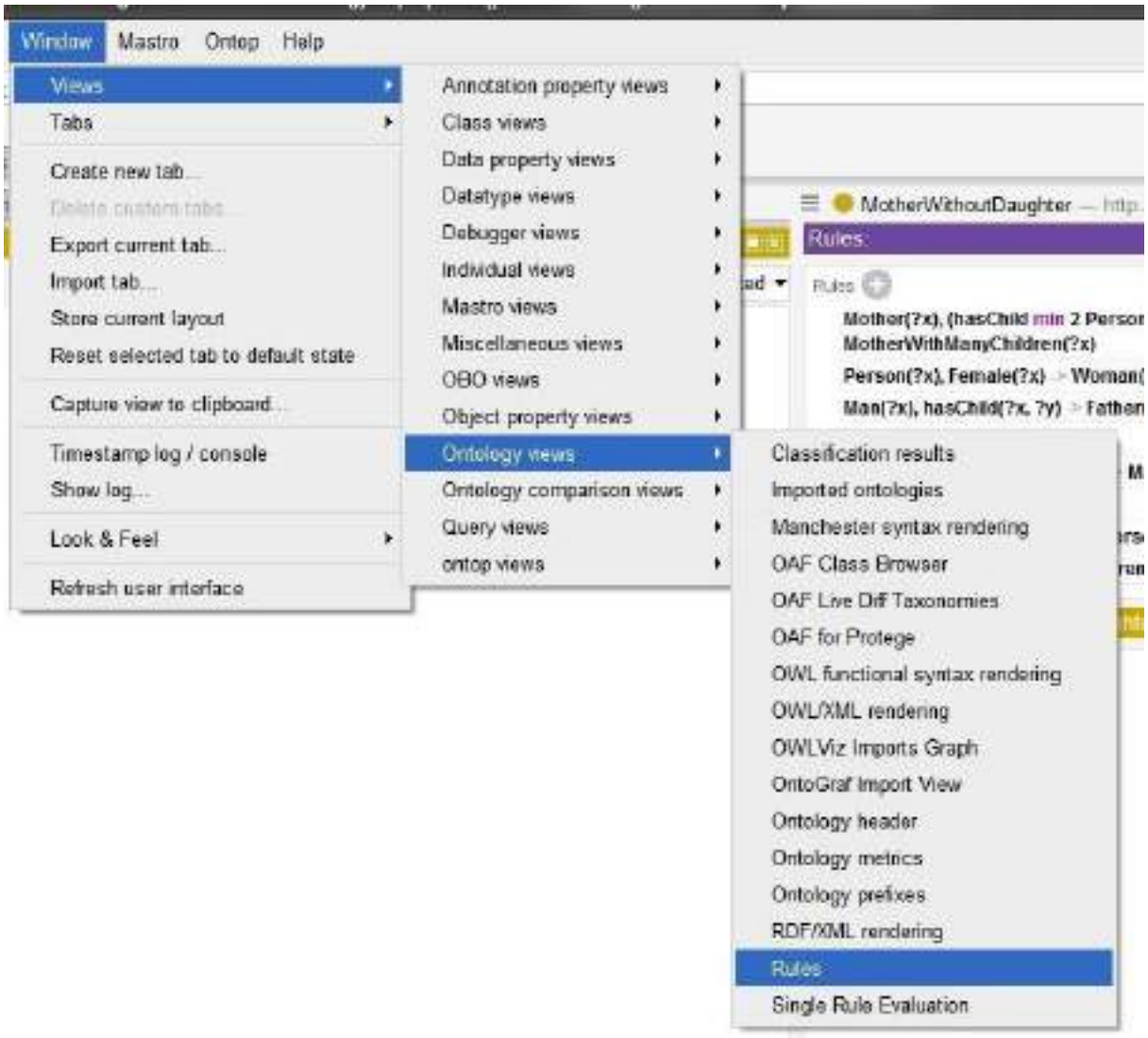


Первоначально в системе только одно общее объектное свойство owl:Think. Оно включает в себя абсолютно все существующие (и несуществующие) в мире объектные свойства.

Как запустить плагин OntoGraf



Как запустить редактор SWRL-правил



Лабораторная работа № 4.

Обработка Web-онтологий

Цель работы: изучение основных элементов языка SWRL, освоение навыков работы в системе Protégé, создание SWRL-онтологии семейных отношений в системе Protégé, логический вывод в онтологии на основе SWRL.

Порядок выполнения работы:

0) Изучить онтологию семейных отношений, заданную формулами дескриптивной логики. Данная онтология задается на уровне классов (*TBox*, терминологический словарь).

$$
\begin{aligned}
\text{Woman} &\equiv \text{Person} \sqcap \text{Female} \\
\text{Man} &\equiv \text{Person} \sqcap \neg\text{Woman} \\
\text{Mother} &\equiv \text{Woman} \sqcap \exists\text{hasChild.Person} \\
\text{Father} &\equiv \text{Man} \sqcap \exists\text{hasChild.Person} \\
\text{Parent} &\equiv \text{Father} \sqcup \text{Mother} \\
\text{Grandmother} &\equiv \text{Mother} \sqcap \exists\text{hasChild.Parent} \\
\text{MotherWithManyChildren} &\equiv \text{Mother} \sqcap \geqslant 3\,\text{hasChild} \\
\text{MotherWithoutDaughter} &\equiv \text{Mother} \sqcap \forall\text{hasChild.}\neg\text{Woman} \\
\text{Wife} &\equiv \text{Woman} \sqcap \exists\text{hasHusband.Man}
\end{aligned}
$$

Fig. A terminology (TBox) with concepts about family relationships.

1) Создать классы онтологии (*Classes*). Следует заметить, что базовыми классами будут классы *Person* и *Female*. Остальные классы будут вычисляемыми.

2) Создать отношения онтологии (*Object Properties*, объектные свойства).

3) Создать *SWRL*-правила для каждого вычисляемого класса, используя встроенный редактор правил. Чтобы добраться до редактора правил, следует пройтись по цепочке меню *Window->Views->Ontology views->Rules*.

4) Разработать простое генеалогическое дерево.

5) В соответствии с этим деревом, создать экземпляры классов и отношений (*individuals, class assertion, object property assertion*).

6) Протестировать созданную онтологию, используя встроенный ризонер (*reasoner*) *Hermit*. При тестировании решается задача классификации, когда

экземпляры автоматически распределяются по классам. Сопоставить полученную классификацию с исходным генеалогическим деревом.

7) Просмотреть и изучить сгенерированный код *OWL*/*SWRL*-онтологии в различных форматах (*OWL*/*XML*, *RDF*/*XML*, *OWL functional syntax*), используя цепочку меню: *Window->Views->Ontology views.*

8) Сгенерировать визуальное представление онтологии, используя плагин *OntoGraf*. Для активизации данного плагина использовать цепочку меню: *Window->Tab->OntoGraf*. Построить несколько визуальных представлений: а) только на уровне классов; б) только на уровне экземпляров; в) смешанное представление, использующее классы и экземпляры.

9) Сохранить разработанную онтологию на диске в формате *RDF*/*XML*.

10) Составить отчет о проделанной работе в Word.

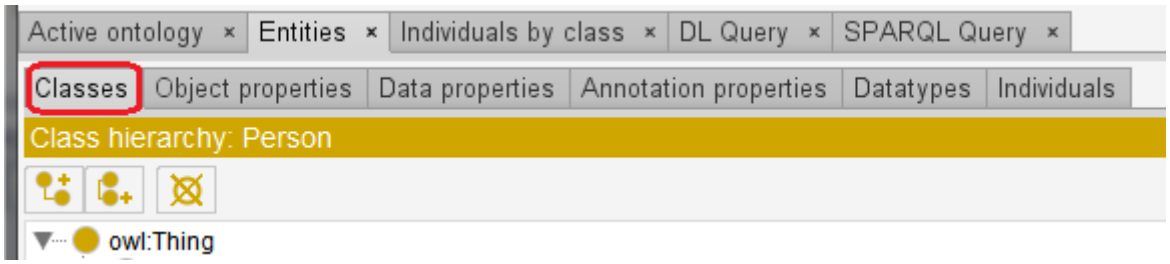
ПРИЛОЖЕНИЕ 1.

Основные сведения по работе в системе Protege

Создание классов (Classes)

Запускаем Protégé

Входим во вкладку Entities->Classes



Первоначально в системе только один общий класс owl:Think. Он включает в себя абсолютно все существующие (и несуществующие) в мире классы.

Для создания нового класса надо навести курсор на класс owl: Think и нажать кнопку  . Появится окно для ввода имени класса.

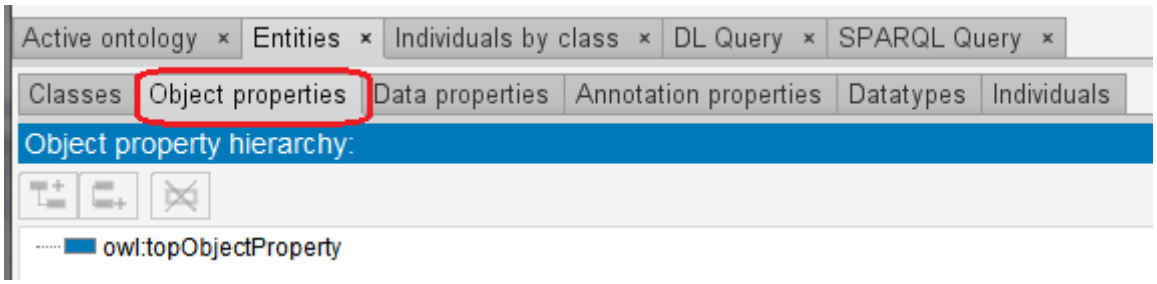
Рекомендуется использовать только латинские буквы, а первую букву лучше делать заглавной (это общепринятые правила хорошего тона).

Для удаления класса надо навести курсор на удаляемый класс и нажать кнопку 

Создание объектных свойств (Object properties)

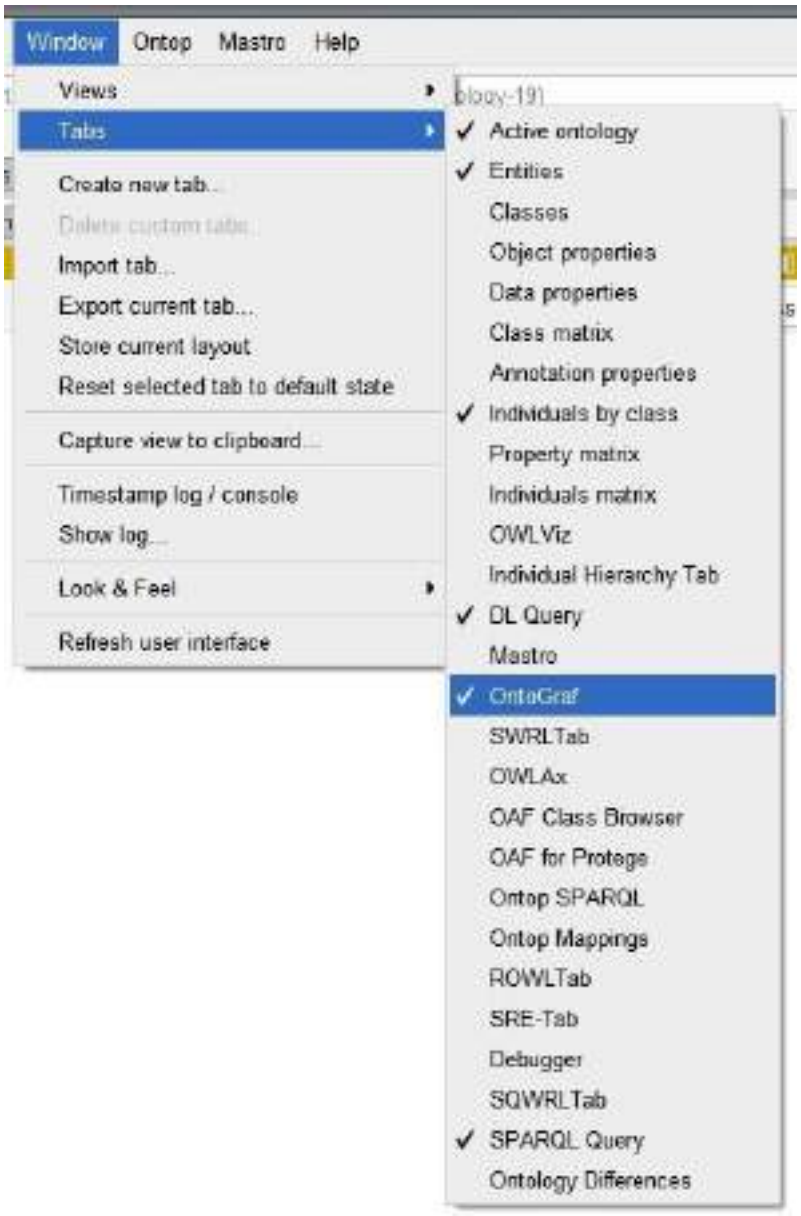
Объектные свойства определяют *отношения* между классами.

Входим во вкладку Entities-> Object properties

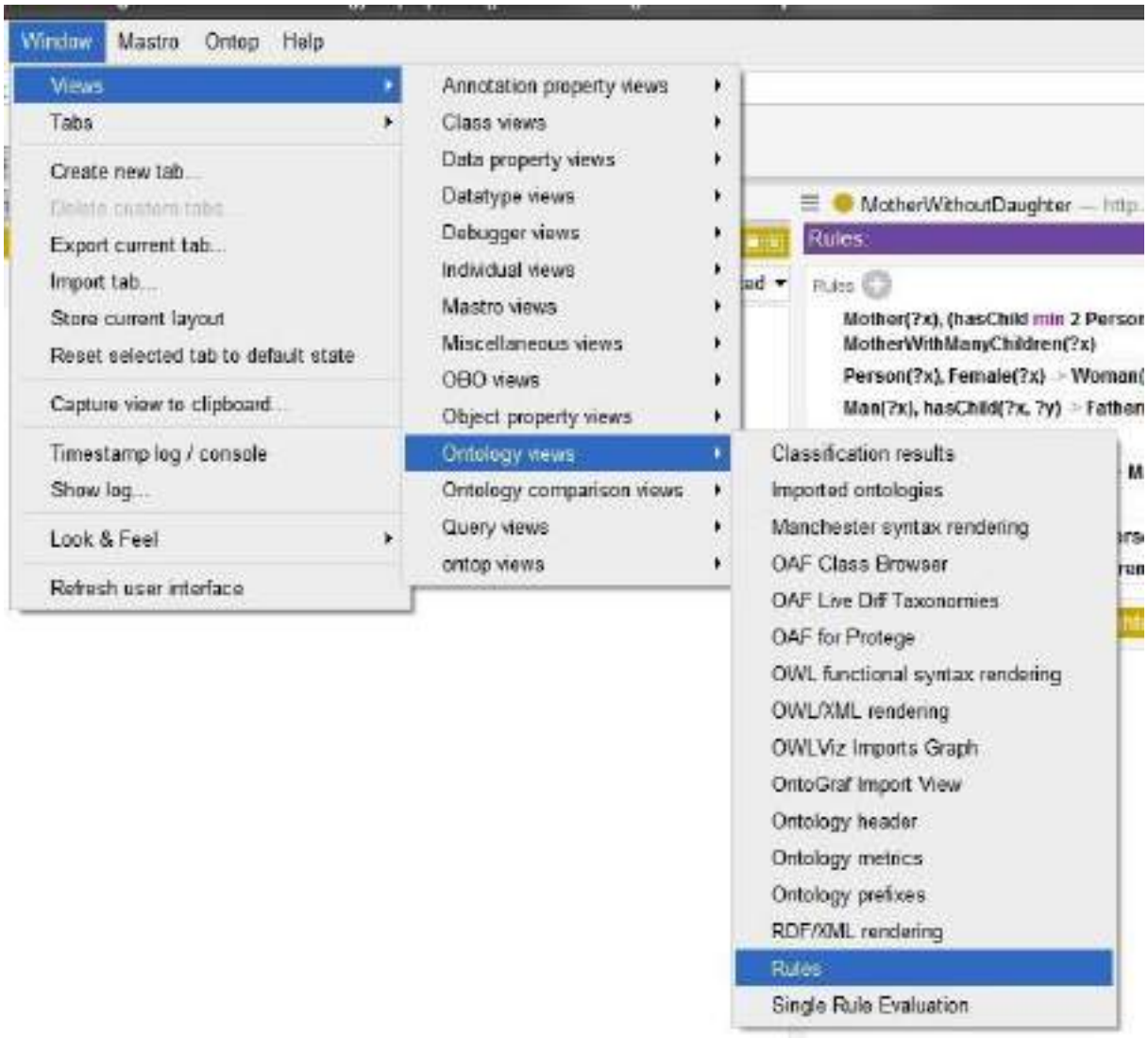


Первоначально в системе только одно общее объектное свойство owl:Think. Оно включает в себя абсолютно все существующие (и несуществующие) в мире объектные свойства.

Как запустить плагин OntoGraf



Как запустить редактор SWRL-правил



Лабораторная работа № 4.

Разработка клиент-серверных приложений на основе Web-сервисов

Часть 1. Создание наборов RDF-данных с использованием Protege

Цели работы:

1) изучение основных элементов RDF(S)- онтологий;

2) освоение навыков работы в системе Protégé;

3) получение навыков онтологического моделирования предметных областей (на примере книготорговой компании) в системе Protégé;

4) освоение метода преобразования реляционных баз данных в онтологическое представление;

5) создание набора RDF- данных с использованием системы Protégé.

Порядок выполнения работы:

1) Скачать и установить систему Protégé для разработки онтологий:
https://protege.stanford.edu/

2) Изучить основы теории онтологий. В определенной мере можно использовать учебное пособие [2].

3) Изучить предметную область, модель «сущность-связь» и схему базы данных о книготорговой компании из методички [1].

Рис. 1. Модель «сущность-связь» книготорговой компании

2) Разработать простую онтологию книготорговой компании на уровне классов (*T-Box*) в системе Protégé, используя только следующие элементы: классы, объектные свойства и свойства по данным. Аксиомы при этом не используются. Классам дать следующие имена: *Author* (автор) *Title* (книга), *Publisher* (издательство).

3) Расширить онтологию экземплярами, создав тем самым *A-Box* онтологии. Данные берутся из таблиц базы данных, приведенных в Приложении методички [1]. Для целей тестирования можно создать только 3-5 экземпляров каждого класса.

4) Записать полученную онтологию в файл в формате RDF. В результате получается RDF-база данных. В дальнейшем эта RDF-база данных будет использоваться для вычисления SPARQL-запросов к ней.

Литература

1. Дубинин В.Н. Работа с базами данных в архитектуре клиент-сервер: Метод. указания к выполнению лабораторных работ. – Пенза: Изд-во Пенз. гос. ун-та. – 2000. – 91 с.

http://alice.pnzgu.ru:8080/~dvn/pubs/uch_posobiya/Dubinin_Rabota_s_bazami_dannyh_v_architekture_client-server_2000.pdf

2. Балашова И.Ю., Прошкина Е.Н. Онтологический инжиниринг в проектировании интеллектуальных информационных систем. Учебное пособие, Пенза: ПГУ, 2019. – 156 с.

https://yadi.sk/i/9aaabTSaYiyfeA

Часть 2. Разработка SPARQL-запросов к наборам RDF- данных

Цели работы:

1) изучение языков запросов SPARQL и SPARQL Update к RDF-данным;

2) освоение метода преобразования SQL-запросов в SPARQL-запросы;

3) получение навыков работы с сервером Apache Jena Fuseki для выполнения SPARQL-запросов.

Порядок выполнения работы:

1) Изучить языки запросов SPARQL и SPARQL Update к RDF-данным. Для изучения языков рекомендуется книга [2];

2) Скачать и установить сервер Apache Jena Fuseki:

https://jena.apache.org/documentation/fuseki2/

Данный сервер работает как Web-сервис, выполняющий SPARQL-запросы. Для формирования SPARQL-запросов и просмотра результатов их вычислений используется Web-браузер (например, Chrome). При работе в локальном варианте

для загрузки человеко-машинного интерфейса в адресной строке браузера надо ввести:

http://localhost:3030

3) В соответствии с номером варианта бригады в лабораторной работе № 1 методички [1] в таблице на страницах 38-39 найти номера SQL-запросов, а на страницах 16-32 найти сами SQL-запросы. Эти SQL-запросы надо преобразовать в SPARQL-запросы.

4) Выполнить SPARQL-запросы на сервере Apache Jena Fuseki. Проанализировать полученные результаты.

5) Добавить RDF-базу данных новые данные (например, новых авторов, новые книги издательства), удалить и модифицировать некоторые данные с использованием языка SPARQL Update. Проанализировать полученные результаты.

Литература

1. Дубинин В.Н. Работа с базами данных в архитектуре клиент-сервер: Метод. указания к выполнению лабораторных работ. – Пенза: Изд-во Пенз. гос. ун-та. – 2000. – 91 с.

http://alice.pnzgu.ru:8080/~dvn/pubs/uch_posobiya/Dubinin_Rabota_s_bazami_dannyh_v_architekture_client-server_2000.pdf

2. DuCharme B. Learning SPARQL. - O'Reilly, 2011. – 256 p. (на английском)

https://yadi.sk/d/1VjzMHzqMa-wXw

Часть 3. Разработка клиентского приложения для выполнения SPARQL-запросов

Версия 1.0

Цели работы:

1) изучение методов взаимодействия приложений со SPARQL-сервером;

2) изучение подходов к разработке клиентских приложений для SPARQL-сервера;

3) получение навыков разработки простого клиентского приложения для выполнения SPARQL-запросов к RDF-данным на сервере Apache Jena Fuseki.

Порядок выполнения работы:

1) Изучить методы взаимодействия приложений со SPARQL-сервером, а также подходы к разработке клиентских приложений для SPARQL-сервера. В качестве основы взять источник [1].

2) Написать простое клиентское приложение для удаленного выполнения SPARQL-запроса на сервере Apache Jena Fuseki. Функции клиентского приложения: а) посылка SPARQL-запроса к RDF-данным, разработанного в предыдущей лабораторной работе, на сервер; б) принятие результатов вычисления запроса и его

отображение на экране. В качестве языка программирования могут использоваться языки: Java, JavaScript, C/C++/C#, Pyton и другие.

Наиболее простым способом взаимодействия является использование протокола HTTP. При этом в клиентском приложении формируется запрос типа GET или POST, который отсылается на сервер с использованием HTTP.

При необходимости функционал клиентского приложения может быть расширен, например, в него может быть добавлена функция выборки и загрузки данных из файла на SPARQL-сервер.

Литература

1. Глава 7. Building Applications with SPARQL: A Brief Tour в книге: DuCharme B. Learning SPARQL. - O'Reilly, 2011. – 256 p. (на английском)

https://yadi.sk/d/1VjzMHzqMa-wXw

Часть 4. Выполнение запросов на удаленных SPARQL-сервисах

Цели работы:

1) изучение подходов к выполнению запросов на удаленных SPARQL-сервисах;

2) получение навыков разработки запросов к удаленным SPARQL-сервисам.

Порядок выполнения работы:

1) Изучить подходы к выполнению запросов на удаленных SPARQL-сервисах. В качестве основы взять источник [1]. Обратить внимание на ключевое слово SERVICE, какова его семантика, а также на федеративные запросы.

2) Распределить наборы RDF-данных по SPARQL-серверам. Как крайний случай, каждый набор данных можно поместить на отдельный SPARQL-сервер. Эти сервера должны иметь различные IP-адреса, поэтому данную лабораторную работу необходимо выполнять в аудитории, где имеется локальная сеть.

3) Составить несколько SPARQL-запросов, в которых используется обращение к удаленным SPARQL-сервисам. Для этого в подзапросе необходимо использовать ключевое слово SERVICE. Среди этих запросов должен быть хотя бы один федеративный запрос (то есть, запрос, содержащий не менее двух подзапросов с ключевым словом SERVICE).

Литература

1. Подраздел «Querying a Remote SPARQL Service» (стр. 95) и «Federated Queries: Searching Multiple Datasets with One Query» (стр. 98) в книге: DuCharme B. Learning SPARQL. - O'Reilly, 2011. – 256 p. (на английском)

https://yadi.sk/d/1VjzMHzqMa-wXw

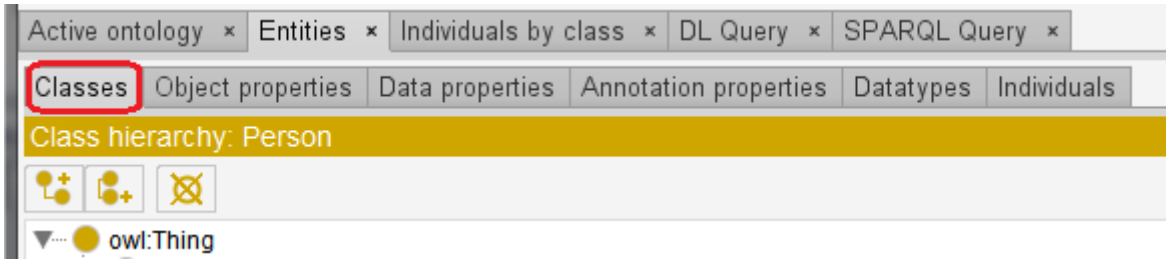
ПРИЛОЖЕНИЕ 1.

Основные сведения по работе в системе Protege

Создание классов (Classes)

Запускаем Protégé

Входим во вкладку Entities->Classes



Первоначально в системе только один общий класс owl:Think. Он включает в себя абсолютно все существующие (и несуществующие) в мире классы.

Для создания нового класса надо навести курсор на класс owl: Think и нажать кнопку  . Появится окно для ввода имени класса.

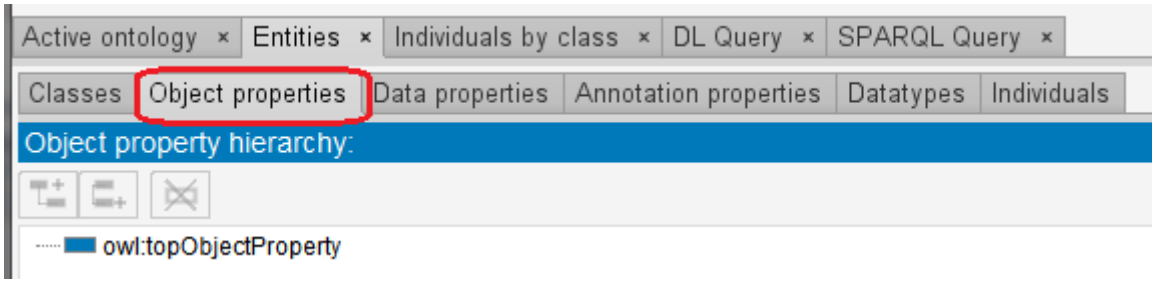
Рекомендуется использовать только латинские буквы, а первую букву лучше делать заглавной (это общепринятые правила хорошего тона).

Для удаления класса надо навести курсор на удаляемый класс и нажать кнопку 

Создание объектных свойств (Object properties)

Объектные свойства определяют *отношения* между классами.

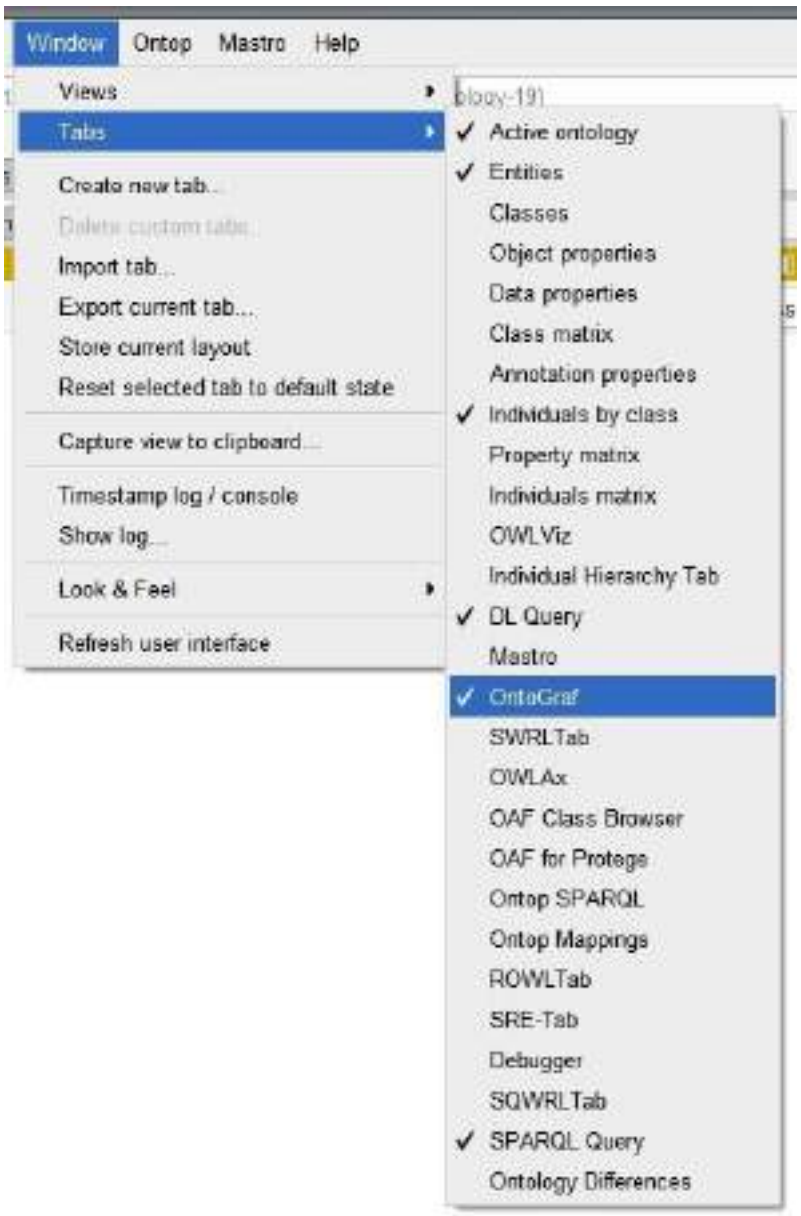
Входим во вкладку Entities-> Object properties



Первоначально в системе только одно общее объектное свойство owl:Think. Оно включает в себя абсолютно все существующие (и несуществующие) в мире объектные свойства.

Как запустить плагин OntoGraf

(для визуализации онтологий)


Window
Ontop
Mastro
Help
Views
Tabs
Create new tab...
Delete custom tabs...
Import tab...
Export current tab...
Store current layout
Reset selected tab to default state
Capture view to clipboard...
Timestamp log / console
Show log...
Look & Feel
Refresh user interface
Active ontology
Entities
Classes
Object properties
Data properties
Class matrix
Annotation properties
Individuals by class
Property matrix
Individuals matrix
OWLViz
Individual Hierarchy Tab
DL Query
Mastro
OntoGraf
SWRLTab
OWLAx
OAF Class Browser
OAF for Protege
Ontop SPARQL
Ontop Mappings
ROWLTab
SRE-Tab
Debugger
SQWRLTab
SPARQL Query
Ontology Differences

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ УПРАВЛЕНИЕ ПРОЕКТАМИ В ПРОФЕССИОНАЛЬНОЙ СФЕРЕ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

Содержание

Лабораторная работа № 1
Нормативная законодательная база, регламентирующая управление проектами

Цель работы – закрепить полученный на лекционных занятиях необходимый минимум теоретических знаний в нормативной законодательной базе, регламентирующей управление проектами и отработать практические навыки в поиске правовой информации.

Общие сведения

Обзор стандартов в области управления проектами

На сегодняшний день различными организациями и инициативными группами разработано достаточно большое количество стандартов, имеющих отношение к проектному менеджменту (рисунок 1).

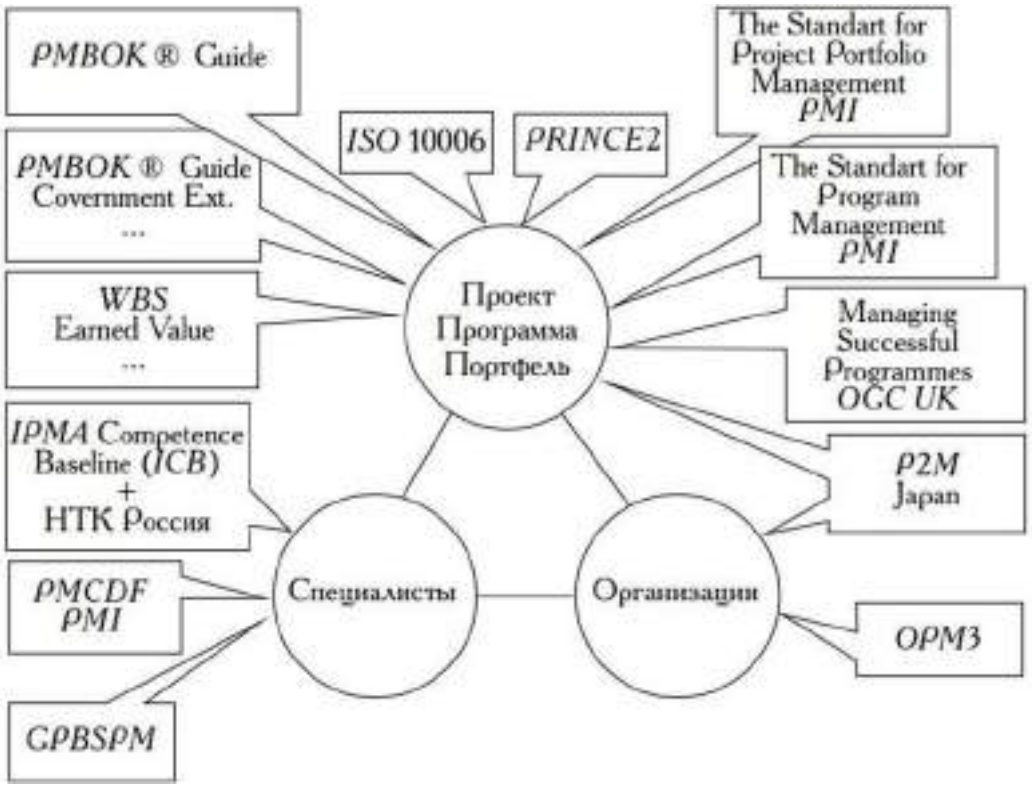


Рисунок 1 - Наиболее известные стандарты в области проектного менеджмента

На рисунке 1 представлены наиболее популярные стандарты в области проектного менеджмента, включая следующие:

– ISO 10006. Системы менеджмента качества. Руководящие указания по менеджменту качества проектов;

– PMBOK Guide. A Guide to the Project Management Body of Knowledge. Руководство к своду знаний по управлению проектами, PMI;

– PMBOK Guide Government Extension. Руководство к своду знаний по управлению проектами для правительственных организаций, PMI;

– WBS. Руководство по разработке иерархической структуры работ проекта, PMI;

– Earned Value. Руководство по применению методики освоенного объема, PMI;

– PRINCE2. Стандарт управления проектами, OGC (Office of Government Commerce), Великобритания;

– The Standard for Portfolio Management, PMI. Стандарт управления портфелем проектов, PMI;

– The Standard for Program Management, PMI. Стандарт управления программой, PMI;

– Managing Successful Programmes, OGC UK. Стандарт управления программой, OGC (Office of Government Commerce), Великобритания;

– P2M Japan. Стандарт управления проектами и программами в организации, Япония;

– OPM3. Модель зрелости организации в области проектного менеджмента, PMI;

– IPMA Competence Baseline (ICB). Международные требования к компетенции менеджеров проектов, IPMA;

– НТК Россия. Основы профессиональных знаний и Национальные требования к компетентности (НТК) специалистов по управлению проектами, СОВНЕТ;

– PMCDF PMI. Структура развития компетенций в проектном менеджменте (Project Management Competence Development Framework), PMI;

– GPBSPM. Общий стандарт оценки проектного персонала на основе опыта (Global Performance Based Standards for Project Management Personnel), GPBSPM Initiative.

Основной стандарт, разработанный ***Международной ассоциации управления проектами*** (*IPMA - international Project Management Association) IPMA*, — ICB, определяет требования к квалификации специалистов в области управления проектами (УП) и является основой для международной сертификации. В соответствии с правилами и требованиями IPMA в России разработаны национальные требования к компетенции менеджера проекта и программа сертификации специалистов по управлению проектами. Специалисты, прошедшие сертификацию по этой системе, получают сертификаты международного образца, которые признаются во всем мире.

Другая авторитетная организация в области проектного менеджмента — ***Институт управления проектами, США*** (*PMI*) с индивидуальной системой членства: насчитывается более 200 тыс. человек в 125 странах мира. PMI имеет наиболее активную и широкую стратегию в области разработки стандартов.

По **основным областям применения** стандарты могут быть разделены на следующие группы:

1) ***применимые к отдельным объектам управления*** (проект, программа, портфель проектов) и регламентирующие соответствующие процессы управления;

2) ***применимые к субъектам управления*** (менеджеры проектов, участники команд УП) и определяющие требования к знаниям и квалификации соответствующих специалистов и процессу оценки квалификации;

3) ***применимые к системе УП и организации в целом и позволяющие оценить уровень зрелости организационной системы менеджмента***.

В лабораторной работе рассмотрим группу стандартов, применимых к отдельным объектам управления (проект, программа, портфель проектов).

Наиболее проработанными по структуре и содержанию и распространенными являются стандарты, регламентирующие процессы управления отдельными проектами. В данной группе стандартов можно выделить:

– *ISO* 10006:2003. Системы менеджмента качества. Руководящие указания по менеджменту качества проектов;

– *PMI. A Guide to the Project Management Body of Knowledge. (PMBOK Guide)*. Руководство к своду знаний по управлению проектами. Третье издание.

***ISO* 10006:2003. Системы менеджмента качества. Руководящие указания по менеджменту качества проектов.** Данный международный стандарт сам по себе не является руководством по УП. В нем приведены руководящие указания по качеству процессов УП. В стандарте приводятся основные принципы и практические методики, которые влияют на качество разработки и реализации проектов. В нем процессы по проекту сгруппированы в две категории: процессы УП и процессы, связанные с продуктом проекта (т.е. такие, как проектирование, производство, проверка). Руководящие указания по качеству процессов, относящихся к продукту проекта, рассматриваются в стандарте *ISO* 9004-1.

Стандарт применим к проектам различной степени сложности, небольшим или большим, краткосрочным или долгосрочным, выполняемым в различных окружающих условиях безотносительно к виду продукта или процесса. Представленные рамочные требования требуют последующей адаптации данного руководства к конкретным условиям разработки и реализации отдельного проекта.

В стандарте разделяются понятия процессов управления и фаз реализации проекта. Проект может быть разделен на различные взаимозависимые процессы и фазы в качестве средств планирования и контроля за реализацией целей и оценкой связанных с этим рисков.

Фазы делят жизненный цикл проекта на управляемые стадии, такие как разработка концепции и проектной документации, реализация, сдача в эксплуатацию.

Процессы проекта — это процессы, необходимые для управления им, а также для реализации продукта проекта. Процессы сгруппированы в соответствии с принципом родственности (например, все процессы, связанные с управлением по временным параметрам, включены в одну группу). Всего в стандарте выделено 11 групп процессов:

- стратегические (определение направления проекта);
- относящиеся к ресурсам и персоналу;
- касающиеся взаимосвязей;
- касающиеся области применения;
- касающиеся времени;
- связанные с затратами;
- связанные с передачей информации;
- касающиеся рисков;
- связанные с закупками.

Отдельно рассматриваются процессы, касающиеся измерений и анализа и постоянного совершенствования. В стандарте содержится описание каждого процесса, а также руководящие указания по менеджменту качества конкретного процесса.

В основе руководящих указаний по менеджменту качества при проектировании, содержащихся в данном международном стандарте, лежат следующие принципы менеджмента качества (см. *ISO* 9000:2015 2.3):

1) ориентация на потребителя;

2) лидерство;

3) взаимодействие работников;

4) процессный подход;

5) улучшение;

6) принятие решений, основанное на свидетельствах;

7) менеджмент взаимоотношений;

Эти общие принципы образуют основу системы менеджмента качества для организации — инициатора и организации — исполнителя проекта.

PMBOK Guide. Руководство к своду знаний по управлению проектами. Институт управления проектами, США. *PMBOK Guide* является американским национальным стандартом УП и широко используется в мире. В основу стандарта положена процессная модель описания деятельности по УП.

В качестве основных целей разработки Руководства называют унификацию терминологического пространства и использование данного документа в качестве базового справочного пособия для сертификации профессионалов по управлению проектами (*PMP*).

В Руководстве определяются:

– структура УП (часть 1). В данной части содержатся основные сведения об УП, определены основные термины и общий обзор глав Руководства. Особое внимание уделяется понятиям жизненного цикла проекта, организационным структурам и окружению проектов;

– стандарт УП (часть 2) включает описание пяти групп управленческих процессов: 1) инициация, 2) планирование, 3) организация исполнения, 4) контроль и 5) завершение. В рамках данных групп процессов описываются 44 базовых управленческих процесса и взаимосвязи между ними;

– области знаний по УП (часть 3) состоят из девяти областей знаний: управление 1) интеграцией, 2) содержанием, 3) сроками, 4) стоимостью, 5) качеством, 6) человеческими ресурсами, 7) коммуникациями, 8) рисками, 9) поставками проекта. В данной части приводится детальное описание для каждого из 44 управленческих

процессов, включая общее описание процесса, входной и выходной информации, а также перечисление рекомендуемых методов и инструментов.

Управление интеграцией проекта – наиболее важная из областей знаний, включающая в рамках групп процессов различные элементы УП. К этой области относятся следующие процессы:

- разработка устава проекта;
- разработка предварительного описания содержания проекта;
- разработка плана УП;
- руководство и управление исполнением проекта;
- мониторинг и управление работами проекта;
- общее управление изменениями;
- закрытие проекта.

Управление содержанием проекта играет скорее вспомогательную роль ввиду того, что план проекта детализирован здесь по составу работ в объеме, необходимом и достаточном для успешного выполнения проекта. В данную область входят следующие процессы:

- планирование содержания;
- определение содержания;
- создание иерархической структуры работ (ИСР);
- подтверждение содержания;
- управление содержанием.

Управление сроками проекта включает процессы управления по временным параметрам для формирования календарного плана проекта в целях соблюдения сроков:

- определение состава операций;
- определение взаимосвязей операций;
- оценку ресурсов операций;
- оценку длительности операций;
- управление календарным планом.

Управление стоимостью проекта нацелено на успешное освоение его бюджета, последовательно реализующее процессы планирования, разработки и контроля затрат. Включает следующие процессы:

- стоимостную оценку;
- разработку бюджета расходов;
- управление стоимостью.

Управление рисками проекта охватывает идентификацию рисков, разработку карт рисков и составление плана реагирования на них, а также содержит следующие процессы управления:

- планирование управления рисками;
- идентификацию рисков;
- качественный анализ рисков;
- количественный анализ рисков;
- планирование реагирования на риски;
- мониторинг и управление рисками.

Управление качеством направлено на удовлетворение требований по качеству как продукта, так и проекта. Учитывает требования Международной организации по стандартизации (*ISO*), а также авторские и общие модели. Область включает следующие процессы:

- планирование качества;
- процесс обеспечения качества;
- процесс контроля качества.

Управление человеческими ресурсами в практике УП играет одну из ключевых ролей, и от того, насколько профессионально будут реализованы перечисленные ниже процессы, зависит полнота достижения целей и в целом обеспечен успех проекта:

- планирование человеческих ресурсов;
- набор команды проекта;
- развитие команды проекта;
- управление командой проекта.

Управление коммуникациями проекта состоит в своевременном и достоверном сборе, распределении, хранении и использования информации для всех участников, входящих в команду в соответствии с их ролями в проекте. Выделяются следующие процессы:

- планирование коммуникаций;
- распространение информации;
- отчетность по исполнению;
- управление участниками проекта.

Управление поставками проекта описывает процессы приобретения и получения продуктов, услуг и результатов, а также процессы управления контрактами. В данную область знаний входят следующие процессы:

- планирование покупок и приобретений;
- планирование контрактов;
- запрос информации у продавцов;
- выбор продавцов;
- администрирование контрактов;
- закрытие контрактов.

Одним из направлений развития стандарта *PMBOK Guide* стала его адаптация к отраслевой специфике. В настоящее время выпущены расширения стандарта для правительственных и строительных проектов (*Government Extension to the PMBOK Guide, Construction Extension to the PMBOK Guide*).

Кроме того, PMI разрабатывает стандарты, связанные с отдельными методиками УП. На сегодняшний день выпущены стандарты, регламентирующие методы разработки иерархической структуры работ проекта и контроля по методу освоенного объема (*Practice Standard for Work Breakdown Structures, Practice Standard for Earned Value Management*).

Следующая область стандартизации – процессы управления такими объектами, как ***программа*** и ***портфель проектов***.

Пионерами в данной области являются стандарты, выпущенные в Великобритании Государственным департаментом коммерции. На

протяжении многих лет стандарты используются в правительственных программах, а также для сертификации менеджеров программ.

Однако стандартов международного уровня в данной области до последнего времени не существовало. На роль общепризнанных могут претендовать стандарты, выпущенные *PMI* в 2006 г.: The Standard for Program Management и The Standard for Portfolio Management. Данные стандарты также построены по процессному принципу.

The Standard for Portfolio Management. Стандарт управления портфелем проектов. Институт управления проектами, США.

Основные цели разработки стандарта – формулирование понятийного пространства управления портфелем проектов, определение типовых процессов и их результатов без привязки к отраслевым особенностям бизнеса, а также описание ключевых ролей управления портфелем, зон ответственности и полномочий. Важное значение придается стратегии организации, возможности отслеживания достижения целей через процессы интегрированного управления портфелями проектов, программами и отдельными проектами. Раскрывается взаимосвязь с функциональными областями управления: финансами, маркетингом, корпоративными коммуникациями, управлением персоналом.

Связь управления портфелем с управлением программами и проектами устанавливается через реализацию следующих функций менеджера портфеля:

- выравнивание компонентов в соответствии со стратегией;
- обеспечение сбалансированности и устойчивости компонентов как частей портфеля, основанных на ключевых индикаторах;
- оценка стоимости и взаимосвязей компонентов портфеля;
- определение доступности ресурсов и расстановка приоритетов;
- включение и исключение портфельных компонентов.

Система управления портфелем для эффективной поддержки выполнения представленных функций предусматривает вовлечение в управление следующих ключевых ролей и подразделений:

наблюдательного совета управления портфелем, клиентов, спонсоров, исполнительных директоров, управления операционной деятельностью, менеджеров программ, офиса управления программами и проектами, менеджеров проектов, функциональных менеджеров, финансовых менеджеров, участников команды проекта.

При идентификации ключевой роли стандарта — менеджера портфеля — выявлены следующие дополнительные функции, определяющие отличительные признаки портфельного управления:

– расстановка приоритетов и выравнивание компонентов управления портфелем в соответствии со стратегическими целями;

– обеспечение ключевых акционеров своевременными результатами оценки, ранней идентификации воздействий на выполнение работ;

– измерение стоимости организации с помощью инвестиционных инструментов, таких как *ROI, NPV, PP*.

Процессы управления портфелем представлены двумя группами:

1) группа процессов формирования портфеля включает процессы управления им, обеспечивающие достижение сбалансированности портфеля проектов со стратегическими целями организации. Группа включает следующие процессы: идентификация проектов, категоризация, оценка, отбор, расстановка приоритетов, балансировка портфеля, авторизация;

2) группа процессов мониторинга и контроля основана на индикаторах деятельности, с помощью которых периодически выравниваются компоненты портфеля относительно стратегических целей. Включает процессы сбора периодической отчетности, анализа состояния портфеля проектов и управления изменениями.

Основные характеристики проекта как системы управления. Понятие проекта, стадии проекта. Функции и структура проекта.

Одним из основоположников современной теории управления проектами является Генри Гантт (Henry Gantt, 1861 - 1919), американский

инженер, предложивший в 1910 г. новую технику календарного планирования с использованием горизонтальных диаграмм. Графики Гантта оказались настолько полезным аналитическим инструментом, что до сих пор повсеместно используются в управлении проектами почти без изменений.

Родоначальником классической теории управления большинство специалистов считают французского инженера Анри Файоля (Henri Fayol, 1841-1925), который, руководя крупной компанией по добыче угля в 1920-х годах, впервые исследовал и описал особый вид деятельности - управление. Стремясь найти правила рациональной управленческой деятельности, он рассматривал организацию как единый организм, характеризующийся наличием определенных видов деятельности или шести функций:

1. Техническая деятельность (производство);
2. Коммерческая деятельность (закупка, сбыт и обмен);
3. Финансовая деятельность (поиск и оптимальное использование капитала);
4. Деятельность по безопасности (защита собственности людей);
5. Деятельность по анализу, учету, статистике;
6. Управление.

Файоль выделил управление как особый вид деятельности, который включает в себя следующие функции, ставшие впоследствии основой управления проектами:

- планирование;
- организацию;
- распорядительство;
- координацию;
- контроль.

Анри Файоль разработал ***14 принципов управления предприятием***, которые не потеряли своего значения и по сей день: разделение труда; власть; дисциплина; единство распорядительства; единство руководства;

подчинение индивидуальных интересов общим интересам; вознаграждение персонала; централизация; цепи взаимодействия; порядок; равенство; стабильность персонала; инициатива; корпоративный дух. На этой основе были сформулированы и базовые принципы концепции управления проектами.

Понятие «проект» объединяет разнообразные виды деятельности, которые имеют ряд общих признаков. Так, любой проект направлен на достижение конкретных целей и определенных результатов; характеризуется координированным выполнением многочисленных, взаимосвязанных действий; имеет ограниченную протяженность во времени, с определенным началом и концом.

Международный стандарт ISO 10006:2003 трактует **проект** как «***уникальный процесс, состоящий из совокупности скоординированных и управляемых видов деятельности, имеющий начальную и конечную даты выполнения, предпринимаемый для достижения цели, соответствующей установленным требованиям, включая ограничение по времени, затратам и ресурсам***.

Отличие проекта от производственной системы заключается том, что проект является однократной, не циклической деятельностью. Проект как система деятельности существует ровно столько времени, сколько требуется для получения конечного результата. Концепция проекта, однако, не противоречит концепции организации и вполне с ней совместима. Более того, проект часто становится основной формой деятельности организации.

С точки зрения системного подхода проект можно рассматривать как процесс перехода из исходного состояния в конечное (результат) при наличии ряда ограничений (нормативно-правовых, финансовых, временных и др.) и механизмов (техника, инструменты, технологии и др.).

Любой проект представляет собой некую задачу с определенными исходными данными и требуемыми результатами (целями) которые определяют способ ее решения. Проект включает в себя ***замысел (проблему), средства его реализации (решения проблемы) и получаемые***

в процессе реализации результаты.

Таким образом, **проект - это целенаправленное, заранее проработанное и запланированное создание или модернизация физических объектов, технологических процессов, технической » организационной документации для них, материальных, финансовых, трудовых и иных ресурсов, а также управленческих решений и мероприятий по их выполнению**. При этом проект рассматривается *как единое целое, как система - совокупность определенных элементов (объектов материального и нематериального характера) и связей между ними, обеспечивающих достижение поставленных целей*.

Проект как система имеет ряд свойств:

– проект возникает, существует и развивается в определенном окружении, называемом внешней средой;

– состав проекта не остается неизменным в процессе его реализации и развития: в нем могут появляться новые элементы (объекты) и из его состава могут удаляться некоторые его элементы;

– проект, как и всякая система, может быть разделен на элементы, при этом между выделяемыми элементами должны устанавливаться и поддерживаться определенные связи.

Основываясь на понимании проекта как системы, можно выделить в нем следующие ***составные части***:

– формирование замысла, постановка целей;

– подготовка (разработка) проекта;

– реализация проекта;

– завершение проекта, т.е. достижение поставленных целей (ввод в эксплуатацию, выход на заданную мощность, внедрение в производство и т.п.).

Все эти элементы определенным образом взаимосвязаны между собой. Эти связи между элементами проекта, возникая и развиваясь во времени, формируют процесс реализации проекта. Таким образом, можно

утверждать, что процесс выполнения проекта есть процесс реализации определенных связей между всеми его элементами.

В процессе управления проектами необходимо:

– определить цели проекта и провести его обоснование;

– выявить структуру проекта (подцели, основные этапы работы, которые предстоит выполнить);

– определить необходимые объемы и источники финансирования;

– подобрать исполнителей - в частности, через процедуры торгов и конкурсов;

– подготовить и заключить контракты;

– определить сроки выполнения проекта, составить график его реализации, рассчитать необходимые ресурсы;

– рассчитать смету и бюджет проекта;

– планировать и учитывать риски;

– организовать реализацию проекта, в том числе подобрать команду проекта;

– обеспечить контроль хода выполнения проекта.

Проект всегда функционирует в определенном окружении, включающем внутренние и внешние компоненты, при этом максимально должны учитываться экономические, политические, социальные, технологические, географические, климатические, нормативные, культурные и другие факторы.

Проект как однократная совокупность действий и задач, характеризуется следующими отличительными признаками:

– четкие цели, которые должны быть достигнуты с одновременным выполнением технических, экономических и других требований;

– внутренние и внешние взаимосвязи операций, задач и ресурсов, которые требуют четкой координации в процессе выполнения проекта;

– конкретные сроки начала и конца проекта;

– ограниченные ресурсы;

– определенная степень уникальности целей проекта, условий его осуществления;

– неизбежность возникновения различных конфликтов в процессе реализации.

Проект всегда нацелен на результат, на достижение определенных целей, на определенную предметную область. Реализация проекта осуществляется полномочным руководством проекта, менеджером проекта и командой проекта, работающей под этим руководством, другими участниками проекта, выполняющими отдельные специфические виды деятельности, процессы по проекту. В работах по проекту, как правило, на условиях частичной занятости, могут участвовать представители линейных и функциональных подразделений организаций, ответственных за выполнение возложенных на них заданий, видов деятельности, функций, включая планирование, руководство, контроль, организацию, администрирование и другие общесистемные функции.

Любые проекты, большие (мегапроекты) и малые (микропроекты), делятся на стадии и имеют определенную структуру циклов. Количество стадий зависит от сложности проекта и области его применения. Все стадии, которые проходит проект, то есть промежуток времени между моментом появления, зарождения проекта и моментом его ликвидации, завершения, называются жизненным циклом проекта. При этом каждая стадия имеет специфический результат или несколько результатов, достижение которого (которых) означает конец фазы. ***Жизненный цикл определяет порядок выполнения задач проекта и суть каждой задачи, учитывая и тот факт, что сложные проекты в действительности включают множество менее масштабных проектов, разрабатываемых параллельно***. Малые проекты связаны между собой воедино общим проектом, а значит, подчиняются общей конечной цели.

Укрупненно жизненный цикл проекта можно разделить на три основные стадии: *прединвестиционную, инвестиционную и эксплуатационную*. Типичный жизненный цикл проекта обычно представляют **четырьмя стадиями**:

– концепция или определение целей (что должно быть выполнено);

– планирование или проектирование (как выполнять);

– выполнение или реализация (выполнение);

– окончание или применение (эксплуатация).

На практике жизненный цикл проекта обычно **распадается на пять логически связанных между собой стадий**:

– концептуальная стадия, которая состоит из формулирования целей,

– анализа инвестиционных возможностей, обоснования осуществимости (технико-экономическое обоснование) и планирования проекта;

– стадия разработки проекта, во время которой проводится определение структуры работ и исполнителей, построение календарных графиков работ, бюджета проекта, разработка проектно- сметной документации, переговоры и заключение контрактов подрядчиками и поставщиками;

– стадия выполнения проекта, когда проходят работы по его реализации (строительство, маркетинг, обучение персонала и др.);

– стадия завершения проекта, состоящая обычно из приемочных испытаний, опытной эксплуатации и сдачи проекта в эксплуатацию;

– эксплуатационная стадия, включающая: приемку и запуск, замену оборудования, расширение, модернизацию, инновацию.

Жизненный цикл проекта является исходным понятием для исследования проблем финансирования работ по проекту и принятия соответствующих решений.

«Руководство по управлению проектами» (PMBOK), подготовленное

PMI, определяет **концептуальную разработку** как «***процесс выбора/документирования наилучшего подхода к достижению целей проекта***». Для того чтобы создать точную модель концептуальной разработки проекта, руководитель проекта и члены его команды должны собрать следующую разнообразную информацию:

Формулировка проблемы или потребности. Управление замыслом проекта, иначе называемое инициацией проекта, начинается с формулировки цели проекта, то есть с ответа на вопрос: для чего предназначен данный проект, почему возникла необходимость в поиске решения посредством этого проекта, в чем состоит суть проблемы. Как утверждается в Руководстве PMBOK, проекты появляются в результате требований рынка, экономических потребностей, запросов покупателей, технического прогресса, юридических требований (новых законов), социальных потребностей.

Сбор информации. На этом этапе производится сбор данных об условиях осуществления проекта и его целях. Нельзя рассчитывать на эффективное начало работы над проектом в том случае, если его руководитель не имеет ясного представления о текущем состоянии дел и о конечной цели проекта.

Ограничения. Формулируя цель проекта, руководитель проекта должен заниматься выявлением ограничений, способных повлиять на возможности достижения желаемых результатов. Так, временные и бюджетные ограничения, требования клиентов по качеству должны приниматься во внимание в первую очередь, так как они могут серьезно влиять на реализацию проекта.

Анализ альтернатив. Обычно для каждой проблемы существует несколько альтернативных методов решения. При управлении проектами анализ альтернатив – это, прежде всего, достижение ясного понимания природы проблемы, а уже затем - разработка различных вариантов ее решения. Эти решения помимо предоставления всем их разработчикам ясного понимания особенностей проекта, предлагают также возможные подходы к реализации проекта. При этом руководители проектов не

должны ограничивать информационный поиск в ходе проведения анализа альтернатив. Если поиск информации был проведен в полном объеме, то конечные цели проекта должны четко обозначиться.

Цели проекта. Концептуальная разработка должна завершаться формированием конечных целей проекта с точки зрения его результатов, требуемых ресурсов и сроков работ. Если предыдущие этапы были выполнены квалифицированно, то формулировка целей проекта станет логическим следствием проведенного анализа. Цели должны быть реальными, осязаемыми, точными, измеряемыми по доступным проверке конечным продуктам, реалистичными и реальными. Поскольку проекты имеют четко очерченные временные рамки, которые определяют дату начала и завершения проекта, то при определении цели проекта это должно учитываться в первую очередь. При формулировании цели проекта необходимо также принимать во внимание требования по качеству конечному продукту проекта, а также объем финансировании проекту. В конечном счете, цели проекта должны быть наглядно представлены, четко сформулированы и документально зафиксированы.

При этом выделяется генеральная цель (миссия) проекта и цели первого уровня, а также подцели/задачи, действия и результаты.

Миссия - ***это генеральная цель проекта, четко выраженная причина его существования. Она детализирует статус проекта, обеспечивает ориентиры для определения целей следующих уровней, а также стратегий на различных организационных уровнях***.

Стратегия проекта - *центральное звено в выработке направлений действий с целью получения обозначенных миссией и системой целей результатов проекта*. ***Подготовку стратегии проекта*** можно условно разделить на **три последовательных процедуры**: *стратегический анализ; разработка и выбор стратегии; реализация стратегии.*

Под результатом проекта понимают продукцию, результаты, полезный эффект проекта. В качестве результата, в зависимости от типа/цели проекта, могут выступать: научная разработка, новый технологический процесс, программное средство, строительный объект,

реализованная учебная программа, реструктурированная организация, сертифицированная система качества и т.д.

Окончанием существования проекта может быть:

– ввод в действие объектов, начало их эксплуатации и использования результатов выполнения проекта;

– достижение проектом заданных результатов.

Обычно как факт начала работ над проектом, так и факт его ликвидации оформляются официальными документами.

Таким образом, **проект - это однократная, не циклическая, система деятельности, нацеленная на получение конкретного, заранее определенного результата, в границах отведенного для этого срока и установленного объема финансирования.** Проект представляет собой единое неразрывное целое - систему, в которой выделяются логически связанные между собой элементы (фазы, стадии), образующие в совокупности жизненный цикл проекта, который определяет порядок выполнения всех задач проекта и суть каждой из них.

Функции управления проектом включают базовые виды деятельности, которые осуществляются на всех уровнях и во всех предметных областях по проекту на протяжении всего жизненного цикла, и нацелены на процессы, процедуры и методы.

Устав проекта.

Одним из документов концептуальной стадии управления проектами является **Устав проекта**. Руководство PMBOK определяет устав проекта как документ, «формально авторизующий» проект, то есть, утверждение Устава означает официальное начало существования проекта. Устав проекта составляется инициатором проекта или спонсором, не входящим в организацию проекта и имеющим достаточные полномочия для финансирования проекта. Составление Устава проекта и авторизация проекта обычно происходит за пределами организации проекта. Это осуществляет какое-либо предприятие, государственное учреждение, организация, занимающаяся программами.

Создание Устава проекта является звеном, соединяющим проект с текущей работой организации. В некоторых организациях Устав проекта формально составляется, и проект инициализируется только после того, как выполняется оценка потребностей, составляется анализ осуществимости проекта, предварительный план или аналогичная форма анализа, которая инициируется отдельно.

Разработка Устава проекта в первую очередь связана с документальным оформлением производственной необходимости, обоснованием проекта, текущим пониманием потребностей заказчика и нового продукта, услуги или результата, призванными удовлетворить эти потребности.

Устав проекта непосредственно или со ссылкой на другие документы должен содержать следующую информацию:

– требования, удовлетворяющие потребности, пожелания и ожидания заказчика, спонсора и других участников проекта;

– производственная необходимость, самое общее описание или требования к продукту, который является предметом проекта;

– цель и или обоснование проекта;

– информацию о назначенном менеджере проекта и уровне его полномочий;

– расписание контрольных событий проекта;

– отношения между участниками проекта;

– функциональные организации и их участие;

– допущения относительно организации и окружения, а также внешние допущения;

– ограничения относительно организации и окружения, а также внешние ограничения;

– реальная бизнес-ситуация, служащая обоснованием проекта с данными о прибыли на инвестиции;

– бюджет проекта.

Содержание работы представляет собой описание поставляемых

проектом продуктов или услуг. Во внутренних проектах инициатор проекта или спонсор обеспечивает содержание работы на основе производственной необходимости и требований к продукту или услуге. Во внешних проектах содержание работы может быть получено от заказчика в качестве составляющей тендерной документации, например, запрос предложения, запрос информации, либо в качестве приложения к контракту. В содержании работы указываются:

– производственная необходимость - практическая необходимость организации может основываться на необходимости обучения, рыночном спросе, техническом прогрессе, юридических требованиях или государственном стандарте;

– определение содержания продукта - документирует требования к продукту и характеристики продукта или услуги, для создания которых был предпринят проект;

– стратегический план - все проекты должны поддерживать стратегические цели организации. При принятии решений по выбору проекта стратегический план исполняющей организации следует рассматривать как один из факторов.

При разработке Устава проекта должны учитываться все факторы внешней среды, окружающие проект и оказывающие влияние на его успешность. Сюда относятся:

– организационная или корпоративная культура и структура;

– государственные или промышленные стандарты (например предписания контролирующих органов, стандарты на продукцию, стандарты качества, стандарты изготовления);

– инфраструктура (например, существующие сооружения и капитальное оборудование);

– существующие человеческие ресурсы (навыки, знания, специализации, такие как проектирование, разработки, юридические вопросы, заключение контрактов, закупки);

– управление персоналом (например, правила приема и увольнения, оценка эффективности работы и обучение персонала;

– информационные системы управления проектами (например, автоматизированные системы, такие как программное обеспечение для управления расписанием, система управления конфигурацией, система сбора и распределения информации и веб-интерфейсы к другим автоматизированным системам, работающим в режиме online).

При разработке Устава проекта и последующей документации по проекту все активы, используемые для влияния на успех проекта, могут быть взяты из активов организационного процесса. У всех и каждой из вовлеченных в проект организаций могут быть свои формальные и неформальные кодексы поведения, процедуры, планы и регламенты, влияние которых необходимо учитывать. Активы организационного процесса представляют собой также опыт и знания, накопленные из предыдущих проектов; например, завершенные расписания, данные о рисках и освоенных объемах. Активы организационного процесса могут быть организованы по-разному, в зависимости от отрасли, организации и области приложения. Например, активы организационного процесса сгруппировать в две категории:

1. Процессы и процедуры организации для проведения работ:

– принятые в организации стандартные процессы, такие как стандарты, корпоративные правила (правила техники безопасности и охраны труда, регламент по управлению проектами), стандартные жизненные циклы продукта и проекта, а также политика и процедуры в отношении качества (аудиты процессов, направления усовершенствования, контрольные списки и стандартизованные определения процессов);

– стандартизированные руководства, рабочие инструкции, критерии оценки предложений и измерения эффективности;

– шаблоны (например, шаблоны рисков, иерархической структуры работ и сетевых диаграмм расписания проекта);

– правила и критерии для адаптации совокупности стандартов организации для удовлетворения конкретных нужд проекта;

– требования к коммуникации (например, имеющаяся коммуникационная технология, разрешенные средства коммуникации, требования к архивированию и защите информации);

– правила или требования к закрытию проекта (например, проведение окончательного аудита проекта, оценки проекта, утверждение продукта и критерии приемки);

– процедуры финансового контроля (например, отчеты об отработанном времени, проверки произведенных расходов, номера бухгалтерских счетов и стандартные положения контрактов);

– процедуры управления проблемами и дефектами, определяющие контроль за проблемами и дефектами, выявление и исправление проблем и дефектов и отслеживание выполненных действий;

– процедуры управления изменениями, в том числе этапы изменения официальных корпоративных стандартов, регламентов, планов и процедур - или любой проектной документации - и способ утверждения и ратификации изменений;

– процедуры управления рисками, в том числе категории рисков, определение и влияние вероятности, а также матрица вероятности и последствий;

– процедуры одобрения и выдачи разрешения на авторизацию работ.

2. Корпоративная база знаний для хранения и извлечения информации:

– база измерений процессов, предназначенная для сбора и предоставления данных об измерениях процессов и продуктов;

– файлы проекта (например, базовые планы по содержанию, стоимости, расписанию и качеству, базовые планы исполнения, календари проекта, сетевые диаграммы расписания проекта, реестры рисков, запланированные ответные меры и определение влияния риска);

– историческая информация и база накопленных знаний (например, проектные записи и документация, вся информация и документация по закрытию проекта, информация, как о результатах отбора, так и об эффективности предыдущего проекта, а также информация о трудоемкости управления рисками);

– база данных управления проблемами и дефектами, включающая в себя статус проблем и дефектов, информацию об управлении ими, их решении и результатах;

– база знаний управления конфигурацией, включающая версии и базовые планы всех официальных корпоративных стандартов, регламентов, процедур и всей проектной документации;

– финансовая база данных, содержащая такую информацию, как количество рабочих часов, расходах, бюджетах и любых перерасходах проектных смет.

3. Информационная система управления проектами

Информационная система управления проектами (ИСУП) представляет собой стандартизированный набор имеющихся в организации автоматизированных инструментов, интегрированных в систему. ИСУП используется командой управления проектом для подготовки Устава проекта, обеспечения обратной связи на этапе его доработки, управления вносимыми в Устав изменениями и издания утвержденного документа.

4. Экспертная оценка

Экспертная оценка часто применяется для оценки первичных документов, необходимых для разработки Устава проекта. Такая оценка и экспертиза применяются ко всем техническим и организационным деталям в ходе этого процесса. Экспертиза осуществляется любым лицом или группой лиц, имеющими специальные знания или подготовку; источники в таких случаях могут быть разными:

– другие отделы данной организации;

– консультанты;

– участники проекта, в том числе заказчики или спонсоры;
– профессионально-технические ассоциации;
– отраслевые группы.

Утвержденный Устав официально санкционирует начало работ по проекту.

Задания и контрольные вопросы

Задания

1. Осуществить поиск нормативных законодательных актов, регулирующих сферу управления проектами (см. список использованных источников), с помощью информационно-поисковых систем (например, СПС «КонсультантПлюс») и составить список найденных документов.
2. Проанализировать найденные нормативные законодательные акты, регулирующие сферу управления проектами: указать сферу применения; наименование глав документа; общее количество терминов и определений и какие из них касаются проектной деятельности; имеются ли унифицированные образцы документов.
3. Найти следующие термины и определения: проект; управление проектом; производственная система; процесс; ресурс; документ; система; проблема (замысел); реализация; цели реализации; средства реализации; стандарт; регламент; устав; техническое задание.
4. Определить 14 принципов управления предприятиями, установленные Анри Файолем.
5. Определить состав и структуру Устава (Регламента) проекта.
6. Показать результат выполнения работы преподавателю.
7. Оформить отчет о проделанной работе (Приложение А).

Контрольные вопросы

1. Перечислите наиболее популярные стандарты в области проектного менеджмента.
2. На какие группы разделяются стандарты по проектному менеджменту?
3. Какие стандарты применимы к отдельным объектам

управления (проект, программа, портфель проектов) и регламентируют соответствующие процессы управления.

4. Для чего предназначен стандарт *ISO* 10006:2003. Системы менеджмента качества. Руководящие указания по менеджменту качества проектов?

5. Какие группы процессов выделены в стандарте *ISO* 10006:2003?

6. Перечислите принципы менеджмента качества, указанные в стандарте *ISO* 10006:2003.

7. На каких принципах основано Руководство PMBOK Guide?

8. Охарактеризуйте структуру Руководства PMBOK Guide.

9. Какие группы управленческих процессов определены в Руководстве PMBOK Guide?

10. Какие области знаний по УП определены в Руководстве PMBOK Guide? Дайте краткую характеристику

11. Дайте краткую характеристику для каждой области знаний по УП.

12. Кто является основоположником современной теории управления проектами?

13. Какие принципы управления предприятиями установлены Анри Файолем?

14. Как трактует понятие «проект» стандарт ISO 10006:2003?

15. Что включает в себя проект?

16. Какие составные части можно выделить в проекте?

17. Какими отличительными признаками характеризуется проект?

18. Что такое жизненный цикл?

19. На какие стадии можно разделить жизненный цикл?

20. Что понимается под миссией, стратегией и результатами проекта?

21. Что является окончанием существования проекта?

22. Что такое Устав проекта и кто его разрабатывает?

23. Какую информацию должен содержать Устав проекта?

Список используемых источников и литературы

I. Источники

1. Федеральный закон «Об информации, информационных технологиях и о защите информации» от 27.07.2006 № 149-ФЗ // «Российская газета», № 165, 29.07.2006.

2. Федеральный закон «О техническом регулировании» от 27.12.2002 № 184-ФЗ.

3. Федеральный закон «О контрактной системе в сфере закупок товаров, работ, услуг для обеспечения государственных и муниципальных нужд» от 05.04.2013 № 44-ФЗ.

4. Федеральный закон «О размещении заказов на поставки товаров, выполнение работ, оказание услуг для государственных и муниципальных нужд» от 21.07.2005 № 94-ФЗ.

5. ГОСТ Р ИСО 9001:2015. Национальный стандарт Российской Федерации. Системы менеджмента качества. Требования.

6. ГОСТ Р ИСО 15489-1-2007. Национальный стандарт Российской Федерации. Система стандартов по информации, библиотечному и издательскому делу. Управление доку-ментами. Общие требования.

7. Руководство к Своду знаний по управлению проектами (Руководство PMBOK), 2013.

II. Литература

8. Беркович Д.М. Формирование науки управления производством. – М.: Экономика, 1973, с.54-55; Кузнецова Н.В. История менеджмента. – Владивосток: Изд-во Владивостокского университета, 2004, с.47-50.

9. Информационные технологии управления проектами: Учебное пособие / Н.М. Светлов, Г.Н. Светлова. - 2-е изд., перераб. и доп. - М.: ИНФРА-М, 2011. - 232 с.

10. Ларин М.В. Документационное обеспечение управления проектами: Учебно-методическое пособие/Ларин М.В., Ларин М.М.; ВНИИДАД, М., 2011. – 192 с.

11. Мир управления проектами / Под редакцией Х. Решке, Х. Шелле/ Пер. с англ - М. : «Аланс», 1993, с. 26.

12. Сооляттэ, А. Ю. Управление проектами в компании: методология, технологии, практика: учебник / А. Ю. Сооляттэ. - М.: Московский финансово-промышленный университет «Синергия», 2012. - (Академия бизнеса).

Приложение А

(информационное)

Требования к оформлению отчета

Отчет по лабораторной работе оформляется в соответствии с ГОСТ 2.105-95 ЕСКД «Общие требования к текстовым документам.

Отчет должен содержать:

1. Титульный лист с указанием названия номера лабораторной работы, номера группы ФИО студентов, должности и ФИО преподавателя.

2. Формулировку цели работы.

3. Порядок выполнения работы.

4. Выводы о проделанной работе.

5. Приложения с практическими результатами работы.

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ
АВТОМАТИЗАЦИЯ ПРОЕКТИРОВАНИЯ АППАРАТНЫХ И ПРОГРАММНЫХ СРЕДСТВ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

Лабораторная работа №1

РАЗРАБОТКА ПОВЕДЕНЧЕСКОЙ МОДЕЛИ ЦИФРОВОГО УСТРОЙСТВА

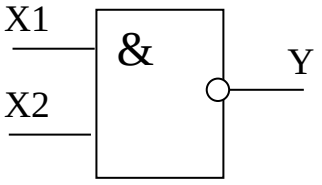
Цель работы: ознакомится с системой моделирования *Active-HDL*, разработать поведенческую модель простейшего цифрового устройства на базе логических элементов и проверить правильность его функционирования посредством моделирования.

Общие сведения

Системное меню *Active-HDL* имеет следующие пункты:

- *File* – для работы с файлами и проектами;
- *Edit* – содержит стандартные пункты для работы с текстом (*Cut, Copy, Paste, Delete*), а также пункты добавления и удаления комментария (*Comment Block, Uncomment Block*);
- *Search* – для поиска строки текста как в текущем файле, так и в файлах с заданным расширением; для перехода между ошибками и маркерами в тексте;
- *View* – содержит пункты для настройки рабочих окон;
- *Design* – предназначен для работы с текущим проектом: компиляция VHDL файла, добавление файла в текущий проект, создание библиотеки, архивации проекта;
- *Simulation* – позволяет запускать и останавливать моделирование;
- *Tools* – инструментальные средства для работы с системой;
- *Help* – помощь.

Работу с системой рассмотрим на примере моделирования логического элемента И-НЕ на 2 входа. Условное обозначение и таблица истинности элемента представлены на рисунке 1.



X1	X2	Y
0	0	1
0	1	1
1	0	1
1	1	0

Рисунок 1 – Условное обозначение и таблица истинности элемента И-НЕ

I. Для моделирования необходимо создать новый проект (*File → New Design*):

1. Ввести имя проекта.
2. Выбрать *Create source files now*.
3. Создать командой *New* новый файл VHDL.
4. Командой *Ports* задаём порты ввода-вывода.
5. Открыть из *Design Browser* ваш файл.
6. Описать функции, реализуемые устройством.

```
--Файл ANDNOT.VHD
--Поведенческое описание логического элемента И-НЕ

entity andnot is
    port (
        x1: in bit;
        x2: in bit;
        y: out bit
             );
end andnot;

architecture andnot of andnot is
begin
   y <= not(x1 and x2);
end andnot;
```

Рисунок 2 – Файл ANDNOT.VHD

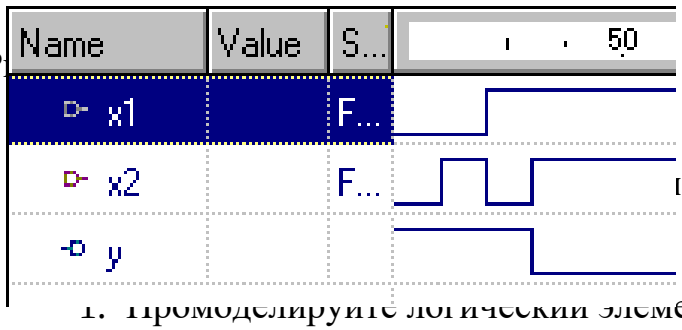
II. Откомпилировать файл ANDNOT.VHD (*Desing → Compile*).

III. В *Design Browser* выбрать закладку *Struc*... В окне выбрать название своего проекта.

- Создать файл сигналов (*File → New → Waveform*).
- В меню *Waveform* выбрать пункт *Add Signals*…
- Добавить сигналы в окно.
- Задать входные сигналы нажав правую кнопку мыши и выбрав *Stimulators*...
- Задать сигналы на входах: в окне *Stimulators Type* выбрать метод задания сигналов.

IV. Запустить на моделирование (*Simulation → Run Until*...).

Р



ия работы

1. Промоделируйте логический элемент И-НЕ.
2. Получите индивидуальное задание и опишите работу цифрового устройства на VHDL. Для этого нужно:
   - Минимизировать функцию, применив любой известный метод минимизации, например, с помощью карт Карно, диаграмм двоичного выбора и т. д.
   - Составить VHDL-модель, употребив логические операторы и операторы назначения сигналов.
   - Составить тестирующую программу, порядок подачи тестирующие воздействий должен соответствовать порядку наборов из левой части таблицы истинности.
3. Составьте и опишите на VHDL входные воздействия, которые полностью проверяли бы функционирование схемы.
4. Выполните моделирование, проверьте результаты.
5. Оформите отчет, включающий: схему, ее поведенческое описание на языке VHDL, таблицу истинности, входные воздействия и результаты моделирования, выводы по работе.

Варианты заданий

Таблица истинности (вариант 1)

x1 x2 x3 x4	y1 y2 y3
0 0 0 0	0 0 0
0 0 0 1	0 0 1
0 0 1 0	0 0 0
0 0 1 1	0 0 0
0 1 0 0	0 0 1
0 1 0 1	0 1 0
0 1 1 0	0 1 1
0 1 1 1	1 0 0
1 0 0 0	0 0 0
1 0 0 1	0 1 1
1 0 1 0	1 0 0
1 0 1 1	1 0 1
1 1 0 0	0 1 1
1 1 0 1	1 0 0
1 1 1 0	1 0 1
1 1 1 1	1 0 0

Таблица истинности (вариант 2)

x1 x2 x3 x4	y1 y2 y3
0 0 0 0	0 0 0
0 0 0 1	0 0 1
0 0 1 0	0 1 0
0 0 1 1	1 0 1
0 1 0 0	0 0 0
0 1 0 1	0 1 0
0 1 1 0	0 1 0
0 1 1 1	1 0 0
1 0 0 0	1 1 0
1 0 0 1	0 1 1
1 0 1 0	1 0 0
1 0 1 1	1 0 1
1 1 0 0	0 1 1
1 1 0 1	1 0 1
1 1 1 0	1 0 1
1 1 1 1	1 1 1

Таблица истинности (вариант 3)

x1 x2 x3 x4	y1 y2 y3
0 0 0 0	1 0 0
0 0 0 1	0 0 1
0 0 1 0	0 1 0
0 0 1 1	1 1 1
0 1 0 0	0 0 0
0 1 0 1	0 1 0
0 1 1 0	0 1 1
0 1 1 1	1 0 0
1 0 0 0	0 0 0
1 0 0 1	0 1 1
1 0 1 0	0 0 0
1 0 1 1	1 0 1
1 1 0 0	0 1 0
1 1 0 1	1 0 0
1 1 1 0	0 0 1
1 1 1 1	1 1 0

Таблица истинности (вариант 4)

x1 x2 x3 x4	y1 y2 y3
0 0 0 0	1 0 0
0 0 0 1	1 0 1
0 0 1 0	1 0 0
0 0 1 1	0 0 0
0 1 0 0	1 0 1
0 1 0 1	0 1 1
0 1 1 0	1 0 1
0 1 1 1	0 0 0
1 0 0 0	0 0 1
1 0 0 1	0 0 1
1 0 1 0	1 0 1
1 0 1 1	1 0 1
1 1 0 0	0 0 1
1 1 0 1	1 0 0
1 1 1 0	1 0 1
1 1 1 1	0 0 0

Таблица истинности (вариант 5)

x1 x2 x3 x4	y1 y2 y3
0 0 0 0	0 0 0
0 0 0 1	0 0 0
0 0 1 0	0 1 0
0 0 1 1	1 1 1
0 1 0 0	0 0 0
0 1 0 1	0 1 0
0 1 1 0	0 1 1
0 1 1 1	1 0 0
1 0 0 0	1 1 0
1 0 0 1	1 1 1
1 0 1 0	1 0 0
1 0 1 1	1 0 1
1 1 0 0	1 1 1
1 1 0 1	0 0 0
1 1 1 0	1 0 1
1 1 1 1	1 1 0

Таблица истинности (вариант 6)

x1 x2 x3 x4	y1 y2 y3
0 0 0 0	0 0 0
0 0 0 1	0 0 1
0 0 1 0	0 0 0
0 0 1 1	0 0 0
0 1 0 0	0 0 0
0 1 0 1	0 1 0
0 1 1 0	0 1 1
0 1 1 1	1 0 0
1 0 0 0	0 1 0
1 0 0 1	0 1 1
1 0 1 0	0 0 0
1 0 1 1	0 0 1
1 1 0 0	0 1 1
1 1 0 1	1 0 0
1 1 1 0	1 0 1
1 1 1 1	0 1 0

Таблица истинности (вариант 7)

x1 x2 x3 x4	y1 y2 y3
0 0 0 0	0 0 0
0 0 0 1	0 0 0
0 0 1 0	1 0 0
0 0 1 1	0 0 1
0 1 0 0	0 0 0
0 1 0 1	0 1 0
0 1 1 0	1 0 0
0 1 1 1	1 0 0
1 0 0 0	1 0 0
1 0 0 1	1 1 0
1 0 1 0	1 0 0
1 0 1 1	1 0 1
1 1 0 0	0 0 0
1 1 0 1	1 0 0
1 1 1 0	0 0 1
1 1 1 1	1 0 0

Таблица истинности (вариант 8)

x1 x2 x3 x4	y1 y2 y3
0 0 0 0	0 0 1
0 0 0 1	0 0 1
0 0 1 0	0 1 0
0 0 1 1	0 1 1
0 1 0 0	0 0 0
0 1 0 1	0 1 1
0 1 1 0	0 1 1
0 1 1 1	1 0 1
1 0 0 0	0 1 1
1 0 0 1	0 1 1
1 0 1 0	1 0 1
1 0 1 1	1 0 1
1 1 0 0	0 1 1
1 1 0 1	1 0 1
1 1 1 0	1 0 1
1 1 1 1	1 1 1

Таблица истинности (вариант 9)

x1 x2 x3 x4	y1 y2 y3
0 0 0 0	0 0 0
0 0 0 1	0 1 1
0 0 1 0	0 1 1
0 0 1 1	0 1 1
0 1 0 0	0 0 0
0 1 0 1	0 1 0
0 1 1 0	0 1 1
0 1 1 1	0 0 0
1 0 0 0	0 1 0
1 0 0 1	0 1 1
1 0 1 0	0 0 0
1 0 1 1	0 1 1
1 1 0 0	0 1 1
1 1 0 1	1 0 0
1 1 1 0	0 0 1
1 1 1 1	1 1 0

Таблица истинности (вариант 10)

x1 x2 x3 x4	y1 y2 y3
0 0 0 0	0 0 0
0 0 0 1	0 0 1
0 0 1 0	1 1 1
0 0 1 1	0 1 1
0 1 0 0	0 0 0
0 1 0 1	1 0 0
0 1 1 0	1 1 1
0 1 1 1	1 0 0
1 0 0 0	0 1 0
1 0 0 1	0 1 1
1 0 1 0	1 0 0
1 0 1 1	1 0 1
1 1 0 0	1 1 1
1 1 0 1	1 0 0
1 1 1 0	1 0 1
1 1 1 1	1 1 1

Таблица истинности (вариант 11)

x1 x2 x3 x4	y1 y2 y3
0 0 0 0	0 0 0
0 0 0 1	0 0 0
0 0 1 0	0 1 0
0 0 1 1	0 1 1
0 1 0 0	0 0 0
0 1 0 1	0 1 0
0 1 1 0	0 1 1
0 1 1 1	1 0 0
1 0 0 0	0 1 0
1 0 0 1	0 1 1
1 0 1 0	1 0 1
1 0 1 1	1 1 1
1 1 0 0	0 1 1
1 1 0 1	1 0 0
1 1 1 0	1 0 1
1 1 1 1	1 1 1

Таблица истинности (вариант 12)

x1 x2 x3 x4	y1 y2 y3
0 0 0 0	1 0 0
0 0 0 1	1 1 1
0 0 1 0	1 1 1
0 0 1 1	0 1 1
0 1 0 0	1 0 1
0 1 0 1	0 1 0
0 1 1 0	0 1 1
0 1 1 1	1 0 1
1 0 0 0	0 1 0
1 0 0 1	0 1 1
1 0 1 0	1 1 0
1 0 1 1	1 1 1
1 1 0 0	1 1 1
1 1 0 1	1 0 1
1 1 1 0	1 0 1
1 1 1 1	1 1 1

Таблица истинности (вариант 13)

x1 x2 x3 x4	y1 y2 y3
0 0 0 0	0 1 0
0 0 0 1	0 1 1
0 0 1 0	0 1 0
0 0 1 1	0 1 1
0 1 0 0	0 1 1
0 1 0 1	0 1 0
0 1 1 0	0 1 1
0 1 1 1	0 1 0
1 0 0 0	0 1 0
1 0 0 1	0 1 1
1 0 1 0	0 1 0
1 0 1 1	0 1 1
1 1 0 0	0 1 1
1 1 0 1	0 1 0
1 1 1 0	0 1 1
1 1 1 1	0 1 0

Таблица истинности (вариант 14)

x1 x2 x3 x4	y1 y2 y3
0 0 0 0	0 0 0
0 0 0 1	0 0 0
0 0 1 0	0 1 0
0 0 1 1	0 1 1
0 1 0 0	1 0 1
0 1 0 1	0 0 0
0 1 1 0	0 1 1
0 1 1 1	1 0 0
1 0 0 0	0 1 0
1 0 0 1	0 0 0
1 0 1 0	1 0 0
1 0 1 1	1 0 1
1 1 0 0	0 1 0
1 1 0 1	1 0 0
1 1 1 0	0 0 1
1 1 1 1	1 1 0

Таблица истинности (вариант 15)

x1 x2 x3 x4	y1 y2 y3
0 0 0 0	0 0 0
0 0 0 1	0 0 1
0 0 1 0	0 0 0
0 0 1 1	0 0 1
0 1 0 0	1 0 1
0 1 0 1	0 1 0
0 1 1 0	0 1 1
0 1 1 1	1 1 0
1 0 0 0	1 0 0
1 0 0 1	1 0 1
1 0 1 0	1 0 0
1 0 1 1	1 0 1
1 1 0 0	1 1 1
1 1 0 1	1 1 0
1 1 1 0	1 1 1
1 1 1 1	1 1 1

Таблица истинности (вариант 16)

x1 x2 x3 x4	y1 y2 y3
0 0 0 0	0 0 0
0 0 0 1	0 0 1
0 0 1 0	0 1 0
0 0 1 1	0 1 1
0 1 0 0	1 0 0
0 1 0 1	0 1 0
0 1 1 0	0 1 1
0 1 1 1	1 0 0
1 0 0 0	0 1 0
1 0 0 1	0 1 1
1 0 1 0	1 0 0
1 0 1 1	1 0 1
1 1 0 0	0 1 1
1 1 0 1	1 0 0
1 1 1 0	1 0 1
1 1 1 1	1 1 0

Таблица истинности (вариант 17)

x1 x2 x3 x4	y1 y2 y3
0 0 0 0	0 0 0
0 0 0 1	0 0 1
0 0 1 0	0 1 0
0 0 1 1	0 1 1
0 1 0 0	1 0 0
0 1 0 1	1 1 0
0 1 1 0	0 1 1
0 1 1 1	1 0 0
1 0 0 0	0 1 0
1 0 0 1	0 1 1
1 0 1 0	1 1 1
1 0 1 1	1 0 1
1 1 0 0	0 1 1
1 1 0 1	1 0 0
1 1 1 0	1 0 1
1 1 1 1	1 1 0

Таблица истинности (вариант 18)

x1 x2 x3 x4	y1 y2 y3
0 0 0 0	0 0 0
0 0 0 1	0 1 1
0 0 1 0	0 1 0
0 0 1 1	1 0 0
0 1 0 0	0 1 1
0 1 0 1	0 1 0
0 1 1 0	0 1 1
0 1 1 1	1 0 0
1 0 0 0	0 1 0
1 0 0 1	1 1 1
1 0 1 0	1 0 0
1 0 1 1	1 0 1
1 1 0 0	0 1 1
1 1 0 1	1 1 0
1 1 1 0	1 0 1
1 1 1 1	1 0 0

Лабораторная работа №2

РАЗРАБОТКА СТРУКТУРНОЙ МОДЕЛИ ЦИФРОВОГО УСТРОЙСТВА

Цель работы: разработать структурную двоичную модель цифрового устройства в соответствии с вариантом задания и провести моделирование.

Общие сведения

Структурная модель описывает элемент или устройство на более низком уровне, в данном случае на уровне вентилей. Рассмотрим структурную модель на примере синхронного RS-триггера, состоящего из 4-х элементов И-НЕ.

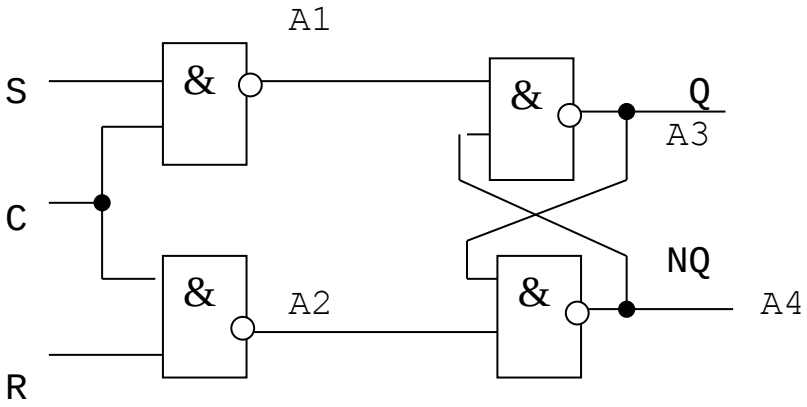
```
--Файл RSTR_S.VHD
--Структурная модель синхронного RS-триггера

entity AND2NOT is
       generic (DELAY: time := 0.1ns);
       port (I1,I2: in bit; O: out bit);
end AND2NOT;

architecture AND_2_NOT of AND2NOT is
begin
process(I1,I2)
begin
       O <= not (I1 and I2) after DELAY;
end process;
end AND_2_NOT;

entity TRIGG is
     port (S,R,C: in bit; Q,NQ: out bit);
end TRIGG;

architecture STRU of TRIGG is
component AND2NOT
  port (I1,I2: in bit; O: out bit);
end component;
signal A1,A2,A3,A4:BIT:='0';
begin
  G1:AND2NOT
   port map(S,C,A1);
  G2:AND2NOT
   port map(C,R,A2);
  G3:AND2NOT
   port map(A1,A4,A3);
  G4:AND2NOT
   port map(A3,A2,A4);
  Q<=A3;
  NQ<=A4;
end STRU;
```



S	R	C	Q	NQ
X	X	0	Q	NQ
0	0	1	Q	NQ
0	1	1	0	1
1	0	1	1	0
1	1	1	X	X

Рисунок 1 – Исходная схема триггера и таблица переходов

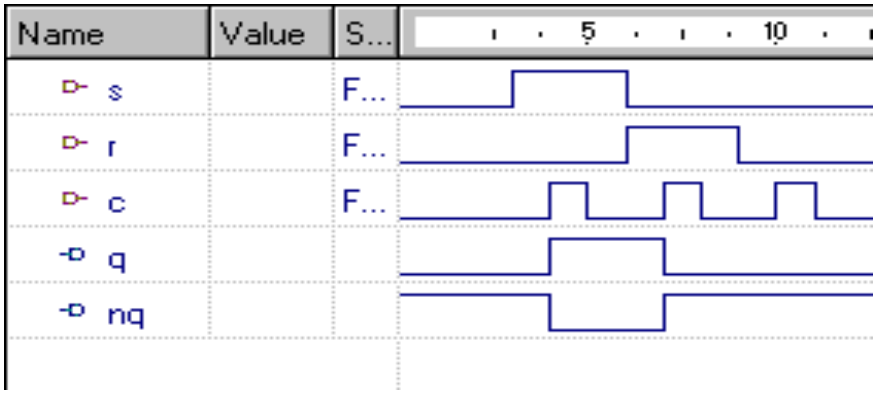


Рисунок 2 – Результат моделирования

Порядок выполнения работы

1. По таблице истинности из задания первой лабораторной работы составьте структурное описание цифрового устройства, вместе с поведенческим описанием необходимых логических элементов, реализующих заданные булевы функции.
2. Составьте и опишите на VHDL входные воздействия, которые полностью проверяли бы функционирование схемы.
3. Выполните моделирование, проверьте результаты.
4. Оформите отчет.

Содержание отчета

1. Схема устройства.
2. Описание устройства на языке VHDL.
3. Таблица истинности.
4. Входные воздействия и результаты моделирования.
5. Выводы по работе.

Лабораторная работа №3

МУЛЬТИПЛЕКСОР И ДЕМУЛЬТИПЛЕКСОР

Цель работы: реализовать VHDL-проекты для мультиплексора 4×1 (3 бита в каждом канале) и для 4-канального демультиплексора с сигналом разрешения выбора номера канала.

Общие сведения

VHDL- проект для мультиплексора:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity mux is
port (
EN: in std_logic; -- Сигнал разрешения выбора канала.
IN0: in std_logic_vector (2 downto 0);
IN1: in std_logic_vector (2 downto 0);
IN2: in std_logic_vector (2 downto 0);
IN3: in std_logic_vector (2 downto 0);
SEL: in std_logic_vector (1 downto 0); -- Выбор канала.
OUT: out std_logic_vector (2 downto 0));
end entity;

architecture mux_arch of mux is
constant NON_ACTIVE : std_logic_vector (2 downto 0) := (others => '0');
begin
process (SEL, EN, IN0, IN1, IN2, IN3)
begin
if (EN = '0') then
OUT <= NON_ACTIVE;
else
case CONV_INTEGER(SEL) is
when 0 => OUT <= IN0;
when 1 => OUT <= IN1;
when 2 => OUT <= IN2;
when 3 => OUT <= IN3;
when others => OUT <= NON_ACTIVE;
end case;
end if;
```

```
end process;
end architecture;
```

VHDL-проект для демультиплексора:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity dmux is port (
EN: in std_logic; -- Сигнал разрешения выбора канала.
SEL: in std_logic_vector (1 downto 0); -- выбор номера канала.
IN1: in std_logic_vector (7 downto 0); -- входной канал.
OUT0: out std_logic_vector (7 downto 0); -- выходной канал 0.
OUT1: out std_logic_vector (7 downto 0); -- выходной канал 1.
OUT2: out std_logic_vector (7 downto 0)); -- выходной канал 2.
OUT3: out std_logic_vector (7 downto 0)); -- выходной канал 3.
end entity;

architecture dmux_arch of dmux is
constant NON_ACTIVE : std_logic_vector (7 downto 0):= (others => '0');
begin
OUT0 <= IN1 when (SEL = 0) and (EN = '1') else NON_ACTIVE; OUT1 <= IN1 when (SEL = 1) and (EN = '1') else NON_ACTIVE; OUT2 <= IN1 when (SEL = 2) and (EN = '1') else NON_ACTIVE; OUT3 <= IN1 when (SEL = 3) and (EN = '1') else NON_ACTIVE;
end architecture;
```

Порядок выполнения работы

1. Реализовать VHDL-проекты мультиплексора и демультиплексора, провести их функциональное моделирование.
2. Подумать, как в проекте для демультиплексора можно использовать оператор выбора *case*.

Содержание отчета

1. Листинги проектов с комментариями, поясняющими выполнение каждого оператора.
2. Диаграммы временного моделирования.
3. Выводы по работе.

Лабораторная работа №4

СУММАТОР

Цель работы: реализовать VHDL-проект для одноразрядных полусумматора и полного сумматора.

Общие сведения

VHDL-проект для полусумматора:

```
library ieee;
use ieee.std_logic_1164.all;

entity half is
      port (a,b : in std_logic ; sum, carry : out std_logic);
end half;

architecture halfadder of half is begin
      sum <= a xor b;
      carry <= a and b;
end halfadder;
```

VHDL-проект для одноразрядного полного сумматора:

```
library ieee;
use ieee.std_logic_1164.all;

entity full is
      port (a,b,c : in std_logic ; sum, carry : out std_logic);
end full;

architecture fulladder of full is begin
      sum <= a xor b xor c;
      carry <= (a and b) or (b and c) or (c and a);
end fulladder;
```

Полный одноразрядный сумматор можно представить как объединение двух полусумматоров. Первый полусумматор служит для сложения двух чисел, принадлежащих одному разряду, и обеспечивает формирование

промежуточной суммы s_i и переноса p_{i+1}. Второй полусумматор складывает перенос с предыдущего разряда.

Порядок выполнения работы

1. Реализовать VHDL-проекты одноразрядных полусумматора и полного сумматора, провести их функциональное моделирование.
2. Предложите другой вариант VHDL-кода для реализации одноразрядного сумматора, основанный на использовании оператора языка, выражающего условие (например, оператора *if*).
3. Предложите и реализуйте проект 4-разрядного сумматора с последовательным переносом. Для реализации этого проекта можно использовать модульное конструирование с помощью операторов *component* и *port map*.
4. Изучите, как выглядит проект 4-разрядного сумматора на RTL-уровне (уровне регистровых передач – register transfer level, RTL).

Содержание отчета

1. Листинги проектов с комментариями, поясняющими выполнение каждого оператора.
2. Диаграммы временного моделирования.
3. Выводы по работе.

Лабораторная работа №5

СЧЕТЧИК ДЖОНСОНА

Цель работы: реализовать проект счетчика Джонсона на уровне VHDL и схемотехническом уровне.

Общие сведения

Если коэффициент пересчета счетчика равен $M=2N$, где N – количество триггеров в схеме, то такой счётчик называется счётчиком Джонсона. По мере поступления тактовых импульсов счётчик сначала заполняется единицами, а потом от них освобождается. В схеме счётчика Джонсона используется перекрестная обратная связь. Таблица работы счетчика имеет вид:

Такт	Q1	Q2	Q3	Q4
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

VHDL – проект для счетчика Джонсона:

```
library ieee;
use IEEE.std_logic_1164.all;

entity jc2_top is
      port (LEFT, RIGHT, STOP, CLK : in std_logic;
      Q: inout std_logic_vector (3 downto 0) := "0000");
end jc2_top;

architecture jc2_top_arch of jc2_top is
      signal DIR: std_logic := '0';
      signal RUN: std_logic := '0';
begin process (CLK)
begin
if (CLK' event and CLK='1') then if (RIGHT='0') then DIR <= '0';
elsif (LEFT='0') then DIR <= '1';
end if;
```

```
if (STOP='0') then RUN <= '0';
elsif (LEFT='0' or RIGHT='0') then RUN <= '1';
end if;
if (RUN= '1') then if (DIR= '1') then
Q(3 downto 1) <= Q(2 downto 0); Q(0) <= not Q(3);
else
Q(2 downto 0) <= Q(3 downto 1); Q(3) <= not Q(0);
end if; end if; end if;
end process;
end jc2_top_arch;
```

Оператор *S'event* возвращает значение *true* типа *boolean*, если на сигнале произошло событие, и *false* в противном случае. При формировании тестового воздействия установите длительность импульсов на входе CLK равной 20 нс, а период повторения – 40 нс.

Для составления принципиальной схемы разрабатываемого счетчика понадобятся четыре D-триггера, инвертор и маркеры ввода/вывода:

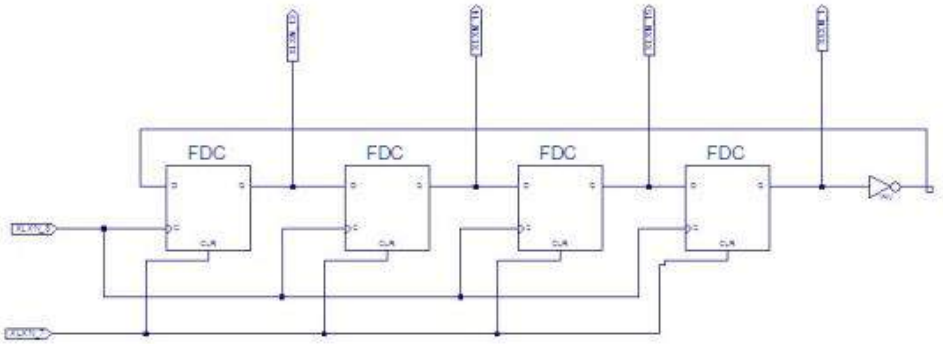


Рисунок 1 – Схема электрическая принципиальная

Порядок выполнения работы

1. VHDL-проект снабдить комментариями, поясняющими выполнение каждого оператора.
2. Проведите функциональное моделирование двух реализаций проекта.

Содержание отчета

1. Листинги проектов с комментариями, поясняющими выполнение каждого оператора.
2. Диаграммы временного моделирования.
3. Выводы по работе.

Лабораторная работа №6

БАЗОВЫЕ ЭЛЕМЕНТЫ ПАМЯТИ

Цель работы: ознакомиться с основными понятиями RTL-синтеза, получить навыки моделирования простейших элементов памяти.

Общие сведения

Проектирование ЭВМ и цифровых систем условно можно разбить на несколько этапов:

1. На этапе системного проектирования определяется структура будущей системы, состав устройств и их основные характеристики: производительность, надежность и др.
2. На этапе структурно-алгоритмического проектирования отрабатывается система команд, алгоритмы функционирования и структурные схемы устройств.
3. На этапе функционально-логического проектирования разрабатываются функциональные и принципиальные электрические схемы, тесты контроля устройств и узлов.
4. На конструкторском этапе производится привязка схем к конструктивным элементам; стойкам, панелям, типовым элементам замены. Осуществляется расстановка корпусов микросхем и трассировка соединений на печатных платах.

Эта последовательность этапов характерна и для разработки больших интегральных схем (БИС). Она отображает процесс проектирования объекта от спецификации его внешних характеристик до схемы и конструкции.

Проектирование «сверху вниз» на базе HDL предполагает переход от поведенческого описания системы к синтезабельному описанию на уровне регистров (RTL-описанию). В дальнейшем это описание специальной программой преобразуется в низкоуровневое описание – схему соединений логических элементов и проводников.

Порядок выполнения работы

1. Составить VHDL-описание бистабильного элемента, произвести его функциональное моделирование (см. рис. 1):

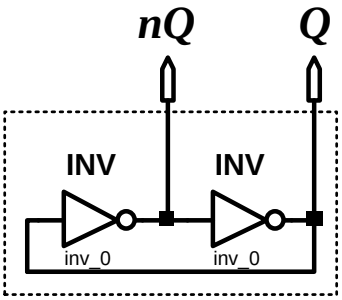


Рисунок 1 – Структурная схема бистабильного элемента

2. Составить структурное и поведенческое VHDL-описание RS-защелки (RS-latch) произвести его функциональное моделирование (см. рис. 2). После чего составить параметрическую модель с транспортными и инерциальными задержками и произвести ее функциональное моделирование:

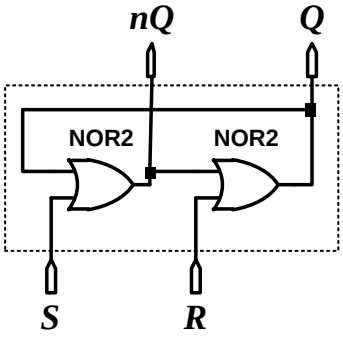


Рисунок 2 – RS-защелка

3. Проделать все действия из пункта 2 с D-защелкой (см. рис. 3):

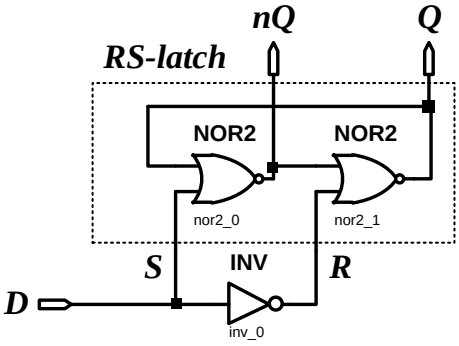


Рисунок 3 – D-защелка

4. Проделать все действия из пункта 2 с D-защелкой с разрешением (см. рис. 4):

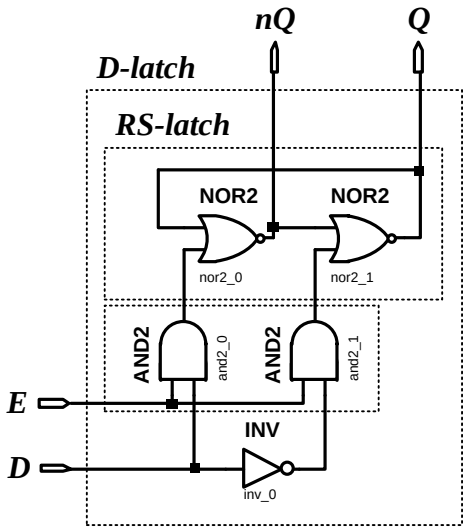


Рисунок 4 – D-защелка с разрешением

5. Выполните индивидуальное задание (составление поведенческой модели и функциональное моделирование).

Содержание отчета

1. Постановка задачи;
2. VHDL-описания выполненных заданий;
3. Анализ результатов RTL-синтеза каждого задания;
4. Выводы.

Варианты заданий

1. D-защелка с возможностью асинхронного сброса;
2. D-защелка с возможностью асинхронной предустановки;
3. D-триггер;
4. D-триггер с разрешением;
5. D-триггер с возможностью асинхронного сброса;
6. D-триггер с возможностью асинхронной предустановки;
7. RS-триггер;
8. JK-триггер;
9. T-триггер.

ЛИТЕРАТУРА

1. IEEE Standard VHDL Language Reference Manual. – New York: IEEE Std 1076-1993
2. Erschienen August 1994, IEEE, Taschenbuch ISBN: 1559373768, 1994. – 186 с.
3. Active-HDL User's Guid. Second Edition. – Aldec Inc. – 1999. – 213 с.
4. ГОСТ Р 50754-95 Язык описания аппаратуры цифровых систем VHDL. Описание языка.
5. Бибило П.Н. Основы языка VHDL. – М.: СОЛОН-ПРЕСС, 2007. – 200 с.
6. Кондратенко Ю.П., Сидоренко С.А., Подопригора Д.М. Поведенческий синтез цифровых устройств в среде Active-HDL. – Николаев: МФНаУКМА. – 2002. – 116 с.
7. Сергиенко А.М. VHDL для проектирования вычислительных устройств. – 2000. – 208 с.

СОДЕРЖАНИЕ

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ АНГЛИЙСКИЙ ЯЗЫК

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

What do you know about scientific investigation? Read the following quotations and express your own ideas about science. What is your field of science?

1. In science the opinions of a thousand are not worth as much as one tiny spark of reason in an individual man. ***(Galileo Galilei)***
2. Our society, in which reigns an eager desire for riches and luxury, does not understand the value of science. It does not realize that science is a most precious part of its moral patrimony. Nor does it take sufficient cognizance of the fact that science is at the base of all the progress that lightens the burden of life and lessens its suffering. ***(Marie Curie)***
3. Our scientific power has outrun our spiritual power. We have guided missiles and misguided men. ***(Martin Luther King, Jr.)***
4. When the number of factors coming into play in a phenomenological complex is too large scientific method in most cases fails. One need only think of the weather, in which case the prediction even for a few days ahead is impossible. ***(Albert Einstein)***
5. The aim of science is to seek the simplest explanations of complex facts. We are apt to fall into the error of thinking that the facts are simple because simplicity is the goal of our quest. The guiding motto in the life of every natural philosopher should be. *Seek simplicity and distrust it.* ***(Alfred North Whitehead)***
6. Science is founded on uncertainty. Each time we learn something new and surprising, the astonishment comes with the realization that we were wrong before… In truth, whenever we discover a new fact it involves the elimination of old ones. ***(Lewis Thomas)***
7. Eureka!" (I found it!) but "That's funny...***(Isaac Asimov)***

Exercise 1.Write down 10-12 sentences expressing your ideas about science.

Exercise 2. Read the definitions of science and choose the one that coincides with your opinion.

1) Science is the study of the nature and behavior of natural things and the knowledge that we obtain about them.
(https://www.collinsdictionary.com/dictionary/english/science)

2) The systematic study of the nature and behavior of the material and physical universe, based on observation, experiment, and measurement, and the formulation of laws to describe these facts in general terms
(https://www.merriam-webster.com/dictionary/science)

3) (knowledge from) the careful study of the structure and behavior of the physical world, especially by watching, measuring, and doing experiments, and the development of theories to describe the results of these activities:

- pure/applied science;
- recent developments in science and technology;
- Space travel is one of the marvels/wonders of modern science.

(https://dictionary.cambridge.org/dictionary/english/science)

Exercise 3. Read the text and come up with an appropriate title.

Science is defined as the observation, identification, description, experimental investigation, and theoretical explanation of natural phenomena. Webster's New Collegiate Dictionary gives the definition of science as "knowledge attained through study or practice" or "knowledge covering general truths of the operation of general laws, esp. as obtained and tested through scientific method [and] concerned with the physical world."

Here are some other common definitions of science:

- A branch of knowledge or study dealing with a body of facts or truths systematically arranged and showing the operation of general laws: for example, mathematical science.
- Systemic knowledge of the physical or material world gained through observations and experimentation
- Systematized knowledge in general
- Any of the particular branches of natural or physical sciences
- Knowledge of facts or principles; knowledge gained by systematic study
- Skill especially reflecting a precise application of facts or principle

The word Science comes from Latin word "scientia" meaning "knowledge" and in the broadest sense it is any systematic knowledge-base or prescriptive practice capable of resulting in prediction. Science can also be understood as a highly skilled technique or practice.

In more contemporary terms, science is a system of acquiring knowledge based on the scientific process or method in order to organize a body of knowledge gained through research.

Science is a continuing effort to discover and increase knowledge through research. Scientists make observations, record measureable data related to their observations, and analyze the information at hand to construct theoretical explanations of the phenomenon involved.

The methods involved in scientific research include making a hypothesis and conducting experiments to test the hypothesis under controlled conditions. In this process, scientists publish their work so other scientists can repeat the experiment and further strengthen the reliability of results.

Scientific fields are broadly divided into natural sciences (the study of natural phenomena) and social sciences (the study of human behavior and society). However, in both these divisions, knowledge is obtained through observation and must be capable of being tested for its validity by other researchers working under similar conditions.

There are some disciplines like health science and engineering that are grouped into interdisciplinary and applied sciences.

Most scientific investigations use some form of the scientific method. The scientific method tries to explain the events of nature in a reproducible way, eventually allowing researchers to formulate testable predictions.

Scientists make observations of natural phenomenon and then through experimentation they try to simulate natural events under controlled conditions. Based on observations, a scientist may generate a model and then attempt to describe or depict the phenomenon in terms of mathematical or logical representation.

Scientist will then gather the necessary empirical evidence to generate a hypothesis to explain the phenomenon.

This hypothesis is used to form predictions which in turn will be tested by experiment or observations using the scientific method. Statistical analysis is commonly used to interpret results of experiments, and evaluations are made to decide whether a hypothesis should be accepted, rejected, or merely examined again with modifications. This inspires ongoing research and the overall accumulation of knowledge in that particular field of science.

(https://explorable.com/definition-of-science)

Exercise 4. a) Compose 5 questions based on the text. Ask your partner your questions. Answer your partner's questions.

Exercise 5. a) Find the underlined words in the text and try to explain their meanings. Use the dictionary if necessary; b) Choose best explanation for the following words.

1) hypothesis

a) a supposition or proposed explanation made on the basis of limited evidence as a starting point for further investigation.
b) a proposition made as a basis for reasoning, without any assumption of its truth.

2) modification

a) the action of modifying something.
b)a change to something, usually to improve it

3) observation

a) the action or process of closely observing or monitoring something or someone.
b) a statement based on something one has seen, heard, or noticed.

4)identification

a) the action or process of identifying someone or something or the fact of being identified.
b)a person's sense of identity with someone or something.

5) investigation

a) the action of investigating something or someone
b) a formal inquiry or systematic study**.**

6)explanation

a) a statement or account that makes something clear.
b)a reason or justification given for an action or belief.

7) description

a) a spoken or written account of a person, object, or event.
b) a type or class of people or things**.**

8)empirical

a) based on, concerned with, or verifiable by observation or experience rather than theory or pure logic.
b)relying on experience or observation alone often without due regard for system and theory

Exercise 6. a) Match the words with their synonyms. b) Prepare a short speech on your major using as many words as possible

A	B
1) method	a) regular
2) systematic	b) method
3) prediction	c) measured
4) technique	d) rational
5) data	e) way
6) engineering	f) arrange
7) mathematical	g) forecast
8) logical	h) information

Exercise 7. Fill in the gaps in the sentences below with the words from the list.

measure facts important tool(s) medicines investigation investigators both experimenting focused

1. Research is scholarly inquiry, often it is done primarily by research scientists or their research assistants, often using one or more specialized _____.
2. When meeting someone addressed by someone else as “Doctor So-and-so,” their answer to the question, “Are you researcher or a physician?” can tell you whether they are a Ph.D. (researcher) or an M.D. (physician) “doctor.” (Only MDs are allowed to prescribe _____.)
3. _____is a type of research, often brought about by some unwanted situation (e.g., why a bridge failed under load, or a sudden, unexplained death).
4. Due to the complexity and significance of many investigations, they are typically done by trained, experienced professional _____ or detectives.
5. Reliability of results is very_____, so if a study is replicated the findings should be similar.
6. Validity, does a test measure what it was designed to _____. For example, do IQ tests really measure 'intelligence'?
7. Investigation generally implies official/thorough examination of _____, possible causes, and results.
8. Scientific investigation and research are_____concepts that apply for search for something about which a new fact or knowledge is sought..
9. Kids in their early ages are often fond of_____ things.
10. Science can cover a lot of different subjects, so depending on the type of science that you are studying, each one will require a unique set

of skills that you need to learn, and you will also have to study areas of knowledge that are _____on a particular subject.

Exercise 8. Match the definition to the word-combinations. Work in pairs and compare your answers.

Formal sciences	– the study of mathematics and logic, which use an a priori, as opposed to factual, methodology
Natural sciences	– the study of human behavior and societies
Social sciences	– the study of natural phenomena (including cosmological, geological, chemical, and biological factors of the universe)

Exercise 9. Read the text and answer the following true/false question.

Scientist vs. scholar

In English (at least in American English) the word scientist has a more narrow meaning than scholar. A scientist can study physics or geology, possibly archaeology, but not history or religion. These areas are called "humanities" as opposed to "sciences", and a person researching, for instance, historical background of the Bible may be a Biblical scholar, but not a Biblical scientist.

Similarly in Russian, Наука includes history, philosophy, art studies etc. Ученый (scientist) may study the Bible, or Shakespeare.

In some sense the American division is rather arbitrary – why is political science a science, athough history is not, even though history can be much more objective than political science?

"Scientist" is used when referring specifically to a person who is an expert in a science, especially physical or natural sciences. "Scholar" is broader, and can be used generally for anyone who has profound knowledge of a particular subject. "Scholar" isn't used for sciences (chemistry, geography), but can be used for History, Language, etc.

True or False?

1) In English the word scientist has a wide meaning than scholar.
2) A scientist can study physics or geology, possibly archaeology as well as history or religion.
3) Physics or geology, possibly archaeology are called "humanities".
4) A person researching historical background of the Bible may be a Biblical scholar.

5) "Scholar" can be used for anyone who has deep knowledge of a particular subject.
6) "Scholar" is used for sciences, but cannot be used for Language.

Exercise 10. a) Look through the text again and find the sentences that use the Passive Voice.
b) Can you change these sentences from Passive Voice into Active Voice? Which sentences are better in Active? Discuss it with your partner.

Exercise 11. Start your "PROFESSIONAL VOCABULARY". Think of definitions, synonyms and antonyms to explain the meaning of these words.

Exercise 12. Read the text and summarize it (5-7 sentences). *(To summarize – to provide the most important information without details and examples).*

Scientific Investigation

Developments in science and technology are fundamentally altering the way people live, connect, communicate and transact, with profound effects on economic development. Science and technology are key drivers to development, because technological and scientific revolutions underpin economic advances, improvements in health systems, education and infrastructure.

Scientific research has an enormous impact on modern society, with its effects felt in many aspects of our lives. But scientists are also part of that society, and can adapt their research topics and methods to reflect its ever-changing priorities. All too often, though, these priorities are dictated by governments or by the private sector, while the views of members of the public aren't heard. However, it's certainly possible for interested individuals to influence the course of scientific research.

The Ebola outbreak in West Africa still defies attempts at containment. A new climate agreement is still in question while global greenhouse gas emissions rise largely unchecked. Millions of people are facing a food crisis as funding dries up and extreme weather events disrupt farming. The global economy is languishing, inequality is putting pressure on social and political systems, and ecosystems are under threat.

Human activity has come to dominate many of the Earth's life-support systems, but are societies prepared to address these challenges? Are we even able to assess the risks they pose to our wellbeing? Science can play an important role, but research programmes must become more interconnected if we are to address today's most pressing developing challenges – from improving health, food security and access to clean energy and water, to understanding how demographic shifts affect our production and consumption patterns.

As someone who studies the impact of science and technology on society, it was clear that whatever issues motivated us to get involved in politics, they were going to change dramatically over the next five to ten years thanks to science and technology. Digital technologies are already making it harder to fund public services, as they enable companies to benefit from "stateless profits", depriving the public purse of much needed tax receipts. Artificial Intelligence (AI) and the advent of robots is going to rapidly transform industry, healthcare and the job market in ways we have never seen before; we need to think deeply about what this means for the skills and education we should offer future generations. And the way highly

paid jobs in the tech industry are usually men's jobs means that the gender pay gap is likely to widen as these industries grow in the future. In short, in the 21st century, science and technology is one of the most powerful drivers of change and shaper of our society. And we cannot govern the country if we don't understand how to govern these technologies.

(https://www.theguardian.com/science/political-science/2017/sep/25/the-science-and-technology-committee-shouldnt-be-filled-with-scientists-female-or-not)

Exercise 13. Read the following text and give the main idea of it in English.

РАЗВИТИЕ РОССИЙСКОЙ НАУКИ

День науки ежегодно отмечается в Российской Федерации 8 февраля. Именно в этот день в 1724 г. Петр Первый издал указ об основании Академии наук, и теперь эта дата стала праздником всех российских ученых.

Наша страна имеет славные традиции развития научного потенциала, что делает Россию одним из мировых лидеров во многих областях знаний. Российская наука – держит уровень и сегодня, а это очень важно, так как от достижений ученых зависит экономика, обороноспособность, медицина и промышленность нашей страны, а значит благосостояние и жизнь большинства российских граждан. Уже почти тысячу лет у нас в стране развивают науку, стараясь не только не отставать от ведущих стран, но и выйти в абсолютные мировые лидеры.

Центрами науки и просвещения на Руси сначала были монастыри. Именно там монахами были написаны работы по математике, истории, лингвистике еще в нач. Только в XVII веке в России появляются первые научные центры, самым известным их них по праву считается Славяно-греко-латинская академия, выпустившая из своих стен много известных выпускников.

Император Петр I поставил цель ликвидировать отставание нашей страны от передовых государств и создал в Санкт-Петербурге Академию наук, куда пригласил работать учёных со всего мира. Особую славу принес себе в России и во всем мире Михаил Ломоносов, который не только создал Московских государственный университет, но и сам активно изучал химию, географию, физику, историю, экономику, лингвистику и многие другие дисциплины, где добился огромных успехов. Позже университеты открылись во многих крупных городах Российской империи.

«Золотым веком» российской науки по праву считался XIX в. Признанных достижений в области математики добивается Н. И.

Лобачевский, в химии Д. И. Менделеев и А. М. Бутлеров, в истории Н.М. Карамзин и С. М. Соловьев, в медицине – С. П. Боткин, и многие-многие другие ученые. В нач. XX в. работали такие исследователи с мировым именем: И. П. Павлов, получивший Нобелевскую премию за исследования в области физиологии пищеварения; И. И. Мечников добившийся её же за свои работы в медицине.

После революции наука была поставлена на службу государству, а именно его оборонительной мощи. Некоторые гуманитарные науки сначала были подвергнуты опале, а ученых-философов, например, и вовсе выслали из страны на пароходе. Большевики поставили себе задачу – как можно быстрее восстановить страну и построить мощную промышленность. Коммунистическим властям удалось создать эффективную систему организации научной деятельности, из-за которой государство быстро стало индустриальным, а также передовым в научной сфере. Созданы были многочисленные НИИ, действовала Академия наук СССР и ее филиалы во многих советских городах. При ВУЗах работали кафедры и институты, которые не только выращивали новые научные кадры, но и успешно занимались исследованиями.

Успехи советских ученых были замечены мировым научным сообществом, многие из них были награждены Нобелевской и другими премиями. Работы И. В. Курчатова, А. Д. Сахарова, С. П. Королева, Л. Д. Ландау, П. Л. Капицы и других советских ученых внесли огромный вклад в мировую науку, плоды которого мы используем и по сей день.

Российские исследователи продолжают славные традиции прошлого в науке. В РФ действует порядка четырех тысяч различных научных организаций и обществ, большинство государственных, которые занимаются научными исследованиями. Самых значительных успехов российские ученые добились в физике, биологии и химии, в то же время по гуманитарным и общественным наукам есть некоторое отставание, которое надеются сократить в самое ближайшее время. Вообще, в современной России особенное внимание уделяют знаниям в сфере безопасности, освоения космоса, военных вооружений, ядерной энергетики, телекоммуникационных систем и прочим.

(http://katehon.com/ru/article/razvitie-rossiyskoy-nauki)

Exercise 14. Prepare a 5 minutes public speech (in Russian) about the role of science in our life. Make up an abstract (5-7 sentences) of the speech and mark 5 key words.

Exercise 15. After presenting your speech give the key words and abstract to other students who will render your speech in English.

When preparing a speech or a presentation as well use Appendix I.

Exercise 16. a) <u>Scan</u> the text below and take brief notes about what these numbers refers to: 1959; 800; 8.

<u>Scan</u> – to read smth. very quickly to find particular information.

b) First read the questions and then scan the text to find the answers.

1) When and where Russian Academy of Sciences was organized?
2) What is known about the membership of Academy of Sciences?
3) What departments does it include?
4) Who was the first Russian member of Academy?
5) What can you say about the society's highest prize?

Academy of Sciences, in full (1917–25 and since 1991) Russian Academy of Sciences, Russian Rossiiskaya Akademiya Nauk, highest scientific society and principal coordinating body for research in natural and social sciences, technology, and production in Russia. The organization was established in St. Petersburg, Russia, on February 8 (January 28, Old Style), 1724. Membership in the academy is by election, and members can be one of three ranks—academician, corresponding member, or foreign member. The academy is also devoted to training students and to publicizing scientific achievements and knowledge. It maintains ties with many international scientific institutions and collaborates with foreign academies. The academy directs the research of other scientific institutions and institutions of higher education. It includes departments of mathematics; physics; power engineering, mechanics, and control processes; information science and computer technology; chemistry and materials; biology; earth sciences; social sciences; and history and philology. Its membership is more than 1,500, with some 800 corresponding members, 500 academicians, and 200 foreign members.

Founded by Peter I the Great, the academy was opened in 1725 by his widow, Catherine I, as the Academy of Sciences and Arts. Later known under various names, it held its present name from 1917 to 1925 and took it once more in 1991. In its early decades, foreign scholars, notably the Swiss mathematicians Leonhard Euler and Daniel Bernoulli, worked in the academy. The first Russian member was Mikhail Vasilyevich Lomonosov, scientist and poet, who was elected in 1742 and contributed extensively to many branches of science. The society's highest prize, the Lomonosov Gold Medal, bears his name; it was first awarded in 1959 and is traditionally given each year to one Russian and one foreign scientist. Under the tsars, the academy was headed by members of court circles and controlled a small number of institutions. After 1917 the academy chose its president and

expanded its activities as new scientific institutions arose throughout the Soviet Union. By 1934, when it transferred from St. Petersburg to Moscow, it embraced 25 institutes. Before the dissolution of the Soviet Union in 1991, the academy directed more than 260 institutions, including laboratories, naval institutes, observatories, research stations, and scientific societies, and its branches were spread throughout the Soviet Union. Since 1999 the date of the academy's founding, February 8, has been observed as a national day of science.

(https://www.britannica.com/topic/Academy-of-Sciences-Russian-organization)

c) Translate the first paragraph of the text into Russian in written form using your dictionary (if necessary).

d) Work in pairs. What information is new to you? Discuss with the partner.

Exercise 17. How is your field of science developing in Russia? What do you expect to see in the future? Write a paragraph (10 – 12 sentences).

Exercise 18. Write an essay (about 15 sentences) using the vocabulary of this unit. Choose one of the following topics:

1) What is Science?
2) Development of Science in Russia.
3) My Scientific Interests. My Future Investigations in Science.

READING

Exercise 1. Read the text and summarize it (9 – 10 sentences).
Exercise 2. Translate the text into Russian.

Alexander Graham Bell

Alexander Graham Bell invented the telephone. Remarkably, he only worked on his invention because he misunderstood a technical work he had read in German. His misunderstanding ultimately led to his discovery of how speech could be transmitted electrically.

Alexander Graham Bell was born March 3, 1847 in Edinburgh, Scotland, UK. His mother's name was Eliza Grace Symonds.

His father, Alexander Melville Bell, was a professor of speech elocution at the University of Edinburgh. His father also wrote definitive books about speech and elocution, which sold very well in the UK and North America.

The young Alexander was home-schooled until he was 11, following which he attended Edinburgh's Royal High School for four years: he enjoyed science, but did not do well academically.

Although his schoolwork was poor, his mind was very active. One day, he was playing at a flour mill owned by the family of a young friend. Bell learned that de-husking the wheat grains took a lot of effort and was also very boring. He saw that it would be possible for a machine to do the work, so he built one. He was only 12 at the time. The machine he built was used at the mill for several years.

Aged 15, he joined his grandfather who had moved to London, England. His grandfather home-schooled him, which seemed to bring out the best in Bell again. When he was 16, he enrolled at Weston House Academy in Elgin, Scotland, where he learned Greek and Latin and also earned some money teaching elocution.

While he was 16, he and his brother tried to build a talking robot. They built a windpipe and a realistic looking head. When they blew air through the windpipe, the mouth could make a small number of recognizable words.

For the next few years, Bell moved to a new school most years, either teaching elocution or improving his own education.

To Canada

While Bell moved around a lot, he continued to carry out his own research into sound and speech. He worked very hard indeed, and by the time he was 20 he was in very poor health and returned to his family home, which was now in London.

By mid-1870, when Bell was 23, both of his younger brothers had died of

tuberculosis. Bell's parents were terrified that Alexander, whose health was fragile, would suffer a similar fate. He was now their only surviving child.

Bell's father had gone to Canada when he was younger and found that his poor health had improved dramatically. He now decided that what was left of his family should move to Canada, and by late 1870, they were living in Brantford, Ontario. Thankfully, Alexander Graham Bell's health began to improve.

While living in Brentford, Bell learned the Mohawk language and put it in writing for the first time. The Mohawk people made him an Honorary Chief.

And the United States

When he was 25, Bell opened his School of Vocal Physiology and Mechanics of Speech in Boston, MA, where he taught deaf people to speak. At age 26, although he did not have a university degree, he became Professor of Vocal Physiology and Elocution at the Boston University School of Oratory.

The Invention of the Telephone

While he was moving jobs and locations around the UK and North America, Bell had developed an overriding desire to invent a machine that could reproduce human speech.

Speech had become his life: his mother had gone deaf, and Bell's father had developed a method of teaching deaf people to speak, which Bell also taught. His research into mechanizing human speech had become a relentless obsession: in the UK it had driven him almost to collapse.

When Bell was only 19 years old, he had described his work in a letter to the linguistics expert Alexander Ellis. Ellis told Bell his work was similar to work carried out in Germany by Hermann von Helmholtz.

A Mistake Puts Bell on the Right Track

Bell eagerly read Helmholtz's work, or tried to read it. It was in German, which he did not understand. Instead, he tried to follow the logic of the book's diagrams. Bell misunderstood the diagrams, believing that Helmholtz had been able to convert all of the sounds of speech to electricity.

In fact, Helmholtz had not been able to do this – he had only succeeded with vowel sounds – but from then on, Bell believed it could be done!

Aged 23, Bell built a workshop in the new family home in Ontario and experimented there with converting music into an electrical signal.

In Boston, aged 25, Bell continued his experiments through the night while working in the day. In summer, he would return to his workshop in Ontario and continue his experiments.

Financial Backing for a Voice Telegraph

In 1874 Bell was 26. The first electrical telegraph lines had been built forty years earlier, in the 1830s. These allowed electrical clicks (Morse code)

to be instantly transmitted over great distances. Bell wanted to transmit human speech instead of clicks, and he was getting close to doing it.

He had found that human speech came in wave like patterns. He now hoped to produce an electrical wave that would follow the same patterns as someone's speech.

He won financial backing from Gardiner Hubbard and Thomas Sanders, two wealthy investors. Hubbard also brought in Anthony Pollok, his patent attorney. The money enabled Bell to hire Thomas Watson, a skilled electrical engineer, whose knowledge would complement Bell's.

Patenting the Telephone

Aged 27, in 1875, Bell and his investors decided the time had come to protect his intellectual property using patents.

Bell had a patent written for transmission of speech over an electrical wire. He applied for this patent in the UK, because in those days UK patents were granted only if they had not first been granted in another country. Bell told his attorney to apply in the USA only *after* the patent had been granted in the UK.

By 1876, things in the USA had become murkier. In February of that year, Elisha Gray applied for a US patent for a telephone which used a variable resistor based on a liquid: salt water.

In the transmitter, the liquid resistor transferred to an electric circuit the vibrations of a needle attached to a diaphragm which had been made to vibrate by sound. The electrical resistance of the circuit changed in tandem with the needle's position in the liquid and so sound was converted into an equivalent electrical signal. The receiver converted the electrical signal back into sound using a vibrating needle in liquid connected to a diaphragm which vibrated to recreate the sound that had been transmitted.

On the same day, Bell's attorney filed his US patent application.

It was only in March 1876 that Bell actually got his invention to work, using a design similar to Gray's. Hence Gray lay claim to have invented the telephone.

On the other hand, Bell had established the concept before Gray, and in all demonstrations of a working phone Bell gave or developed commercially he used his own setup rather than a water based variable resistor. In fact, in 1875, Bell had filed a patent for a liquid mercury based variable resistor, predating Gray's liquid variable resistor patent.

Bell had to fend off around 600 lawsuits before he could finally rest in bed at night as the legally acknowledged inventor of the telephone.

"Mr. Watson, come here. I want to see you." **THE FIRST WORDS SPOKEN IN A TELEPHONE CALL: ALEXANDER GRAHAM BELL**

Inventor

By summer 1876, Bell was transmitting telephone voice messages over a line several miles long in Ontario.

Making Money

Near the end of 1876, Bell and his investors offered to sell their patent to Western Union for $100,000. Western Union ran America's telegraph wires, and its top people believed the telephone was just a fad. They thought it would not be profitable.

How spectacularly wrong they were!

By 1878, Western Union's opinion had altered dramatically. They now thought that if they could offer $25 million to get the patent, they would have gotten it cheaply.

Unfortunately for Western Union, in 1877, the Bell Telephone Company had been launched. And the rest, as they say, is history.

Not Just the Telephone

Alexander Graham Bell had a restless mind. The telephone made him wealthy and famous, but he wanted new challenges, and he continued inventing and innovating.

The Photophone, or Optical Telephone

Today, it is standard practice to transmit huge amounts of data using photons of light through optical fiber.

In 1880, Bell and his assistant Charles Summer Tainter transmitted wireless voice messages a distance of over 200 meters in Washington D.C. The voice messages were carried by a light beam, and Bell patented the photophone. This was two decades before the first radio messages were sent without wires and a century before optic fiber communications became commercially viable.

The receiver of Bell's photophone. In Bell's opinion, the photophone was his best invention.

The Metal Detector

In 1881, after President James Garfield was shot, Bell invented the metal detector to locate the bullet precisely. The rudimentary metal detector worked in tests, but the bullet in the President's body was too deep to be detected by the early detecting equipment.

National Geographic Society

In 1888 Bell was one of the founders of the National Geographic Society. In 1897, he became its second president.

The End

Alexander Graham Bell died aged 75 on August 2, 1922 in Nova Scotia, Canada. He had been ill for some months with complications from diabetes. He was survived by his wife, Mabel, and two daughters – Elsie and Marian.

Every phone in North America was silenced during his funeral in his honor.

The unit of sound intensity, the bel, more usually seen as the smaller unit, the decibel, was named after Bell: it was conceived of in the Bell Laboratories.

(https://www.famousscientists.org/alexander-graham-bell/)

UNIT 2

Studying Computer Engineering

What do you know about computer engineering? Read the following quotations and express your own ideas about it. Why have you decided to be a computer engineer?

1. "Part of what made the Macintosh great was that the people working on it were musicians, poets, and artists, and zoologists, and historians. They also happened to be the best computer scientists in the world. But if it hadn't been computer science, these people would have been doing amazing things in other fields." ***(Steve Jobs)***

2. "Computers are useless. They can only give you answers." ***(Pablo Picasso)***

3. "The question of whether computers can think is like the question of whether submarines can swim." ***(Edsger W. Dijkstra)***

4. "Never trust a computer you can't throw out a window."***(Steve Wozniak)***

5. "Software suppliers are trying to make their software packages more 'user-friendly'… Their best approach so far has been to take all the old brochures and stamp the words 'user-friendly' on the cover." ***(Bill Gates)***

6. Computer science education cannot make anybody an expert programmer any more than studying brushes and pigment can make somebody an expert painter. ***(Eric S. Raymond)***

7. Computer Science is a science of abstraction -creating the right model for a problem and devising the appropriate mechanizable techniques to solve it. ***(Alfred Aho)***

Exercise 1.Write down 10-12 sentences expressing your ideas about computer engineering.

Exercise 2. Students at your faculty are known to study the following programs:

- Informatics and Computer Engineering
- Information Systems and Technologies
- Software Engineering
- Computer Systems, Complexes, Networks

• Applied Informatics

What is your specialty? Why have you chosen it?

Exercise 3. Read the following definitions and translate them into Russian.

1. Computer science is the study of the theory, experimentation, and engineering that form the basis for the design and use of computers. It is the scientific and practical approach to computation and its applications and the systematic study of the feasibility, structure, expression, and mechanization of the methodical procedures (or algorithms) that underlie the acquisition, representation, processing, storage, communication of, and access to information. An alternate, more succinct definition of computer science is the study of automating algorithmic processes that scale. A computer scientist specializes in the theory of computation and the design of computational systems. (source: Computer Science @ Wikipedia)

2. Informatics is a branch of information engineering. It involves the practice of information processing and the engineering of information systems, and as an academic field it is an applied form of information science. The field considers the interaction between humans and information alongside the construction of interfaces, organisations, technologies and systems. As such, the field of informatics has great breadth and encompasses many subspecialties, including disciplines of computer science, information systems, information technology and statistics. Since the advent of computers, individuals and organizations increasingly process information digitally. This has led to the study of informatics with computational, mathematical, biological, cognitive and social aspects, including study of the social impact of information technologies. (source: Informatics @ Wikipedia)

3. An information system is software that helps you organize and analyze data. This makes it possible to answer questions and solve problems relevant to the mission of an organization.

4. Information system, an integrated set of components for collecting, storing, and processing data and for providing information, knowledge, and digital products. Business firms and other organizations rely on information systems to carry out and manage their operations, interact with their customers and suppliers, and compete in the marketplace. (https://www.britannica.com/topic/information-system)

5. Software engineering is the application of principles used in the field of engineering, which usually deals with physical systems, to the design, development, testing, deployment and management of software systems. (https://whatis.techtarget.com/definition/software-engineering)

6. A computer system is a basic, complete and functional computer, including all the hardware and software required to make it functional for a user. (https://www.techopedia.com/definition/593/computer-system)

7. A system of interconnected computers that share a central storage system and various peripheral devices such as a printers, scanners, or routers. Each computer connected to the system can operate independently, but has the ability to communicate with other external devices and computers. (http://www.businessdictionary.com/definition/computer-system.html)

8. Development of procedures and systematic applications that are used on electronic machines. Software engineering incorporates various accepted methodologies to design software. This particular type of engineering has to take into consideration what type of machine the software will be used on, how the software will work with the machine, and what elements need to be put in place to ensure reliability.

9. Higher education degree program, which usually requires a certain number of courses to be completed in order to receive certification or a degree. (http://www.businessdictionary.com/definition/software-engineering.html)

Exercise 4. Read the text and come up with an appropriate title.

What is informatics? Informatics is the study of the structure, behavior, and interactions of natural and engineered computational systems. Informatics studies the representation, processing, and communication of information in natural and engineered systems. It has computational, cognitive and social aspects. The central notion is the transformation of information - whether by computation or communication, whether by organisms or artifacts. Understanding informational phenomena - such as computation, cognition, and communication - enables technological advances. In turn, technological progress prompts scientific enquiry. The science of information and the engineering of information systems develop hand-in-hand. Informatics is the emerging discipline that combines the two. In natural and artificial systems, information is carried at many levels, ranging, for example, from biological molecules and electronic devices through nervous systems and computers and on to societies and large-scale distributed systems. It is characteristic that information carried at higher levels is represented by informational processes at lower levels. Each of these levels is the proper object of study for some discipline of science or engineering. Informatics aims to develop and apply firm theoretical and mathematical foundations for the features that are common to all computational systems.

In its attempts to account for phenomena, science progresses by defining, developing, criticizing and refining new concepts. Informatics is developing its own fundamental concepts of communication, knowledge, data, interaction and information, and relating them to such phenomena as computation, thought, and language. Informatics has many aspects, and encompasses a number of existing academic disciplines – Artificial Intelligence, Cognitive Science and Computer Science. Each takes part of Informatics as its natural domain: in broad terms, Cognitive Science concerns the study of natural systems; Computer Science concerns the analysis of computation, and design of computing systems; Artificial Intelligence plays a connecting role, designing systems which emulate those found in nature. Informatics also informs and is informed by other disciplines, such as Mathematics, Electronics, Biology, Linguistics and Psychology. Thus Informatics provides a link between disciplines with their own methodologies and perspectives, bringing together a common scientific paradigm, common engineering methods and a pervasive stimulus from technological development and practical application. Three of the truly fundamental questions of Science are: "What is matter?", "What is life?" and "What is mind?" The physical and biological sciences concern the first two. The emerging science of Informatics contributes to our understanding of the latter two by providing a basis for the study of organization and process in biological and cognitive systems. Progress can best be made by means of strong links with the existing disciplines devoted to particular aspects of these questions.

Computational systems, whether natural or engineered, are distinguished by their great complexity, as regards both their internal structure and behavior, and their rich interaction with the environment. Informatics seeks to understand and to construct (or reconstruct) such systems, using analytic, experimental and engineering methodologies. The mixture of observation, theory and practice will vary between natural and artificial systems. In natural systems, the object is to understand the structure and behavior of a given computational system. The theoretical concepts underlying natural systems ultimately are built on observation and are themselves used to predict new observations. For engineered systems, the object is to build a system that performs a given informational function. The theoretical concepts underlying engineered systems are intended to secure their correct and efficient design and operation. Informatics provides an enormous range of problems and opportunities. One challenge is to determine how far, and in what circumstances, theories of information processing in artificial devices can be applied to natural systems. A second challenge is to determine how far principles derived from natural systems are applicable to

the development of new kinds of engineered systems. A third challenge is to explore the many ways in which artificial information systems can help to solve problems facing mankind and help to improve the quality of life for all living things. One can also consider systems of mixed character; a question of longer term interest may be to what extent it is helpful to maintain the distinction between natural and engineered systems.

(https://www.ed.ac.uk/files/atoms/files/what20is20informatics.pdf)

Exercise 5. Make a short summary of the text.

Exercise 6. a) Find the underlined words in the text and try to explain their meanings. Use the dictionary if necessary; b) Choose best definition for the following words.

1) system

a) a set of things working together as parts of a mechanism or an interconnecting network; a complex whole.
b) a set of principles or procedures according to which something is done; an organized scheme or method.

2) communication

a) the imparting or exchanging of information by speaking, writing, or using some other medium.
b) means of sending or receiving information, such as telephone lines or computers.

3) cognition

a) concerned with the act or process of knowing, perceiving, etc.
b) of or relating to the mental processes of perception, memory, judgment, and reasoning, as contrasted with emotional and volitional processes.

4) large-scale

a) involving many people or things.
b) covering or involving a large area.

5) to aim

a) point or direct (a weapon or camera) at a target.
b) have the intention of achieving.

6) to refine

a) elegant and cultured in appearance, manner, or taste.
b) developed or improved so as to be precise or subtle.

7) intelligence

a) the ability to acquire and apply knowledge and skills.
b) the collection of information of

8) to emulate

a) to imitate.
b) to reproduce the function or action of (a different computer, software system, etc.).

10) challenge

a) a call to someone to participate in a competitive situation or fight to decide who is superior in terms of

military or political value.

9) environment

a) the surroundings or conditions in which a person, animal, or plant lives or operates.
b) the natural world, as a whole or in a particular geographical area, especially as affected by human activity.

ability or strength.
b) a call to prove or justify something.

Exercise 7. Match the words with their synonyms.

A	**B**
study	education
natural	digital
transformation	normal
computational	comprehension
link	conversion
understanding	planning
organization	connection
artificial	fake

Exercise 8. Fill in the gaps in the sentences below with the words from the list.

operating; database; browser; system; peripheral; microprocessors; microwaves; radio waves; contains; computer security; consist of; layer; top

1. The principal system software is the _____.
2. The power of the _____at the heart of computing devices has been doubling approximately every 18 to 24 months.
3. The_____equipment are magnetic or solid-state storage disks, input-output devices, and telecommunications gear.
4. Wireless technologies, predominantly based on the transmission of _____and _____ , support mobile computing.
5. The Internet – type services can be provided within an organization and for its exclusive use by various intranets that are accessible through a _____.
6. A_____ is a collection of interrelated data organized so that individual records or groups of records can be retrieved to satisfy various criteria.

7. Technical personnel include development and operations managers, business analysts, systems analysts and designers, database administrators, programmers,_____ _____specialists, and computer operators.
8. Information systems_____ three layers: operational support, support of knowledge work, and management support.
9. Operational support forms the base of an information system and various transaction processing systems for designing, marketing, producing, and delivering products and services.
10. Support of knowledge work forms the middle_; it contains subsystems for sharing information within an organization.
11. Management support, forming the___layer, contains subsystems for managing and evaluating an organization's resources and goals.

Exercise 9. Match the explanation to the word-combinations. Work in pairs and compare your answers.

	– forms the middle layer; it contains subsystems for sharing information within an organization.
Management support	– forms the base of an information system and contains various transaction processing systems for designing, marketing, producing, and delivering products and services.
Support of knowledge work	– forms the top layer and contains subsystems for managing and evaluating an organization's resources and goals.
Operational support	

Exercise 10. Read the text and answer the following true/false questions.

Operating systems

Operating systems manage a computer’s resources—memory, peripheral devices, and even CPU access—and provide a battery of services to the user’s programs. UNIX, first developed for minicomputers and now widely used on both PCs and mainframes, is one example; Linux (a version of UNIX), Microsoft Corporation’s Windows XP, and Apple Computer’s OS X are others.

One may think of an operating system as a set of concentric shells. At the center is the bare processor, surrounded by layers of operating system routines to manage input/output (I/O), memory access, multiple processes, and communication among processes. User programs are located in the outermost layers. Each layer insulates its inner layer from direct access, while

providing services to its outer layer. This architecture frees outer layers from having to know all the details of lower-level operations, while protecting inner layers and their essential services from interference.

Early computers had no operating system. A user loaded a program from paper tape by employing switches to specify its memory address, to start loading, and to run the program. When the program finished, the computer halted. The programmer had to have knowledge of every computer detail, such as how much memory it had and the characteristics of I/O devices used by the program.

It was quickly realized that this was an inefficient use of resources, particularly as the CPU was largely idle while waiting for relatively slow I/O devices to finish tasks such as reading and writing data. The earliest operating systems were small supervisor programs that did just that: they coordinated several programs, accepting commands from the operator, and provided them all with basic I/O operations. These were known as multiprogrammed systems.

A multiprogrammed system must schedule its programs according to some priority rule, such as “shortest jobs first.” It must protect them from mutual interference to prevent an addressing error in a program from corrupting the data or code of another. It must ensure noninterference during I/O so that output from several programs does not get commingled or input misdirected. It might also have to record the CPU time of each job for billing purposes.

Modern types of operating systems:

An extension of multiprogramming systems was developed in the 1960s, known variously as multiuser or time-sharing systems. Time-sharing allows many people to interact with a computer at once, each getting a small portion of the CPU’s time. If the CPU is fast enough, it will appear to be dedicated to each user, particularly as a computer can perform many functions while waiting for each user to finish typing the latest commands.

Multiuser operating systems employ a technique known as multiprocessing, or multitasking (as do most single-user systems today), in which even a single program may consist of many separate computational activities, called processes. The system must keep track of active and queued processes, when each process must access secondary memory to retrieve and store its code and data, and the allocation of other resources, such as peripheral devices.

Since main memory was very limited, early operating systems had to be as small as possible to leave room for other programs. Virtual memory gives each process a large address space (memory that it may use), often much larger than the actual main memory. This address space resides in

secondary memory (such as tape or disks), from which portions are copied into main memory as needed, updated as necessary, and returned when a process is no longer active. Even with virtual memory, however, some "kernel" of the operating system has to remain in main memory. Early UNIX kernels occupied tens of kilobytes; today they occupy more than a megabyte, and PC operating systems are comparable, largely because of the declining cost of main memory.

Operating systems have to maintain virtual memory tables to keep track of where each process's address space resides, and modern CPUs provide special registers to make this more efficient. Indeed, much of an operating system consists of tables: tables of processes, of files and their locations (directories), of resources used by each process, and so on. There are also tables of user accounts and passwords that help control access to the user's files and protect them against accidental or malicious interference.

Thin systems:

While minimizing the memory requirements of operating systems for standard computers has been important, it has been absolutely essential for small, inexpensive, specialized devices such as personal digital assistants (PDAs), "smart" cellular telephones, portable devices for listening to compressed music files, and Internet kiosks. Such devices must be highly reliable, fast, and secure against break-ins or corruption—a cellular telephone that "freezes" in the middle of calls would not be tolerated. One might argue that these traits should characterize any operating system, but PC users seem to have become quite tolerant of frequent operating system failures that require restarts.

Reactive systems:

Still more limited are embedded, or real-time, systems. These are small systems that run the control processors embedded in machinery from factory production lines to home appliances. They interact with their environment, taking in data from sensors and making appropriate responses. Embedded systems are known as "hard" real-time systems if they must guarantee schedules that handle all events even in a worst case and "soft" if missed deadlines are not fatal. An aircraft control system is a hard real-time system, as a single flight error might be fatal. An airline reservation system, on the other hand, is a soft real-time system, since a missed booking is rarely catastrophic.

Many of the features of modern CPUs and operating systems are inappropriate for hard real-time systems. For example, pipelines and superscalar multiple execution units give high performance at the expense of occasional delays when a branch prediction fails and a pipeline is filled with unneeded instructions. Likewise, virtual memory and caches give good

memory-access times on the average, but sometimes they are slow. Such variability is inimical to meeting demanding real-time schedules, and so embedded processors and their operating systems must generally be relatively simple.

Operating system design approaches:

Operating systems may be proprietary or open. Mainframe systems have largely been proprietary, supplied by the computer manufacturer. In the PC domain, Microsoft offers its proprietary Windows systems, Apple has supplied Mac OS for its line of Macintosh computers, and there are few other choices. The best-known open system has been UNIX, originally developed by Bell Laboratories and supplied freely to universities. In its Linux variant it is available for a wide range of PCs, workstations, and, most recently, IBM mainframes.

(https://www.britannica.com/technology/computer)

True or False?

1) User programs are located in the outermost layers.
2) Early computers had a good operating system.
3) The earliest operating systems were small supervisor programs.
4) Early operating systems had to be as big as possible to leave room for other programs.
5) Embedded systems are known as “hard” real-time systems.
6) Many of the features of modern CPUs and operating systems are suitable for hard real-time systems.
7) Virtual memory and caches give good memory-access times on the average.
8) Mainframe systems have not been proprietary and not supplied by the computer manufacturer.

Exercise 11. a) Look through the text again and find all the words with “-ing” forms. Explain what do these grammatical forms mean. You may go to Grammar Appendix (if necessary).
b) Look through the text again and find all the modal verbs used in it. How many modal verbs you have found? Discuss it with your partner.

Exercise 12. a) Start your “PROFESSIONAL VOCABULARY LIST”. Think of definitions, synonyms and antonyms to explain the meaning of these words.
b) Explain the meaning of the words but don’t name them. Let your partner guess the words.

Exercise 13. a) Insert the prepositions from the list and read the text aloud. b) Read the text again and summarize the information from it (5-7 sentences). *(To summarize – to provide the most important information without details and examples).*

between; for (2); of; at (2); on (2); in.

Software Engineering vs. Computer Science

There is significant overlap _____ software engineering and computer science degree programs. Professionals in the two fields often compete _ the same positions. There are distinctions in their education, however, and in the future, there may be distinctions in what roles they are allowed to perform.

Software engineering is an engineering discipline. A licensing examination is under development; stakeholders believe that within two years, a number_states will be licensing those software developers whose work impacts safety and public welfare. Moreover, many people in the field believe that software development should be approached as an engineering discipline, even when it doesn't have a direct bearing on public safety. Computer Scientists test theories and work the edge of the unknown. Engineers start with knowledge that has already proven reliable.

Software development should follow the principles of engineering. Scientists research and extend scientific knowledge; they test theories and work the edge of the unknown. Engineers start with knowledge that has already proven reliable. Their job is to create designs that work. Engineers need breadth, or broad scope, to their education while scientists need depth in narrow branches of knowledge. It is better to have a degree in computer science than to work in software development without a related degree. Still, a traditional computer science degree, with its focus__the theoretical aspects of the profession, does not always provide the best education.

There is some confusion among future software developers about the distinctions between programs.

Some schools offer separate tracks in computer science and software engineering. The two tracks will generally include some common courses, but a different overall architecture. Software engineering programs include a design project; through this project, a future software engineer displays his or her ability to apply engineering principles in the real world. Students and graduates of computer science and software engineering programs cite a variety of differences. Courses computer science often place greater

emphasis __ the theoretical. They emphasize algorithms. Traditionally, they have been considered good preparation___ graduate level research.

(https://www.softwareengineerinsider.com/articles/computer-science-vs-software-engineering.html)

Exercise 14. Read the following text and give the main idea of it in English.

Цифровая экономика даст России шанс на рывок в будущее

По словам Владимира Путина, развитие цифровой экономики предстоит реализовать, "опираясь на накопленный технологический, интеллектуальный потенциал"

Российский лидер заметил, что, возможно, среди россиян есть те, кто задается вопросом: "Зачем вообще нужен переход к цифровой экономике, ведь у нас все есть: у нас есть нефть, газ, уголь, металлы, причем самые разные, и черные, и цветные, и золото, и платина, и алмазы?". В ходе заседания совета он объяснил, какое значение имеет цифровая экономика для России и каждого ее жителя.

Цифровая экономика станет основой для реформ в масштабах всей страны и затронет каждую компанию и каждого гражданина России, отметил Путин. "Цифровая экономика - это не отдельная отрасль, по сути это уклад жизни, новая основа для развития системы государственного управления, экономики, бизнеса, социальной сферы, всего общества. Формирование цифровой экономики - это вопрос национальной безопасности и независимости России, конкуренции отечественных компаний", - заявил он.

Согласно проекту программы развития цифровой экономики, к 2025 году 97% российских домохозяйств должны иметь широкополосный доступ в интернет (100 МБит/с). К этому времени во всех городах с населением от 1 млн. человек должны быть развернуты сети 5G. Планируется также, что до 2025 года в России появятся десять предприятий в сфере высоких технологий и столько же цифровых платформ для основных отраслей экономики, а вузы будут выпускать более 100 тыс. специалистов в сфере IT в год.

России для эффективного развития цифровой экономики нужно подготовить миллион IT-специалистов. По его словам, сейчас у России есть 500 тыс. программистов. Однако для цифровой экономики нужны специалисты с компетенциями в широком смысле слова: не только разработчики программного обеспечения, но высококвалифицированные специалисты в разных сферах, которые обладают "цифровыми" знаниями, навыками и опытом. При этом министр считает, что Российскую Федерацию нужно сделать

привлекательной юрисдикцией для разработчиков перспективных цифровых технологий.

Цепью по бюрократии: как блокчейн может сделать революцию в документообороте

По данным Института статистических исследований и экономики знаний НИУ ВШЭ, в России доля специалистов в сфере информационно-коммуникационных технологий в общей численности занятых не превышает 2%. В то же время в таких странах, как Финляндия, Швеция и Великобритании, этот показатель достигает 5-6%, отмечается в информационном бюллетене "Цифровые навыки населения".

Директор направления "Молодые профессионалы" Агентства стратегических инициатив Дмитрий Песков (отвечал за разработку раздела программы по кадрам и образованию) в ходе заседания совета отметил, что люди - это действительно ключевое ограничение программы. Однако для прорыва, по его мнению, не требуется просто миллион программистов, а только 120 тысяч высококвалифицированных программистов и инженеров. По словам Пескова, в будущем низкоквалифицированных программистов заменит искусственный интеллект, и этот момент "ни в коем случае нельзя проспать". Безработными специалисты, которых заменит искусственный интеллект, не останутся - в программе "Цифровая экономика" предусмотрено создание "системы цифровых ваучеров, которые направлены на получение нужных для цифровой экономики компетенций через онлайновую систему образования", пояснил представитель АСИ.

По словам президента, за последние годы Россия заметно продвинулась по многим направлениям цифрового развития. Так, по динамике распространения широкополосного доступа и беспроводных сетей РФ находится на уровне ведущих стран. При этом средняя скорость интернета в России в 2016 году увеличилась на 29%, что сопоставимо с Францией и Италией.

Путин также добавил, что к началу 2017 года российский рынок коммерческих центров хранения и обработки данных вырос до 14,5 млрд руб. По его словам, благодаря высокому уровню компетенций российских IT-специалистов компании РФ предлагают уникальные программные решения, которые используются в том числе и при создании "умных городов".

(http://tass.ru/ekonomika/4390974)

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ ВЕРИФИКАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

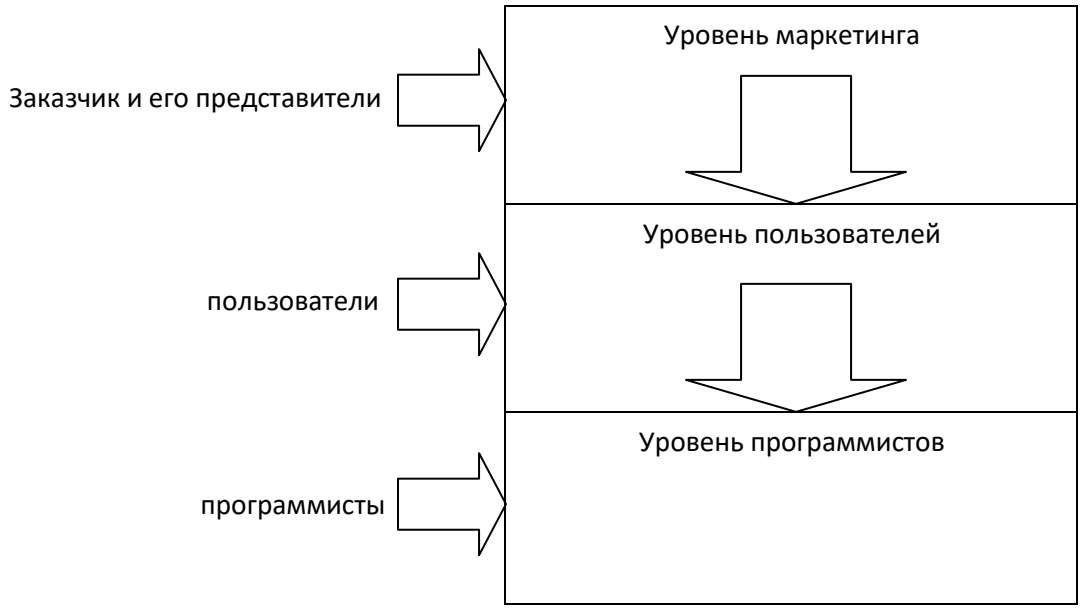
2021

Лабораторная работа номер 1- Тестирование требований

Тестирование программного обеспечения должно начинаться с самого раннего этапа – этапа формулировки требований. Основными проблемами требований являются:

- Некорректность
- Двусмысленность
- Неполнота
- Непроверяемость
- Несвязанность между требованиями разных уровней (нетрассируемость)
- Непонятность

Требования к программному обеспечению формулируются на трех уровнях. Самый верхний – уровень маркетинговых требований – содержит самые общие требования. Следующий уровень – уровень пользователей – содержит, обычно, описание бизнес-процессов. На самом низком уровне – уровне программистов – используются мелкие технические требования.



В данной работе все требования считаются сформулированными на уровне маркетинга.

Задание на лабораторную работу.

Вы – системный аналитик. Ваша задача написать техническое задание на предмет, маркетинговые требования к которому Вам были переданы в соответствии с вариантом (в рамках лабораторной работы делать этого не надо, нужно только решить, достаточно ли для этого информации). Для этого

Вы должны раскрыть требования маркетинга на уровень пользователей. В процессе этого у Вас могут возникать различные проблемы.

В лабораторной работе необходимо проанализировать требования заказчика, указать на проблемы в требованиях (каждую ошибку отнести к соответствующей категории) и скорректировать требования таким образом, чтобы в результате получился предмет указанного наименования. Для каждого пункта требований описать, каким образом будет производиться его проверка.

Указания.

Типовой проблемой при выполнении лабораторной работы является попытка придраться к каждому слову спецификации. На самом деле, предполагается моделирование реальной ситуации – Вам прислали требования заказчика и Вы делаете по ним техническое задание, по мере выполнения которого обнаруживаете разнообразные проблемы, препятствующие этому.

Заказчик никогда сам не пишет техническое задание! Когда в ответ на требование вида “разработать двухкамерный холодильник” Вы пишете замечания вида «неизвестно какого он должен быть размера, цвета и формы» Вы, тем самым, говорите заказчику, что он не сделал Вашу работу, потому что определить размер, цвет и форму, если они не указаны закахчиком, – задача аналитика.

Еще одной важной задачей является произвести на заказчика впечатление грамотного специалиста, а не человека, который не умеет делать свою работу.

Пример выполнения.

Спецификация на разработку стула.

1. Стул должен иметь четыре ножки и горизонтальную поверхность для сидения.
2. Стул должен иметь возможность регулирования высоты
3. Стул должен быть удобным.
4. Стул должен иметь высоту 60 сантиметров.
5. Стул должен весить не более 500 грамм
6. Стул должен быть легко перемещаем по помещению.
7. Стул не должен царапать паркет при перемещении
8. Стул должен использовать только нетоксичные материалы.

Рассмотрим требования по очереди.

Требование номер один интересно тем, что в нем упоминаются только ножки и поверхность для сидения. Стул, у которого нет спинки, называется “табурет”. Имеет смысл уточнить, имеется ли в самом деле в виду табурет или спинка была просто забыта при описании.

Требование номер два обычно используется для стульев, имеющих одну ножку, хотя принципиальных проблем с реализацией для четырехногого стула не имеется.

Требование номер три невозможно проверить, его можно скорректировать разнообразными способами, например, “дизайн стула утверждается заказчиком” в том смысле, что сперва будет утвержден дизайн и только после этого будет продолжена реализация.

Требование номер 4 явно противоречит требованию номер два, необходимо указать диапазон изменения высоты либо отказаться от требования номер два. Кроме того, возникает вопрос – является ли указанная высота высотой сиденья или спинки.

1 Спецификация на разработку холодильника

Необходимо разработать двухкамерный холодильник на базе системы андроид, отвечающий следующим требованиям:

- Холодильник двухкамерный
- При захлопывании дверцы она всегда обеспечивает плотное прижатие, вне зависимости от того, с какой силой было произведено это действие.
- Холодильник имеет интерфейс через сенсорный дисплей с локал изацией, поддерживающий следующие языки: Русский, Английский
- Управление температурой в холодильной и морозильной камерах осуществляется с дисплея.
- Когда дверца холодильника открыта, дисплей показывает предупреждающее сообщение и не разрешает управление температурой
- Когда дверца холодильника закрыта, дисплей отображает текущую температуру в холодильной и морозильной камерах.
- При изменении температуры на N градусов фактическая температура в камере должна измениться через N минут

2 Спецификация на разработку пылесоса

Необходимо разработать пылесос на базе системы андроид, отвечающий следующим требованиям:

- Пылесос способен убирать пыль и мелкий мусор
- Пылесос обеспечивает всасывание воздуха с мощностью 1600 Ватт
- Масса пылесоса в процессе работы не должна превышать 5 килограмм
- Пылесос может быть использован для сбора пыти на любых поверхностях и под любыми предметами мебели
- На пылесосе должна быть предусмотрена ручка
- Заряда пылесос должно хватать на 1 час работы

3 Спецификация на разработку стиральной машины

Необходимо разработать стиральную машину, отвечающую следующим требованиям:

- Стиральная машина должна уметь стирать белье.
- Минимальная загрузка должна составлять пять килограмм
- Стирка должна осуществляться в двух режимах – быстрая и полная, а также машина должна уметь осуществлять полоскание
- Стиральная машина должна подключаться к водопроводной трубе, сама закачивать воду, нагревать ее до нужной температуры, по окончании стирки – сливать
- У машины должен быть дисплей, демонстрирующий пользователю полезную информацию.
- Машина должна подключаться к WiFi

4 Спецификация на разработку микроволновой печи

Необходимо разработать микроволновую печь, отвечающую следующим требованиям:

- Микроволновая печь должна уметь нагревать продукты, а также осуществлять разморозку.
- Микроволновая печь должна иметь дверцу
- Микроволновая печь во включенном состоянии не должна производить опасное для человека излучение
- Нагревать продукты в микроволновой печи нужно при закрытой дверце
- Микроволновая печь должна позволять помещать вместе с продуктами металлические столовые приборы и посуду, при этом они не должны нагреваться или искрить
- Микроволновая печь должна иметь подсветку.
- Микроволновая печь не должна сушить продукты, которые в ней готовятся

5 Спецификация на разработку электрокофеварки

Необходимо разработать электрокофеварку, отвечающий следующим требованиям:

- Кофеварка должна уметь варить кофе
- Кофеварка должна иметь кнопку включения, ручку и крышку над емкостью для наливания воды
- Кофеварка должна работать от электричества
- Кнопка должна включаться, только если крышка закрыта
- Воду в кофеварку можно наливать, только если крышка закрыта
- Кофеварка должен поддерживать протокол HTCPCP (RFC 2324). Заказчик особенно настаивает на этом требовании и отказывается его убирать.
- Кофеварка должна быть красивой

6 Спецификация на разработку газовой плиты

Необходимо разработать газовую плиту, отвечающую следующим требованиям:

- Плита должна иметь четыре конфорки и духовку со стеклянной дверцей.
- Когда духовка открыта, газ подаваться не должен.
- На передней панели плиты должен быть расположен датчик температуры
- Духовка должна быть оборудована таймером, выключающим газ, и зуммером с настраиваемой мелодией, срабатывающим по завершении
- На плите можно готовить весь набор популярных блюд
- Плита должна исключать возможность возникновения пожара

7 Спецификация на разработку велосипеда

Необходимо разработать велосипед, отвечающий следующим требованиям:

- У велосипеда должно быть удобное, регулирующееся по высоте сиденье
- Велосипед должен поддерживать 16 скоростей
- Велосипед не должен сильно подпрыгивать на небольших бугорках
- Велосипед должен позволять перевозить двух пассажиров

8 Спецификация на разработку дырокола

Необходимо разработать дырокол, отвечающий следующим требованиям:

- Дырокол должен делать два отверстия в листе бумаги на стандартном для папок расстоянии
- Дырокол должен позволять вставить не менее 20 листов бумаги
- Габариты дырокола не должны превышать 20 сантиметров в ширину и 10 сантиметров в высоту
- Вес дырокола со вставленной бумагой не должен быть более 200 грамм
- Дырокол не должен иметь острых или выступающих краев, которыми можно пораниться или поранить другого
- Дырокол должен иметь информационное табло, отображающее актуальную информацию

9 Спецификация на разработку шкафа-купе

Необходимо разработать шкаф-купе, отвечающий следующим требованиям:

- Шкаф имеет три вертикальные секции и три двери
- Двери шкафа должны крепиться на горизонтальных направляющих
- Высота шкафа должна составлять 2 метра
- Двери шкафа должны быть созданы с таким расчетом, чтобы ими ничего нельзя было прищемить
- Глубина шкафа должна быть выбрана таким образом, чтобы в него помещался велосипед
-

A Задание повышенной сложности

Самостоятельно разработать спецификацию, содержащую осмысленные ошибки и описать их.

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ ВСТРАИВАЕМЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СРЕДСТВА

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

Содержание

Введение

ARM (Advanced RISC Machines – усовершенствованная RISC машина) – архитектура процессоров с сокращённой системой команд, предложенная британской компанией ARM. Эта компания не производит сама микросхемы, а разрабатывает и лицензирует собственные технологии третьим фирмам для производства микросхем. Множество известных фирм приобретает лицензии на использование разработанных ARM ядер и использует в своих разработках.

Ядра ARM имеют три линейки: M (microcontroller), R (real-time) и A (application). Ядра M предназначены для применения во встраиваемой технике, а ядра A и R – в более мощных устройствах с операционными системами типа Linux, WinCE.

Линейка ядер Cortex-M содержит 4 ядра:

Cortex-M0 и Cortex-M1 примерно одинаковые, рассчитаны на замену 8-ми битных контроллеров – обеспечивают минимальное энергопотребление, имеют минимальный набор команд, простую архитектуру контроллера (одна шина связи с памятью/периферией).

Cortex-M3 предназначен для замены обычных 16-ти и 32-х битных контроллеров, имеет расширенный набор команд, более развитую связь с периферией (3 шины)

Cortex-M4 предназначен для замены DSP (цифровой сигнальный процессор) контроллеров – имеет встроенный FPU (floating point unit – сопроцессор, выполняющий операции с плавающей точкой), команды обработки данных с накоплением, может работать с арифметикой «с насыщением»

Фирма STMicroelectronics по лицензии выпускает контроллеры STM32F30x с использованием ядра Cortex-M4, для изучения работы с которыми предлагаются данные методические указания. Российская фирма МИЛАНДР также приобрела права на использование микроконтроллерного ядра фирмы ARM, на основе которого разработала серию микроконтроллеров 1986BE9x с использованием ядра Cortex-M3, со схожими с микросхемой STM32F30x свойствами. Так что методические указания могут быть использованы и при изучении особенностей работы и с российскими микроконтроллерами. В процессе изучения используется среда проектирования Keil μVision и платы STM32F3DISCOVERY. Предусмотрено подключение к плате дополнительных модулей периферии: клавиатуры, индикации, датчиков. В рамках лабораторных работ предполагается знакомство со схемотехникой простейших устройств на основе микроконтроллера STM32F30x и особенностями его программирования. Предполагается применение методических указаний при выполнении лабораторных работ по нескольким дисциплинам: «Проектирование встраиваемых систем», Проектирование встраиваемых систем на микроконтроллерах» и «Методы и средства отладки встроенных систем на микроконтроллерах».

Общие сведения о микроконтроллерах STM32F30x

Архитектура ARM контроллеров

Ядра ARM Cortex-M построены по классической RISC архитектуре: блок из 12 регистров общего назначения, указатель стека, указатель команд. При изучении материала предлагается пользоваться различными руководящими указаниями, собранными в папке «Datasheet». Ядро имеет (рис. 1) встроенный контроллер прерываний (NVIC), блок отладки (по JTAG или SWD) (Debug access port) и матрицу шин (Bus matrix). Более сложные ядра могут иметь также блок защиты памяти (позволяет задавать и контролировать использование блоков памяти – например, защитить от записи), и модуль FPU – математического сопроцессора.

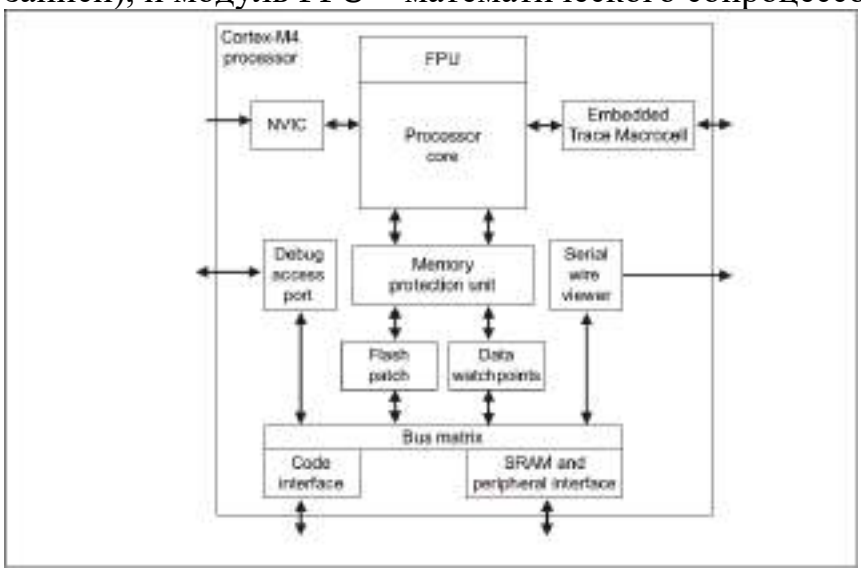


Рисунок 1. – Ядро ARM контроллера

Адресное пространство памяти программ, памяти данных и регистров периферийных устройств общее с объёмом 4ГБ. Таким образом, работа с периферийными устройствами осуществляется так же, как и с памятью данных.

Тактирование ядра и периферии производится либо от внешнего тактового генератора на кварцевом резонаторе, либо от внутреннего RC генератора. И в том и в другом случае схема тактирования позволяет умножить частоту генератора на степень двойки специальной схемой PLL и затем поделить на любое число для получения требуемой частоты. Такой способ тактирования используется для ядра, контроллера USB (если есть), контроллера UART и некоторой другой периферии. Основная часть периферии подключается к тактовой частоте ядра с выбранным делителем.

Ядро имеет **контроллер прерываний-исключений**. Первые слова памяти программ хранят вектора прерываний и исключений. Большинство из них являются маскируемыми и могут иметь различный приоритет. Однако есть ряд немаскируемых прерываний (исключений), одно из которых – сброс.

Также к ядру относится **таймер SysTick,** позволяющий генерировать временные интервалы путём подсчёта импульсов тактовой частоты ядра. Для данного таймера задаётся начальное значение, от которого он считает вниз.

По достижении нуля значение загружается повторно и генерируется прерывание (или устанавливается флаг).

Периферийные модули контроллера подключаются к матрице шин и адресуются как обычная память. При этом для использования какого-либо модуля необходимо подключить его к матрице шин (к тактированию), подключить его к схеме тактирования (для модулей, требующих этого), и разрешить его работу (для большинства модулей).

Подключение к матрице шин делает доступным регистры модуля, подключение к тактированию, как правило, требуется для модулей, работающих с какой-либо частотой (АЦП, таймеры, UART). При этом частота тактирования модуля может отличаться от частоты ядра и от частоты шины.

Программирование контроллера возможно либо через интерфейс отладки (если программа не блокирует данную возможность и не «завешивает» ядро контроллера), либо через UART или USB порты. В последнем случае используется специальный загрузчик, постоянно прошитый в памяти контроллера. Для запуска загрузчика нужно подать на определенную ножку контроллера (BOOT0) определенный уровень. После сброса управление в этом случае получит загрузчик, который не может быть поврежден пользователем, и даже при ошибочно написанной программе позволит стереть и запрограммировать память контроллера.

Для ускорения разработки фирма ARM предоставляет библиотеку CMSIS – написанную на Си библиотеку, дающую стандартный доступ к ресурсам ядра контроллера. Внутри данной библиотеки описана и стандартизирована библиотека драйверов периферийных устройств (SDL), также написанная на Си. Практически, при использовании данных библиотек программист не видит и не вникает в тонкости программирования ядра, за исключением программирования периферии. В практической части занятий будет использоваться именно эта библиотека. Одна из причин – легкость в начальном изучении других ARM-CortexM контроллеров. Например, в настоящее время актуальным является использование контроллера 1986BE93 (BE9x). Данный контроллер построен на ядре ARM Cortex-M3. Рассматриваться будет контроллер STM32F303, построенный на ядре ARM Cortex-M4. В основном приемы программирования на языке Си одинаковы для обоих контроллеров, контроллеры 1986BE93 также имеют библиотеку CMSIS.

Необходимо помнить, что использование чистого языка С не всегда даёт доступ ко всем возможностям контроллера – особенно это касается DSP расширений и арифметики с насыщением. Для использования этих возможностей предоставляются специализированные библиотеки.

Библиотека CMSIS

Библиотека содержит две основные части – собственно CMSIS и драйвера периферии SDL.

CMSIS включает в себя файлы, описывающие ядра (каталог CMSIS\Include), общие для многих контроллеров, и файл, описывающий конкретный контроллер (Device\ST\STF32F30x\Include\stm32f30x.h).

В этих файлах следует искать функции для работы с контроллером прерываний, системным таймером, блоком отладки. Также в этих файлах определены структуры, описывающие периферию, и переменные, позволяющие обращаться к периферийным устройствам. Периферийные устройства контроллера сделаны таким образом, что все относящиеся к устройству регистры находятся в адресном пространстве рядом, и для каждого периферийного устройства может быть задан базовый адрес. Поэтому удобно определить интерфейс к регистрам периферии как структуру, и объявить экземпляр структуры по конкретному адресу.

Так как в основном эта часть CMSIS описывает данные, а описываемые функции очень просты и сделаны встраиваемыми, хватает только заголовочных файлов.

Для конфигурации файлов используется:

1) определение USE_STDPERIPH_DRIVER, которое может быть пояснено в файле stm32f30x.h или внесено в проект вручную. Это определение говорит о том, что нужно подключать заголовочные файлы SDL.
2) Определение HSE_VALUE – определяет частоту кварца. Данное определение нужно для правильного расчёта текущей тактовой частоты контроллера служебными функциями. Его можно определить в проекте, либо оно будет задано величиною в 8МГц.
3) Ряд других определений, в том числе определение LSE_VALUE – определяет различные частоты, в основном менять их не нужно.

Также CMSIS имеет расширенные возможности, такие, как работа с DSP, RTOS, на данный момент они не рассматриваются.

Драйверы периферии SDL хранятся в каталоге STM32F30x_StdPeriph_Driver. Для каждого периферийного блока имеется заголовочный файл в каталоге STM32F30x_StdPeriph_Driver\inc и файл кода в каталоге STM32F30x_StdPeriph_Driver\src. Файл кода необходимо подключать к проекту вручную, все заголовочные файлы подключаются к файлу Device\ST\STF32F30x\Include\stm32f30x.h, если было определено USE_STDPERIPH_DRIVER. Узнать, какой файл нужно подключить для конкретной периферии, можно по окончанию имени файла. Оно однозначно определяет описываемый периферийный блок (Таблица 1).

Таблица 1

№	Файл	Описание
1	stm32f30x_adc	АЦП
2	stm32f30x_can	Контроллер шины CAN

3	stm32f30x_comp	Управление аналоговыми компараторами
4	stm32f30x_crc	Аппаратный расчет CRC с помощью задаваемого полинома
5	stm32f30x_dac	ЦАП
6	stm32f30x_dma	Контроллер прямого доступа к памяти. Используется в основном с другими периферийными блоками.
7	stm32f30x_exti	Контроллер внешних входов прерываний
8	stm32f30x_flash	Управление встроенной FLASH памятью, кроме функций записи/чтения/стирания содержит полезные функции настройки задержек (в зависимости от частоты контроллера), и ряд функций работы с «байтами настройки», позволяющими задать начальное состояние некоторых узлов контроллера и ядра при включении питания.
9	stm32f30x_gpio	Порты ввода-вывода. Обязательно подключать, т.к. только тут можно отконфигурировать выводы контроллера.
10	stm32f30x_i2c	Шина I2C
11	stm32f30x_iwdg	Сторожевой таймер (обычный)
12	stm32f30x_misc	Работа с прерываниями и таймером SysTick на уровне SDL (чуть проще, чем через саму CMSIS)
13	stm32f30x_opamp	Управление встроенными операционными усилителями
14	stm32f30x_pwr	Работа с режимами питания – спящий режим, контроль питания.
15	stm32f30x_rcc	Схема тактирования. Обязательно подключать, так как она позволяет подключать к шине другие периферийные блоки. Тут же находятся функции включения тактирования отдельных блоков, требующих этого.
16	stm32f30x_rtc	Часы реального времени
17	stm32f30x_spi	Шина SPI
18	stm32f30x_syscfg	Управление конфигурацией системы. Может влиять на разные устройства, в том числе: I2C, DMA, подключение контроллера прерываний к входам контроллера, и т.д.
19	stm32f30x_tim	Работа с таймерами (кроме SysTick)
20	stm32f30x_usart	UART – последовательный интерфейс (несколько штук)
21	stm32f30x_wwdg	«Оконный» сторожевой таймер – позволяет

		ограничить минимальную и максимальную частоту сброса сторожевого таймера

Порты ввода-вывода, конфигурация выводов контроллера

Шестнадцатиразрядные порты ввода-вывода, количество которых зависит от числа ножек конкретного контроллера, подключаются к выводам контроллера. Упрощённая схема одного разряда порта предложена на рисунке 2. Эта схема показывает, что каждая ножка контроллера может выполнять функцию входа или выхода разряда у порта PORTx, функцию аналогового входа АЦП, операционного усилителя или другую альтернативную функцию (до 16 разных).

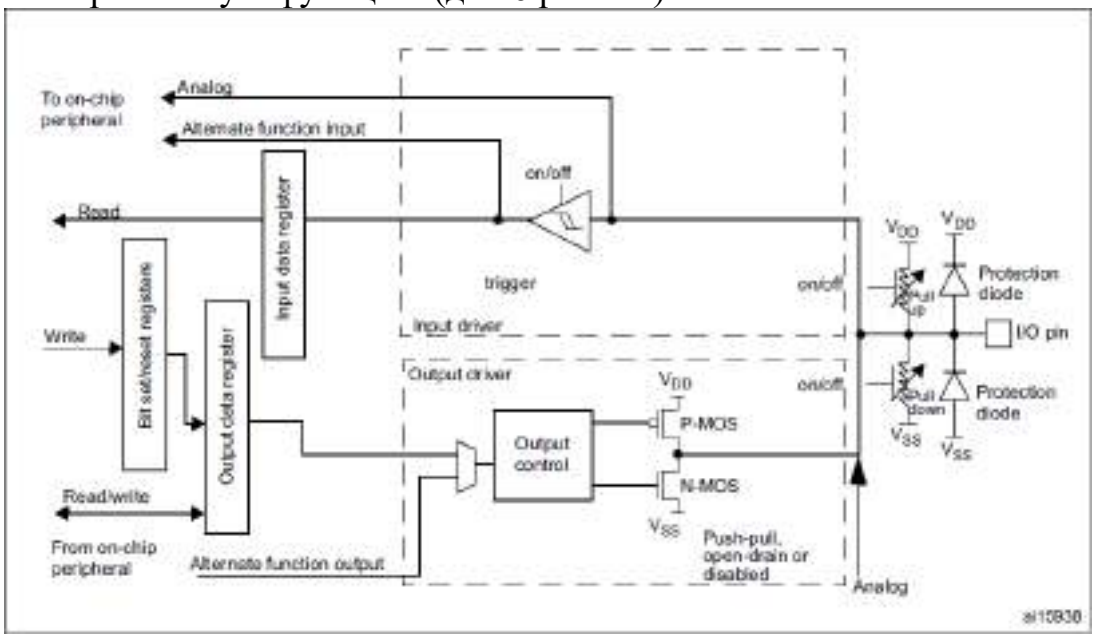


Рисунок 2. – Схема вывода порта

При работе в основном режиме ножка PORTx_Pin_y подключена к порту ввода-вывода GPIOx_pin_y. Она может работать как вход или как выход. Альтернативные и аналоговые функции при этом отключены.

При работе в альтернативной функции вывод подключен к соответствующему модулю.

После подачи питания все выводы подключены к альтернативной функции AF0.

Независимо от выбранной функции каждая ножка может быть запрограммирована на разные скорости работы (меняется скорость нарастания сигнала), на работу в режиме «открытый коллектор или исток» (open-drane) или «полный порт» (push-pull) и к ней могут быть подключены внутренние резисторы подтяжки к положительной (pull up) или отрицательной (pull done) линии питания. Включение данных функций нужно производить в соответствии со здравым смыслом. Например, резистор подтяжки на аналоговой ножке внесёт дополнительную ошибку, но не будет отключен автоматически.

Скорость работы определяет максимальную скорость переключения ножки: 2, 10 и 50 МГц. Рекомендуется по возможности выбирать меньшую

скорость, т.к. это уменьшит потребление и уменьшит вероятность выхода ножки из строя при емкостной нагрузке из-за большого тока.

Основы языка Си

Поскольку при составлении программ для контроллеров широко используется язык Си, обратим внимание на некоторые моменты этого языка с учётом применения к контроллеру:

- Оформление процедур
- Объявление переменных, массивов, указателей. Основные типы данных. Модификаторы const, volatile.
- Выполнение вычислений, автоматическое приведение типов.
- Вызов процедур, передача и возврат параметров. Возврат параметров через указатели. Операция return.
- Условия if then else, выбор switch, команда прерывания break.
- Циклы со счётом for(), с предусловием while () {}, с постусловием do {}while().

Основные типы данных и допустимые с ними операции

Единственной задачей программы является обработка данных – данные, их правильное представление и обработка - это самое основное в программе.

Рассмотрим основные типы данных языка Си, применительно к 32-х битным системам.

Все типы данных можно разбить на несколько групп:

- целочисленные типы
- Числовые вещественные типы
- Указатели
- Массивы
- Перечисления
- Структуры
- Объединения

Первые два типа являются «основными» - все остальные типы так или иначе используют основные типы в своем определении.

Здесь намеренно опущен «символьный» тип – его можно отнести к целочисленным типам, к нему применимы все те же самые операции и преобразования.

Перечисления, структуры и объединения рассматриваться не будут. Коротко о них:

Переменные перечислимого типа (enum) могут содержать одно из множества указанных при объявлении значений. Такие типы широко используются для организации сложных ветвлений, однако легко могут быть заменены на любой целочисленный тип, что будет использоваться в нашей работе.

Структуры (struct) позволяют создать составной тип, содержащий поля разных типов, хранящиеся под одним именем. Вещь очень важная и полезная

– позволяет улучшить процесс обработки данных, сделать его более наглядным. Рекомендуется для дальнейшего изучения.

Объединения (union) позволяют создать составной тип, все поля которого отображаются на одну и ту же память. Может быть применено для экономии памяти или для приведения одних структур в другие.

Типы данных используются не только для объявления переменных, но и для объявления параметров функций. Однако объявление переменных – самая распространенная операция, все примеры будут приводиться для объявления переменных.

Объявление переменных

Шаблон для объявления переменных:

<модификаторы> <тип> <имя переменной>;

<модификаторы> <тип> <имя переменной>=<значение>;

Здесь модификаторы и тип – предопределенные слова, а имя переменной может быть любым, начинающимся с символа.

Основные модификаторы:

signed, unsigned – явно определяют наличие или отсутствие знакового бита у целочисленных переменных.

const – определяет, что объявленная переменная является константой и больше не может быть изменена.

volatile – определяет, что работа с переменной не оптимизируется, то есть переменная всегда хранится в основной памяти. Это используется например при работе с переменной как внутри программы так и в прерываниях, поскольку без данного определения компилятор может оптимизировать работу и сохранить данную переменную в регистрах ядра, что приведет к неверной работе. Однако, данное определение не является определением «критической секции» - работа с переменной всё равно производится побайтно, если прерывание произойдет в процессе обработки переменной, и в прерывании будет изменено её значение – может произойти потеря данных.

static – определяет, что переменная создаётся только один раз и сохраняет своё местоположение всё время работы программы.

register – определяет, что переменную следует хранить в регистре ядра. Не рекомендуется для использования.

extern – определяет, что указанное объявление лишь ссылка на объявленную в другом файле переменную. В этом случае после слова extern должна идти полная копия определения переменной.

Целочисленные типы данных

Переменные таких типов обозначают области памяти, содержащие 1, 2, 4 или 8 байт, работа с которыми осуществляется как с одним целым числом.

По умолчанию считается, что переменная является знаковой, и данные представлены в дополнительном коде. Для гарантии может быть применен модификатор signed, в основном есть смысл объявлять явно наличие знака

при использовании типа char – не во всех компиляторах по умолчанию данный тип является знаковым.

Если применяется модификатор **unsigned** то данные считаются беззнаковыми, и хранятся в прямом коде, то есть старший разряд не является знаковым.

Модификаторы signed и unsigned не рекомендуется применять с типами uint16_t, int16_t, uint32_t, int32_t, где первая буква указывает на знаковость определяемой переменной. Данные типы появились позже основного стандарта, но широко применяются в программировании ARM контроллеров, так как позволяют избежать проблем с различной разрядностью.

Перечислим основные типы данных, с указанием числа используемых битов:

Тип	Разрядность		Комментарий
char	8		«Символьный» тип – в основном предполагается работа с данными как с символом, а при работе с массивом – как со строкой.
Int	16	32	Разная разрядность для 8-ми и 16-ти, и для 32-х и 64-х битных систем
Long	32		
int16_t	16		
uint16_t	16		Беззнаковое
int32_t	32		
uint32_t	32		Беззнаковое

Примеры объявлений:

char c; - переменная, занимающая 1 байт

unsigned int num; - переменная, занимающая 4 байта, беззнаковая

int32_t counter; - переменная, занимающая 4 байта, знаковая

Основные операции, допустимые с целочисленными данными:

Операции	Операция
+ - / *	Основные арифметические операции
%	Результат – остаток от деления первого операнда на второй операнд. Так, 13%5 = 3
++	Увеличение значения на единицу
--	Уменьшение значения на единицу
|	Логическое побитовое сложение
&	Логическое побитовое умножение
^	Логическое побитовое «исключающее или»
~	Побитовая инверсия
<<	Арифметический сдвиг первого операнда на несколько бит влево. Число бит сдвига определяется вторым операндом.

>>	Арифметический сдвиг первого операнда на несколько бит вправо. Число бит сдвига определяется вторым операндом.

Также доступны логические операции над переменными, рассматриваемыми как «истина» или «ложь». В этом случае если число равно нулю, его значение рассматривается как «ложь», если не нулю – как «истина». Побитовый анализ не производится.

Операции	Операция
||	Логическое сложение
&&	Логическое умножение
^^	«исключающее или»
!	Инверсия. Результат – 0 или 1

Задание целочисленных констант:
int a=12345; - десятичное число
int b=0x4A7B; - шестнадцатеричное число

Вещественные типы данных

Позволяют работать с числами с плавающей запятой. Всегда знаковые.

Тип	Разрядность	Комментарий
float	32	
double	64	

Основные операции - +, -, *, /, операция сравнения.

Важно помнить, что операция сравнения даст «единицу», или «истину», только при полном побитовом равенстве, что далеко не всегда выполнимо.

Задание вещественных констант:
float a=3.14159;
Важно помнить и понимать, что следующая строчка:
float a=4;
Не вызовет проблем, но на самом деле будет создана целочисленная константа 4, которая затем будет приведена к вещественному типу.

Указатели

Указатели используются для хранения ссылки на какой-либо тип данных. Для определения указателя к типу данных добавляется символ «*». Для хранения указателей в ARM контроллерах всегда выделяется 4 байта.

Пример:
int * pnt; - объявляет указатель pnt на переменную типа int.
Особенно нужно выделить объявления нетипизированного указателя:

void * <имя>

Данный указатель также содержит 4 байта, но размер типа данных не определен. С ним не допустимы операции «+» и «-».

Операции с указателями:

Операции	Операция
&	Разименование – возвращает указатель на переменную, например: a = &count – присваивает указателю a адрес переменной count.
*	Доступ к данным по указателю, например: *a = 5 присваивает данным, по указателю a, значение 5.
++	Перемещение указателя на следующий элемент данных.
--	Перемещение указателя на предыдущий элемент данных.
+	Перемещение указателя на несколько элементов данных вперед.
-	Перемещение указателя на несколько элементов данных назад.

Массивы

Массивы используются для хранения последовательности элементов. Для объявления массива к названию переменной добавляется две квадратные скобки, внутри которых написано число элементов массива:

int a[10];

Возможно объявление массива массивов, например:

int b[10][10];

определяет массив из 10 строк по 10 элементов каждый.

Имя массива в общем случае является указателем на первый элемент массива.

Для выбора одного из элементов массива также используются квадратные скобки:

a[4]=b[4][3];

Элементы в любых массивах нумеруются с нуля. Таким образом, для массива int a[10]; допустимыми будут индексы a[0]..a[9]. Фиксация выхода за границы массива не производится, что нужно помнить при написании программы - могут быть повреждены случайные данные при выходе за границы массива.

Задание массивов:

int a[10]={1,2,3,4,5,6,7,8,9,10};

Отдельно рассмотрим строковый массив:

char string[]=”УРА!!!”;

создает массив string длиной 7 символов – 6 значащих и завершающий ноль.

Арифметические и логические выражения

Выше уже были приведены специфичные для разных типов данных операции. Ещё некоторые операции, выполнение которых возможно с любыми типами данных:

Операции	Операция
=	Присваивает первому операнду значение второго операнда
==	Сравнивает два операнда, возвращает не ноль, если они равны
!=	Сравнивает два операнда, возвращает ноль, если они равны
<=, >=, >, <	Операции сравнения. Возвращают не ноль в соответствующих случаях.

Арифметические и логические выражения строятся как в большинстве языков программирования:

A = B + C / (D-5)

Однако, важно понимать, что оператор присвоения ‘=’ – такой же оператор, может быть выполнен в любом месте выражения. Его даже может не быть (результат выражения в этом случае может быть потерян, ошибки или предупреждения выведено не будет):

A = (B=5) + 6;

после этой строчки B присваивается значение 5, а A – 11.

В выражении можно использовать вызов функции - <Имя функции>(<парметры>) – при вычислении будут рассчитаны параметры функции, вызвана функция, и результат её выполнения использован в выражении.

Приоритет выполнения операций

Есть множество мест, где можно посмотреть сравнительный приоритет операций, однако рекомендуется лишний раз определить приоритет скобками, в явном виде. Укрупненная таблица приоритетов, в порядке понижения приоритета:

Операции	Описание
[], (), ., ->	Взятие индекса, вызов функции, выбор данных из структуры
(<тип>)	Приведение типов
*, /, %	Арифметические операции типа «умножение»
-, +, <<, >>	Арифметические операции типа «сложение», «вычитание»
<, >, ==, !=	Операции сравнения
&, |, ^, &&, ||	Логические операции

=	Присвоение

Особые операции

Особой строкой стоят операции “++” и “—“ – это одноместные операции, как правило полностью заменяемые на команды процессора inc и dec переменной. Выполнение данных операций производится либо **до** вычисления выражения, если они написаны перед операндом, либо **после** выполнения, если за операндом:

```
int a=5;
int b;
b=(a++)+5;  // Результат – 10
b=(++a)+5;  // Результат – 11
```

Данные операции очень распространены, так как быстро выполняются и позволяют реализовывать перебор и простые счетчики.

Интересна следующая операция

<условие>?<операнд1>:<операнд2>

Результатом этой трехместной операции является операнд1, если условие истинно, и операнд2, если ложно.

Также, для ускорения вычислений предусмотрены «операции с накоплением»:

+=, -=, \=, *=, >>=, <<=

Синтаксис таких операций, например:

<Операнд1>+=<Операнд2>

Эта операция равносильна записи:

<Операнд1>=<Операнд1>+<Операнд2>

Однако, на многих процессорах, операция += выполнится быстрее, так как значение операнда1 не будет извлекаться из памяти.

Приведение типов данных

Язык Си автоматически приводит типы данных при разборе выражений. Это нужно учитывать при написании программы, особенно при выполнении операций с целыми числами.

Неявное приведение типов, в котором не происходит потеря данных по мнению компилятора, осуществляется без предупреждений. Если происходит явная потеря данных, компилятор выдаёт предупреждение.

Неявно без проблем выполняются следующие преобразования:

Целое – в вещественное.

Знаковое – в беззнаковое.

Тип с меньшей разрядностью – в тип с большей разрядностью.

Массив – в указатель.

Неявно выполняется, но с предупреждениями:

Вещественное – в целое (отбрасыванием всего после запятой).

Знаковое – в беззнаковое.\

Явное приведение выполняется в гораздо более широких рамках. Для явного приведения необходимо перед приводимым значением указать требуемый тип в скобках:

```
int a;
float b;
b=5.5;
a=b;  - данная строчка вызовет предупреждение «потеря точности»
a=(int)b; - пройдёт без предупреждений
```

Результат в обеих случаях будет равен 5.

Важно понимать, что неявное приведение выполняется только при необходимости, логику выражения компилятор не понимает. Следовательно, результат следующей программы:

```
int a=33;
float b;
b=a/6;
```

В результате b будет равно 5, так как деление будет выполнено в целочисленной арифметике.

Для верного результата можно написать:

```
b=a/6.0;
```

6.0 в данном случае определяет число с плавающей запятой, и приведение будет выполнено не к целому типу, а к числу с плавающей запятой.

Структура программы

С точки зрения языка си, программа представляет собой ряд арифметических выражений, для которых вычисляется результат. Результат может быть использован, или потерян – это не важно. Окончание выражения или оператора определяется символом «;». Вся программа состоит из функций. Большинство базовых операций, такие как вывод и ввод символа, вычисление синуса – такие же функции, но описанные в стандартных библиотеках.

Кроме того, определены несколько базовых операторов:

```
условие if (..) .. else ..;
переключатель switch (..) ..;
оператор прерывания break;
цикл с перебором for (;;) ..;
цикл с предусловием while (..) ..;
цикл с постусловием do .. while(..);
продолжение continue;
выход из функции return ..;
переход goto …;
```

Рассмотрим кратко каждый из операторов

if (<условие>**)** <выражение1> **else** <выражение2>**;**

Выполняет вычисление условия, и затем выполняет выражение1 если условие истинно, и выражение2 – если ложно.

Допускает сокращение - **if (**<условие>**)** <выражение1>**;** В этом случае выражение1 выполнится, только если условие даёт не нулевой результат.

```
switch (<выражение>) {
        case (<значение1>):
                …секция1;
        case (<значение2>):
                …секция2;
        default:
                …секция3;
};
```

Переключатель выполняет вычисление выражения, и переход к одной из меток **case()** в зависимости от значения выражения. Требуемое значение пишется в скобочках после метки, также доступна метка **default** для обозначения точки, в которую выполнение передаётся, если ни одна метка **case** не удовлетворяет условию.

Нужно понимать, что совершается именно переход внутрь текста программы. Таким образом, после перехода например на секцию2, будет выполнена и секция3. Чтобы этого не происходило, используют оператор прерывания **break**.

break;

Этот оператор вызывает немедленный выход из операторов цикла и оператора переключателя switch.

for (<выражение1>**;**<условие>**;**<выражение2>**)** <выражение3>**;**

Оператор цикла со счетчиком на самом деле выполняет следующие операции:

1. Перед началом цикла выполняет выражение1.
2. Выполняет выражение3.
3. Выполняет выражение2.
4. Вычисляет условие. Если условие истинно, возвращается на пункт2 – выполнение выражения3.

Обычное использование данного цикла – цикл со счетчиком:

```
int a;
for (a=0;a<10;a++) …;
```

выполнится ровно 10 раз, если конечно в теле цикла не менять значение a.

Любое из выражений и условий данного оператора может быть пустым. Условие в этом случае считается истинным. Таким образом, строка

```
for (;;) ;
```

допустима и приведет к зависанию программы – будет бесконечно выполняться пустой цикл.

while (<условие>**)** <выражение1>**;**

Цикл с предусловием вычисляет значение условия, и если результат не нулевой – выполняет выражение1 и возвращается к подсчету условия.

do <выражение1> **while (**<условие>**);**

Цикл с постусловием выполняет выражение, затем вычисляет значение условия, и если результат не нулевой возвращается к выполнению выражения.

continue;

Данная операция позволяет пропустить всё тело цикла, перейдя сразу в конец цикла. Дальнейшие действия зависят от типа цикла.

return <выражение>;

Вычисляет выражение и возвращает полученный результат как результат выполнения функции.

goto <метка>;

Выполняет безусловный переход по метке. Использование данной операции не рекомендуется, так как сложно проконтролировать, куда будет совершен переход и является ли он безопасным. Так, можно совершить переход из одного составного оператора в другой составной оператор, что приведет к ошибке, если использовались локальные для одного из операторов переменные.

Составной оператор

Составной оператор - часть программы, считающаяся для компилятора одной командой. Составной оператор начинается с символа «{» и заканчивается символом «}».

Все выражения, написанные внутри составного оператора, выполняются вместе, поочередно, и считаются одним выражением.

Составной оператор широко применяется вместе с базовыми операторами. Так, если в условии требуется выполнить только одно выражение, оно может быть написано само по себе:

```
if (a==b) c=d;
```

Если же требуется выполнить несколько действий, требуется использовать составной оператор:

```
if (a<b) {
        c=a;
        a=b;
        b=c;
};
```

Есть смысл использовать составной оператор и сам по себе – это позволяет объявить область видимости переменных (см. далее).

Допустимо многократное вкладывание операторов:

```
if (…) {
        if (……) {
        }
} else {
        for (;;) {
        }
}
```

Оформление функции и общая структура функций.

Основным элементом программы в Си является функция. Функция – это именованная часть программы, выдающая определенный результат по заданным входным параметрам.

Общая структура определения функции:

<тип результата> <имя функции>(<тип аргумента1> <аргумент1>, <тип аргумента2> <аргумент2> , …) <Составной оператор>

Типы результата и аргументов – такие же как у переменных. С точки зрения текста программы внутри функции аргументы – это такие же переменные, переданные в функцию по значению (т.е. передаются копии параметров. При необходимости передать аргумент в явной форме, чтобы была возможность изменить его в функции – нужно передать указатель на него). Аргументы могут быть изменены внутри функции – на данных, доступных вне функции, это не отразится.

Возможно также указание void вместо возвращаемого параметра или вместо списка аргументов. В этом случае соответственно функция не возвращает значений или не получает значений. Также, допустимо оставлять список аргументов пустым, но некоторые компиляторы в этом случае выдают предупреждение.

Для возврата результата функции используется специальный оператор return <результат>. Если функция не возвращает значения, оператор используется без параметра. Если функция выполнилась до конца, т.е. выполнился весь составной оператор, а функции return не было, также выполняется выход из функции, при этом если ожидается, что функция возвращает результат – выдаётся предупреждение.

Функция становится доступной только ниже своего объявления. Чтобы избежать этого, а также для представления функции в других файлах, используют прототип функции – полное определение функции, где вместо составного оператора сразу за закрывающейся скобкой стоит точка с запятой. Также прототип функции может не содержать имен аргументов, только их типы. Объявление прототипа не обязательно.

Пример:

```
// Прототип функции
```

```
int sum(int,int);

// Описание функции
int sum(int a,int b) {
    return a+b;
}
```

Варианты передачи параметров

Между функцией и вызывающей программой осуществляется обмен в двух направлениях: передача аргументов функции и возврат результата работы функции.

Передача аргументов может осуществляться с помощью аргументов функции, либо с использованием глобальных переменных.

Во втором случае функция использует в своей работе значения некоторых глобальных переменных, заданные основной программой. Такое решение является некрасивым, поскольку усложняет анализ программы и ухудшает её универсальность – невозможно использовать функцию, не задав нужное значение данных глобальных переменных. Однако, такое решение оправдано, если необходимо сэкономить память или быстродействие – в этом случае не тратится время и память на копирование аргументов, ведь их передача ведется по значению. Также, такой подход оправдан при написании рекуррентных функций.

В первом случае, при использовании аргументов, также возможны два подхода – передача по значению и передача по ссылке.

Передача по значению осуществляется по умолчанию для всех базовых типов и для указателей. Однако структуры, массивы и объединения передаются по ссылке, то есть их изменение приводит к изменению данных основной программы, а не копии данных.

Передача по ссылке используется, если функция должна изменить какие-либо данные в основной программе. В этом случае передаётся не само значение, а указатель на него – соответственно, функция может обратиться к данным по указателю и изменить их.

Возвращать параметры функция также может двумя способами – либо оператором **return**, либо изменив значение параметров основной программы, как описано выше.

Создание переменных

Области видимости, глобальные переменные

Переменные в языке Си могут быть созданы как глобальные или как локальные переменные.

Глобальные переменные описываются в тексте программы вне каких-либо функций. Память под такие переменные выделяется при запуске программы, если переменные инициализированы – задание начальных значений происходит также при запуске программы. Имена таких переменных доступны в любой точке файла, ниже объявления глобальной

переменной, но могут быть перекрыты локальным именем переменной. Также, использование extern позволяет сделать глобальные переменные доступными и в других файлах.

Локальные переменные объявляются внутри составного оператора, например, внутри функции, после открывающейся фигурной скобки. Для локальных переменных память выделяется в стеке и адресуется относительно указателя стека. Имена таких переменных доступны внутри составного оператора, где они описаны, и во всех вложенных в него составных операторов. При завершении составного оператора память, выделенная под переменные, освобождается.

Отдельно стоят локальные переменные, объявленные как **static**. Память под них также выделяется при запуске программы, но инициализация производится при первом входе в блок, в котором описана переменная. Память данных переменных никогда не освобождается, но доступны они только внутри того блока, где описаны.

Примеры:

```
int a;                  // Глобальная переменная a

int fun2(int b) {
        int a;          // Локальная переменная a, первая
        a=5;            // Обращение к локальной переменной a, первой
}

int fun(int b) {
        int a;          // Локальная переменная a, вторая
        a++;            // Обращение к локальной переменной a, второй
        fun2(a);        // вызов функции, внутри которой тоже есть переменная a
}
```

Базовые директивы препроцессора

Перед компиляцией текстовый файл, который представляет собой программа, обрабатывает препроцессор. Его задача – преобразовать текст, преобразовав ряд директив в соответствующий текст, понятный компилятору.

Базовые директивы

```
//
```

Двойной слеш – символ комментария. Двойной слеш показывает, что всё далее написанное в этой строчке должно быть отброшено

```
/*
…
```

*/

Данная комбинация – блок комментария. Всё, между /* и */ - считается комментарием и отбрасывается из текста программы.

#include <имя файла>

Полностью включает текст из указанного файла в текст программы. Имя файла должно быть указано в двойных кавычках или в угловых скобках. Способ указания влияет на путь поиска файла, сейчас это почти не важно.

#define <имя> <значение>

Текстовое определение. Приводит к тому, что далее по тексту все лексемы, равные <имя>, будут заменены на <значение>

#ifdef <имя>

…

#endif

Условный оператор – если <имя> где-то определено, то текст между #ifdef и #endif включается в текст для компилятора. Иначе – отбрасывается как ненужный.

Пример программы

```
#include <stdio.h>

// Функция посимвольно выводит строку с переданного
// адреса, до момента, когда в строке встретится ‘0’
void puts(char * p) {
  while (*p)
    putch(*(p++));
}


void OutNumber(int zn) {
  char buf[20];
  char bpos=19;
  buf[bpos--]=0;
  do {
    char cz;
    cz=zn%10;
    zn/=10;
    buf[bpos--]=cz+’0’;
  } while (zn!=0);
  puts(buf+bpos);
}
```

```
void main(void) {
  OutNumber(1472);
}
```

Программа выводит целое десятичное число посимвольно. Предполагается, что где-то описана функция putch

Структурирование программы

Когда размер программы превышает определенный, написать и осмыслить всю программу за один раз становится невозможно. Ещё интереснее, когда одну программу пишет несколько человек. В этих случаях очень помогает разбиение программы на логические модули. Каждый модуль должен выполнять четко определенную функцию, интерфейс модуля должен определяться функцией, а не аппаратными и программными возможностями контроллера. Это отнимает некоторые дополнительные ресурсы – память, быстродействие, но даёт возможность повторного использования модуля в других проектах, возможность изменения структуры модуля даже с полной заменой аппаратной составляющей без изменения остального кода проекта. Также, это помогает при коллективном труде над программой.

Как правило, в языке Си такой модуль содержит как минимум один заголовочный файл, описывающий интерфейс с модулем, и один компилируемый файл, содержащий код модуля.

Хорошим тоном является сокрытие внутренних переменных модуля за функциями типа «получить значение», «установить значение» - это помогает при переделке модуля. Хотя в некоторых случаях это неприемлемо, например при обработке больших объемов данных.

Также, хорошим тоном является использование имен переменных и функций, помогающих без труда определить их назначение.

При выполнении лабораторных заданий предлагается составлять программы с учетом возможного деления на модули и продумывать интерфейсы между модулями и основной программой.

Например, для светодиода, меняющего цвет, предлагается написать модуль, содержащий функции InitLed и SetColor.

Для светодиодного индикатора – функции InitSegm, setpos, putch, clrscr, repaint.

Лабораторная работа №1

Знакомство со схемотехническими особенностями
применения контроллера

Цель работы: знакомство с назначениями выводов контроллера STM32F30x, со схемотехникой подключения контроллера к цепям питания, синхронизации, программирования и к периферийным модулям системы, получение навыков поиска информации в документации на контроллер и приобретение базовых знаний о возможностях и требованиях контроллера. Подключение простейшей периферии к контроллеру.

Описание используемых модулей и микросхем

Используя документацию, предложенную в папке Distrib\Datasheet, необходимо проанализировать предложенную в описании платы STM32F3DISCOVERY схему (рисунки 11…14), изучить назначения и краткие характеристики используемых микросхем. Содержащаяся в папке Distrib\Datasheet документация включает в себя:

DSH_STM32F303.pdf – описание контроллера, включающее в себя электрические параметры, расположение выводов, структуру и общие сведения. По тексту для ссылки на этот файл будет использоваться наименование DSH.

RM_STM32F303.pdf – инструкция по использованию контроллера, включающая подробное описание периферийной части контроллера. По тексту для ссылки на этот файл будет использоваться наименование RM.

STM32F3DISCOVERY.pdf – описание используемой отладочной платы.

L3GD20.pdf – описание цифрового гироскопа, установленного на плате.

LSM303DLHC.pdf – описание цифрового компаса, установленного на плате. Имеется документация и на другие компоненты.

Схема платы построена необычно в виде фрагментов, соединённых между собой с помощью идентификаторов цепей. Плата описана на четырёх листах. Обратим внимание на некоторые моменты. На листе 1 (рисунок 11 документа) показано подключение выводов портов контроллера и питания к разъёмам P1, P2. Показаны, какие выводы контроллера используются при работе с отладчиком-программатором ST_LINK, показанным в левом углу условно квадратом, а какие выводы используются при работе с периферией, в виде блока справа в верхнем углу схемы. На втором листе чертежа предложена схема блока ST_LINK и некоторые схемотехнические особенности. В блоке PWR показано формирование напряжения питания контроллера с помощью стабилизатора LD3985M33R. Причём, питание контроллера формируется после диода Шотки BAT60J.., что позволяет при необходимости подавать через другой подобный резистор питание от другого источника. На контроллере STM32F103 реализован блок ST_LINK, который

использует для обмена с компьютером USB ST_LINK разъём, расположенный в середине короткой стороны платы напротив расположенных по кругу восьми светодиодов. Именно этот разъём используется при подключении к компьютеру. Выводы разъёма CN3, помеченные буквами SWD также могут использоваться с одноимённым интерфейсом SWD. Диод LD1 индицирует наличие питания контроллера, а светодиоды LD2 своим миганием указывают на активную работу блока ST_LINK.

На листе 3 показано подведение питания контроллера и подключение резонаторов.

На листе 4 показана периферия и её подключение к контроллеру. Согласно информации, предложенной на этом листе, восемь светодиодов подключены к разрядам PE[15/8]. Пользовательский USB разъём CN2, помеченный названием USB User, подключен непосредственно к входам контроллера STM32F303. Диоды D4, D5 при подключении разъёма CN2 к компьютеру позволяют запитать плату от компьютера, если отсутствует питание платы от постороннего источника с напряжением 5В. Если же питание платы от постороннего источника присутствует, диоды исключают возможность соединения двух источников питания параллельно, что недопустимо. Имеются схемы подключения кнопки сброса контроллера B2 и подключенной к ножке PA0 кнопки пользователя B1. Две микросхемы L3GD20 и LSM303DLHC, подключенные с помощью последовательных интерфейсов к контроллеру, используются для ввода информации в контроллер при необходимости.

Для формирования десятичных символов находят широкое применение семисегментные индикаторы.

В зависимости от обслуживания различных разрядов индикатора или различных сегментов различают индикацию статическую и динамическую.

Статическая индикация предполагает постоянное действие на каждый сегмент индикатора управляющего напряжения. Если предполагается индивидуальное управление каждым сегментом с целью сохранения возможности формирования специальных символов, то каждый семисегментный индикатор должен в этом случае управляться восьмиразрядным регистром

Динамическая индикация предполагает поразрядный вывод информации с периодизацией вывода с частотой более $n \cdot 8$Гц, где n – число разрядов. То есть в динамической индикации в конкретный момент времени выдаётся информация только в одном разряде многоразрядного индикатора, а остальные разряды пассивны. Динамическая индикация при большом числе разрядов оказывается выгодной, поскольку требует меньшего объёма аппаратных средств. Периодизация вывода информации может происходить не только от разряда к соседнему, но и от выдачи информации посегментно. То есть в данный момент времени производится активизация, например, сегментов A всех разрядов индикации, затем сегментов B и так далее. При поразрядном управлении в случае индикации только десятичных цифр

потребуется применение одного дешифратора десятичного кода в сигналы управления сегментами индикаторов. Динамическая индикация может быть реализована с использованием программного обмена, обмена по прерыванию и прямого доступа к памяти. Многоразрядные семисегментные светодиодные индикаторы конструктивно выполняются как одно изделие с разделением в расчёте на динамическое управление цепей управления сегментами и цепей разрешения работы каждого разряда.

Для ввода информации в микропроцессорную систему широко используется клавиатура, которая строится с использованием клавиш или кнопок, имеющих два состояния: кнопка нажата или отпущена.

Можно различать матричную и линейную организацию клавиатуры. При матричной организации кнопки соединяют в матрицу, которая при большом количестве кнопок требует минимального числа проводов для подключения к электронным схемам. Линейная организация целесообразна при небольшом количестве кнопок и подобна организации одной строки матричной клавиатуры. В зависимости от процедуры обслуживания клавиатуры может предусматриваться программный обмен или обмен по прерыванию. Следует помнить, что при использовании простейших кнопок с помощью резисторов необходимо задавать состояние входов контроллера, если кнопки не нажаты. Для этой цели могут быть использованы резисторы подтяжки, включение которых программируется в выбранном контроллере.

Прежде, чем выполнять работу постарайтесь ответить на контрольные вопросы, что облегчит проектирование схемы.

Порядок выполнения работы

Используя знания схемотехники, документацию на микроконтроллер STM32F303 с 48 контактами, результаты анализа схемы STM32F3DISCOVERY.pdf и возможность копировать фрагменты схемы из этой документации необходимо на базе макета принципиальной схемы stm32f3_Э3.vsd (рисунок 3) построить принципиальную схему модуля, в котором решены предложенные ниже задачи. На схеме предложены условные обозначения контроллера, разъёма питания X2, стабилизатора напряжения, четырёхразрядного индикатора, энкодера или измерителя углового положения AS5046. При необходимости следует в схему добавлять требуемые детали. Ставится задача подготовить аппаратную платформу для модуля измерителя угла, который, работая в периодическом режиме, выводит на семисегментный светодиодный индикатор с общими анодами значение измеренного угла. При этом потребуется определить набор функций, на которые ориентированы ножки используемого контроллера, чтобы разумно использовать его выводы. Эта информация предложена в таблице 11 документа DSH_STM32F303.pdf.

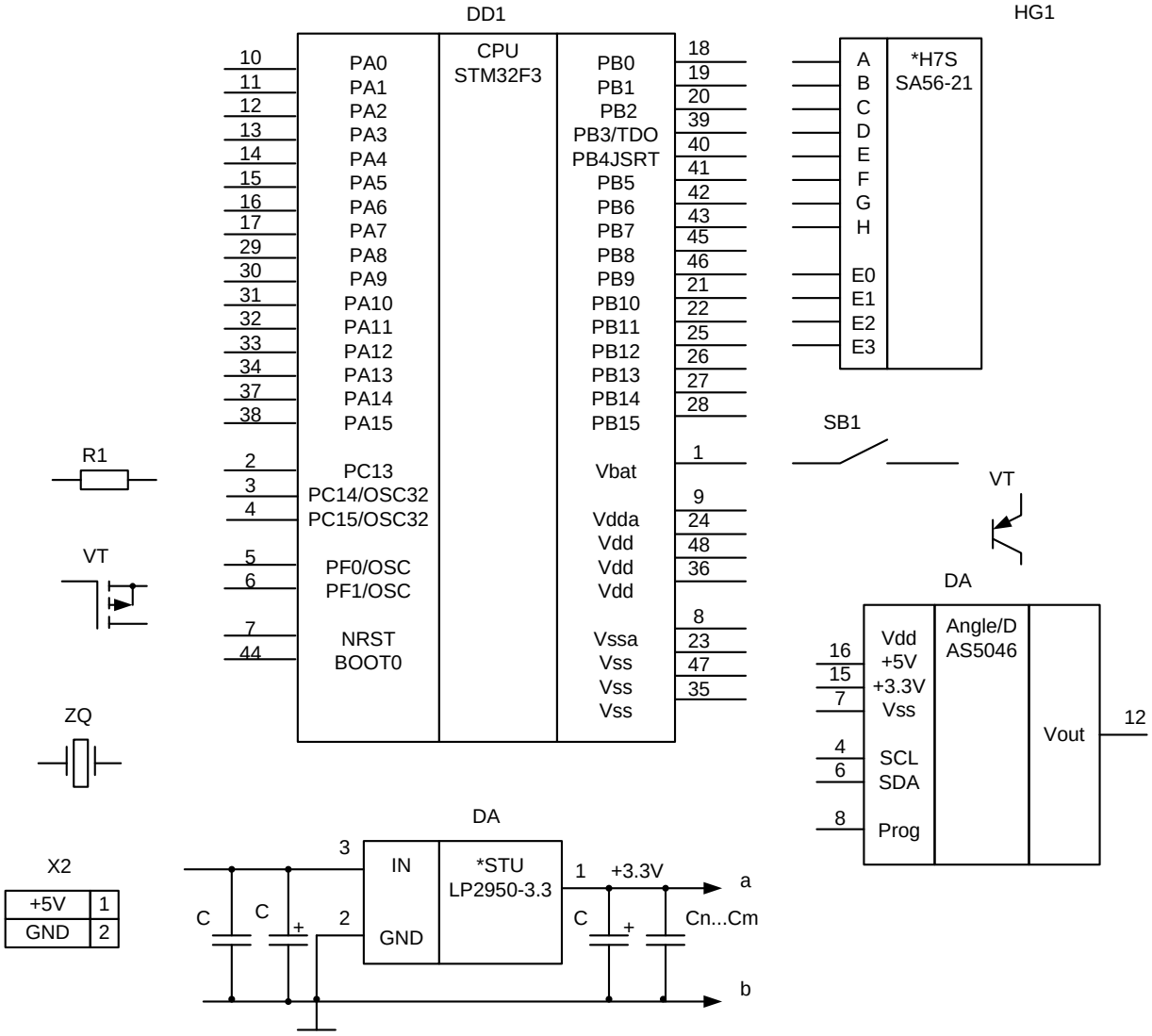


Рисунок 3. – Схема макета stm32f3_Э3.vsd

В том же документе предлагается схема подачи питания на рисунке 11, подключение резонаторов показано на рисунках 16 и 17, а на рисунке 25 показано использование интерфейса I2C. Используйте при необходимости и другие файлы.

Непременно решите следующие задачи. Качество исполнения чертежа должно быть достаточным, чтобы монтажник смог собрать макет.

1. Подать питание на контроллер и на другие элементы в предположении, что с помощью разъёма X2 на плату подаётся напряжение в +5Вольт. Для этого следует найти подходящий компонент, имеющийся на макете.
2. Подключите батарейку для сохранения некоторых функций контроллера при выключении питания (каких функций?).
3. Подключить резонаторы на 8МГц и на 32кГц.
4. Подключите кнопку в цепи внешнего сброса контроллера.
5. Подключите датчик AS5046, используя соответствующие входы контроллера с учётом требуемого интерфейса.

6. Подключите индикатор с учётом требуемого для его работы алгоритма. Не забудьте о необходимости ограничения токов в цепях светодиодов.
7. Подключите 4 клавиши клавиатуры, выбрав способ её обслуживания.
8. Подключите разъём для обеспечения возможности подключения к компьютеру.
9. Подключите вход BOOT0 с расчётом сохранения возможности программирования контроллера. Найдите в документации, при каком состоянии этого сигнала будет выполняться программирование контроллера?
10. Подключите светодиод к цепи питания контроллера, чтобы индицировать состояние напряжения питания.
11. Составьте таблицу при анализе схемы STM32F3DISCOVERY, в которой запишите соответствие номеру светодиода и его цвету номер контакта порта PE, к которому подключен светодиод. Определите, при каком состоянии выходного разряда порта светодиод будет светиться.
12. Составьте возможный алгоритм работы модуля.

Необходимо по результатам работы оформить отчёт.

Могут быть поставлены дополнительные задачи.

Контрольные вопросы

1. Каково напряжение питания контроллера?
2. Почему используются два резонатора и на какие частоты они настроены?
3. Будет ли контроллер функционировать, если резонаторы не будут подсоединены?
4. Какие интерфейсы используют датчики, используемые на плате STM32F3DISCOVERY?
5. В чём отличие статической и динамической индикации?
6. Предложите способ подключения четырёх клавиш к контроллеру, не имеющему резисторов подтяжки, в двух вариантах: клавишы организованы в матрицу и в линию.
7. Предложите схему динамической индикации на 4 разряда и опишите алгоритм её обслуживания.
8. Определите выводы контроллера, имеющего 48 ножек, которые используются при организации интерфейсов типа USB, UART, I2C.

Лабораторная работа №2

Создание проекта в среде Keil

Цель работы: знакомство с особенностями создания проекта в среде Keil и изучение особенностей программирования портов контроллера.

Общие сведения

Особенности работы ARM контроллеров с периферией

Ядро контроллера имеет сквозную адресацию памяти и периферийных устройств в одном адресном пространстве размером 4Гб (32 бита на хранение адреса). Общение с любыми устройствами, а также с памятью осуществляется с помощью «Bus Matrix» (стр. 11 файла DSH) - матрицу шин, набор шин для связи ядра и периферии. Память и некоторые устройства подключены к шине напрямую, но в основном требуется «подключить к шине», причем к нужной, блок, который нужно использовать.

«Подключение к шине» фактически разрешает тактирование данного блока. Пока на блок не подано тактирование, его энергопотребление минимально, поэтому включать лишние блоки не приветствуется. За подключение к шине, также как и за тактирование ядра, отвечает блок «Reset & clock control». Некоторые устройства, кроме подключения к шине, могут требовать подключения тактовой частоты с данного блока для внутренних нужд – это производится отдельной операцией.

Работа «Reset & clock control» описана в файле SDL: stf32f30x_rcc.c.

Особенности портов ARM контроллеров

Порты ввода-вывода ARM контроллеров (стр. 135 файла RM) могут работать в разных режимах. Каждая ножка контроллера (кроме служебных) относится к какому-либо порту ввода-вывода (GPIO), но при этом может иметь ещё альтернативную или аналоговую функцию. Выбор функции пользователь должен произвести в явной форме, то есть если используется блок, имеющий свои входные цепи – например UART, то соответствующие выводы должны быть переведены в состояние альтернативной функции вручную. При этом некоторые блоки подключены одновременно к нескольким выводам, что позволяет разработчику выбирать, куда ему лучше развести линии данного блока.

Кроме функции выводы имеют ряд настроек, доступных независимо от функции (нужно сверяться со здравым смыслом – использование резистора подтяжки при включенной аналоговой функции, например, смысла не имеет, хотя и возможно). Эти настройки:

- Выход с открытым коллектором
- Резисторы подтяжки к «+ питанию» и «земле» (вверх и вниз), включаемые раздельно
- Контроль скорости выхода – три режима, 2, 10 и 50МГц. Рекомендуется выбирать малую скорость, если иного не требуется. В этом случае емкостная нагрузка не сможет повредить выходной каскад порта.

Также для рассматриваемого контроллера доступна «фиксация» настроек порта на случай сбоя программы. Данная функция не рассматривается.

Использование библиотеки SDL как справочного пособия

Использование библиотеки SDL позволяет сильно упростить написание программы, поскольку сами заголовочные файлы библиотеки как правило содержат всю необходимую информацию в доступной форме.

Для использования какого-либо периферийного блока нужно подключить соответствующий .c файл к проекту. Соответствующие заголовочные .h файлы подключаются к проекту внутри файла stm32f30x.h. Можно также добавить их в проект для простоты доступа к ним – компилировать их смысла нет, но легче становится открыть соответствующий заголовочный файл.

В начале каждого .c файла есть краткая инструкция по работе с данным файлом.

В конце каждого заголовочного файла есть список функций, которые доступны для работы с данным устройством.

На рисунке 4 показаны фрагменты файла «stm32f30x_gpio.c» с комментариями, который используется при работе с портами контроллера.

```
* @file    stm32f30x_gpio.c
  * @author  MCD Application Team
  * @version V1.0.1
  * @date    23-October-2012
  * @brief   This file provides firmware functions to manage the following
  *          functionalities of the GPIO peripheral:
  *           + Initialization and Configuration functions - (функции инициализации и конфигурации)
  *           + GPIO Read and Write functions – (порт функции чтение и запись)
  *           + GPIO Alternate functions configuration functions – (альтернативные функции)

 *
==============================================================================
                ##### How to use this driver #####  - (Как пользоваться этим драйвером)
==============================================================================
   (#) Enable the GPIO AHB clock using RCC_AHBPeriphClockCmd() – (подключение)
   (#) Configure the GPIO pin(s) using GPIO_Init() – (конфигурирование побитное)
       Four possible configuration are available for each pin:
       (++) Input: Floating, Pull-up, Pull-down. – (вход, свободный или подключён резистор)
       (++) Output: Push-Pull (Pull-up, Pull-down or no Pull), - (выход логический или с резисторами подтяжки)
                    Open Drain (Pull-up, Pull-down or no Pull). – (выход с открытым стоком с резисторами подтяжки или без них)
          In output mode, the speed is configurable: Low, Medium, Fast or High. – (скорость или быстродействие на данном выходе)

==============================================================================
          ##### Initialization and Configuration ##### - (инициализация и конфигурация)
==============================================================================
```

```
else if(GPIOx == GPIOE)
  {
   RCC_AHBPeriphResetCmd(RCC_AHBPeriph_GPIOE, ENABLE); - (чтобы включить порт GPIOE – порт E)
   RCC_AHBPeriphResetCmd(RCC_AHBPeriph_GPIOE, DISABLE); - (чтобы этот порт выключить)
...
```

Рисунок 4. – Фрагменты файла «stm32f30x_gpio.c»

В основном конфигурация устройства включает три стадии:

- подключение к шине, производится функциями файла stf32f30x_rcc, например RCC_APB1PeriphClockCmd(uint32_t RCC_APB1Periph, FunctionalState NewState) – в качестве RCC_APB1Periph указывается константа – идентификатор периферийного блока, в качестве Functional State – ENABLE.

- конфигурация блока, функцией …_Init (три точки – наименование блока). Для конфигурации необходимо заполнить определенную структуру, поля которой можно посмотреть в том же h файле. Перед конфигурацией рекомендуется выполнять функцию DeInit. Ёсли сброс произошел не по включению питания, это поможет вернуть периферию в нормальное состояние.

- Для многих блоков, хотя и не для всех, нужно включить блок, подав ему команду …_Cmd(ENABLE) (три точки – наименование блока). Команда разрешит работу блока.

При использовании библиотеки ни в коем случае (кроме явно оговоренных мест) нельзя использовать числовые константы – нужно найти и использовать константы, описанные в соответствующем h файле.

Для поиска функций нужно посмотреть окончание h файла соответствующего блока, а вот для поиска определений структур и констант можно использовать следующие функции (нужно хотя бы раз откомпилировать проект с подключенными файлами, чтобы было составлено дерево ссылок):

- При нажатии правой кнопки на типе структуры (имени функции, переменной…) доступна команда «Go to definition of ‘’» - перейти на определение данной структуры. Перейдя на определение, можно узнать поля структуры. Константы, описывающие поля, как правило, определены выше или ниже структуры, в непосредственной близости. Иногда, к ним также можно применить указанный способ поиска.

- При заполнении структуры после ввода имени и символа «.» можно нажать Ctrl+Пробел, появится список полей (Если включено автозаполнение, то список появится сам).

- Если ничего из выше перечисленного не помогло, то можно использовать поиск (Ctrl+F) или поиск по всем файлам проекта (Ctrl+Shift+F).

Создание проекта

Необходимая для работы информация содержится в нескольких папках:

Distrib\Drivers – дистрибутив драйвера отладчика ST-Link, интегрированного в отладочную плату.

Distrib\Library – библиотека CMSIS для контроллера STM32F303

Docs – документация.

Так как библиотека CMSIS и SDL содержит много мелких файлов, копировать её из проекта в проект смысла не имеет. Поэтому рекомендуется следующая структура папок проекта (Таблица 2):

Таблица 2

Projects\CMSIS	Содержит функции работы с ядром, описания регистров контроллера
Projects\STM32F30x_StdPeriph_Driver	Содержит блоки для работы с периферийными устройствами контроллера. Для работы с конкретным устройством нужно подключить к проекту .c файл с соответствующим именем из каталога Projects\STM32F30x_StdPeriph_Driver\src, и добавить в программу ссылку на соответствующий заголовочный файл из каталога Projects/STM32F30x_StdPeriph_Driver\inc.
Projects\Task1 … Projects\Taskn	Ваши проекты

То есть работа начинается с создания на локальном диске D в папке «users\ARM» папки с вашими проектами с названием, например, «Projects», в которую поместите папки с библиотеками: «CMSIS» и «STM32F30x_StdPeriph_Driver» и в которой вы можете располагать свои проекты с названиями, например, «Taskn». Это позволяет при необходимости переносить на другой компьютер в папке «Projects» свои проекты. Библиотека при этом будет всегда рядом с проектами, а без неё вы не откроете проект.

В рамках данной лабораторной работы необходимо познакомиться с процедурой создания проекта в среде Keil и с некоторыми особенностями использования библиотеки при работе с портами. В результате создаваемый проект должен заставить некоторые светодиоды платы светиться.

1. Создаём в папке Projects папку Task1, а в ней папку - Source
2. Запускаем Keil

3. Создаём проект в среде Keil: в меню Project -> New uVsion Project

Выбираем созданную папку Task1, затем в списке контроллеров – выбираем производителя STMicroelectronics и контроллер STM32F303VC (буквы задают в том числе объём памяти). Соглашаемся на копирование startup_stm32f30x.s файла и добавление его в проект. Этот файл заведует векторами прерывания контроллера. Подробнее этот файл будет разобран на других занятиях.

4. Создаём файл программы – меню File -> New, в открывшемся окне набираем минимальный текст:

```
#include <stm32f30x.h>

int main(void) {
  for (;;);
}
```

Сохраняем файл в папке Source

При написании программы рекомендуется следовать правилам:

1) каждый уровень команд сдвинут на 2 пробела (что при указанных выше настройках равно 1 табуляции)
2) открывающая фигурная скобочка находится на той же строке, что и команда
3) Закрывающая фигурная скобочка находится под первым символом команды
4) Имена функций и переменных должны однозначно определять их использование и назначение.
5) Комментарии должны быть там, где нужно, но не нужно комментировать очевидное.
6) Желательно, чтобы функции не были слишком большие – в идеале, умещались на экране. Если основная программа не разбита на функции, нужно, чтобы каждая конкретная логическая часть программы не была слишком большой – также умещалась на экране. Можно разбивать программу на логические части комментариями. Использование лишних вызовов функций может уменьшить быстродействие – неизвестно, будет ли компилятор оптимизировать программу, выбрасывая ненужные вызовы функций.
7) Рекомендуется не экономить на скобочках, ни на фигурных, ни на обычных – не полагаться лишний раз на приоритет выполнения операций в выражениях.

Такие правила написания программ позволят уместить на один экран как можно больше значащего текста, что облегчает анализ и понимание программы.

5. Копируем файлы:

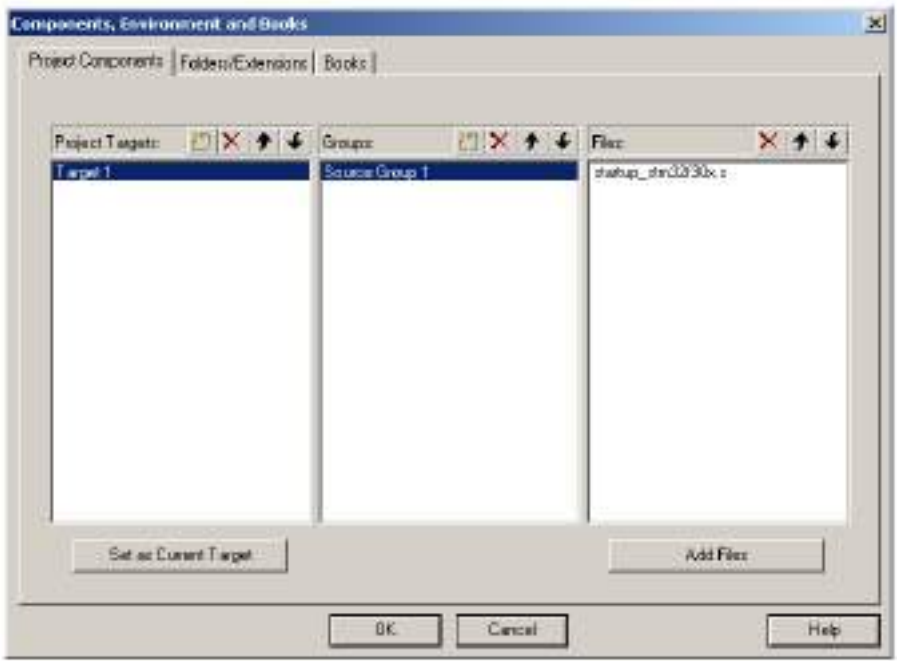
CMSIS\Device\ST\STM32F30x\Source\Templates\system_stm32f30x.c
CMSIS\Device\ST\STM32F30x\Source\Templates\ stm32f30x_conf.h

в каталог Task1\Source

Первый файл проводит начальную инициализацию контроллера – в основном тактового генератора.

Второй файл конфигурирует SDL, позволяет включить контроль входных параметров функций настройки периферии. Это можно сделать на время отладки, сейчас данная возможность отключена.

6. Добавляем файлы в проект и наводим в нём порядок, разбив файлы на две группы – Source и SDL. Для этого нужно нажать правой кнопкой на окне проекта и выбрать «Manage Components» или использовать соответствующую пиктограмму. Откроется окно «Components, ..» с открытой папкой «Project Components».



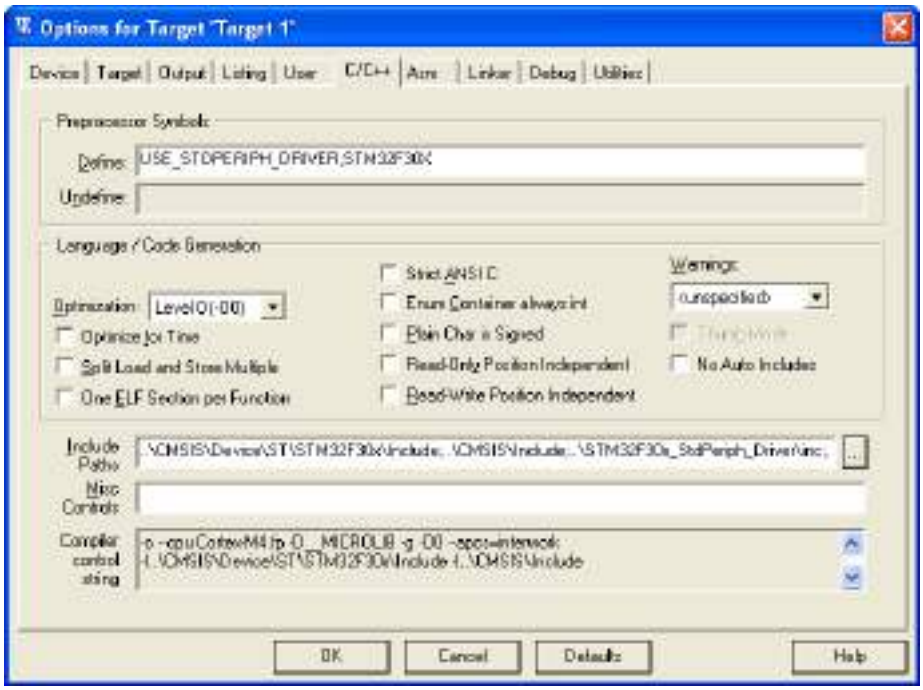
В левом списке будет наименование проекта, в среднем – группы, а в правом – файлы, включенные в группу, выбранную в среднем списке. Группу по умолчанию переименовываем в Source, и создаём группу SDL.

В группу Source добавляем файлы из каталога Source и файл startup_stm32f30x.s из корневого каталога проекта.

В группу SDL в будущем будут добавлены файлы для работы с конкретными периферийными устройствами контроллера из каталога STM32F30x_StdPeriph_Driver\src.

7. Добавляем пути к заголовочным файлам и конфигурируем опции компиляции следующим образом:

Правой кнопкой мыши на имени проекта Target1 открываем окно Options for Target 1. Выбираем вкладку C/C++, добавляем в открывшейся вкладке в строке «Define» определения (набираем вручную) USE_STDPERIPH_DRIVER, STM32F30X. Эти определения конфигурируют CMSIS на подключение некоторых файлов, в том числе stm32f30_conf.h.

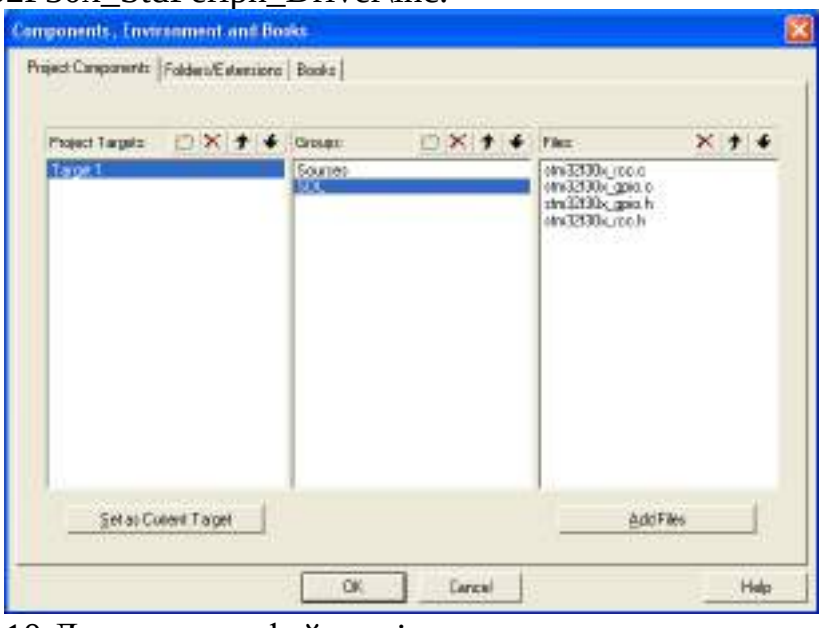


Далее редактируем список папок с заголовочными файлами, для чего нажимаем кнопку правее строки «Include Paths». Добавляем каталоги из

папок CMSIS, SDL. Для этого в открывшемся окне щёлкаем дважды левой клавишей мышки на верхней строке. Справа на этой строке появляется клавиша, помеченная тремя точками, нажав на которую, откроем дерево папок и файлов. Находя необходимый объект, помещаем его в проект.

8. Компилируем нажатием кнопки F7 или соответствующей пиктограммки . Программа должна безошибочно откомпилироваться. В нижней части экрана в окне «Build Output» в нижней строке появится текст: 0Error(s), 0Warning(s)

9. Добавляем в группу SDL файлы stm32f30x_rcc.c и stm32f30x_gpio.c из каталога STM32F30x_StdPeriph_Driver\src. Можно, но не обязательно, добавить соответствующие файлы .h из каталога STM32F30x_StdPeriph_Driver\inc.



10.Дописываем файл main.c:

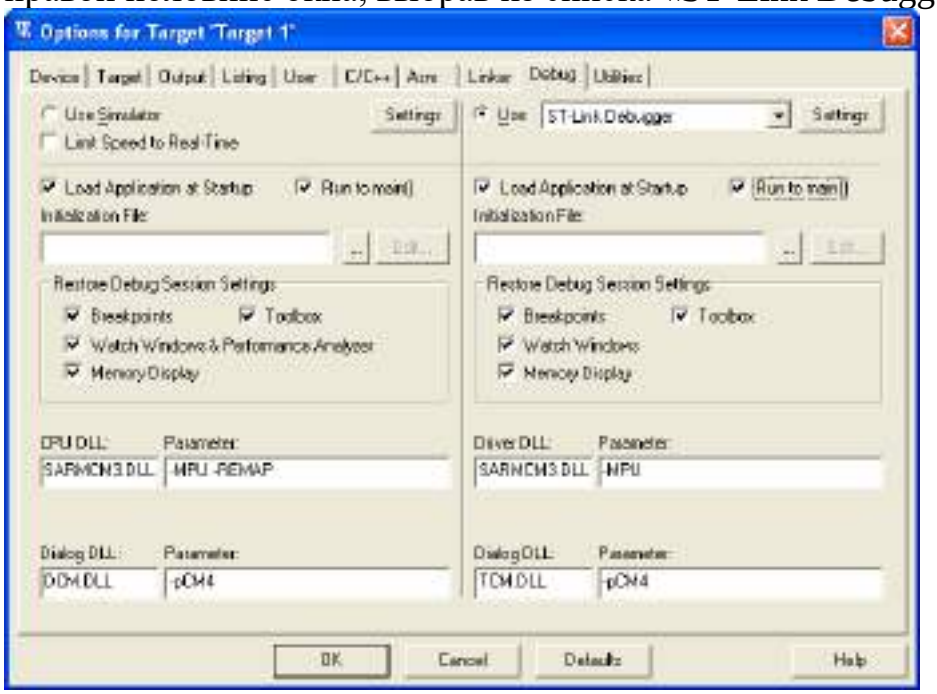
```
#include <stm32f30x.h>

int main(void) {
  GPIO_InitTypeDef GPIO_InitStruct; // Структура для настройки порта
```

```
// Инициализация порта со светодиодами
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOE,ENABLE);
  GPIO_InitStruct.GPIO_Mode=GPIO_Mode_OUT;
  GPIO_InitStruct.GPIO_OType=GPIO_OType_PP;
  GPIO_InitStruct.GPIO_Pin=GPIO_Pin_9|GPIO_Pin_13;
  GPIO_InitStruct.GPIO_PuPd=GPIO_PuPd_NOPULL;
  GPIO_InitStruct.GPIO_Speed=GPIO_Speed_2MHz;
  GPIO_Init(GPIOE,&GPIO_InitStruct);
// Включение светодиодов
  GPIO_SetBits(GPIOE,GPIO_Pin_9|GPIO_Pin_13);
  for (;;);
}
```

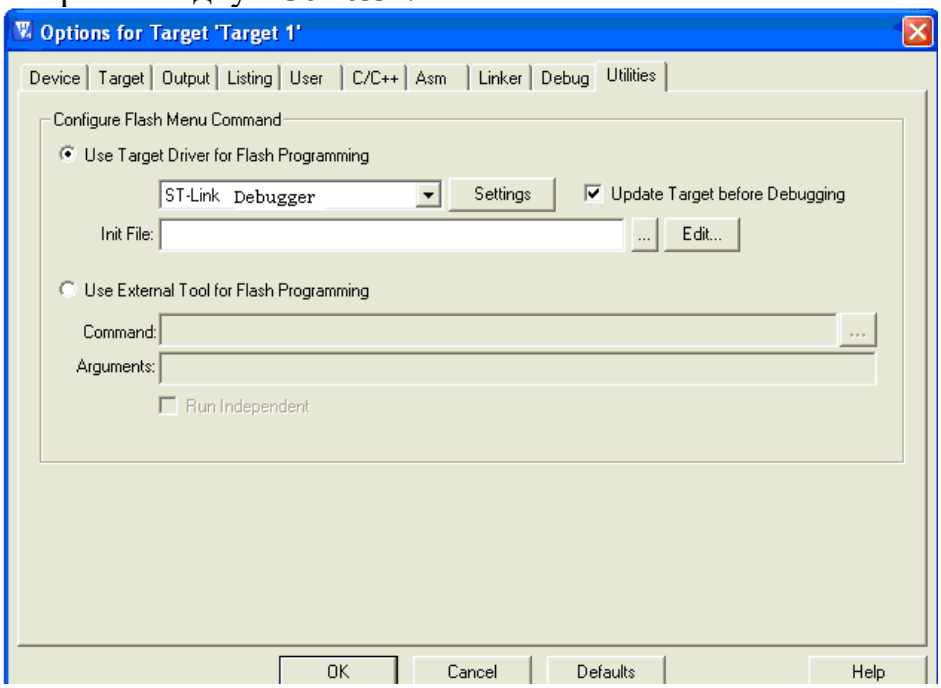
11.Подключаем к проекту отладчик, предполагая надичиеплаты.

Для этого в опциях проекта выбираем «Debug», и ставим галочку в правой половине окна, выбрав из списка «ST-Link Debugger».



Также, можно нажать кнопку «Settings» рядом со списком отладчиков и установить порт для работы: Port - SW

Затем нужно настроить утилиту программирования перед отладкой, выбрав вкладку «Utilites»:



Далее нужно выбрать устройство, которое будет загружаться, нажав Settings и выбрав папку Flash Download:

12.Компилируем программу, выбрав в меню Project -> Rebuild all targets или нажав соответствующую пиктограммку .

13.Запускаем программу в отладку, выбрав в меню Debug -> Start/stop debug session или нажав соответствующую пиктограммку .

14.Если программа загружена нормально, интерфейс должен изменить свой вид. Запускаем программу на выполнение кнопкой F5 или пиктограммкой .

Программу можно остановить, можно посмотреть переменные или состояние памяти. Интерфейс достаточно понятен. Для перекомпиляции программы, если текст был изменён, нужно выйти из режима отладки, откомпилировать программу, и войти в режим отладки повторно. ВНИМАНИЕ! Вход в режим отладки не включает в себя компиляцию – если она не была проведена вручную, будут загружены старые данные.

Порядок выполнения работы

1. Создать описанный проект, заставить запрограммированную плату работать должным образом, зафиксировать в отчёте результаты работы платы вместе с копиями экрана.
2. Внести комментарий в программу, объясняя, что выполняется каждой командой. Результат работы поместить в отчёт.
3. Изменить программу таким образом, чтобы включались другие светодиоды с учётом номера бригады (таблица 3). Информация о подключении светодиодов и их цвете предлагается в таблице 4. Светодиоды включаются «единицей».

Таблица 3

Вариант	1	2	3	4	5	6
Светятся диоды	LD3, LD10	LD4, LD9	LD5, LD8	LD6, LD7	LD4, LD5	LD8, LD9

Таблица 4

Светодиод	Разряд порта E
LD3 – red	PE9
LD4 – blue	PE8
LD5 – orange	PE10
LD6 – green	PE15
LD7 – green	PE11
LD8 – orange	PE14
LD9 – blue	PE12
LD10 – red	PE13

4. Подключить кнопку USER. При нажатии и удержании кнопки должны отключаться ранее подключенные два диода, а три других должны включиться.

Задачу решить самым простым способом.

Кнопка подключена к выводу PA0 и подтягивает вывод к плюсу. При этом у кнопки есть резистор подтяжки к минусу, поэтому на выводе PA0 формируется 0, если кнопка не нажата и 1, если кнопка нажата. Необходимо настроить работу порта A.

В зависимости от считанного значения кнопки нужно сбрасывать в ноль состояния одной группы выводов и устанавливать единицы на других ножках. Если возникают проблемы с применением Си и библиотек контроллера, то можно разобраться в предложенном ниже фрагменте программы. Анализируя команды программы, внесите вместо вопросов комментарий, указывающий, что делает выбранный кусочек. Желательно внести комментарии для каждой команды.

```
// ?????
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA,ENABLE);
  GPIO_InitStruct.GPIO_Mode=GPIO_Mode_IN;
  GPIO_InitStruct.GPIO_Pin=GPIO_Pin_0;
  GPIO_Init(GPIOA,&GPIO_InitStruct);
// ?????
  for (;;) {
    if (GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_0)) {

GPIO_SetBits(GPIOE,GPIO_Pin_8|GPIO_Pin_10|GPIO_Pin_12|GPIO_Pin_14);

GPIO_ResetBits(GPIOE,GPIO_Pin_9|GPIO_Pin_11|GPIO_Pin_13|GPIO_Pin_15);
    } else {

GPIO_ResetBits(GPIOE,GPIO_Pin_8|GPIO_Pin_10|GPIO_Pin_12|GPIO_Pin_14);

GPIO_SetBits(GPIOE,GPIO_Pin_9|GPIO_Pin_11|GPIO_Pin_13|GPIO_Pin_15);
    }
  };
```

```
}
```

Совет по написанию программы

В структуре для настройки портов задаётся, какие именно ножки запрограммировать. При инициализации разных портов или разных ножек одного порта можно пользоваться тем, что структура после выполнения функции GPIO_Init не меняется – можно изменить только нужные поля, остальные останутся в ранее установленных значениях, и выполнить GPIO_Init для других ножек с измененными условиями.

Если выражение не помещается на одну строку, можно перенести его на другую, разбив по знаку операции. Рекомендуется в конце строки оставлять аргумент, а следующую строку начинать со знака операции, например:

```
z=a+b+c
+d+e+f;
```

В этом случае если по ошибке в конце предыдущей строки будет установлена точка с запятой, компиляция завершится ошибкой.

Должно получиться два блока настройки порта – для светодиодов и для кнопки.

Внутри бесконечного цикла нужно вставить условие, примерно следующего вида:

```
for (;;) {
  if (…) {
    …
  } else {
    …
  }
}
```

Контрольные вопросы

1. Каков порядок настройки выбранного порта?
2. Какова разрядность портов?
3. Как обозначаются порты при написании программы?
4. Зачем нужны библиотеки CMSIS, SDL?
5. Какие варианты организации выходных и входных цепей портов вы знаете?
6. Как организовать выход с открытым стоком?
7. Термин «Push-Pull» о чём говорит?
8. Что будет делать контроллер при выполнении предложенных команд:

```
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOE,ENABLE);
GPIO_InitStruct.GPIO_Pin=GPIO_Pin_0;
GPIO_InitStruct.GPIO_Speed=GPIO_Speed_2MHz;
```

Лабораторная работа №3

Изучение работы с таймерами и системой обработки прерываний

Цель работы: знакомство с особенностями использования и программирования универсальных таймеров, системного таймера SysTick и контроллера прерываний NVIC.

Общие сведения

Все устройства, работающие в контроллере или в системе под управлением контроллера, используют одно адресное пространство, то есть работа с любым устройством аналогична записи или чтению из памяти.

В основном устройства характеризуются базовым адресом и смещением рабочих регистров относительно базового адреса. В Си это описывается как структура.

Это позволяет описывать одинаковые устройства одинаковыми структурами, изменяя только базовые адреса.

Ссылки типа GPIOA, GPIOE – это как раз базовые адреса, описанные как структуры.

Ядро контроллера имеет **механизм обработки «исключений»**. Исключения бывают различных типов:

- Немаксируемые исключения, такие как Reset, NMI, HardFault, MemFault…
- Маскируемые, вызываемые аппаратурой или программно – прерывания

В дальнейшем немаскируемые исключения продолжим называть исключениями, а маскируемые, пользовательские исключения будем называть прерываниями.

Для обработки исключений используется таблица, по умолчанию хранящаяся с адреса 0x00000000 (Первая ячейка таблицы – не вектор прерывания, а начальное значение указателя стека SP). Каждая ячейка таблицы содержит адрес функции, вызываемой при исключении. Первые 15 исключений – немаскируемые, далее идут маскируемые прерывания, их количество и назначение определяется уже не ядром, а контроллером.

В библиотеке CMSIS принято нумеровать прерывания с нуля, а немаскируемые исключения нумеровать отрицательными числами (см. стр. 37 (PM0214.pdf)).

При создании первого проекта использовались файлы startup_stm32f30x.s и system_stm32f30x.c.

Файл startup_stm32f30x.s содержит ассемблерный текст, описывающий таблицу векторов прерываний, выделяющий память под стек и «кучу». Также этот файл содержит обработчик исключения «Reset_handler» - запуск программы, и обработчики по умолчанию для остальных исключений и прерываний.

Обработчики по умолчанию содержат только одну команду – переход на саму себя. Т.е. если произойдёт прерывание или исключение, не предусмотренное программистом – контроллер зависнет.

Обработчик исключения «Reset_handler» вызывается при сбросе (и при включении питания). Этот обработчик вызывает функцию SystemInit (описанную во втором файле - system_stm32f30x.c) и передаёт управление основной программе main.

В файле startup_stm32f30x.s можно найти имя функции, которую нужно описать для включения в программу своего обработчика прерываний. Например для таймера SysTick это SysTick_Handler. Описанная в тексте программы функция с таким именем будет вызываться по каждому прерыванию таймера SysTick.

Файл system_stm32f30x.c содержит две функции и одну глобальную переменную:

SystemInit() – инициализирует тактовый генератор. В данном случае – пытается включить кварц на 8МГц, затем включает умножитель и формирует частоту 72МГц. На плате кварца нет, поэтому частота остаётся 8МГц.

SetSysClock() – эту функцию нужно вызвать для обновления значения SystemCoreClock после конфигурации тактовых цепей контроллера.

SystemCoreClock – текущее значение частоты ядра HCLK.

Кроме значения SystemCoreClock для определения текущей частоты контроллера можно использовать функцию RCC_GetClocksFreq(…) драйвера stm32f30x_rcc.h.

Все объявления, касающиеся контроллера, выполняются в файле **stm32f30x.h**. Данный файл хранится в дебрях каталога CMSIS и содержит описания всех структур и констант, описывающих аппаратную часть контроллера.

Также, в конце файла есть подключение заголовочного файла stm32f30x_conf.h – данный файл включает заголовочные файлы библиотеки драйверов и подключается, если вы в проекте указали USE_STDPERIPH_DRIVER.

Именно это определение, с определением STM32F30X – типом контроллера – нужно указать в проекте.

Таймер SysTick – единственный таймер, принадлежащий ядру а не периферии. Таймер 24-битный, считает от заданного значения до 0 и перезагружает заданное значение.

Для использования таймера нужно:

1. Задать значение перезагрузки таймера в регистре STK_LOAD
2. Обнулить значение таймера в регистре STK_VAL
3. Выбрать режим работы в регистре STK_CTRL - Разрешить счёт и прерывания в регистре STK

Все эти действия одновременно выполняет функция SysTick_Config(uint32_t), описанная в файле core_cm4.h. Параметром функции является значение регистра перезагрузки, т.е. число тактов, через

которое будет вызываться прерывание. Функция возвращает 0 при успехе, и не ноль – если затребованное значение счетчика слишком велико.

Следует учесть, что функция SysTick_Config меняет приоритет прерывания от SysTick на 15, т.е. самый низкоприоритетный.

Вызов данной функции автоматически разрешает прерывания от SysTick.

Рекомендуется не задавать число тактов в явной форме, а использовать функцию RCC_GetClocksFreq, возвращающую значения различных частот в текущей конфигурации, и рассчитывать требуемое число тактов:

```
RCC_GetClocksFreq(&RCC_Clocks);
SysTick_Config(RCC_Clocks.HCLK_Frequency/2);
```

Можно проверить результат конфигурации таймера функцией SysTick_Config: если он не нулевой, таймер не отконфигурирован.

Для написания функции, вызываемой по прерыванию от SysTick, нужно узнать её имя. Имя функции можно посмотреть в файле startup_stm32f30x.s.

Данный файл содержит ряд служебной информации:

- размер стека, размер кучи
- таблицу векторов
- Начальную программу, вызывающую сначала запуск функции SystemInit а затем функции main
- Обработчики исключений «по умолчанию», вызывающие зависание контроллера
- Обработчики прерываний «по умолчанию», также вызывающие зависание контроллера.

В таблице векторов можно найти имя функции – обработчика прерываний от таймера: SysTick_Handler

Для написания обработчика нужно в любом месте программы (вне тела функций, конечно же) описать функцию с данным именем, она и будет вызываться при прерывании (благодаря файлу .s):

```
void SysTick_Handler(void) {
…
}
```

Прерывания от периферии

Прерывания от периферийных узлов формируются по предложенной на рисунке 5 схеме.

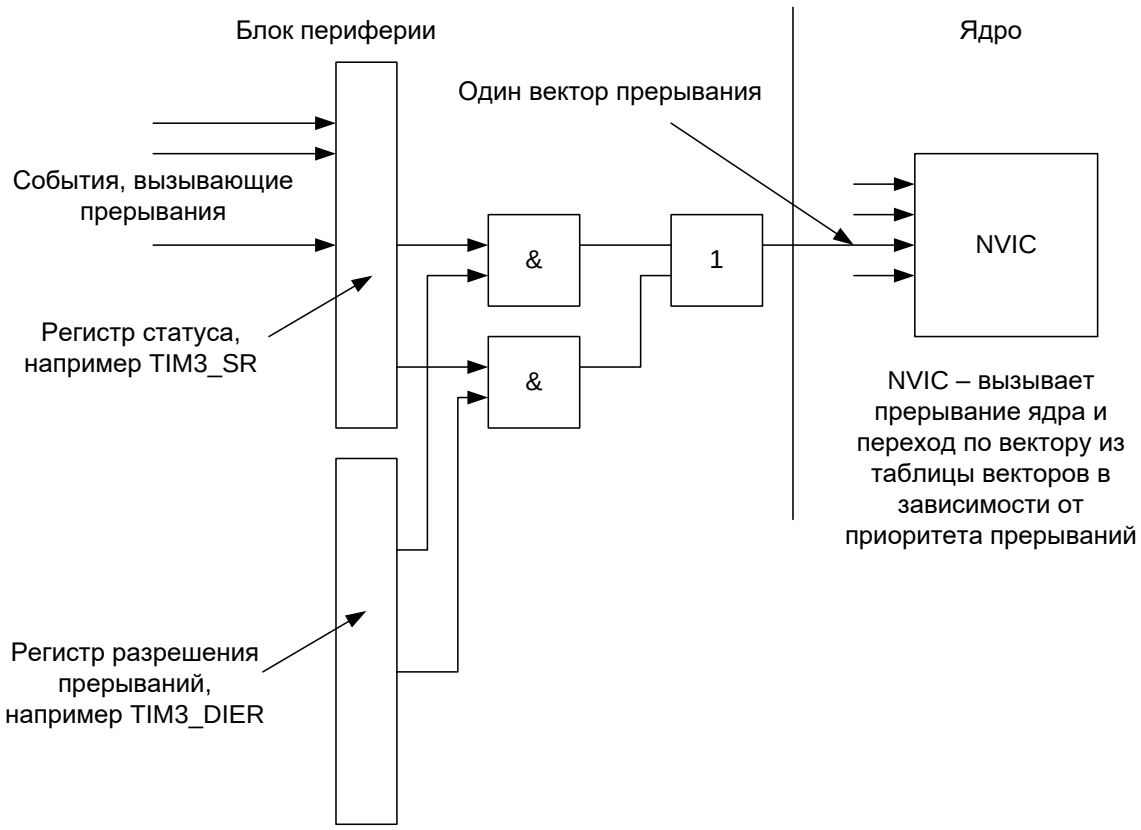


Рисунок 5. – Схема обработки прерываний

В каждом периферийном блоке есть несколько источников событий, например переполнение таймера, совпадение значения таймера с регистром сравнения, и т.д. При возникновении данных событий устанавливается в «1» соответствующий флаг в регистре статуса. Данный флаг может быть сброшен программно.

Если необходимо по данному событию сформировать прерывание, нужно установить «1» в соответствующем бите регистра разрешения. Если флаг события установлен и прерывание разрешено, формируется запрос на прерывание в контроллер прерывания ядра. Запрос на прерывание – общий для данного периферийного блока (в некоторых случаях – даже общий на несколько блоков).

Таким образом, для разрешения прерывания нужно:

- Разрешить прерывание в периферийном устройстве
- Разрешить линию прерывания от периферийного устройства в контроллере NVIC

При обработке прерывания нужно:

- Опрашивая возможные источники прерывания по данному вектору (регистры статуса) найти флаг, вызвавший прерывание
- Выполнить требуемые действия
- Сбросить обработанный флаг

В противном случае, сразу после выхода из процедуры прерывания контроллер повторно войдёт в процедуру прерывания, так как запрос на прерывание не будет снят.

Контроллер NVIC ядра поддерживает программируемые приоритеты для всех прерываний, кроме некоторых исключений. Приоритет задаётся числом от 0 до 15, при этом приоритетным является прерывание с меньшим значением приоритета. Внутри контроллера NVIC приоритеты задаются в старших 4-х битах регистров IPR, каждый 32-х битный регистр делится на 4 восьмибитных значения.

При использовании CMSIS на стр.195 файла PM0214 описаны функции для управления контроллером NVIC. Основные:

NVIC_EnableIRQ, NVIC_DisableIRQ – разрешение и запрещение линии прерываний

NVIC_SetPriority() – установка приоритета линии прерываний.

Универсальные таймеры

Периферия контроллера включает 13 универсальных таймеров. Для их использования можно напрямую программировать регистры контроллера – поля структур TIMx, либо использовать драйвер stm32f30x_tim.

Таймер, то есть счетчик таймера, может работать как на увеличение, так и на уменьшение значения, а также изменять направление счета каждый цикл. Подсчёт ведется либо внутренней частоты, либо внешних импульсов. Возможно добавление делителя перед счетным входом таймера.

Как правило таймер имеет ряд входов с регистрами захвата, позволяющими по внешнему сигналу сохранить значение таймера в регистре TIMx_CCRy, и ряд выходов, изменяющих своё состояние при совпадении значения таймера со значением TIMx_CCRy

Каждый таймер подключен к одному из входов прерывания контроллера (вектору прерывания), но внутри таймера есть множество источников прерывания, поэтому пользователь должен в обработчике прерывания разобрать причину прерывания и удалить соответствующий флаг.

Возможности таймера благодаря этому очень широки. Любой таймер может:

- Измерение частоты (подсчётом импульсов за известный промежуток времени)
- Измерение временных интервалов (захватом значения таймера при приходе внешних сигналов)
- Генерация внутренних временных интервалов (прерываний через заданный промежуток времени) (TIMx_CCRy+=константа)
- Генерация ШИМ сигналов различной сложности

Также, имеются специальные возможности для:

- Приема квадратурных сигналов
- Формирования сигналов для управления транзисторными мостами, с формированием «мертвого времени» для закрытия транзисторов.

Программирование таймера TIM3 с помощью SDL

1. Подключить к шине таймер – поиск сочетания TIM3 в файле stm32f30x_rcc.h даст константу и соответственно наименование функции, которой подключать к шине.
2. Создать структуру с типом TIM_TimeBaseInitTypeDef и заполнить все её поля (кроме последнего – оно не для этого таймера).

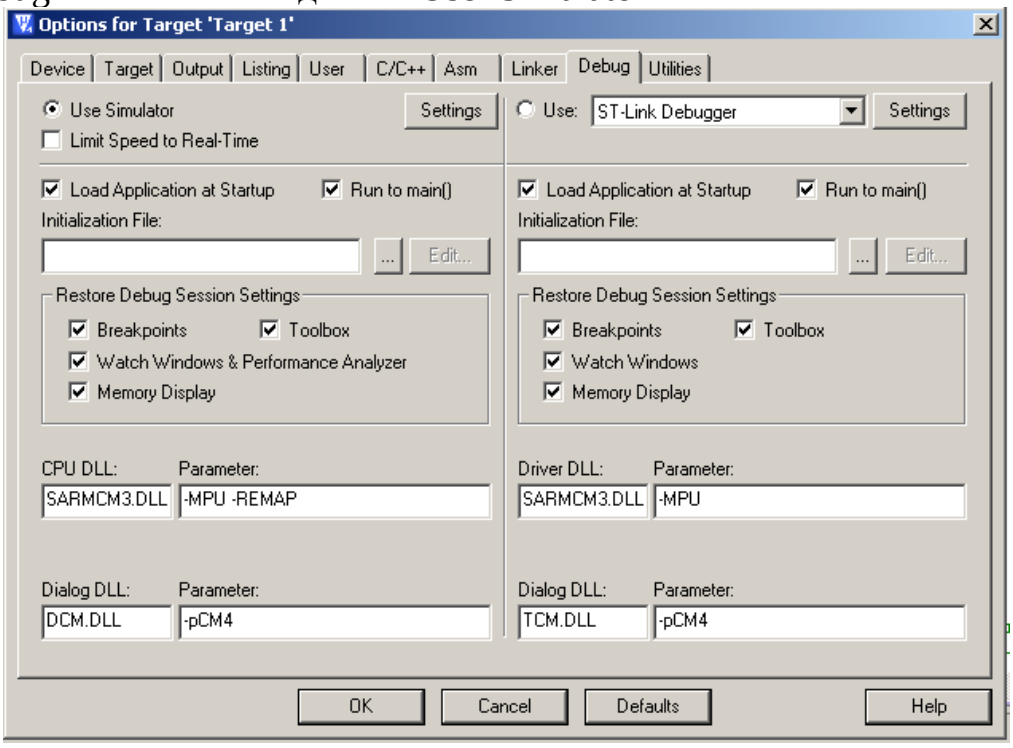
TIM_Prescaler=4000;
TIM_CounterMode=TIM_CounterMode_Down;
TIM_Period=36000;
TIM_ClockDivision=TIM_CKD_DIV1;

3. Функцией TIM_TimeBaseInit инициализировать таймер TIM3
4. Функцией TIM_ITConfig разрешить запрос прерываний от обновления таймера
5. Функцией NVIC_EnableIRQ(TIM3_IRQn) разрешить прерывания от таймера 3
6. Разрешить работу таймера TIM3 функцией TIM_Cmd

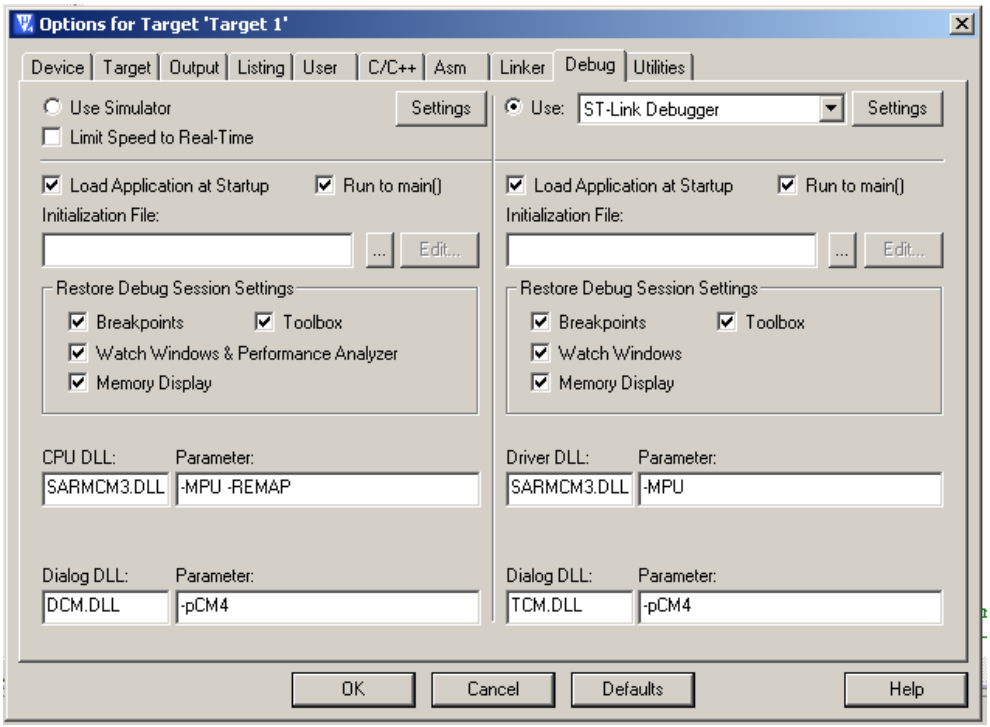
Отладка программ в среде Keil

Инструментальная среда разработки Keil предоставляет широкие возможности по отладке кода программ. Благодаря этому имеется возможность получать информацию о корректности выполнения программы, проводить всевозможные тесты, не используя при этом микросхему, для которой предназначается разрабатываемое ПО, что значительно экономит затрачиваемое время, а также материальные и другие ресурсы.

В этом случае необходимо выбрать в окне «Options for Target …» папку «Debug» и отметить надпись «User Simulator» в левой половине окна.



В то же время отладчик позволяет работать и с аппаратной поддержкой, что позволяет учесть аппаратные особенности системы. Для настройки этого режима следует в том же окне поставить галочку в правой половине окна, выбрав из списка «ST-Link Debugger».



Основные функции, доступные в режиме отладки

Все доступные функции можно выбрать в папке “Debug” (рисунок 6).

Для некоторых наиболее часто используемых функций можно воспользоваться быстрыми клавишами, которые подписаны рядом с соответствующими пунктами меню. Все функции меню “Debug” продублированы на панели инструментов для более удобного доступа к ним. Опишем эти функций:

Run: Запускает выполнение программы до первой точки останова (breakpoint), либо до принудительной остановки выполнения.

Stop: Останавливает процесс выполнения программы.

Step: Данная функция позволяет перейти к выполнению следующей команды. В случае если текущей командой является команда вызова подпрограммы (например: call), то указатель переместится на первую инструкцию в этой подпрограмме.

Step Over: Действует аналогично предыдущей функции, но не переходит внутрь подпрограмму, а проходит через неё на следующую инструкцию.

Step Out: Позволяет выйти из текущей подпрограммы и перейти на следующую инструкцию после вызова этой подпрограммы.

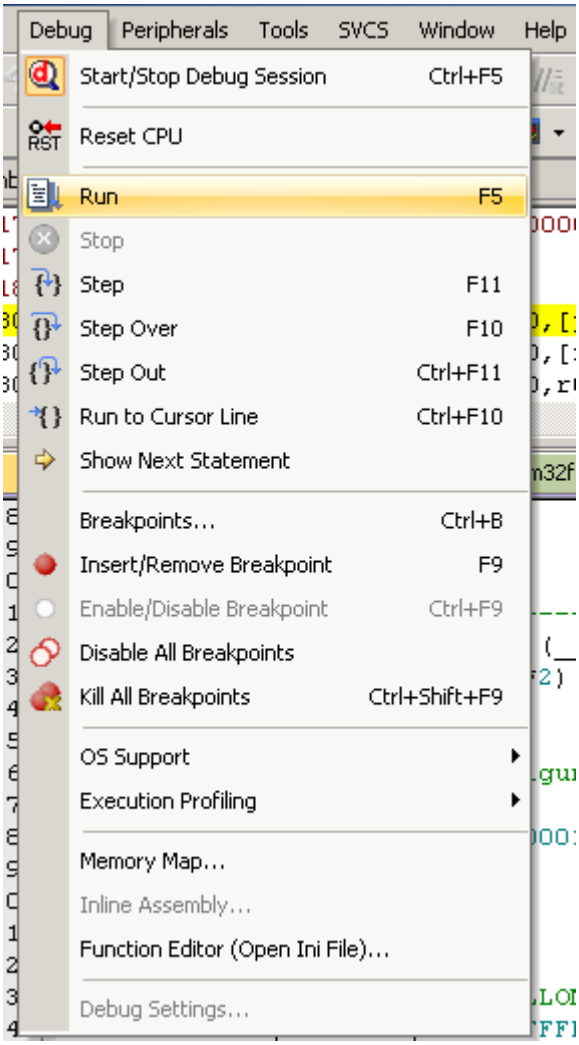


Рисунок 6. – Папка “Debug”

Run to Cursor line: Данная функция аналогична функции “Run”, но выполнение происходит до места, в котором находится курсор. Этой функцией также можно воспользоваться, открыв в окне кода контекстное меню и выбрав там пункт “ Run to Cursor line ”.

Insert/Remove Breakpoint: позволяет выставить в коде программы точки останова, в которых выполнение программы будет останавливаться.

Enable/Disable Breakpoint: Позволяет включить или выключить точку останова. Эта функция полезна, когда точка останова в данный момент не нужна, но может пригодиться в дальнейшем.

Kill All Breakpoints: Убирает все точки останова из программы.

Часто для анализа работы программы необходимо знать, как устройство взаимодействует с другими устройствами через свои периферийные интерфейсы. Для этого можно воспользоваться элементами меню “Peripherals”. Вывод содержимого памяти устройства

Для просмотра памяти устройства нужно воспользоваться окном “Memory Window”, которое можно найти в папке “View” (рисунок 7). В этой

же папке имеются и другие возможности просмотра нужной информации вплоть до логического анализатора.

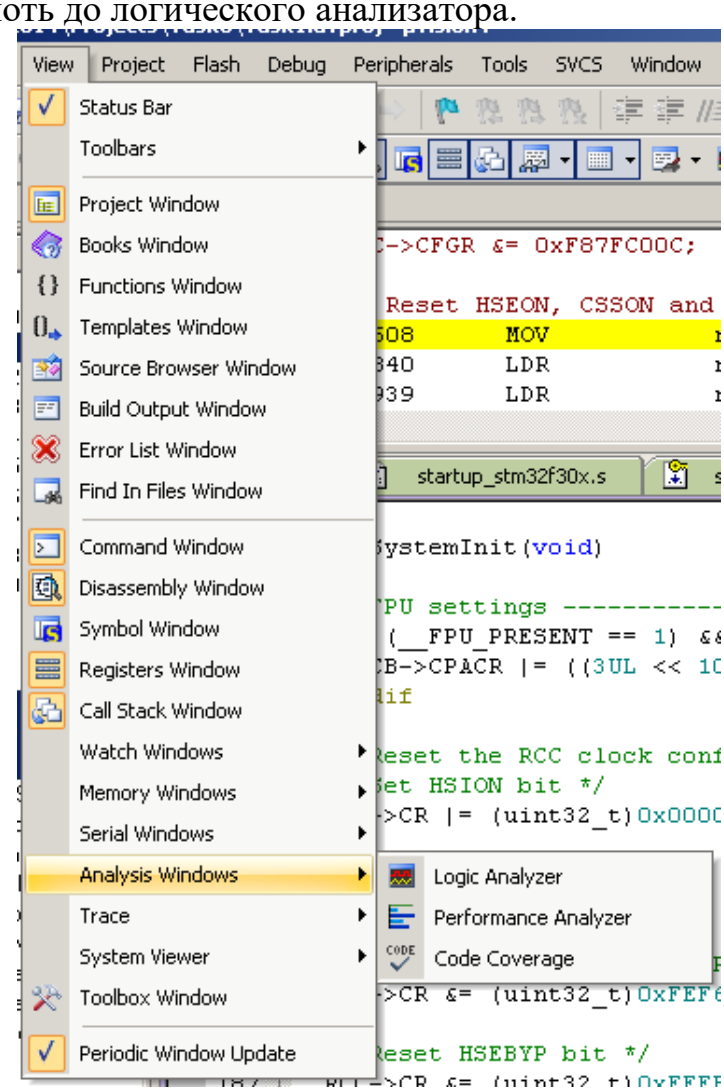


Рисунок 7. – Папка “View”

В отладчике можно просматривать переменные непосредственно при исполнении программы. Для этого нужно добавить окно просмотра. Для этого после запуска отладчика, до запуска программы, открыть окно просмотра переменных View\Watch window.

В открывшемся окне можно ввести имя переменной, и в нём же отобразится её значение. Глобальные переменные видны всегда. Локальные переменные могут быть недоступны из-за оптимизации.

Также в отладчике можно посмотреть состояние регистров периферии. Для этого выберите View\System viewer\Нужная периферия.

Порядок выполнения лабораторной работы

Поскольку задание достаточно сложное предусмотрен вариант анализа готовой программы, если с программированием возникнут проблемы. В процессе составления программы необходимо использовать средства отладки.

Необходимо выполнить следующие пункты:

1. Написать обработчик прерываний от SysTick и отконфигурировать его таким образом, чтобы светодиод LD6 загорался с частотой 5 Гц. Учесть, что такой большой период напрямую задать не получится – нужно сделать программный делитель.
   При входе в прерывание зажигать светодиод LD4, при выходе из прерывания – гасить его
2. Убедиться что светодиоды LD7, LD6 мигают как положено.
3. Добавить в тело прерывания от таймера 3 (TIM3) программную задержку:

```
uint32_t ps;
…
for (ps=0;ps<5000000;ps++)
```

4. Убедиться, что внутри прерывания от Таймера-3 прерывание от системного таймера не вызывается.
5. Установить приоритеты прерываний SysTick и TIM3 таким образом, чтобы внутри прерываний от Таймера-3 вызывались прерывания от SysTick.

По результатам работы составляется отчёт.

Возможный вариант программы предлагается на рисунке 8.

```
#include <stm32f30x.h>

void SysTick_Handler(void) {
  static uint8_t a=0;
  static uint8_t dd=10;
//  uint32_t ps;
  GPIO_SetBits(GPIOE,GPIO_Pin_8);
  if (!(--dd)) {
    dd=10;
    if (a) {
      a=0;
      GPIO_SetBits(GPIOE,GPIO_Pin_15);
    } else {
      a=1;
      GPIO_ResetBits(GPIOE,GPIO_Pin_15);
    }
  }
//  for (ps=0;ps<50000;ps++);
  GPIO_ResetBits(GPIOE,GPIO_Pin_8);
}

uint8_t prt;

void TIM3_IRQHandler(void) {
  static uint8_t a=0;
```

```
  uint32_t ps;
  TIM_ClearFlag(TIM3,TIM_IT_Update);
  GPIO_SetBits(GPIOE,GPIO_Pin_10);
  if (a) {
    a=0;
    GPIO_SetBits(GPIOE,GPIO_Pin_11);
  } else {
    a=1;
    GPIO_ResetBits(GPIOE,GPIO_Pin_11);
  }
  for (ps=0;ps<5000000;ps++);
  GPIO_ResetBits(GPIOE,GPIO_Pin_10);
}

int main(void) {
  GPIO_InitTypeDef GPIO_InitStruct; // Структура для настройки порта
  TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStruct;
// Инициализация порта со светодиодами
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOE,ENABLE);
  GPIO_InitStruct.GPIO_Mode=GPIO_Mode_OUT;
  GPIO_InitStruct.GPIO_OType=GPIO_OType_PP;

GPIO_InitStruct.GPIO_Pin=GPIO_Pin_9|GPIO_Pin_11|GPIO_Pin_13|GPIO_Pin_
15
                         |GPIO_Pin_8|GPIO_Pin_10|GPIO_Pin_12|GPIO_Pin_14;
  GPIO_InitStruct.GPIO_PuPd=GPIO_PuPd_NOPULL;
  GPIO_InitStruct.GPIO_Speed=GPIO_Speed_2MHz;
  GPIO_Init(GPIOE,&GPIO_InitStruct);
// Инициализация кнопки
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA,ENABLE);
  GPIO_InitStruct.GPIO_Mode=GPIO_Mode_IN;
  GPIO_InitStruct.GPIO_Pin=GPIO_Pin_0;
  GPIO_Init(GPIOA,&GPIO_InitStruct);

  SysTick_Config(SystemCoreClock/100);
  NVIC_SetPriority(SysTick_IRQn,0);

// Таймер 3
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3,ENABLE);
  TIM_TimeBaseInitStruct.TIM_Prescaler=4000;
  TIM_TimeBaseInitStruct.TIM_CounterMode=TIM_CounterMode_Down;
  TIM_TimeBaseInitStruct.TIM_Period=36000;
  TIM_TimeBaseInitStruct.TIM_ClockDivision=TIM_CKD_DIV1;
  TIM_TimeBaseInitStruct.TIM_RepetitionCounter=0; // Можно не заполнять
  TIM_TimeBaseInit(TIM3,&TIM_TimeBaseInitStruct);
```

```
  TIM_ITConfig(TIM3,TIM_IT_Update,ENABLE);
  NVIC_EnableIRQ(TIM3_IRQn);
  NVIC_SetPriority(TIM3_IRQn,2);
  TIM_Cmd(TIM3,ENABLE);

  prt=NVIC_GetPriority(SysTick_IRQn);
// Включение светодиодов
  for (;;) {
  };
}
```

Рисунок 8. Вариант программы

Контрольные вопросы

1. Каково число таймеров общего применения в контроллере?
2. Каково назначение таймера SysTick и способы его применения?
3. Назначение контроллера NVIC?
4. Каково количество внешних векторных прерываний, которое может обслужить контроллер прерываний?
5. Какова разрядность таймеров общего применения и SysTick?
6. Сколь быстро обслуживает прерывание контроллер?
7. Какие задачи позволяет решать отладчик?
8. Каково число уровней приоритета?

Лабораторная работа №4

Изучение методов программирования и использования АЦП

Цель работы: знакомство с особенностями использования и программирования АЦП, его калибровки, программирование прямого доступа к памяти.

Общие сведения

Для обработки аналоговых сигналов контроллер имеет:

- четыре 12-ти битных АЦП,
- два 12-ти битных ЦАП
- 4 rail-to-rail операционных усилителя, работающих в том числе с известным коэффициентом усиления (Programmable Gain Amplifier). Также есть возможность переключения входов синхронно с ШИМ (реализация синхронного детектора, например).
- 7 «быстрых» компараторов.
- 24 канала измерения ёмкости, однако данная схема не является точной (предназначена для построения емкостной клавиатуры).

Есть варианты контроллеров с 16-ти битным сигма-дельта АЦП (stm32f373). Analog devices выпускает контроллеры с ядром ARM и более развитыми аналоговыми средствами, в том числе 24-х битным АЦП.

Аналого-цифровой преобразователь

АЦП предназначен для измерения входного напряжения в диапазоне Vss…Vref+. Опорные напряжения общие для всех 4-х каналов, минимальное значение Vref+ 1.8В. В зависимости от корпуса, Vref+ может быть не выведено наружу, при этом за опору берется аналоговое напряжение питания Vdda.

На входе каждого канала АЦП имеется аналоговый мультиплексор, позволяющий подключать канал к одному из нескольких источников: аналоговым входам, внутренним источникам:

- опорное напряжение Vref
- датчик температуры
- ½ напряжения на входе Vbat
- Выходы внутренних ОУ контроллера

Имеется возможность использования дифференциальных входов. Фактически, большинство входов – дифференциальные, но в обычном режиме отрицательный вход подключен к Vss=0В.

Имеется возможность синхронизировать каналы для измерения двух сигналов синхронно.

Имеется возможность программирования «аналоговых сторожевых таймеров» - схем, генерирующих прерывание при выходе аналогового сигнала за некоторые рамки.

Максимальная скорость измерения АЦП – 5.1Ms/s (5 миллионов отсчетов в секунду). Скорость измерения зависит от тактовой частоты АЦП.

Запуск АЦП может производиться программно, циклически, от внешних сигналов, от внутренних событий (например, от таймера). Каждый канал АЦП имеет специальный блок формирования последовательности измеряемых каналов, позволяющий по одному сигналу запуска провести серию измерений с разных входов мультиплексора.

Последовательность измеряемых входов есть «обычная» и «добавленная» (injected). Т.е. возможно в ряд постоянных измерений вклинить дополнительную последовательность входов.

Как и почти вся периферия, выход АЦП может быть подключен к контроллеру прямого доступа к памяти (DMA) для автоматического размещения результата в ОЗУ.

Возможно два варианта выравнивания – по правой и по левой части 16-ти битного значения.

В АЦП есть флаг «Overrun» - если он установлен, новое значение будет стирать старое значение в регистре DR, даже если оно ещё не считано.

Рассматривать будем один из самых простых вариантов использования АЦП – измерение одного аналогового сигнала и температуры кристалла.

Одно измерение АЦП включает, как правило, следующие фазы:

1. Выбор требуемого канала – переключение внутреннего мультиплексора.
2. Переходные процессы (время выборки) – заряд ёмкости схемы АЦП. Зависят от внутреннего сопротивления источника сигнала. АЦП данного контроллера имеет весьма высокое входное сопротивление – до 100к, в зависимости от требуемой скорости оцифровки сигнала (если есть желание получить максимальную скорость и максимальную точность – входное сопротивление падает до 18 Ом). В документации приведены значения времени выборки для разных сопротивлений подключенной аналоговой схемы.
3. Запуск измерения – может выполняться и до пункта (2), если это позволяет АЦП. В данном контроллере – позволяет.
4. Измерение (как правило, по 1 циклу частоты АЦП на разряд).

Можно сказать, что АЦП производит сравнение входного напряжения с опорным. Результат работы АЦП – значение A – соответствует соотношению:

$A/(2^n) = Vin/Vref+$ (1)

Здесь n – число двоичных разрядов АЦП

A – результат работы АЦП

Vin – входное напряжение

Vref+ - опорное напряжение

Отсюда можно вычислить значение Vin:

$Vin=A * Vref+ / (2^n)$ (2)

Если в схеме использован внешний стабильный опорный источник Vref+ - проблем нет. Но если использован обычный стабилизатор, или даже выхода Vref+ нет, и используется напряжение питания – зависимость результата от температуры будет слишком большой.

При использовании АЦП присутствует ряд погрешностей, основные:

- нестабильность опорного напряжения от времени
- нестабильность опорного напряжения от температуры
- погрешность квантования
- погрешности самого АЦП – смещение, ошибка масштаба, нелинейность. Чаще всего эти ошибки перекрывают ошибку квантования и могут быть учтены как «фактически имеющееся разрешение». Для используемого АЦП, например, заявлено 12 бит разрядность, но при этом в зависимости от требуемой скорости реально разрешение будет от 10.3 до 11.7 бит.

Для исключения (уменьшения) температурной зависимости источника Vdda, а также для исключения необходимости калибровки в контроллере есть внутренний **опорный источник Vref** с напряжением =1.2±0.04В (3.5%) и температурным коэффициентом 100ppm. Перед использованием этот опорный источник **нужно включить**, после чего можно измерить значение данного напряжения с помощью АЦП. При расчетах можно использовать значение кода АЦП, соответствующее данному напряжению при Vref=3.3В, измеренное при производстве контроллера.

Оно хранится по адресу 0x1FFF F7BA - 0x1FFF F7BB (два байта).

Для доступа к данной константе можно использовать например такое выражение: VRet=(*(uint16_t*)(0x1FFFF7BA)).

Измерив два значения, результат АЦП преобразования входного сигнала A и результат АЦП преобразования Vref VR, вместе с эталонным значением VRet, можно получить значение входного напряжения с основной погрешностью порядка 0.03% и дополнительной температурной погрешностью 0.1%. Процесс пересчета можно представить например как расчет точного значения Vref через значение VRet, затем расчет текущего значения Vdda через значения VR и точного Vref, и расчет Vin по формуле (2). Соответствующие формулы:

$VRet/(2^n) = Vref/3.3$ –> $Vref=3.3*VRet/(2^n)$

$VR/(2^n) = Vref/Vref+$ -> $Vref+=Vref*(2^n)/VR$

$Vin=A * Vref+ / (2^n)$

Всё это можно сократить:

$Vin=A * 3.3*VRet/((2^n) *VR)$

В контроллер встроен датчик температуры, подключенный к ADC1_IN16. Точность датчика – ±1.5°C. Для использования его также необходимо включить специальной командой.

Температура рассчитывается по формуле:

$((V_{25}-V_{TS})/AVG\_Slope)+25$

V_{TS} - Напряжение на термодатчике

V_{25} - Калибровочное напряжение на термодатчике при 25 градусах

AVG_Slope - Наклон кривой термодатчка

V_{25} и AVG_Slope можно найти в документации.

Также, в контроллер прошиты две калибровочные точки температурного датчика, измеренные при изготовлении микросхемы:

0x1FFF F7B8 - 0x1FFF F7B9 – значение кода АЦП при питании 3.3В и температуре 30°C

0x1FFF F7C2 - 0x1FFF F7C3 – значение кода АЦП при питании 3.3В и температуре 110°C

По данным значениям можно рассчитать текущую температуру кристалла.

Для учёта индивидуальных особенностей АЦП необходимо перед включением АЦП провести калибровку нуля и смещения. Это специальная операция, описанная в руководстве.

Процесс включения АЦП

1. Подать тактовую частоту на АЦП командой RCC_ADCCLKConfig. Частота подаётся с PLL, нужно выбрать делитель. В простейшем случае выбирается максимальный делитель.
2. Подключить АЦП к шине командой RCC_AHBPeriphClockCmd().
3. Настроить вывод, используемый АЦП, как аналоговый вход.
4. Настроить АЦП функцией ADC_Init(). На этом этапе при заполнении структуры нужно выбрать разрядность результата, выравнивание результата, всё неизвестное – запрещать.
5. Включить регулятор напряжения АЦП.
6. Наверное, не мешает выполнить паузу для установления напряжения АЦП. Но про это нигде не сказано, поэтому переходим к 7.
7. Запустить калибровку АЦП. Предварительно можно выбрать режим – дифференциальный или обычный.
8. ОБЯЗАТЕЛЬНО дождаться окончания калибровки – можно использовать строчку while (ADC_GetCalibrationStatus(ADC1));
9. Включить АЦП командой ADC_Cmd(ADC1,ENABLE)
10. ОБЯЗАТЕЛЬНО дождаться включения АЦП, контролируя готовности, например строчкой
    while (!ADC_GetFlagStatus(ADC1,ADC_FLAG_RDY));
11. Настроить подключенные каналы.
12. Настроить DMA если используется. Для настройки DMA не забыть подключить её к шине и включить канал после настройки.
13. Запустить измерение, либо выбрать источник запуска

Контроллер прямого доступа к памяти DMA

Контроллер прямого доступа в память позволяет переписывать данные из периферии в память, либо наоборот, по событиям, без участия ядра. Возможно также копирование из памяти в память.

Контроллер имеет два блока DMA, в каждом – 12 каналов. Для настройки каждого канала нужно задать ряд параметров:

- Адрес источника
- Адрес приемника
- Шаг прибавления адреса источника
- Шаг прибавления адреса приемника
- Размер буфера приемника
- Размер буфера передатчика

Каждый канал имеет ряд подключенной периферийной аппаратуры, при этом связь задана жестко. Например АЦП1 подключен всегда к каналу 1. К тому же каналу 1 подключено ещё много другой аппаратуры. Поэтому разрешать использование каналов DMA нужно с оглядкой, не занят ли канал другой периферией.

Канал может работать как в однократном, так и в циклическом режиме.

Цифроаналоговый преобразователь

Использовать ЦАП в простейшем случае очень просто. Код ЦАП всегда задаётся от 0 до Vref+. ЦАП имеет выходной буфер, который может быть включен или выключен. Включенный буфер понижает выходное сопротивление, но при этом вносит ограничения на сигнал – вблизи нуля и напряжения питания сигнал ухудшается.

Для использования достаточно проинициализировать, затем задавать выходное значение командой DAC_SetChannel1Data. Хотя ЦАП имеет также встроенные средства генерации сигналов различной формы, также, к ЦАП может быть подключен DMA для выдачи сигнала из памяти.

Порядок выполнения работы

При выполнении работы используется АЦП 1, у которого активизируется в качестве входа вывод порта GPIOA_pin_0. К этому входу подключается потенциометр внешним монтажом для задания измеряемого напряжения. Работа выполняется в несколько этапов.

<u>1) Простое включение АЦП – проект Task4_1</u>

Проект нужно дописать – заполнить пропущенное.

Основа в файле Task4_1 (рисунок 9).

1. Включить и настроить АЦП на правое выравнивание результата, все injected режимы отключить, использовать однократный запуск.
2. Настроить только один канал, вход с RA0
3. В цикле запускать АЦП и дожидаться окончания измерения (сброса флага измерения). Например, написать в основной программе такой код:

```
ADC_StartConversion(ADC1);
```

while (ADC_GetStartConversionStatus(ADC1));

4. В отладчике добавить регистры АЦП на просмотр и убедиться, что регистр измеренного значения DR изменяется при вращении потенциометра.

2) Использование прямого доступа в память

Используется проект Task4_2

Файл проекта Task4_2 предложен на рисунке 10.

1. Проект нужно изменить. Вместо одного канала, нужно настроить считывание трёх каналов: ADC1, температура, опорное напряжение. Не забыть включить термодатчик и опорное напряжение. Значения также должны получаться в массиве adrez.
2. В зависимости от положения переменного резистора выводить результат на светодиоды в двоичном виде, либо включать один из светодиодов, положение которого определяется уровнем сигнала. Для определения момента расчёта можно использовать синхронизацию по прерыванию от флага EOS. Как более легкий вариант – рассчитывать значения постоянно или с паузой.
3. Дописать расчёт входного напряжения, пользуясь считанной константой VRet и текущим измерением опорного напряжения. Расчёт вести во float, результат отобразить в глобальную переменную и просмотреть в отладчике. Проверить вольтметром.
4. Дописать расчёт текущей температуры в комнате, также во float, вывод – в глобальную переменную.

4) Сделать анализ проекта Task4 (Рисунок 11)

Введите в программу дополнительные пояснения. Выполните программирование платы и убедитесь в вашей правоте. Если необходимо, измените программу для получения нужного результата.

Контрольные вопросы

1. Каково количество АЦП в используемом контроллере?
2. Какова разрядность АЦП и каково быстродействие?
3. Зачем после включения АЦП требуется время до момента запуска процедуры измерения?
4. От чего зависит диапазон измерения напряжения?
5. Что такое прямой доступ к памяти?
6. В каком регистре формируется результат измерения?

Task4_1

```
#include <stm32f30x.h>

int main(void) {
  GPIO_InitTypeDef GPIO_InitStruct; // Структура для настройки порта
  ADC_CommonInitTypeDef ADC_CommonInitStruct;
  ADC_InitTypeDef  ADC_InitStruct;
// Инициализация порта со светодиодами
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOE|RCC_AHBPeriph_GPIOA,ENABLE);
  GPIO_InitStruct.GPIO_Mode=GPIO_Mode_OUT;
  GPIO_InitStruct.GPIO_OType=GPIO_OType_PP;
  GPIO_InitStruct.GPIO_Pin=GPIO_Pin_9|GPIO_Pin_11|GPIO_Pin_13|GPIO_Pin_15
                          |GPIO_Pin_8|GPIO_Pin_10|GPIO_Pin_12|GPIO_Pin_14;
  GPIO_InitStruct.GPIO_PuPd=GPIO_PuPd_NOPULL;
  GPIO_InitStruct.GPIO_Speed=GPIO_Speed_2MHz;
  GPIO_Init(GPIOE,&GPIO_InitStruct);
// Вход АЦП
  GPIO_InitStruct.GPIO_Mode=---;
  GPIO_InitStruct.GPIO_OType=GPIO_OType_PP;
  GPIO_InitStruct.GPIO_PuPd=GPIO_PuPd_NOPULL;
  GPIO_InitStruct.GPIO_Speed=GPIO_Speed_Level_1;
  GPIO_InitStruct.GPIO_Pin=GPIO_Pin_0;
  GPIO_Init(GPIOA,&GPIO_InitStruct);
// Тактирование АЦП


  RCC_AHBPeriphClockCmd(---,ENABLE);
  RCC_ADCCLKConfig(RCC_ADC12PLLCLK_Div256);

// Калибровка АЦП -
  ADC_DeInit(ADC1);
  ADC_StructInit(&ADC_InitStruct);
  ADC_InitStruct.ADC_ContinuousConvMode=ADC_ContinuousConvMode_Disable;
  ADC_InitStruct.ADC_Resolution=ADC_Resolution_12b;
  ADC_InitStruct.ADC_DataAlign=ADC_DataAlign_Right;
  ADC_InitStruct.ADC_OverrunMode=ADC_OverrunMode_Enable;
  ADC_InitStruct.ADC_AutoInjMode=ADC_AutoInjec_Disable;
  ADC_InitStruct.ADC_NbrOfRegChannel=1;
  ADC_Init(ADC1, &ADC_InitStruct);

  ADC_VoltageRegulatorCmd(ADC1, ENABLE);
  ADC_StartCalibration(ADC1);
// Ожидаем окончания калибровки
  while (ADC_GetCalibrationStatus(ADC1));
// Разрешаем работу АЦП
  ADC_Cmd(---);
  while (!ADC_GetFlagStatus(ADC1,ADC_FLAG_RDY));

// Конфигурируем каналы
  ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 1, ADC_SampleTime_61Cycles5);
```

```
  for (;;) {
    uint32_t n;
    if (GPIO_ReadInputDataBit(GPIOE,GPIO_Pin_8))
      GPIO_ResetBits(GPIOE,GPIO_Pin_8);
    else
      GPIO_SetBits(GPIOE,GPIO_Pin_8);
///
    ADC_StartConversion(---);
    while (ADC_GetStartConversionStatus(ADC1));
    for (n=0;n<500000;n++);
  };
}
```

Рисунок 9. – Файл проекта Task4_1

Task4_2

```
#include <stm32f30x.h>

uint16_t adrez[1];

int main(void) {
  DMA_InitTypeDef DMA_InitStructure;
  GPIO_InitTypeDef GPIO_InitStruct; // Структура для настройки порта
  ADC_InitTypeDef  ADC_InitStruct;
// Инициализация порта со светодиодами
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOE|RCC_AHBPeriph_GPIOA,ENABLE);
  GPIO_InitStruct.GPIO_Mode=GPIO_Mode_OUT;
  GPIO_InitStruct.GPIO_OType=GPIO_OType_PP;
  GPIO_InitStruct.GPIO_Pin=GPIO_Pin_9|GPIO_Pin_11|GPIO_Pin_13|GPIO_Pin_15
                          |GPIO_Pin_8|GPIO_Pin_10|GPIO_Pin_12|GPIO_Pin_14;
  GPIO_InitStruct.GPIO_PuPd=GPIO_PuPd_NOPULL;
  GPIO_InitStruct.GPIO_Speed=GPIO_Speed_2MHz;
  GPIO_Init(GPIOE,&GPIO_InitStruct);
// Вход АЦП
  GPIO_InitStruct.GPIO_Mode=GPIO_Mode_AN;
  GPIO_InitStruct.GPIO_OType=GPIO_OType_PP;
  GPIO_InitStruct.GPIO_PuPd=GPIO_PuPd_NOPULL;
  GPIO_InitStruct.GPIO_Speed=GPIO_Speed_Level_1;
  GPIO_InitStruct.GPIO_Pin=GPIO_Pin_0;
  GPIO_Init(GPIOA,&GPIO_InitStruct);


// Тактирование АЦП
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_ADC12,ENABLE);
  RCC_ADCCLKConfig(RCC_ADC12PLLCLK_Div256);

// Настройка АЦП
  ADC_DeInit(ADC1);

  ADC_StructInit(&ADC_InitStruct);
  ADC_InitStruct.ADC_ContinuousConvMode=ADC_ContinuousConvMode_Enable;
  ADC_InitStruct.ADC_Resolution=ADC_Resolution_12b;
  ADC_InitStruct.ADC_DataAlign=ADC_DataAlign_Right;
```

```
  ADC_InitStruct.ADC_OverrunMode=ADC_OverrunMode_Enable;
  ADC_InitStruct.ADC_AutoInjMode=ADC_AutoInjec_Disable;
  ADC_InitStruct.ADC_NbrOfRegChannel=1;
  ADC_Init(ADC1, &ADC_InitStruct);

  ADC_VoltageRegulatorCmd(ADC1, ENABLE);
  ADC_StartCalibration(ADC1);
// Ожидаем окончания калибровки
  while (ADC_GetCalibrationStatus(ADC1));
// Разрешаем работу АЦП
  ADC_Cmd(ADC1,ENABLE);
  while (!ADC_GetFlagStatus(ADC1,ADC_FLAG_RDY));

// Датчик температуры и опорное напряжение
//  ADC_TempSensorCmd(ADC1, ENABLE);
//  ADC_VrefintCmd(ADC1, ENABLE);

// Конфигурируем каналы
  ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 1, ADC_SampleTime_61Cycles5);


  // Включение, начальная инициализация
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1,ENABLE);
     DMA_DeInit(DMA1_Channel1);
     DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)&(ADC1->DR);
     DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)adrez;
     DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
     DMA_InitStructure.DMA_BufferSize = 1;
     DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
     DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
     DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
     DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
     DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
     DMA_InitStructure.DMA_Priority = DMA_Priority_Medium;
     DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
     DMA_Init(DMA1_Channel1, &DMA_InitStructure);
// Enable ADC_DMA
     ADC_DMACmd(ADC1, ENABLE);
     ADC_DMAConfig(ADC1,ADC_DMAMode_Circular);
     DMA_Cmd(DMA1_Channel1, ENABLE);

    ADC_StartConversion(ADC1);


// Включение светодиодов
  for (;;) {
  };
}
```

Рисунок 10. – Файл проекта Task4_2

Task4

```
#include <stm32f30x.h>

uint16_t adrez[3];

int main(void) {
  DMA_InitTypeDef DMA_InitStructure;
  GPIO_InitTypeDef GPIO_InitStruct; // Структура для настройки порта
  ADC_InitTypeDef  ADC_InitStruct;
// Инициализация порта со светодиодами
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOE|RCC_AHBPeriph_GPIOA,ENABLE);
  GPIO_InitStruct.GPIO_Mode=GPIO_Mode_OUT;
  GPIO_InitStruct.GPIO_OType=GPIO_OType_PP;
  GPIO_InitStruct.GPIO_Pin=GPIO_Pin_9|GPIO_Pin_11|GPIO_Pin_13|GPIO_Pin_15
                          |GPIO_Pin_8|GPIO_Pin_10|GPIO_Pin_12|GPIO_Pin_14;
  GPIO_InitStruct.GPIO_PuPd=GPIO_PuPd_NOPULL;
  GPIO_InitStruct.GPIO_Speed=GPIO_Speed_2MHz;
  GPIO_Init(GPIOE,&GPIO_InitStruct);
// Вход АЦП
  GPIO_InitStruct.GPIO_Mode=GPIO_Mode_AN;
  GPIO_InitStruct.GPIO_OType=GPIO_OType_PP;
  GPIO_InitStruct.GPIO_PuPd=GPIO_PuPd_NOPULL;
  GPIO_InitStruct.GPIO_Speed=GPIO_Speed_Level_1;
  GPIO_InitStruct.GPIO_Pin=GPIO_Pin_0;
  GPIO_Init(GPIOA,&GPIO_InitStruct);


// Тактирование АЦП
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_ADC12,ENABLE);
  RCC_ADCCLKConfig(RCC_ADC12PLLCLK_Div256);


// Настройка АЦП
  ADC_DeInit(ADC1);

  ADC_StructInit(&ADC_InitStruct);
  ADC_InitStruct.ADC_ContinuousConvMode=ADC_ContinuousConvMode_Enable;
  ADC_InitStruct.ADC_Resolution=ADC_Resolution_12b;
  ADC_InitStruct.ADC_DataAlign=ADC_DataAlign_Right;
  ADC_InitStruct.ADC_OverrunMode=ADC_OverrunMode_Enable;
  ADC_InitStruct.ADC_AutoInjMode=ADC_AutoInjec_Disable;
  ADC_InitStruct.ADC_NbrOfRegChannel=3; // Связано с конфигурацией каналов
  ADC_Init(ADC1, &ADC_InitStruct);

  ADC_VoltageRegulatorCmd(ADC1, ENABLE);
  ADC_StartCalibration(ADC1);
// Ожидаем окончания калибровки
  while (ADC_GetCalibrationStatus(ADC1));
// Разрешаем работу АЦП
  ADC_Cmd(ADC1,ENABLE);
  while (!ADC_GetFlagStatus(ADC1,ADC_FLAG_RDY));
```

```
// Датчик температуры и опорное напряжение
  ADC_TempSensorCmd(ADC1, ENABLE);
  ADC_VrefintCmd(ADC1, ENABLE);

// Конфигурируем каналы
  ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 1, ADC_SampleTime_61Cycles5);
  ADC_RegularChannelConfig(ADC1, ADC_Channel_TempSensor, 2,
ADC_SampleTime_61Cycles5);
  ADC_RegularChannelConfig(ADC1, ADC_Channel_Vrefint, 3,
ADC_SampleTime_61Cycles5);


  // Включение, начальная инициализация
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1,ENABLE);
     DMA_DeInit(DMA1_Channel1);
     D MA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)&(ADC1->DR);
     DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)adrez;
     DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
     DMA_InitStructure.DMA_BufferSize = 3;
     DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
     DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
     DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
     DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
     DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
     DMA_InitStructure.DMA_Priority = DMA_Priority_Medium;
     DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
     DMA_Init(DMA1_Channel1, &DMA_InitStructure);
// Enable ADC_DMA
     ADC_DMACmd(ADC1, ENABLE);
     ADC_DMAConfig(ADC1,ADC_DMAMode_Circular);
     DMA_Cmd(DMA1_Channel1, ENABLE);

    ADC_StartConversion(ADC1);


// Включение светодиодов
  for (;;) {
  };
}
```

Рисунок 11. – Файл проекта Task4

Лабораторная работа №5

Изучение методов подключение внешних устройств

Цель работы: знакомство с особенностями использования и программирования таймеров для формирования сигнала с широтно-импульсной модуляцией (ШИМ), программирование вывода информации на семисегментный индикатор и подключение клавиатуры.

Общие сведения

Широтно-импульсная модуляция

ШИМ находит широкое применение для управления внешними устройствами типа электродвигателя, для формирования аналоговых сигналов.

Принцип формирования ШИМ сигнала поясняется на рисунке 12.

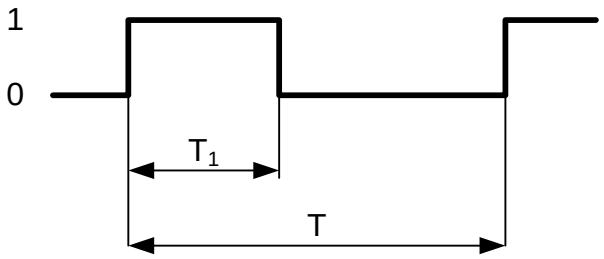


Рисунок 12. – Сигнал ШИМ

Сигнал имеет две основные характеристики: частоту и скважность.

Частота сигнала определяется его периодом T: F=1/T.

Скважность сигнала определяется соотношением полного периода и периода, когда сигнал активен: $K=T_1/T$

Такой сигнал широко применяется для:

- Формирования напряжения при отсутствии ЦАП, путём установки после ШИМ фильтра;
- Изменения яркости свечения индикации (светодиодов, индикаторов, подсветки). За счёт инерционности зрения человек не видит мигания, а видит изменение яркости;
- Управления скоростью электродвигателей;
- Управления импульсными источниками питания и другими преобразователями. Активный период ШИМ открывает ключ, а изменение скважности позволяет изменять характеристики преобразователя.

С помощью микроконтроллера можно формировать ШИМ программно (когда его частота невелика) и аппаратно, с помощью таймера.

Программно ШИМ можно реализовать либо с помощью также таймера и схемы сравнения, но с использованием прерываний вместо изменения состояния ножки, либо с помощью только таймера, выдающего прерывания с гораздо большей, чем частота ШИМ, частотой.

В любом случае нужно понимать, что при цифровом формировании ШИМ всегда будет иметь место дискретность возможных состояний ШИМ (рисунок 13, тонкие линии):

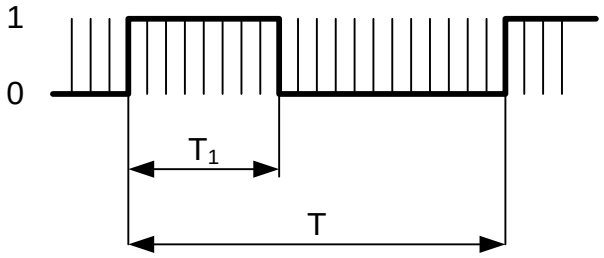
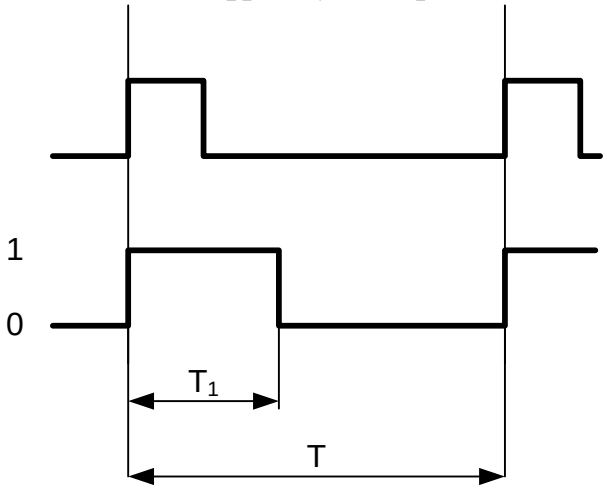


Рисунок 13. – Дискретные моменты времени, в которые возможна смена состояний ШИМ.

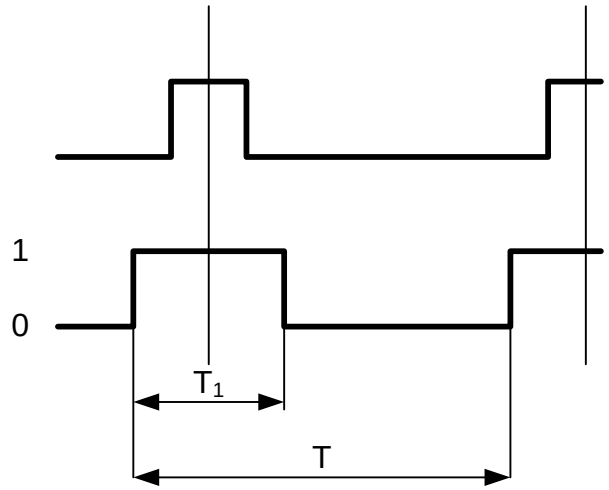
Из-за дискретности, скважность может иметь строго определенный набор вариантов. Например, для рисунка 10 возможны всего лишь 20 коэффициентов заполнения.

При программном формировании ШИМ, в зависимости от кода программы и выбранного варианта формирования, нужно учитывать, что после смены состояния выхода из-за программных задержек формируется время, в течении которого повторная смена состояния невозможна (например, контроллеру нужно выйти из процедуры обработки прерывания и войти повторно, для обратной смены полярности). Поэтому, если данное время окажется больше, чем дискретность коэффициента заполнения, нужно ограничивать минимальное и максимальное значения коэффициента заполнения. Если значение менее минимального – сразу выдавать только «ноль», если больше максимального – сразу выдавать «единицу». При формировании ШИМ таймером данной проблемы нет.

При использовании одного канала ШИМ больше никаких хитростей нет. Но при использовании параллельно нескольких каналов может потребоваться их выравнивание. Основные варианты приведены на рисунке 14: выравнивание по фронту и выравнивание по центру.



а) выравнивание по фронту



б) выравниване по центру

Рисунок 14. – Методы выравнивания при формировании ШИМ сигнала

Формирование сигнала ШИМ таймером микроконтроллера STM32F303 связано с использованием схемы сравнения таймера. Возможны различные варианты генерации ШИМ, в том числе с задаваемым сдвигом по фазе между каналами. Будем использовать самый простой режим: PWM edge-aligned mode.

Период ШИМ определяется регистром перезагрузки таймера TIMx_ARR (рисунок 15). Чем больше будет данное значение, тем больше будет дискрет ных значений коэффициента заполнения, тем больше будет точность задания сигнала.

Коэффициент заполнения ШИМ K определяет значение регистра сравнения таймера TIMx_CCRy:

TIMx_CCRy=TIMx_ARR*K

Можно заметить, что ставить значение TIMx_CCRy больше TIMx_ARR бесполезно.

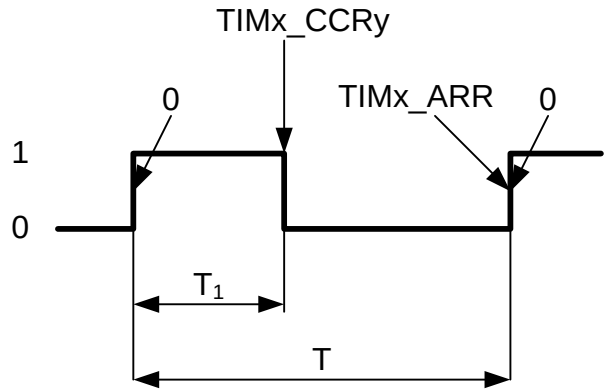


Рисунок 15. Назначения регистров TIMx_CCRy и TIMx_ARR

При изменении коэффициента заполнения или периода ШИМ, то есть при изменении регистров TIMx_CCRy или TIMx_ARR, возможны ошибки, приводящие к генерации ложных импульсов. Например, пусть период таймера равен 100, текущее заданное значение регистра сравнения, то есть ширина импульса – 50. Пусть необходимо уменьшить значение ширины до 20. Если выполнять прямую запись в регистр TIMx_CCRy, возможна

ситуация, что в момент записи значение 20 счетчиком уже пройдено, а значение 50 – ещё не достигнуто. При этом данный период вместо 20 или хотя бы 50% заполнения будет заполнен полностью, т.к. совпадения таймера и регистра сравнения не произойдет. Для исключения такой ситуации каждый из регистров TIMx_CCRy и TIMx_ARR состоит из буферного и теневого регистра. Если буферные регистры разрешены, при записи в регистры TIMx_CCRy и TIMx_ARR на самом деле запись будет производиться в буферные регистры, а перезапись в теневые регистры будет производиться при переполнении таймера (досчете до TIMx_ARR).

Для включения данных регистров нужно использовать флаги:

OCxPE регистра TIMx_CCMRy для включения буферного регистра TIMx_CCRy

ARPE регистра TIMx_CR1 для включения буферного регистра TIMx_ARR

Для принудительной записи из буферных регистров в теневые нужно установить флаг UG регистра TIMx_EGR.

Порядок программирования таймера

1. Конфигурируем ножку, соответствующую выходу схемы сравнения таймера, на альтернативную функцию, и выбираем альтернативную функцию.

   Конфигурация выполняется как обычно функцией GPIO_Init.

   Альтернативная функция ножки выбирается функцией GPIO_PinAFConfig(…). При настройке альтернативной функции следует помнить, что функция выполняется для каждой ножки в отдельности, и что константа выбора номера ножки GPIO_PinSource… не совпадает с константами GPIO_Pins_...
2. Функцией TIM_TimeBaseInit инициализируем таймер, задав предделитель, счет в одну из сторон (вверх или вниз), значение регистра переполнения TIMx_ARR.
3. Функцией TIM_OCxInit(…) конфигурируем канал захвата/сравнения/шим таймера на работу в режиме ШИМ. Многие поля соответствующей структуры не используются, поэтому рекомендуется предварительно заполнить структуру функцией TIM_OCStructInit(…). В структуре нужно предварительно задать значение коэффициента заполнения ШИМ TIMx_CCRy.
4. Функцией TIM_OC…PreloadConfig() разрешаем буферизацию регистров TIMx_CCRy
5. Включаем таймер командой TIMx_Cmd(…).
6. Включаем канал генерации ШИМ командой TIM_CCxCmd(…)
7. Для изменения коэффициента заполнения ШИМ в программе используем функции TIM_SetCompare…(…)

Индикация

Для светодиодной семисегментной индикации рекомендуется выделить в ОЗУ буфер, по одному байту на разряд. При динамической индикации буфер необходим, при статической буфер позволит обновить все разряды в том случае, если программа заменила только часть из выводимой информации. Информацию в буфере выгодно хранить именно по одному биту на сегмент, тогда при выводе не будет требоваться дополнительное преобразование.

В практическом использовании индикации удобно, если она имеет интерфейс, аналогичный потоковым устройствам ввода-вывода, т.е. в индикацию можно выдавать символы последовательно, как на терминал. Например, это даёт возможность использовать функцию printf, перенаправив вывод на индикатор.

Для поддержки таких функций нужно хранить текущее положение позиции вывода («текстовый курсор»). При выводе нужно перекодировать информацию из символьной, например Win1251, в светящиеся или не светящиеся сегменты. Есть символы, требующие особого обращения – «точка», 0x13, 0x10, 0x08 (стирание последнего символа). Требуется решить, какие символы поддерживать, а какие – нет.

Так как может потребоваться вывести информацию в любое место индикатора, нужна функция установки текстового курсора. Также желательна функция стирания индикатора.

Отличием статической индикации, предлагаемой в качестве задания, является невозможность вывода части информации – только полный вывод всех сегментов. Поэтому приходится вводить дополнительную функцию, которая вызывается после обновления данных индикаторов в буфере (в ОЗУ) – данная функция выводит буфер на индикатор.

Применяемые драйверы светодиодов записывают информацию в сдвигающий регистр по нарастающему фронту.

Также по нарастающему фронту информация переписывается из сдвигающего регистра в защелки на выходе.

На вывод OE драйверов нужно подавать «ноль» для отображения информации. Данный вывод может быть подключен к ШИМ для регулировки яркости индикатора.

Простой алгоритм вывода целого числа N:

1. Указатель P устанавливается на младший выводимый разряд.
2. На каждом этапе в разряд, на который указывает указатель, выводится остаток от деления на 10 текущего значения N.
3. Затем рассчитывается новое значение N = N / 10, а указатель переводится на один разряд левее.
4. Если текущее значение N не равно нулю, процесс повторяется с 2-го пункта.
5. Разряды левее текущего добиваются «пробелами» до нужной длины выводимой информации.

Порядок выполнения работы

Работа выполняется в три этапа: формирование ШИМ сигнала для управления двухцветным светодиодом, управление семисегментным индикатором, считывание состояния клавиш клавиатуры. В помощь приводится программа Task5 (рисунок 16).

1. Формирование ШИМ сигнала.

   В качестве основы берется код занятия, получающий положение переменного резистора.

   Задание – включить двухцветный светодиод таким образом, чтобы его цвет свечения задавался переменным резистором.

   Светодиод подключаем к выводам PC6, PC7 контроллера. На данные ножки выдаётся два канала ШИМ с таймера TIM3.

   Частоту ШИМ нужно выбрать в пределах 100 Гц – 2 КГц.

   Коэффициенты заполнения обоих каналов должны изменяться в зависимости от положения переменного резистора, один – расти, другой – уменьшаться.

2. Управление семисегментным индикатором.

   Требуется выполнить два задания:

   1) Написать функции работы с индикатором. Вывести какое-нибудь осмысленное изображение;

   2) Написать программу для вывода целого числа на индикатор. Выводить значение кода АЦП, считанного с переменного резистора после каждого измерения.

3. Опрос состояния клавиш клавиатуры.

   Требуется выполнить два задания:

   1) Разработать процедуру опроса клавиатуры. Каждой клавише сопоставить символ от 1 до 6;
   2) Написать программу, выводящую очередную нажатую кнопку на индикатор по принципу «печатающей машинки» - смещение курсора после каждого нажатия. При достижении окончания разрядов индикатора, возвращаться на первый разряд индикатора.

Контрольные вопросы

1. Что такое ШИМ?
2. Для решения каких задач используется ШИМ?
3. Какое число ШИМ сигналов может сформировать контроллер одновременно?
4. Для какой цели используются резисторы подтяжки при подключении клавиатуры?
5. Сколько управляющих цепей требуется для управления четырёхразрядным семисегментным индикатором?

Task5_pwm

```
#include <stm32f30x.h>
#include <math.h>

#define BRIG GPIOC,GPIO_Pin_1
#define LOAD GPIOC,GPIO_Pin_3
#define SYNC GPIOA,GPIO_Pin_1
#define DATA GPIOA,GPIO_Pin_3

uint16_t adrez[1];

int32_t curpwm1=0;  // Коэффициент заполнения от 0 до 10000
uint32_t side=0;

void SetColor(float cl) {
  float red,green;
  red=cl;
  green=1-cl;
  TIM_SetCompare1(TIM3,10000.0*sqrt(red));
  TIM_SetCompare2(TIM3,10000.0*sqrt(green));
}

void InitLed() {
  GPIO_InitTypeDef GPIO_InitStruct; // Структура для настройки порта
  TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStruct;
  TIM_OCInitTypeDef TIM_OCI;

// Инициализация выходных ножек
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOC,ENABLE);
  GPIO_InitStruct.GPIO_Mode=GPIO_Mode_AF;
  GPIO_InitStruct.GPIO_OType=GPIO_OType_PP;
  GPIO_InitStruct.GPIO_Pin=GPIO_Pin_6|GPIO_Pin_7;
  GPIO_InitStruct.GPIO_PuPd=GPIO_PuPd_NOPULL;
  GPIO_InitStruct.GPIO_Speed=GPIO_Speed_2MHz;
  GPIO_Init(GPIOC,&GPIO_InitStruct);
  GPIO_PinAFConfig(GPIOC,GPIO_PinSource6,GPIO_AF_2);
  GPIO_PinAFConfig(GPIOC,GPIO_PinSource7,GPIO_AF_2);

// Таймер 3
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3,ENABLE);
  TIM_TimeBaseInitStruct.TIM_Prescaler=40;
  TIM_TimeBaseInitStruct.TIM_CounterMode=TIM_CounterMode_Down;
  TIM_TimeBaseInitStruct.TIM_Period=10000;
  TIM_TimeBaseInitStruct.TIM_ClockDivision=TIM_CKD_DIV1;
  TIM_TimeBaseInitStruct.TIM_RepetitionCounter=0; // Можно не заполнять
  TIM_TimeBaseInit(TIM3,&TIM_TimeBaseInitStruct);
  TIM_Cmd(TIM3,ENABLE);

  TIM_OCStructInit(&TIM_OCI);
  TIM_OCI.TIM_OCMode=TIM_OCMode_PWM1;
  TIM_OCI.TIM_Pulse=5000;
  TIM_OC1Init(TIM3,&TIM_OCI);
```

```
  TIM_OC1PreloadConfig(TIM3,TIM_OCPreload_Enable);
  TIM_OC2Init(TIM3,&TIM_OCI);
  TIM_OC2PreloadConfig(TIM3,TIM_OCPreload_Enable);
  TIM_CCxCmd(TIM3, TIM_Channel_1, TIM_CCx_Enable);
  TIM_CCxCmd(TIM3, TIM_Channel_2, TIM_CCx_Enable);

}

void SysTick_Handler(void) {
//  uint32_t ps;
  GPIO_SetBits(GPIOE,GPIO_Pin_8);
  if (!side) {
    curpwm1+=50;
    if (curpwm1>=10000) {
      curpwm1=10000;
      side=1;
    }
  } else {
    curpwm1-=50;
    if (curpwm1<=0) {
      curpwm1=0;
      side=0;
    }
  }
  SetColor(((float)curpwm1)/10000);
  if (GPIO_ReadInputDataBit(BRIG))
    GPIO_WriteBit(BRIG,Bit_RESET);
  else
    GPIO_WriteBit(BRIG,Bit_SET);
  GPIO_ResetBits(GPIOE,GPIO_Pin_8);
}

uint8_t prt;

/*void TIM3_IRQHandler(void) {
  static uint8_t a=0;
  uint32_t ps;
  TIM_ClearFlag(TIM3,TIM_IT_Update);
  GPIO_SetBits(GPIOE,GPIO_Pin_10);
  if (a) {
    a=0;
    GPIO_SetBits(GPIOE,GPIO_Pin_11);
  } else {
    a=1;
    GPIO_ResetBits(GPIOE,GPIO_Pin_11);
  }
  for (ps=0;ps<5000000;ps++);
  GPIO_ResetBits(GPIOE,GPIO_Pin_10);
}*/


#define IDTNUM 4  // Число разрядов
```

```
uint8_t idt[IDTNUM]; // Буфер информации на индикаторе
uint8_t idtpos=0;    // Позиция курсора

void InitSegm() {
  GPIO_InitTypeDef GPIO_InitStruct; // Структура для настройки порта
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA|RCC_AHBPeriph_GPIOC,ENABLE);
  GPIO_InitStruct.GPIO_Mode=GPIO_Mode_OUT;
  GPIO_InitStruct.GPIO_OType=GPIO_OType_PP;
  GPIO_InitStruct.GPIO_PuPd=GPIO_PuPd_NOPULL;
  GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz;
  GPIO_InitStruct.GPIO_Pin=GPIO_Pin_3|GPIO_Pin_1;
  GPIO_Init(GPIOA,&GPIO_InitStruct);
  GPIO_Init(GPIOC,&GPIO_InitStruct);
  idtpos=0;
}


void Repaint() {
  uint8_t obit=0x01;
  uint8_t opos=IDTNUM;
  GPIO_WriteBit(SYNC,Bit_RESET);
  do {
    opos--;
    obit=0x01;
    do {
      GPIO_WriteBit(DATA,(obit&idt[opos])?Bit_SET:Bit_RESET);
      GPIO_WriteBit(SYNC,Bit_SET);
      obit<<=1;
      GPIO_WriteBit(SYNC,Bit_RESET);
    } while (obit);
  } while (opos);
  GPIO_WriteBit(LOAD,Bit_SET);
  for (opos=0;opos<50;opos++);
  GPIO_WriteBit(LOAD,Bit_RESET);
}

const uint8_t digit[10]={0xFC,0x60,0xDA,0xF2,0x66,0xB6,0xBE,0xE0,0xFE,0xF6};

void OutDigit(uint32_t d) {
        int8_t p;
        uint8_t c;
        p=3;
        if (d>9999) d=9999;
        do {
          c=d%10;
                idt[p--]=digit[c];
                d=d/10;
  } while (d);
        while (p>=0) idt[p--]=0;
}
```

```
int main(void) {
  GPIO_InitTypeDef GPIO_InitStruct; // Структура для настройки порта
  DMA_InitTypeDef DMA_InitStructure;
  ADC_InitTypeDef  ADC_InitStruct;
// Инициализация порта со светодиодами
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOE,ENABLE);
  GPIO_InitStruct.GPIO_Mode=GPIO_Mode_OUT;
  GPIO_InitStruct.GPIO_OType=GPIO_OType_PP;
  GPIO_InitStruct.GPIO_Pin=GPIO_Pin_9|GPIO_Pin_11|GPIO_Pin_13|GPIO_Pin_15
                          |GPIO_Pin_8|GPIO_Pin_10|GPIO_Pin_12|GPIO_Pin_14;
  GPIO_InitStruct.GPIO_PuPd=GPIO_PuPd_NOPULL;
  GPIO_InitStruct.GPIO_Speed=GPIO_Speed_2MHz;
  GPIO_Init(GPIOE,&GPIO_InitStruct);

// Тактирование АЦП
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_ADC12,ENABLE);
  RCC_ADCCLKConfig(RCC_ADC12PLLCLK_Div256);

// Настройка АЦП
  ADC_DeInit(ADC1);

  ADC_StructInit(&ADC_InitStruct);
  ADC_InitStruct.ADC_ContinuousConvMode=ADC_ContinuousConvMode_Enable;
  ADC_InitStruct.ADC_Resolution=ADC_Resolution_12b;
  ADC_InitStruct.ADC_DataAlign=ADC_DataAlign_Right;
  ADC_InitStruct.ADC_OverrunMode=ADC_OverrunMode_Enable;
  ADC_InitStruct.ADC_AutoInjMode=ADC_AutoInjec_Disable;
  ADC_InitStruct.ADC_NbrOfRegChannel=1;
  ADC_Init(ADC1, &ADC_InitStruct);

  ADC_VoltageRegulatorCmd(ADC1, ENABLE);
  ADC_StartCalibration(ADC1);
// Ожидаем окончания калибровки
  while (ADC_GetCalibrationStatus(ADC1));
// Разрешаем работу АЦП
  ADC_Cmd(ADC1,ENABLE);
  while (!ADC_GetFlagStatus(ADC1,ADC_FLAG_RDY));

// Датчик температуры и опорное напряжение
//  ADC_TempSensorCmd(ADC1, ENABLE);
//  ADC_VrefintCmd(ADC1, ENABLE);

// Конфигурируем каналы
  ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 1, ADC_SampleTime_61Cycles5);


  // Включение, начальная инициализация
  RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1,ENABLE);
     DMA_DeInit(DMA1_Channel1);
     DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)&(ADC1->DR);
     DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)adrez;
     DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
```

```
    DMA_InitStructure.DMA_BufferSize = 1;
    DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
    DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
    DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
    DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
    DMA_InitStructure.DMA_Priority = DMA_Priority_Medium;
    DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
    DMA_Init(DMA1_Channel1, &DMA_InitStructure);
// Enable ADC_DMA
    ADC_DMACmd(ADC1, ENABLE);
    ADC_DMAConfig(ADC1,ADC_DMAMode_Circular);
    DMA_Cmd(DMA1_Channel1, ENABLE);

   ADC_StartConversion(ADC1);


  SysTick_Config(SystemCoreClock/100);

  InitLed();
  InitSegm();

  idt[0]=0xEE;
  idt[1]=0x1C;
  idt[2]=0x3E;
  idt[3]=0x02;
  for (;;) {

                if (ADC_GetFlagStatus(ADC1,ADC_FLAG_EOS)) {
                        uint16_t r;
                        r=adrez[0];
                        ADC_ClearFlag(ADC1,ADC_FLAG_EOS);
                        OutDigit(r);
    Repaint();
                  SetColor((float)(r)/4096);
                }

  };
}
```

Рисунок 16. – Базовая программа для лабораторной работы

Литература

1. Шумилин С. Новая серия отечественных 32-разрядных высокопроизводительных микроконтроллеров семейства 1986 на базе процессорного ядра ARM Cortex_M3 // Компоненты и технологии. 2008. № 10.
2. Шумилин С. Характеристики производительности микроконтроллеров на базе ядра ARM Cortex-M3 // Электронные компоненты. 2009. № 8.

3. Предварительный вариант спецификации. Серия 1986BE9x высокопроизводительных 32-разрядных микроконтроллеров на базе процессорного ядра ARM Cortex-M3.
4. Руководство по эксплуатации отладочной платы для МК 1986BE91T.
5. Комплект инструментальных средств для микроконтроллеров ЗАО "ПКК Миландр" 1986BE91. Быстрый старт.
6. Пакет инструментальных средств CodeMaster- ARM для микроконтроллеров ПКК "Миландр" 1986BE91xx с ядром Cortex-M3.
7. Инструментальные средства для разработки и отладки систем на базе микроконтроллеров Cortex-M3/M1/M0, ARM7/ARM9. Руководство пользователя.
8. www.phyton.ru
9. Спецификация. Жидкокристаллический модуль MT-12864J.
10. www.milandr.ru

Используемые документы (папка Distrib\Datasheet):

1) AN2606_boot_modes.pdf – режимы загрузки и работа загрузчика
2) DSH_STM32F303.pdf – общее описание контроллера
3) FT232RL.pdf – описание преобразователя USB-UART
4) RM_STM32F303.pdf – подробное описание блоков контроллера
5) STM32F3DISCOVERY.pdf – описание отладочной платы
6) PM0214.pdf – описание ядра CortexM4
7) The Defi nitive Guide to the ARM Cortex-M3 Joseph Yiu. ARMCortexM3Guide.pdf

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

ВВЕДЕНИЕ

В настоящее время идёт стремительное развитие технологии СБИС. Интеграция элементов и частота их синхронизации достигли фантастических результатов. Однако человеческая мысль не стоит на месте и технологические усовершенствования подкрепляются новыми структурными подходами к увеличению производительности вычислительных систем. Вопросам структурных методов повышения производительности и посвящено это учебное пособие. Исторически первыми появились методы, повышающие производительность процессоров и однопроцессорных систем. Конвейерные, матричные, векторные вычисления составляли основу скалярных и суперскалярных процессоров и суперЭВМ. Сейчас на первый план выходят многопроцессорные системы как симметричные (SMP - системы) с разделяемой памятью, так и с массовым параллелизмом (MPP - системы). Предложено множество различных вариантов построения таких систем и осуществлена их практическая реализация. В пособии авторы попытались обобщить многообразие публикаций в данной области и систематизировать материал.

1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ

1.1. КЛАССИФИКАЦИЯ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

Первые три поколения ЭВМ строились на одном и том же принципе обработки данных. Суть этого принципа составлял последовательный метод управления вычислительным процессом и постоянную структуру вычислительной системы. Задачи, решаемые на однопроцессорной ВС, имели относительно небольшую сложность. На входе ВС они представлялись в виде заданий, которые реализовывались в ВС независимыми процессами.

Структура однопроцессорной ВС, если следовать классификации М. Флинна, представляет собой класс "Одиночный поток Команд - Одиночный поток Данных" (ОКОД) или Singl Instraction – Singl Date (SIMD), который для трех основных устройств машины - памяти, устройства обработки (УО) и устройства управления (УУ) - можно представить схемой, приведенной на рис. 1.1.

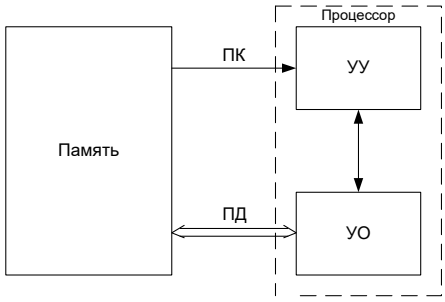


Рис. 1.1

Одинарная линия обозначает потоки команд (ПК), а двойная линия - потоки данных (ПД). Причем устройство управления и устройство обработки образуют в данном случае полноценный процессор. УУ процессора читает из памяти команды, дешифрирует их и направляет в УО (в обычном представлении - АЛУ).

Устройство обработки выбирает из памяти данные (операнды) над которыми выполняются операции, заданные текущей командой; результат возвращается в память.

Появление четвертого поколениия ВС связано с необходимостью решения более сложных задач, которые могут быть решены достаточно быстро лишь при условии применения вычислительных средств в виде некоторого коллектива вычислителей. Точно такие же потребности возникают и при решении задач в реальном масштабе времени (РМВ). При этом решение сложных задач предъявляет высокие требования к производительности (10^{12} - 10^{15}опер/с), надежности и живучести ВС, а также к качеству математического обеспечения и, прежде всего, к качеству операционной системы (ОС). Поскольку возможности по увеличению скорости переключения логических элементов ограничены, поэтому получение высокой производительности невозможно без включения параллелизма в структуру вычислительной системы. Кроме того, требуется динамическое подключение специализированных высокопроизводительных устройств обработки для выполнения трудоёмких операций (например, матричных) в ходе реализации вычислительного процесса.

В связи с изложенным, ВС должны обладать следующими свойствами:

1) автоматическим изменением структуры ВС в зависимости от структуры решаемых задач;

2) параллельно-последовательными алгоритмами управления параллельными вычислительными процессами;

3) возможностью наращивания вычислительной мощности;

4) достаточной надежностью и живучестью;

5) технической и программной модульностью;

6) аппаратной реализацией функций операционных систем.

Пятое поколение ВС связывается с еще более сложными задачами-системами, известными под общим названием "проблемы искусственного интеллекта".

Для удовлетворения этих требований ВС четвертого и пятого поколений строятся по принципам параллельной обработки. Параллельная обработка информации представляет собой одновременное выполнение некоторого множества независимых программ или частей одной и той же программы на некотором множестве процессоров. Параллельную обработку следует отличать от мультипрограммной, основная цель которой, как известно, заключается в разделении времени оборудования между двумя или более программами, функционирующими квазиодновременно (т.е. создается иллюзия одновременности для пользователей) и размещенными полностью или частично в оперативной памяти. Следует отметить, что возможен мультипрограммный режим и при параллельной обработке.

Вычислительные системы параллельной обработки, или просто параллельные ВС, содержат два или более процессоров (компьютеров) и подразделяются в основном на три класса в соответствии с той же классификацией М. Флинна: "Одиночный поток Команд - Множественный поток Данных" (ОКМД) или Singl Instraction – Multiplle Date (SIMD), "Множественный поток Команд - Одиночный поток Данных" (МКОД) или Multiplle Instraction – Singl Date (MISD) и "Множественный поток Команд - Множественный поток Данных" (МКМД) или Multiplle Instraction – Multiplle Date (MIMD).

В структурах параллельных ВС (ПВС) с организацией типа ОКМД (рис. 1.2) одно устройство управления осуществляет управление работой множества процессорных модулей, так что каждый из них одновременно выполняет сначала одну команду, затем вторую и т.д. Другими словами, здесь реализуется синхронный параллельный вычислительный процесс. Часто такой вид параллелизма называют параллелизмом на уровне данных. К таким ВС относятся ассоциативные, матричные, векторные процессоры и ВС, а также процессоры с длинным командным словом (VLIW- процессоры).

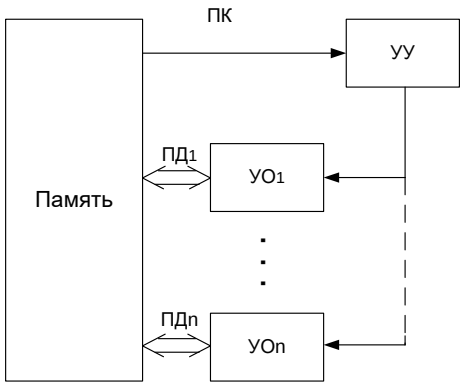


Рис. 1.2

Для структуры ПВС с МКОД процесс обработки разбивается на несколько этапов, каждому из которых соответствует своё УУ и УО (рис. 1.3). К таким ПВС относятся конвейерные ПВС, построенные по одному из трех принципов обработки: арифметико – конвейерному, командно – конвейерному или макро – конвейерному.

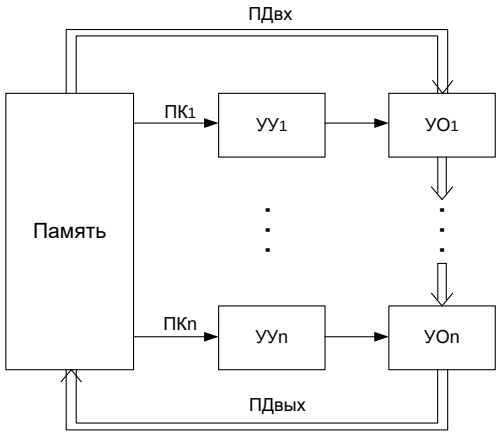


Рис. 1.3

В структурах ПВС с МКМД (рис. 1.4) несколько УУ осуществляют управление одновременным выполнением различных участков одной и той же программы, т.е. здесь реализуется асинхронный параллельный вычислительный процесс. К таким ПВС относятся прежде всего многопроцессорные и многомашинные ВС (МПВС и ММВС). Кроме того, сюда относятся распределенные ВС и вычислительные сети.

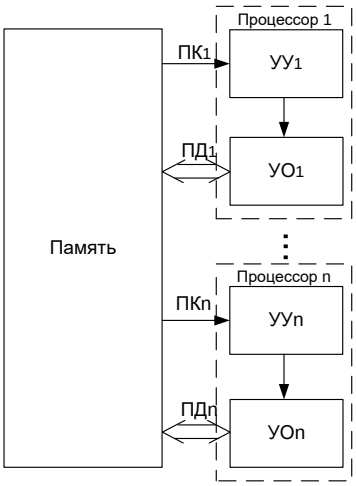


Рис.1.4

Классификационная схема М. Флинна, схема четырех классов сочетаний из различных ПК и ПД, не отражает все возможные структурные решения ПВС. Например, классификацию ПВС можно представить схемой, изображенной на рис. 1.5.

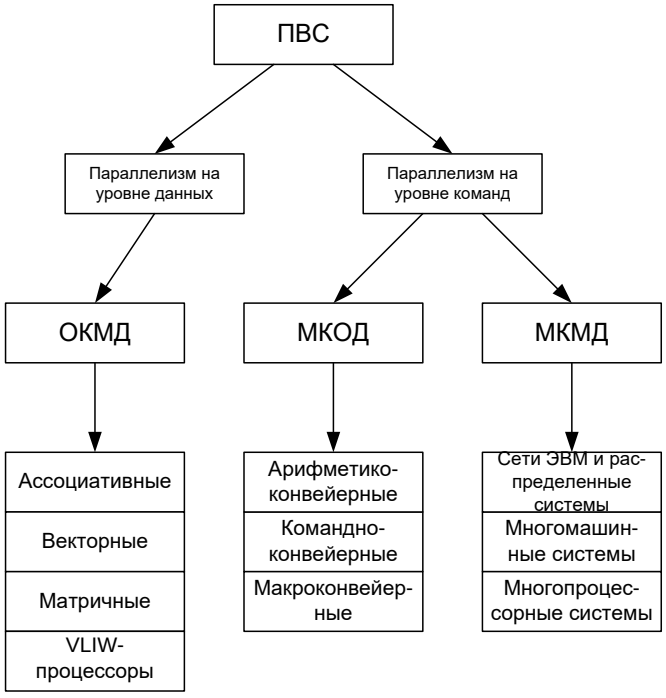


Рис.1.5

Существуют и другие классификации, фактически дополняющие и уточняющие классификацию М. Флинна [Воеводин].

Следует заметить, что в высокопроизводительных ВС и процессорах применяют не один принцип обработки, а совмещают сразу несколько. Так в суперскалярном процессоре Pentium 4 фирмы Intel процессор построен на базе 20-ти стадийного арифметического конвейера и блоков для выполнения целочисленных операций с фиксированной точкой. Для выполнения команд с плавающей точкой используются специализированные блоки FPU (Floating Point Unit). Кроме того в составе процессора имеются блоки расширения MMX и SSE с векторной технологией обработки данных. Все блоки обработки включаются в вычислительный процесс динамически, т.е. в момент появления соответствующей команды в программе. Если текущая команда выполняет операцию с плавающей точкой, то конвейер коммутируется на свободный блок FPU, если текущая команда выполняет 64 – х битную векторную операцию, то подключается блок MMX и т.д.

1.2. УРОВНИ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ

Проблема параллельной обработки выдвигает четыре основных направления исследований:

1) формальная теория параллельных вычислений, исследующая общие свойства параллельных программ;

2) языки и методы параллельного программирования;

3) построение (в том числе и автоматическое) параллельных программ, заключающееся в выявлении сегментов (ветвей программ), пригодных для одновременного исполнения;

4) организация параллельных вычислительных процессов, в частности, распределение ветвей программы по процессорам.

Рассмотрим основные вопросы, касающиеся наиболее близкого к аппаратным средствам и интенсивно развивающегося в настоящее время третьего направления.

Виды параллелизма алгоритмов. Обычный процесс вычислений и решения задач на ЭВМ представляет собой последовательность операций, в результате которых получаются искомые данные. Вычислительная математика в течение многих лет создавала алгоритмы решения задач, которые соответствуют этой последовательности. Такие алгоритмы носят название последовательностных алгоритмов. Именно в соответствии с такими алгоритмами действует человек, решая задачу вручную, именно такой характер алгоритмов нашел отражение в структуре наиболее распространенных алгоритмических языков ПАСКАЛЬ, СИ, ФОРТРАН и др., применяемых в однопроцессорных системах.

Необходимость создания параллельных ВС выдвинуло проблему организации параллельных вычислений, которая в основном сводится к созданию методов разработки алгоритмов программ, пригодных для параллельного исполнения на нескольких процессорах. Поэтому в отличие от обычного последовательного программирования начали развиваться методы параллельного программирования.

Параллелизм, встречающийся в вычислительных задачах, можно разделить на четыре уровня: задачи, подзадачи, операции и микрооперации. Для первого уровня на множестве устройств обработки выполняется параллельно несколько независимых задач, для второго уровня - несколько подзадач одной задачи, для третьего - несколько операций и т.д.

Под независимыми понимаются задачи, результаты которых могут использоваться самостоятельно. К ним, например, можно отнести независимые задачи различных пользователей или одни и те же задачи, которые решаются параллельно в системах повышенной надежности. Распараллеливание на уровне задач (процессов) не требует специальных средств. Такие задачи пригодны для решения на многомашинных и многопроцессорных системах.

Суть параллелизма на уровне подзадач, или, как его еще называют, параллелизм независимых ветвей (потоков) состоит в том, что в программе решения задачи на тех или иных этапах могут быть выделены независимые части-ветви, которые при наличии в вычислительной системе соответствующих средств могут выполняться параллельно.

Понятие независимости включает в себя следующее. Ветвь В программы (подзадача В) не зависит от ветви А (подзадача А), если удовлетворяются следующие условия:

1) отсутствие связи между ветвями по данными и по управлению, т.е. в первом случае ни одна из входных переменных для ветви В не является выходной переменной ветви А (или другой ветви, от нее зависящей) и, во втором случае, условие выполнения ветви В не зависит от признаков, вырабатываемых при завершении работы по ветви А (или другой, от нее зависящей);

2) независимость в программном отношении (т.е. ветви должны выполняться по разным программам);

3) независимость по рабочим полям памяти (не должна производиться запись в одни и те же ячейки памяти для разных ветвей).

Для решения задач, распараллеливающихся на независимые ветви, используются в основном многопроцессорные системы. Только в некоторых случаях, когда решается крупная задача, которая имеет длинные независимые ветви и редкий взаимообмен между взаимодействующими ветвями, возможно использование многомашинных или распределенных ВС. Эффективность использования той или иной вычислительной системы для решения конкретной задачи определяется связностью подзадач (ветвей). Например, если в задаче одно из условий, перечисленных выше, не выполняется, то эффективность ее решения на МПВС снижается. Поэтому возникает необходимость использования других принципов обработки. Так, если не выполняется условие 2, т.е. ветви задачи выполняются по одной и той же программе, то будут возникать коллизии за доступ к единственному программному ресурсу, поэтому необходимо применять либо меры к устранению конфликтов (например, копирование программных блоков), что несомненно приведет к увеличению времени выполнения и возрастанию числа необходимых ресурсов ВС, либо использовать матричные или векторные вычисления.

Если не выполняется условие 3 (т.е. данные являются общими или их разделяемыми для множества процессов), то потребуется либо создание дополнительных копий в процессорных узлах (репликация программ или данных),

либо использование дополнительных программных или аппаратных средств для синхронизации процессов. Однако в первом случае также потребуются дополнительные аппаратные или программные средства для обеспечения когерентности (согласованности) разделяемых данных.

Примером задач, распараллеливающихся на ветви, выполняемые по одинаковой программе, являются задачи, которые сводятся к операциям над *n*-мерными векторами, над матрицами, решетчатыми функциями и др. Большинство операций, выполняемых в таких задачах, представляют собой последовательность из одних и тех же команд, которые одновременно выполняются над совокупностью различных данных.

Например, вычисление скалярного произведения для *n*-мерных векторов

$$A \bullet B = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + a_n b_n$$

состоит фактически из операций двух типов: вычисления *n* попарных произведений соответствующих компонент векторов, затем их суммирование. Программа представляет собой цикл, повторяющийся *n* раз, причем тело цикла будет состоять из операций присвоения и умножения. В однопроцессорной системе эта часть программы выполнится за *n* проходов тела цикла.

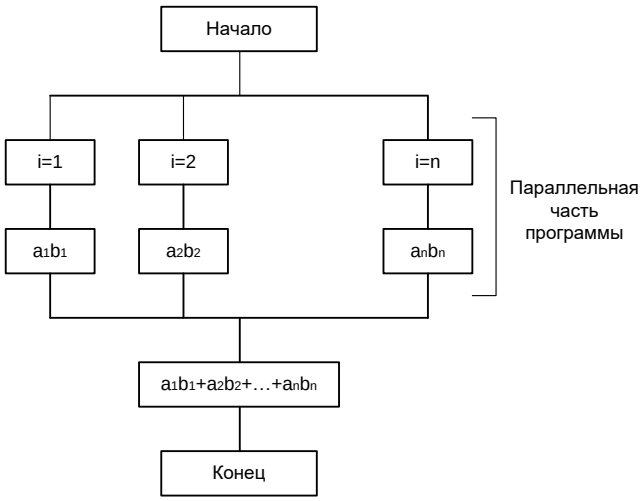


Рис. 1.6

Параллельный алгоритм вычисления произведения векторов A и B можно представить граф - схемой изображенной на рис. 1.6. Из граф – схемы видно, что ветви с 1-й по *n*- ю выполняются параллельно по одинаковой программе, состоящей из двух операций: присвоения и умножения, но с разными данными. Подобные задачи могут обрабатываться многопроцессорной ВС, но при этом каждому процессору желательно иметь копию параллельной программы. В системах типа ОКМД достаточно одной копии, хранящейся в общем устройстве управления. Устройство управления выбирает из памяти программ очередную команду, размножает ее и отправляет в каждое устройство обработки (элементарный процессор) на исполнение. Если число устройств обработки равно *n*, то для выполнения параллельной части программы потребуется время, достаточное для реализации одного прохода тела цикла.

Если в задаче не выполняется условие 1, то для ее решения возможно применение макроконвейерной системы типа МКОД, если выполнение одной и той же программы многократно повторяется. Возможность появляется в том случае, если условия выполнения и исходные данные для выполнения очередной (*i*+1)-й ветви возникают при исполнении *i*-й ветви, (*i*+2)-й ветви при исполнении(*i*+1)-й ветви и т.д. В этом случае можно совместить время исполнения *i*-й ветви с исполнением ветвей (*i*+1), (*i*+2), и т.д.

Вышесказанное иллюстрируется примером, изображенным на рис. 1.7.

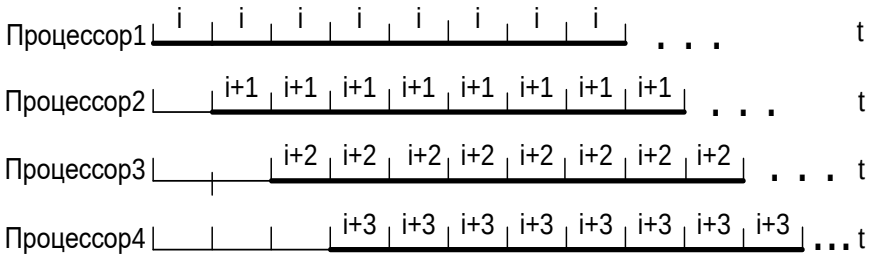


Рис. 1.7

Для упрощения принято, что задачи разбиваются на 4 ветви: i; (i+1); (i+2) (i+3). Условие выполнения (*i*+1) - й ветви возникает после выполнения *i*-й ветви и т.д. Поэтому, начиная с *i*-й ветви можно запускать в конвейер вычислений.

Процессорам 1,...,4 каждый раз после выполнения ветви вновь назначается та же ветвь (одной и той же циклически повторяющейся задачи), только с другими данными. Поэтому, начиная с (i+3) ветви, совмещается во времени выполнение всех ветвей задачи и вычисления производятся одновременно на 4-х процессорах.

Параллелизм на уровне операций и микроопераций (его называют параллелизмом смежных операций и микроопераций) в принципе не отличается от параллелизма ветвей. Отличие заключается в удельном весе, который они занимают в программе. Однако это отличие приводит к совершенно иным структурам процессоров или вычислительных систем. При параллелизме смежных операций используются командно - конвейерные, при параллелизме смежных микроопераций - арифметико-конвейерные процессоры. Однако при внешнем сходстве конвейерного выполнения, структурная организация разных уровней конвейеризации полностью отличаются.

К понятию уровня параллелизма тесно примыкает понятие гранулярности. Это мера отношения объема вычислений, выполненных в параллельной задаче, к объему коммуникаций (для обмена сообщениями). Степень гранулярности варьируется от мелкозернистой до крупнозернистой. Понятия крупнозернистого, среднезернистого и мелкозернистого параллелизма заключаются в следующем. При крупнозернистом параллелизме каждая ветвь параллельной программы достаточно независима от остальных, причем требуется относительно редкий обмен информацией между отдельными ветвями. Ветви могут включать сотни тысяч команд. Этот уровень параллелизма обеспечивается программистом.

При среднезернистом параллелизме параллельными ветвями могут являться вызываемые процедуры, включающие в себя несколько сотен команд. Обычно организуется как программистом, так и компилятором.

Мелкозернистый параллелизм состоит обычно из десятков команд. Распараллеливаемыми единицами являются элементы выражения или отдельные итерации цикла, имеющие небольшие зависимости по данным. Характерная особенность мелкозернистого параллелизма заключается в приблизительном равенстве интенсивности вычислений и интенсивности взаимодействия между

ветвями. Этот уровень параллелизма часто организуется распараллеливающим компилятором.

Эффективное параллельное исполнение параллельной программы зависит от уровня гранулярности. Так, например, задачи с крупной гранулярностью могут эффективно решаться на многопроцессорных и многомашинных системах, среднегранулярные – только на многопроцессорных. Мелкогранулярные задачи требуют применения матричных, векторных или командно-конвейерных процессоров.

Представление параллельных алгоритмов.

Для представления параллельных алгоритмов используются разные методы: язык *p*-схем, разработанный Э.В. Евреиновым и Ю.Г. Косаревым, аппарат ярусно-параллельных форм, предложенный Д.А. Поспеловым и др. Все эти методы в общем близки друг к другу.

Например, при использовании аппарата ярусно-параллельных форм программа представляется в виде "ярусов" на графе. Причем в 0-й ярус входят все те ветви программы, каждая из которых не зависит ни от одной ветви, в 1-й ярус - ветви, зависящие только от ветвей 0-го яруса, во 2-й ярус - ветви, зависящие от ветвей 1-го яруса и, может быть, ветвей 0-го яруса и т.д.

На рис. 1.8 в виде примера изображена некоторая программа, представленная графом в ярусно-параллельной форме. Кружками отмечены операторные вершины графа, обозначающие ветви программы, внутри операторов проставлены номера ветвей, а рядом - длины этих ветвей. Дугами показаны функциональные связи (связи по данным) между ветвями. Дуга, входящая в некоторый оператор, обозначает входную переменную для соответствующей ветви. Если дуга не выходит ни из какого другого оператора, то она соответствует входным данным программы. Дуга, выходящая из некоторого оператора, обозначает выходную переменную соответствующей ветви. Если эта дуга не входит ни в какой другой оператор, то она обозначает выходные данные программы

Разветвления дуг показывают, что одни и те же данные являются входными для двух или более ветвей.

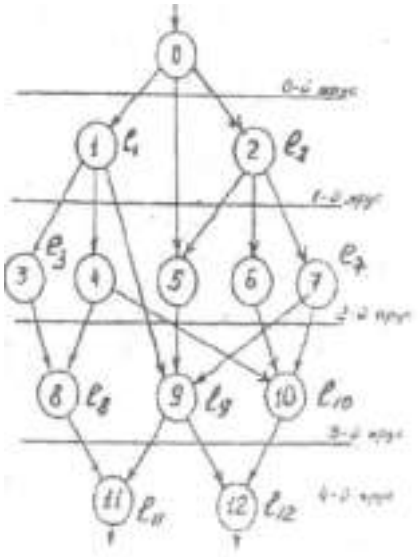


Рис. 1.8

Такое представление алгоритмов используют для оценки возможностей организации параллельных вычислений. Для этого вводят ряд количественных характеристик для программ, представленных в ярусно-параллельной форме: b_i - ширина i-го яруса (т.е. количество независимых ветвей в i-м ярусе); B- ширина графа (максимальная ширина яруса, т.е. $B=max\{bi\}$); l_i -длина яруса (обычно представляется трудоемкостью ветви, имеющей набольшую длину в этом ярусе); L - длина графа (максимальная длина пути, ведущего из нулевого в конечное состояние, называемая *критическим путем*); S_{ij} - связанность операторов или ветвей задачи, определяется количеством информации, передаваемой между i-й и j-й ветвями задачи в процессе ее решения; S - связанность графа (представляет связанность всех ветвей задачи) определяется матрицей связности $S=S_{ij}$.

Несвязанные, слабосвязанные и сильносвязанные задачи. Будем для упрощения считать, что сложная задача, решаемая ВС, состоит из набора простых задач (которые могут быть независимыми задачами или независимыми ветвями сложной задачи). Две или более простые задачи называются связанными, если данные одной задачи используются не только в собственном, но и в вычислительном процессе какой-нибудь задачи.

Сложная задача, состоящая из n простых задач $P_н = \{P_1, P_2,..., P_n\}$, называется несвязанной, если все элементы матрицы связности S равны нулю, т.е. $S_{ij} = 0$, $i,j=1,n$. Несвязанные задачи образуют простейший класс сложных задач, который является естественным обобщением класса одиночных задач. Одиночные простые задачи соединяются в некоторые наборы задач для их решения одними им теми же вычислительными средствами. Класс несвязанных задач является довольно распространенным. Они характерны для информационно-вычислительных систем, автоматизированных систем управления и т.п.

Сложная задача $P_c = \{P_1, P_2,..., P_n\}$называется слабосвязанной, если не все элементы ее матрицы связности S равны нулю. Слабосвязанные задачи образуют обширный класс задач, различающихся как типом графа, описывающего связи между задачами так и объемом данных, передаваемых между простыми задачами. Считается, что общий объем обменных взаимодействий в слабосвязанных задачах много меньше объема вычислений одной простой задачи.

Наличие связанности в сложной задаче, несмотря на незначительный объем обменных взаимодействий, не позволяет эффективно решать задачи такого типа на одной ЭВМ или на совокупности не связанных между собой ЭВМ.

Сложная задача $P_k = \{P_1, P_2,..., P_n\}$называется сильносвязанной, если все элементы ее матрицы связности S не равны нулю. В сильносвязанных задачах резко возрастает число обменных взаимодействий, причем они осуществляются не только после решения набора простых задач, составляющих сложную, но и в процессе решения простых задач, в предельном случае - на каждом шаге вычислений.

Наличие связности в сложных задачах налагает определенные требования на пропускную способность соединительной сети связи между модулями параллельной ВС (ЭВМ, процессорами, процессорными элементами и др.).

Средства определения параллелизма в программе.

При программировании задач на общепринятых алгоритмических языках, допускающих параллельное решение для обозначения независимых частей используются дополнительные операторы типа разветвление (FORK) и объединение (JOIN). При этом от программиста требуется, чтобы он сам расставил в тексте

программы операторы FORK<список ветвей>, обозначающие, что, начиная от данной точки программы, можно одновременно запускать ветви, перечисленные в списке, и операторы JOIN <список ветвей>, обозначающие, что дальнейшее продолжение программы возможно после выполнения всех ветвей, перечисленных в списке.

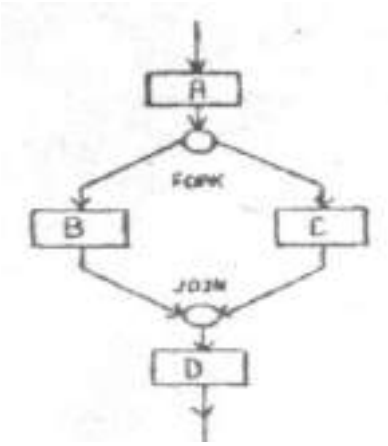


Рис. 1.9

Использование операторов FORK и JOIN поясняется на рис. 1.9. Допустим, что выполнение ветвей B и C некоторой программы может быть начато одновременно на разных процессорах, а обработка ветви D не может начаться раньше, чем закончится обработка ветвей B и C. Тогда к ветвям B и C применяют оператор FORK B, C. По окончании обработки применяют оператор JOIN B, C, объединяющий эти ветви.

Очевидно, что кроме средств разветвления и объединения в программе должны быть предусмотрены средства, которые могли бы определять момент окончания обработки наиболее трудоемкой ветви. Эти средства называют барьерной синхронизацией.

Метрики параллельных вычислений

При реализации задач на параллельных ВС очень важным является оценка эффективности параллельных вычислений. Для этого используют метрики, наиболее распространенные из которых рассматриваются ниже.

Профиль параллелизма программы

Число процессоров многопроцессорной системы, параллельно участвующих в выполнении программы в каждый момент времени t, определяют понятием *степень параллелизма* $D(t)$. Графическое представление параметра $D(t)$ как функции

времени называют *профилем параллелизма программы.* Профиль параллелизма показывает уровень загрузки процессоров в течение времени решения задачи. Он зависит от многих факторов (алгоритма, доступных ресурсов, степени оптимизации, обеспечиваемой компилятором и т. д.). Пример профиль параллелизма для параллельного алгоритма показан на рис. 1.10. В интервалы времени t_1-t_3 и t_{23}-t_{26} загружен один процессор, в интервале t_3-t_5 загружены два процессора и т.д.

Пусть вычислительная система состоит из n одинаковых процессоров; максимальный параллелизм в профиле равен m, причем $n > m$. Производительность P одиночного процессора системы выражается в количестве операций, выполняемых в единицу времени и не учитывает издержек, связанных с обращением к памяти и пересылкой данных. Если за наблюдаемый период загружены i процессоров, то $D(t) = i$.

Рис. 1.10

Общий объем вычислительной работы θ (количество операций), выполненной с момента старта t_s до момента завершения t_e, пропорционален площади под кривой профиля параллелизма. Профиль параллелизма на рисунке за время наблюдения (t_s, t_e) возрастает от 1 до пикового значения m = 8, а затем спадает до 0. Средний параллелизм (загрузка процессоров) ρ = (1 x 5 + 2 x 3 + 3 x 4+4 x 6 + 5 x 2 + 6 x 2 + 8 x 3)/(5+ 3+4 + 6 + 2 + 2 + 3) = 93/25 = 3,72.

Фактически общая рабочая загрузка ρ и представляют собой нижнюю границу асимптотического ускорения.

Ускорение, эффективность, загрузка и качество

Предположим, что параллельная система состоит из n процессоров. Определим ускорение, которое создается параллельной программой. Пусть $T(n)$ — время выполнения параллельной программы на n- процессорной системе, а $T(1)$ - время выполнения этой же программы на однопроцессорной системе.

Ускорение - это отношение времени, требуемого для выполнения программы на одном процессоре и времени параллельного вычисления на n процессорах. Без учета различного рода издержек ускорение определяется как

$$\psi = \frac{T(1)}{T(n)}$$

Как правило, ускорение удовлетворяет условию $\psi < n$.

Эффективность n – процессорной системы — это ускорение, приходящееся на один процессор, определяемое выражением

$$\varepsilon = \frac{\psi}{n}$$

Эффективность отвечает условию $1/n \leq \varepsilon_n \leq 1$, что означает, что обычно часть мощности процессора теряется на решение системных проблем, связанных с конфигурированием в параллельную систем, т.е. организация вычислений на n процессорах связана с издержками вычислительного процесса. Поэтому имеет смысл ввести понятие потерь производительности, отражающее степень снижения производительности процессора в составе параллельной системы.

$$\pi = 1 - \varepsilon$$

Рассмотрим пример. Положим, что последовательный алгоритм выполняется за 12 с., а параллельный алгоритм реализуется на 8-ми процессорах за 2 с. Тогда:

$$\psi = 12/2 = 6$$

$$\varepsilon = 6/8 = 0{,}75$$

$$\pi = 1 - 0{,}75 = 0{,}25$$

Если ускорение, достигнутое на n процессорах, равно n, то говорят, что алгоритм показывает линейное ускорение. В исключительных ситуациях ускорение ψ может

быть больше, чем *n*. В этих случаях иногда применяют термин суперлинейное ускорение. Данное явление имеет шансы на возникновение в двух следующих случаях:

- Если последовательная программа и данные размещены в небольшом сегменте памяти, вызывая, таким образом, частый свопинг (подкачку данных с диска).
- Если кэш имеет малые размеры, что приводит к большему числу кэш-промахов.

В параллельных же программах ветви короче, поэтому зачастую используют много меньший набор данных.

Факторы, ограничивающие ускорение

- **Программные издержки.** Параллельные алгоритмы имеют дополнительные программные издержки по сравнению с последовательными даже если они выполняют одни и те же вычисления. Это связано в первую очередь с расширением функций операционных систем, например, функцией распределения задач по процессорам, которая отсутствует в однопроцессорных системах. Параллельные программы также могут вызывать различного рода конфликты при их исполнении, отсутствующие в алгоритмах последовательных, и требующие дополнительного программного кода для их разрешения. Такая ситуация может возникнуть, например, при доступе к программному коду операционной системы или его отдельным частям, которые разделяется между всеми процессорами.
- **Коммуникационные издержки.** Межпроцессорные коммуникации снижают ускорение, если обмен информацией и вычисления не перекрываются во времени. Поэтому важен уровень гранулярности, определяющий объем вычислительной работы, выполняемой между коммуникационными фазами алгоритма. Достичь уменьшения коммуникационных издержек можно в том случае, когда вычислительные гранулы являются достаточно крупными, а коммуникации имеет сравнительно небольшой объем.
- **Издержки из-за дисбаланса загрузки процессоров.** Между точками барьерной синхронизации ветви программ каждого из процессоров должны обладать одинаковой трудоемкостью, иначе часть процессоров будут ожидать, пока остальные завершат свои операции. Эта ситуация называется *дисбалансом*

загрузки. Таким образом, время выполнения программы, и, следовательно, ускорение определяется длиной критического пути на ярусно - параллельном графе алгоритма.

- **Издержки из-за последовательных участков программ** Параллельная вычислительная система обеспечивает значительное повышение скорости вычислений за счет разделения вычислительной нагрузки на множество независимо работающих процессоров. В лучшем случае система из n процессоров могла бы ускорить вычисления в n раз**.** В действительности достичь такого показателя по ряду причин не удается. Главная из этих причин заключается в невозможности полного распараллеливания задач. Как правило, в каждой программе имеется фрагмент кода, который принципиально должен выполняться последовательно и только одним из процессоров. Это может быть часть программы, например, отвечающая за распределение ветвей задачи по процессорам. Таким образом, добиться увеличения производительности прямо пропорционально числу процессоров ни коем образом не удастся.

Закон Амдала

Амдал предложил формулу, отражающую зависимость ускорения вычислений, достигаемого на параллельной ВС, от числа процессоров и соотношения между последовательной и параллельной частями программы. Пусть ускорение ψ - это отношение времени t_1, затрачиваемого на проведение вычислений на однопроцессорной ВС, ко времени t_n решения той же задачи на n – процессорной системе:

$$\psi = \frac{t_1}{t_n}$$

Определим ускорение, которое дает n – процессорная система с учетом последовательных участков программ (рисунок 1.11).Будем считать, что объем решаемой задачи остается неизменным независимо от числа процессоров, участвующих в ее решении. Программный код решаемой задачи состоит из двух частей: последовательной и параллельной. Обозначим долю операций, которые должны выполняться последовательно одним из процессоров, через f, где $0 \le f \le 1$ (под долей понимается отношение трудоемкости последовательной части

программы, измеряемой в машинных операциях, к трудоемкости всей программы в целом). Отсюда доля, приходящаяся на распараллеливаемую часть программы, составит $1 - f$. Если $f = 0$, то это соответствуют полностью параллельным, а $f = 1$, то полностью последовательным программам. Параллельная часть программы содержит одинаковые ветви и равномерно распределяется по всем процессорам вычислительной системы.

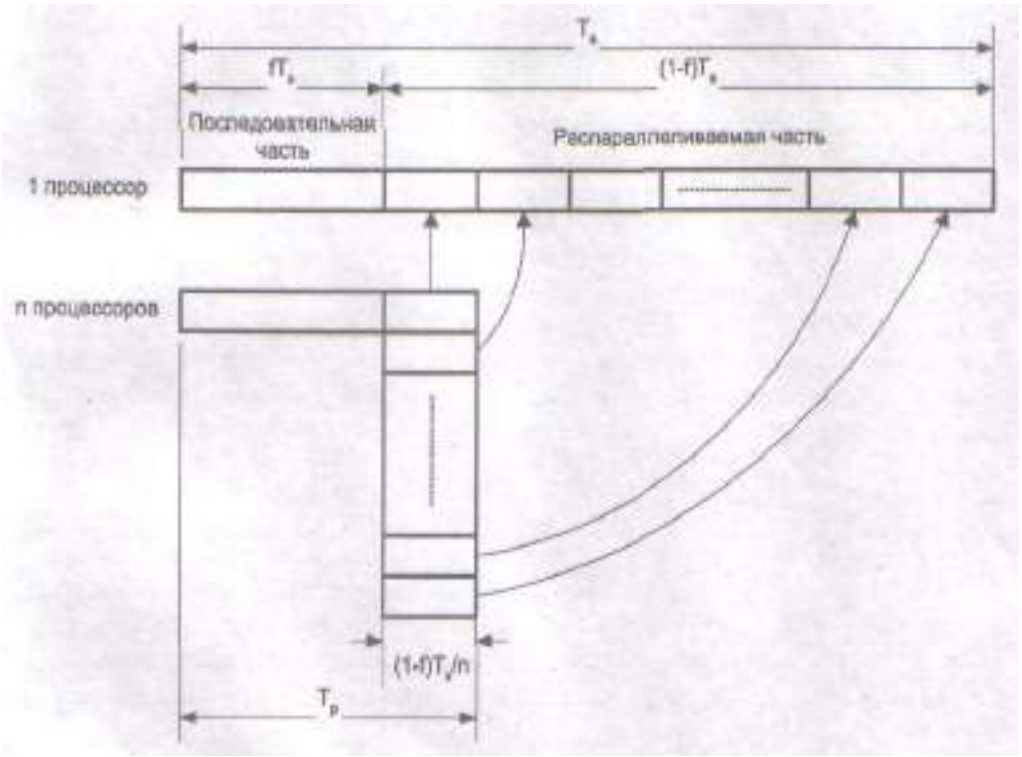


Рисунок 1.11- Определение величины ускорения в законе Амдала

$$t_n = ft_1 + \frac{(1-f)t_1}{n}$$

$$\psi = \frac{t_1}{t_n} = \frac{n}{1+f(n-1)}$$

Если устремить число процессоров к бесконечности, то в пределе получим:

$$\lim_{n\to\infty} \psi = \frac{1}{f} \quad .$$

Характер зависимости ускорения от числа процессоров и доли последовательной части программы показан на рис. 1.12.

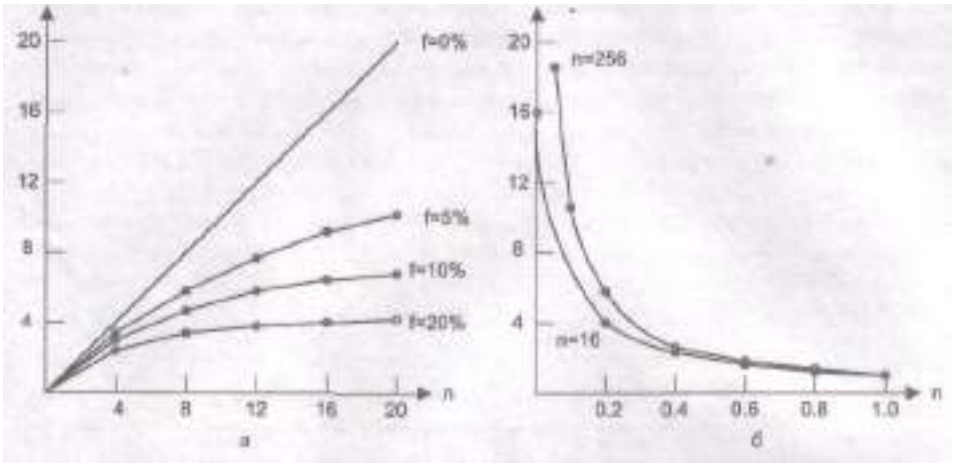


Рисунок 1.12 – Графики зависимости ускорения от доли последовательной части программы (а) и числа процессоров (б)

Закон Амдала показывает, что если в программе 10% последовательных операций (то есть $f = 0{,}1$), то, увеличения скорости работы программы более чем в десять раз нельзя получить, сколько бы процессоров не использовалось. Это верхняя оценка самого лучшего случая, когда никаких других видов издержек нет.

Закон Густафсона

Исследователям Густафсоном замечено, что при решении крупных задач на большой системе, состоящей из 1024 процессоров доля последовательного кода f лежала в пределах от 0,4 до 0,8 %, он получил значения ускорения по сравнению с однопроцессорным вариантом, равные более 1000. Согласно заколу Амдала для данного числа процессоров и диапазона f, ускорение не должно было превысить величины порядка 200. Густафсон пришел к выводу, что причина кроется в том, что в основе закона Амдала стоит повышение производительности ВС с увеличением числа процессоров при неизменном объеме задачи. Пользователь же обычно, получая в свое распоряжение более мощную ВС, стремится не сократить время вычислений, а

старается пропорционально мощности ВС увеличить объем решаемой задачи. И тут оказывается, что наращивание общего объема программы касается главным образом распараллеливаемой части программы. Это ведет к сокращению значения *f*. Примером может служить решение дифференциального уравнения в частных производных. Если доля последовательного кода составляет 10 % для 1000 узловых точек, то для 100 000 точек доля последовательного кода снизится до 0,1 %. Сказанное иллюстрирует рис. 1.13, который отражает тот факт, что, оставаясь практически неизменной, последовательная часть в общем объеме увеличенной программы имеет уже меньший удельный вес

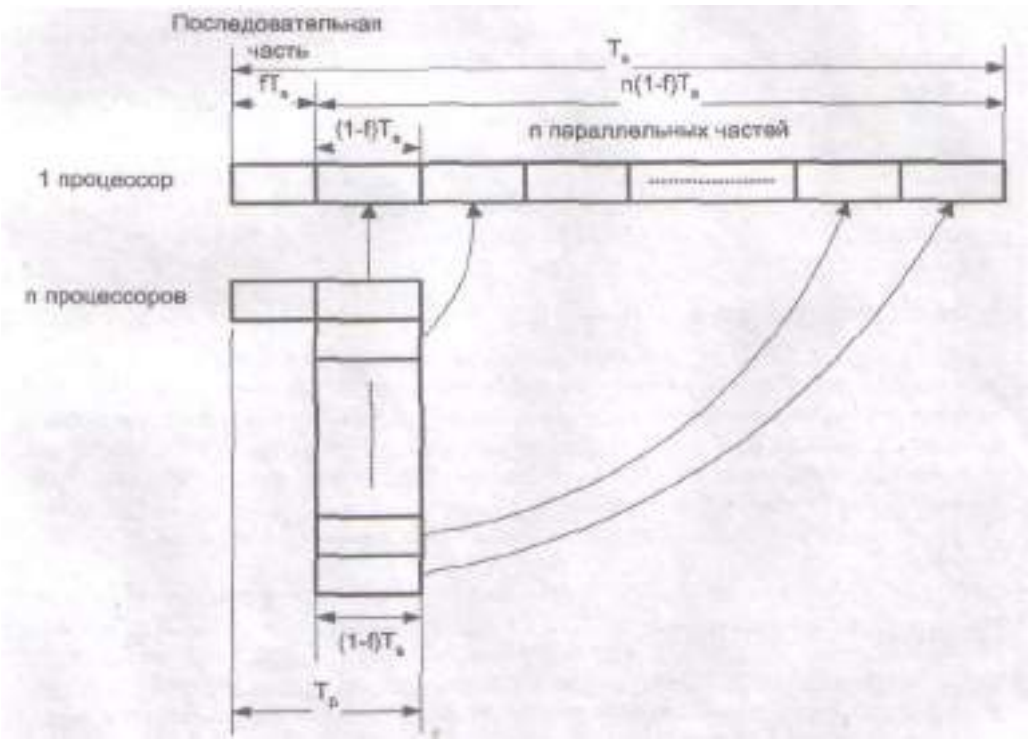


Рисунок 1.13- Определение величины ускорения в законе Густафсона

Было отмечено, что в первом приближении объем работы, которая может быть произведена параллельно, возрастает линейно с ростом числа процессоров в системе. Для того чтобы оценить возможность ускорения вычислений, когда объем последних увеличивается с ростом количества процессоров в системе (при постоянстве общего времени вычислений), Густафсон рекомендует использовать выражение, предложенное Е. Барсисом:

$$\psi = \frac{t_1}{t_n} = \frac{ft_1 + n(1-f)t_1}{ft_1 + (1-f)t_1} = n + (1-n)f.$$

Данное выражение известно как *закон масштабируемого ускорения* или *закон Густафсона* (иногда его называют также законом Густафсона-Барсиса). Закон Густафсона не противоречит закону Амдала. Различие состоит лишь в том, как используется дополнительная мощность вычислительной системы, возникающая при увеличении числа процессоров.

2. Организация памяти вычислительных систем

Иерархическая структура памяти

Желательно, чтобы память ЭВМ обладала как можно большей емкостью и как можно большим быстродействием. Трудно, однако, найти тип памяти, который бы удовлетворял этим противоречащим друг другу требованиям одновременно. Тенденция развития памяти такова, что более быстродействующие из них имеют и более высокую стоимость. Поэтому реализовать память, обладающую большой емкостью и высоким быстродействием, можно при совместном использовании дешевых запоминающих устройств (ЗУ) большой емкости и небольших по емкости, но быстродействующих ЗУ. В оптимальном сочетании таких ЗУ и состоит суть многоуровневой структуры памяти.

На рисунке 2.1 показан классический пример памяти с многоуровневой структурой. Емкость памяти на каждом из уровней этой памяти увеличивается в направлении от процессора в следующей последовательности: регистры процессора, быстродействующая буферная память, основная память, внешняя память, а повышение быстродействия этих устройств идет в обратном порядке.

С точки зрения программиста основная память рассматривается как ЗУ с произвольной выборкой и одномерной адресацией. Наличие буферного ЗУ лишь увеличивает эквивалентную скорость выборки из основного ЗУ, не внося никаких изменений в используемую программой систему адресации. Буферное ЗУ называется кэш-памятью.

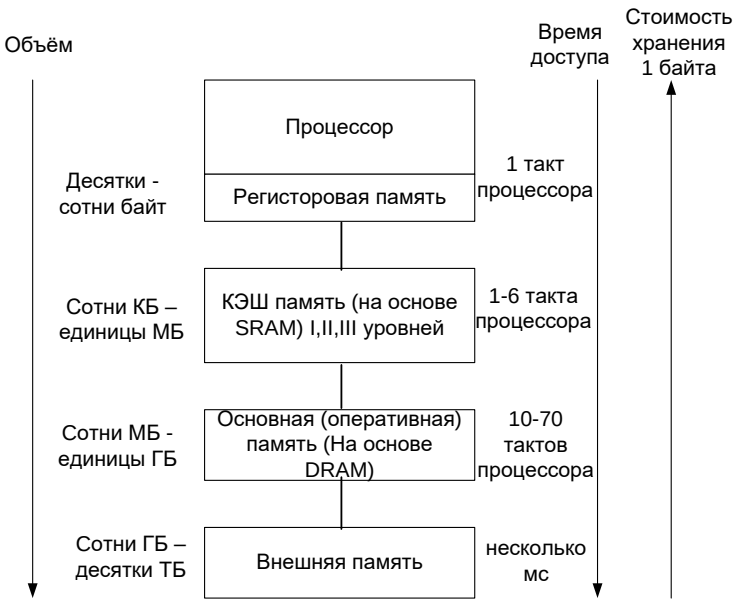


Рисунок 2.1- Многоуровневая структура памяти

Память, в организации которой используется механизм расширения ограниченной емкости основной памяти с помощью устройств памяти, например накопителей на магнитных дисках, называется виртуальной памятью. В виртуальной памяти без увеличения физической емкости основного ЗУ, образуется единое и расширенное, с точки зрения программиста, фиктивное адресное пространство, адресация которого не зависит от физических характеристик составляющих его устройств, и тем самым обеспечивается использование программой большой емкости памяти.

Между основным и внешним ЗУ можно также ввести буферное ЗУ. Оно называется дисковым КЭШем и предназначено для повышения эквивалентной скорости обращения к ЗУ на магнитных дисках (МД).

Как отмечалось выше, многоуровневая организация связана с передачей информации между разнотипными ЗУ и обеспечивает одновременно и быстродействие, и большую емкость памяти ЭВМ. Но в основе такой организации заложен принцип локального обращения к ЗУ (принцип локальности ссылок). Это означает, что область памяти, к которой происходят обращения работающих программ в течение некоторого

интервала времени, является, как правило, небольшой. Для этого программы и данные, обращение к которым происходит часто, размещаются в быстродействующих ЗУ, что способствует повышению быстродействия многоуровневой памяти в целом. При этом данные, обращение к которым происходит часто, размещаются в быстродействующих ЗУ, что способствует повышению быстродействия многоуровневой памяти в целом.

Рассмотрим в качестве модели многоуровневой памяти структуру на рис. 2.2, состоящую из ЗУ двух типов *M*1 и *M*2. Пусть их емкости равны соответственно n_1 и n_2, причем $n_2 >> n1$, а времена выборки t_1 и t_2. Эквивалентное время выборки $t_Э$ многоуровневой памяти, состоящей из М1 и М2, определяется по формуле

$$t_Э = t_1 + pt_2$$

где $p = 1/N$ (коэффициент промаха) - вероятность обращения к *M*2 из-за отсутствия данных в *M*1. Она показывает, что на *N* обращений к *M*1 в среднем приходится одно обращение к *M*2. Значение *N* увеличивается по мере увеличения n_1, причем стремятся обеспечить его больше 10 (обычно *N* = 10 - 100, что соответствует вероятности промаха *p* = 0,1- 0,01). В результате $t_Э$ принимает значение, близкое к t_1.

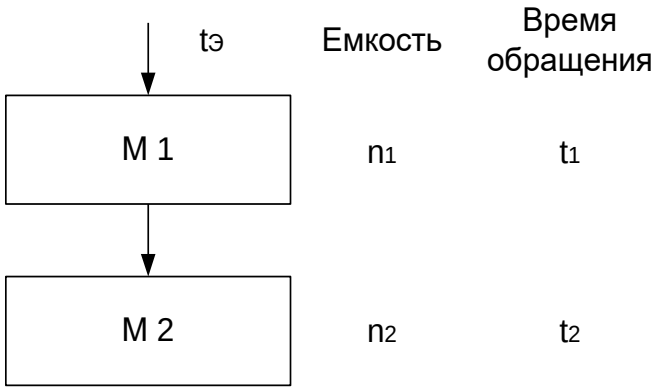


Рис.2.2. Модель двухуровневой памяти.

Расслоение памяти

Традиционно самым медленным устройством в подсистеме процессор-память является основная (оперативная) память. При обращении к основной памяти с целью ускорения этого процесса запись и считывание нескольких байтов могут осуществляться за один раз. При этом большей скорости доступа можно достичь за счет одновременного доступа ко многим независимым модулям памяти (независимыми являются модули, имеющие собственные схемы адресации и буферизации данных). С этой целью в большинстве современных ЭВМ основная память делится на несколько независимых модулей (блоков), т.е. применяется процедура *расслоения памяти*. В ее основе лежит так называемое *чередование адресов*, заключающееся в изменении системы распределения адресов между модулями памяти. Суть этой процедуры состоит в том, что при числе модулей n поступающий адрес a относится к модулю с номером $(a) \bmod n$.

Например, при четырех блоках памяти адресация выполняется в соответствии со структурой, показанной на рис. 2.3. Прием чередования адресов базируется на ранее рассмотренном свойстве локальности по обращению (локальности ссылок), согласно которому последовательный доступ в память обычно производится к ячейкам, имеющим смежные адреса. Иными словами, если в данный момент выполняется обращение к ячейке с адресом 4, то следующее обращение с большой вероятностью будет к ячейке с адресом 5, затем 6 и т. д. В самом деле, программы и данные располагаются в памяти обычно в последовательных адресах, что создает условия для высокоэффективной работы метода чередования адресов.

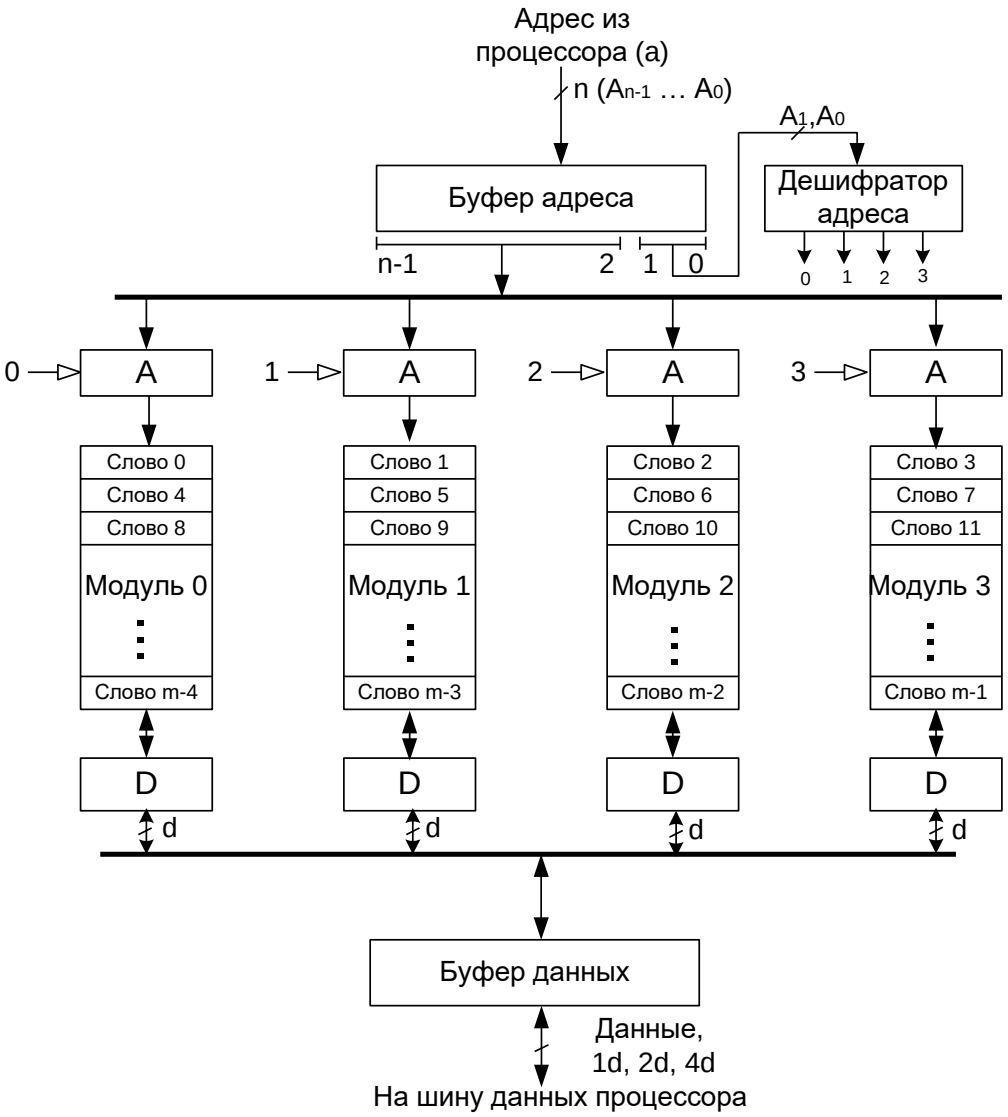


Рисунок 2.3- Схема памяти с расслоением.

Чередование адресов обеспечивается за счет циклического разбиения адреса. Для выбора модуля используются младшие разряды адреса. В приведенной на рисунке 2.3 схеме используются два младших разряда адреса A_1 и A_0 для выбора модуля памяти (см. рисунок 2.4)**,** а для выбора ячейки в модуле - старшие разрядов (A_{n-1} - A_2)**.**

A_1	A_0	№ модуля памяти
0	0	0
0	1	1

1	0	2
1	1	3

Рисунок 2.4- Таблица выбора модулей памяти по значению младших разрядов адреса

Хотя адреса памяти могут генерироваться с тактовой частотой процессора, но в каждом такте на шине адреса может присутствовать адрес только одной ячейки, т.е. параллельное обращение к нескольким модулям невозможно. Поэтому исполнительный адрес поступает на буфер адреса со смещением на один процессорный такт $t_т$. Адрес ячейки запоминается в индивидуальном регистре адреса А, на который указывает дешифратора адреса. Дальнейшие операции по доступу к ячейке в каждом модуле протекают независимо. В начальный момент времени процессор обращается к модулю 0 и запускается цикл памяти. Не дожидаясь окончания цикла памяти, производится обращение к модулю 1 и т.д. конвейерным методом. Следующее обращение к модулю памяти должно состояться не ранее чем закончится цикл этого модуля $t_ц$ (рисунок 2.5). При обращениях к последовательным соседним адресам можно добиться *n*- кратного увеличения скорости обращения за счет параллельного действия всех блоков. Подобная процедура распределения адресов по блокам называется *n*- расслоением обращений к памяти.

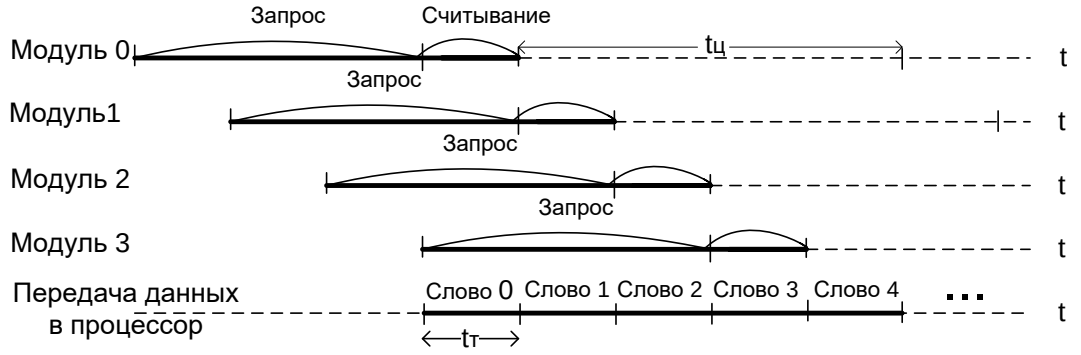


Рисунок 2.5 - Временная диаграмма работы памяти с расслоением.

Передача слов между регистрами D и буфером данных осуществляется через мультиплексор/демультиплексор (на рисунке 2.3 не показан). Прочитанные из памяти данные направляются в процессора с размерностью в одно слово (1d), двойного слова (2d) или четверного слова (4d). Передача данных в режиме записи производится пословно. Схема управления записью/ чтением данных (на рисунке не показана) является общей для всех блоков и функционирует в режиме разделения времени. Таким образом, конструктивно расслоенная память представляет собой единое устройство.

Если же запросы к одному и тому же модулю следуют друг за другом, каждый следующий запрос должен ожидать завершения обслуживания предыдущего. Такая ситуация называется *конфликтом по доступу.* При частом возникновении конфликтов по доступу метод чередования адресов становится неэффективным.

Устройство памяти с расслоением может применяться как в одно- так и многопроцессорных системах, где требуется очень высокая пропускная. Число модулей памяти *n* варьируется обычно в пределах 2—16. В некоторых случаях их число может достигать 64 или даже 128. В некоторых ВС возможен отказ от применения кэш, если используется память с высокой степенью расслоения. Это обстоятельство важно для многопроцессорных систем, где использование кэш в процессорных узлах наряду с положительными качествами создает проблемы, например, проблему обеспечения согласованности разделяемых (общих для всех процессоров) данных или кэш-когерентности (см. раздел ?).

Банковая организация памяти

Традиционные способы расслоения памяти хорошо работают в рамках одной задачи, для которой характерно свойство локальности. В многопроцессорных системах с общей памятью, где запросы на доступ к памяти

достаточно независимы, чаще применяют другой подход, который можно рассматривать как развитие идеи расслоения памяти. Для этого в систему включают несколько банков памяти со своими контроллерами, что позволяет отдельным модулям работать полностью автономно. Причем каждый модуль может быть выполнен с чередованием адресов, что обеспечивает дополнительное повышения пропускной способности памяти.

Обычно такая архитектура памяти применяется в многопроцессорных системах с общей шиной. Структурно-функциональная организация памяти показана на рисунке 2.6. Процессоры взаимодействуют с памятью через внешнюю общую шину (на схеме не показана), работающую в режиме временного мультиплексирования и расщепления транзакций (см. раздел?). Процессоры выставляют адрес памяти на шину адреса, причем старшие разряды используются для выбора модуля с помощью дешифратора адреса, а младшие - для адресации внутри модуля последовательно как в обычной памяти.

Эффективность банковой организации памяти напрямую зависит от частоты обращений процессоров к разным модулям. Уменьшение вероятности последовательных обращений к одному и тому же модулю памяти можно добиться при большом числе модулей. Так, в суперкомпьютере NEC SX/3 основная память состоит из 128 модулей. Кроме того, в целях сокращения конфликтов по доступу необходимо, чтобы компилятор или операционная система, учитывая архитектуру памяти, распределяли сегменты программ и данных в разные модули.

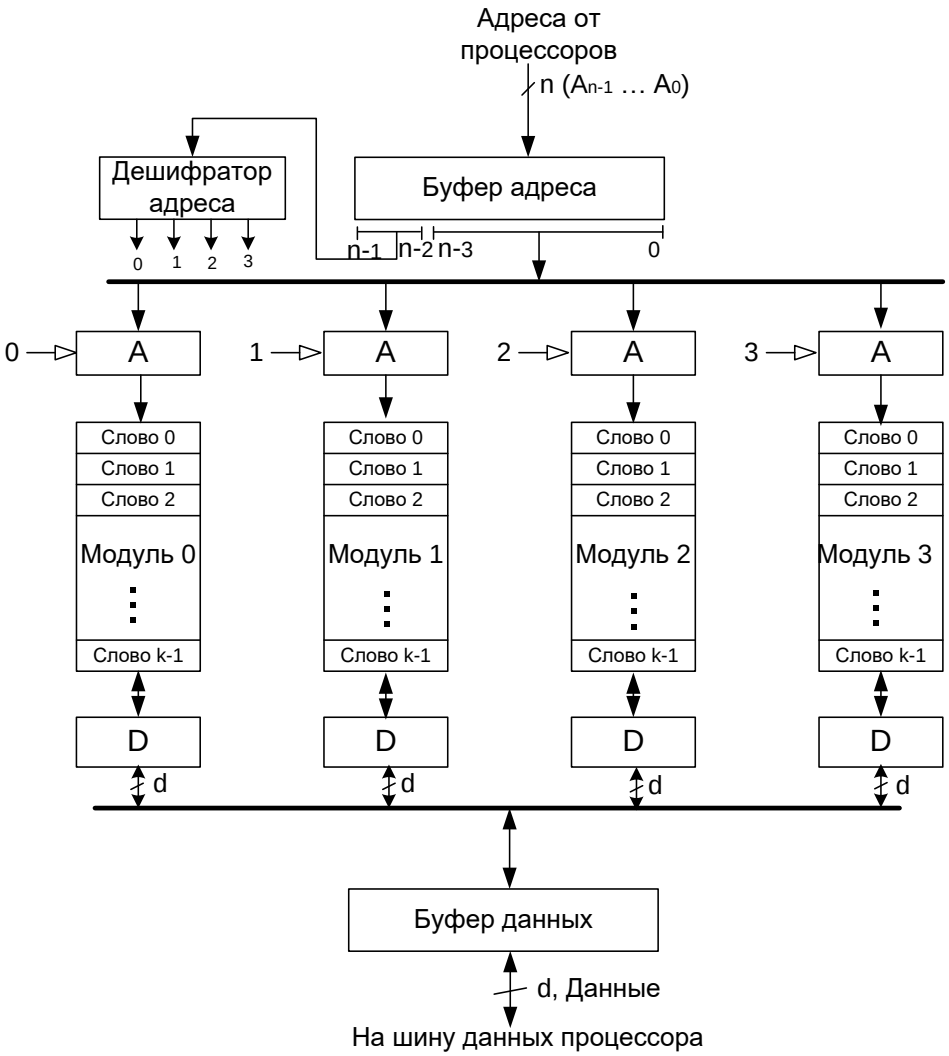


Рисунок 2.6- Схема банковой памяти

Многовходовая память.

Дуальная память. Часто процессоры работают с непрерывными потоками данных большой размерности. Поскольку время доступа к оперативной памяти составляет десятки процессорных тактов, то процессор вынужден часто простаивать, теряя реальную производительность. Особенно эта ситуация недопустима в системах реального времени, где должна быть задействована вся мощность процессора, чтобы успеть обработать быстротекущие процессы.

Один из способов повышения пропускной способности памяти заключается в использовании дуальной или переключающейся памяти.

Организуется такая память из двух или более буферных блоков сверхбыстродействующей памяти с произвольным доступом, которые соединяются так, что пока одни из буферов осуществляет обмен с оперативной памятью (ОП), другие остаются подключенным к центральному процессору (ЦП) и снабжают его данными. По окончании обменов ОП и процессора с подключенными к ним буферами, связи этих буферов взаимно переключаются. ЦП получает доступ к новой партии данных, а ОП получает возможность обмена данными с другими буферными модулями (рис.2.7).

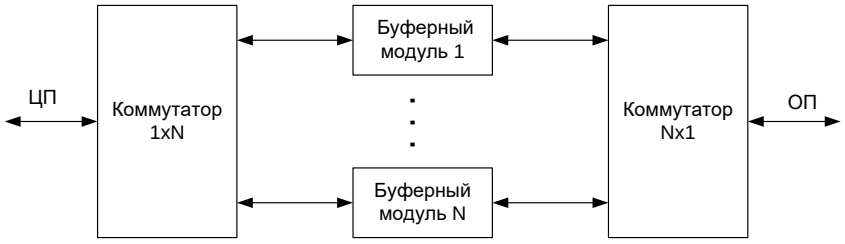


Рисунок 2.7- Структура дуальной памяти с N буферными модулями

Многопортовая память. Стандартная однопортовая память имеет по одной шине адреса, данных и управления и в каждый момент времени обеспечивает доступ к ячейке памяти только одному устройству. В отличие от стандартной в *n*-портовой памяти имеется *n* независимых наборов шин адреса, данных и управления, гарантирующих одновременный и независимый доступ к ОЗУ *n* устройствам. Данное свойство позволяет существенно упростить создание многопроцессорных, многомашинных и конвейерных вычислительных систем, где многопортовая память выступает в роли общей или совместно используемой памяти. В настоящее время серийно выпускаются двух- и четырехпортовые микросхемы, среди которых наиболее распространены первые. Поскольку архитектурные решения в обоих случаях схожи, дальнейшее изложение будет вестись применительно к двухпортовым ОЗУ.

Запоминающий элемент (ЗЭ) двухпортового ОЗУ (см. рис. 2.8) содержит шесть транзисторов, в отличие от стандартного ЗЭ, где число транзисторов

ограничивается четырьмя. Транзисторы Т3, Т4, Т5 и Т6 используются в качестве внутреннего коммутатора выходов запоминающего элемента с прямыми и инверсными линиями битов $P1$ ($\overline{P1}$) $P2$ и ($\overline{P2}$). Запоминающие элементы объединяются в матрицу, которую называют запоминающим массивом.

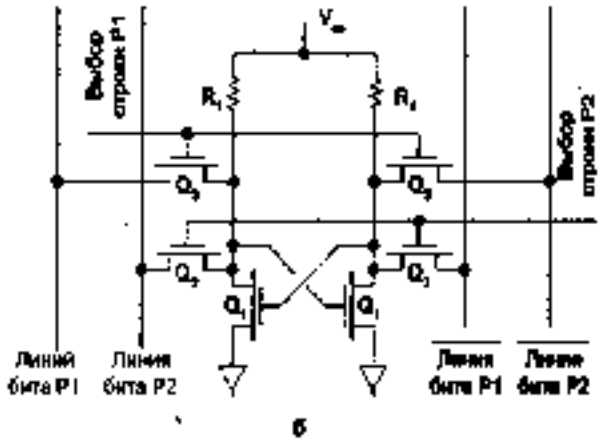


Рисунок 2.8- Запоминающий элемент двухпортового статического ОЗУ

Линии битов запоминающего массива образуют две шины данных модуля памяти, а линии выбора строк соответственно две шины адреса. Таким образом, в двухпортовой памяти имеются два набора адресных шин, шин данных и управляющих шин, каждая из которых обеспечивает доступ к общему массиву ЗЭ (рис. 2.9). Соответственно в ОЗУ с произвольным доступом имеется по два комплекта логических схем дешифрации адресов и буферных регистров данных (портов), которые обеспечивают одновременный доступ к её различным словам со стороны двух устройств (например, к видеопамяти со стороны процессора и видеоконтроллера) через два порта чтения/записи. Стоимость таких устройств высока, что ограничивает их практическое применение специализированными устройствами.

Поскольку двухпортовому ОЗУ свойственна симметричная структура, в дальнейшем наборы шин будем называть «левым» и «правым». В целом организация матрицы ЗЭ остается традиционной. Доступ к ячейкам возможен как через левую, так и через правую группу шин, причем, если адреса различны, никаких конфликтов не возникает. Проблемы потенциально возможны, когда левые и правые устройства одновременно обращаются по одному и тому же

адресу и хотя бы одно из этих устройств пытается выполнить операцию записи. В этом случае, если один из портов читает информацию, а другой производит запись в ту же ячейку, вероятно считывание недостоверной информации.

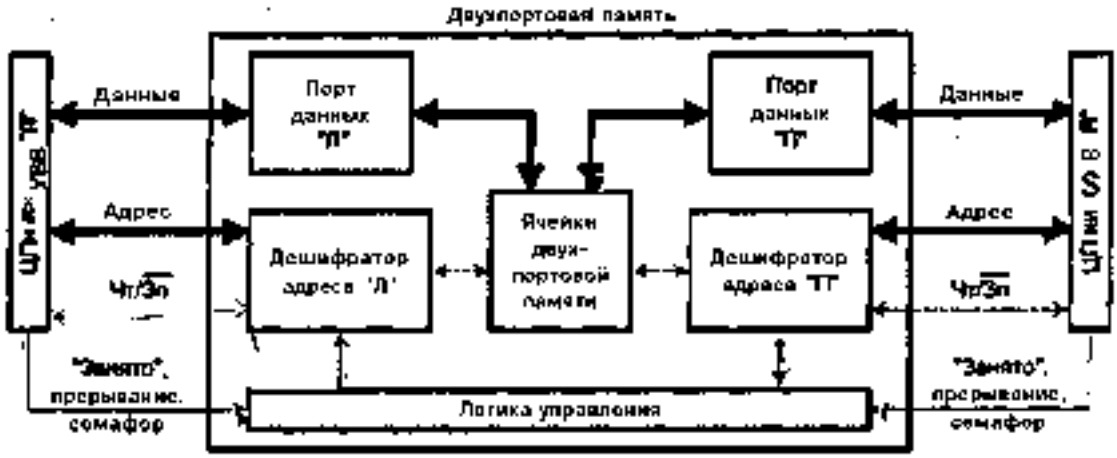


Рис. 5.13 Структура двухпортового ОЗУ

Рисунок 2.9- Функциональная организация двухпортовой памяти

При попытке единовременной записи в одну ячейку с двух направлений в нее может быть занесено неверное слово. Несмотря на то, что вероятность подобных ситуаций по оценкам не превышает 0,1%, такой вариант необходимо учитывать, для чего в двухпортовой памяти имеется схема арбитража.

Логика арбитража в микросхеме реализована аппаратными средствами. Схема обеспечивает формирование сигнала «Занято», запрещающего запись в ячейку со стороны другого порта, на котором адрес появится позже. Кроме того, арбитр производит выбор одного из входных портов при одновременном поступлении адресов одновременно на оба порта. Для этого арбитр снабжают двумя комплектами компараторов адреса, двумя буферами задержки (левые и правые) и триггером-защелкой для фиксации пары сигналов «блокировка» (левый и правый). Триггер-защелка выполняет роль битового семафора, обеспечивающего защиту общего ресурса - ячейки памяти - от одновременного доступа к ней со стороны двух запрашивающих устройств. Семафор при захвате ячейки одним из устройств, блокирует другое

запрашивающее устройство на время выполнения обмена с памятью первого устройства.

Ортогональная память.

Структурная организация памяти этого типа приведена на рис.2.10. В такой памяти данные, расположенные в запоминающих элементах (ЗЭ) строки, рассматриваются как элементы единого информационного сообщения - слова. Доступ к данным в ортогональной памяти, во-первых, обеспечивается указанием номера (адреса) строки. В этом случае обеспечивается параллельный доступ ко всем ЗЭ, расположенным в строке с указанным номером. Таким образом, доступ к данным в памяти выполняется последовательно по словам. Во-вторых, обеспечена возможность доступа к ЗЭ, расположенным в одном столбце. Запоминающие элементы являются битовыми и могут объединятся в байтовые ячейки. В первом случае данные столбца называют битовым срезом, во втором – байтовым срезом. Доступ к данным столбца обеспечивается указанием номера (адрес) столбца. В итоге, в памяти с ортогональной организацией выполняется высокоскоростной доступ к данным последовательно по словам или последовательно по срезам (битовым или байтовым).

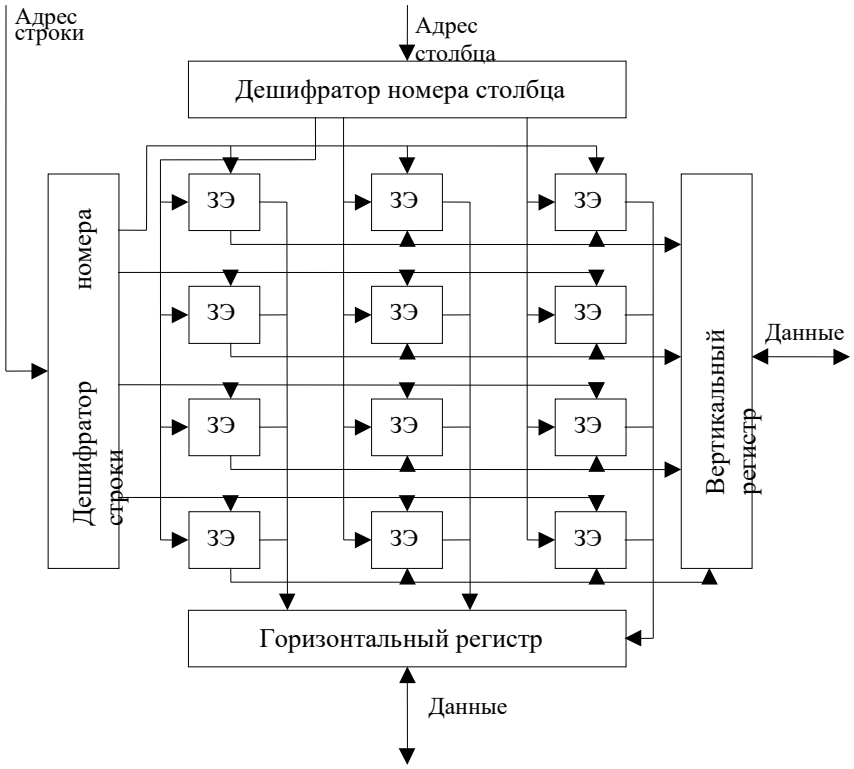


Рисунок 2.10 – Структура ортогональной памяти

Ассоциативная память.

При использовании адресной памяти операнды записываются в память и считываются из памяти поочередно (последовательно), в то время как при решении многих задач эти операции целесообразно выполнять одновременно (параллельно). С целью упрощения поиска операндов и обеспечения возможности одновременного доступа ко многим ячейкам памяти используется адресация по содержанию. Ввиду высокой стоимости и технической сложности память с такой адресацией в ВС используется в качестве локальной памяти и применяется для выполнение операций свертки, поиска и сортировки.

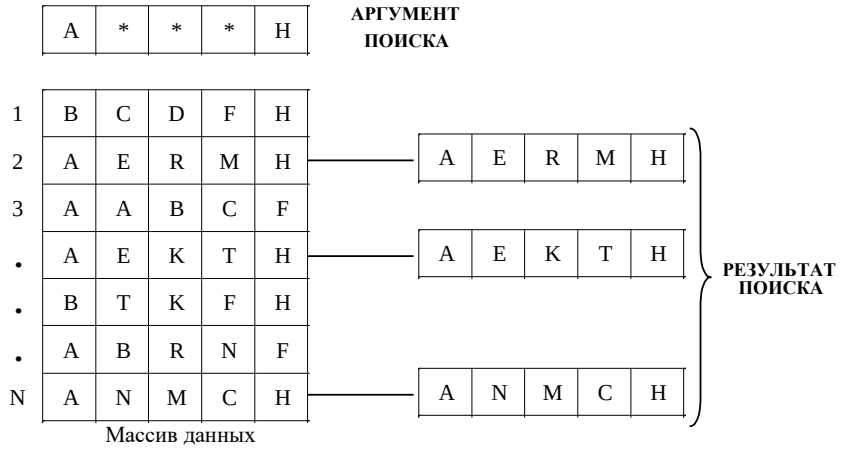


Рисунок 2.11- Принцип адресации по содержанию

Сущность принципа адресации по содержанию заключается в следующем (рис.2.11). Имеется массив данных емкостью N слов. Требуется найти в массиве все слова, которые начинаются с символа "А" и кончаются символом "Н". В этом случае аргументом поиска является слово А***Н, где значком * отмечены разряды, не влияющие на результат поиска. Запоминающий массив на аппаратном уровне строится таким образом, что на выходе ячеек памяти, содержимое которых совпадает со значением поступившего аргумента поиска, появляется сигнал - указатель совпадения. Далее по выработанным сигналам совпадения выполняется выборка ячеек памяти.

Реализация ассоциативного доступа к данным может быть организована одним из двух способов:

1) *программным, основанным на распределении памяти в зависимости от содержания данных и реализуемым с помощью программных средств (метод хеширования. При этом предполагается, что информация хранится в обычном запоминающем устройстве произвольного доступа со стандартной системой адресации;*
2) *аппаратным, опирающимся на применение специальных аппаратных средств, предназначенных для хранения и ассоциативного поиска элементов данных.*

В методах программного поиска доступ к любой информационной единице (записи, документу, массиву или элементу данных и т. п.) осуществляется непосредственно: либо по специальному ключу, либо по некоторому фрагменту самой информационной единицы. Такие методы называют методами хеширования (или другие названия их: адресация в разброс, адресация с перемешиванием, ключевые преобразования и т.п.).

Аппаратная реализация ассоциативной памяти должна обеспечить, как минимум, следующие функции:

- *осуществлять операцию поиска элемента по некоторому аргументу;*
- *выполнять считывание (запись) информации;*
- *возвращать в исходное состояние память фиксации реакций;*
- *анализировать многократность совпадения и осуществлять выборку по совпадению.*

В параллельных ассоциативных ЗУ (АЗУ) процесс поиска данных по содержанию организуется следующим образом. Каждая ячейка памяти модуля памяти АЗУ обеспечивает выполнение функций приема, хранения данных, сравнения хранимой информации с аргументом поиска и выработку сигналов о результате сравнения (рис.2.12).

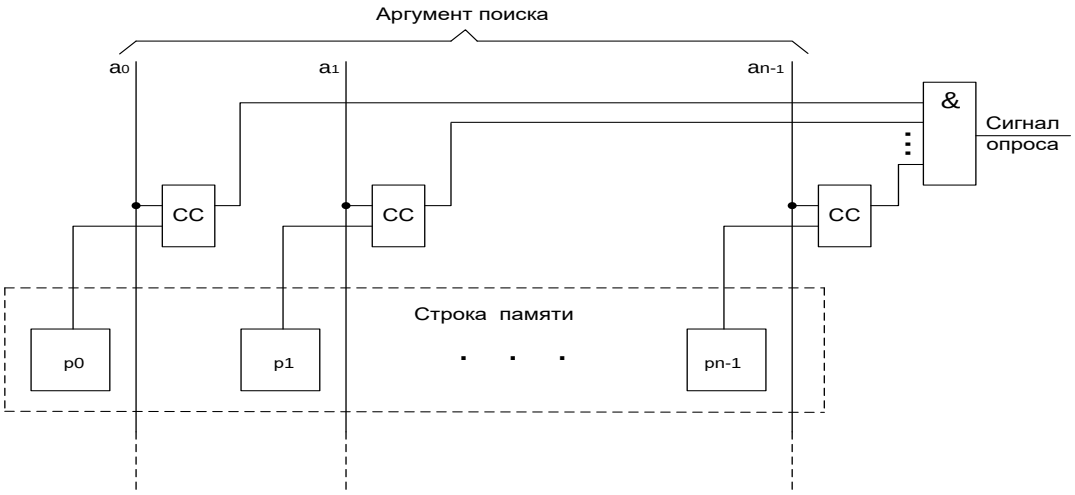


Рисунок 2.12- Упрощенная функциональная схема ячейки АЗУ: СС- схемы сравнения; $p_{0,}...,p_{n-1}$- разряды строки памяти, $a_0,...,a_{n-1}$- разряды аргумента поиска

Модуль памяти организован таким образом, что на каждом цикле работы аргумент поиска поступает параллельно во все ячейки АЗУ. В результате в модуле памяти АЗУ выполняется массовая операция сравнения содержимого ячейки памяти с аргументом поиска и установка - сигналов указателей о совпадении на выходе. Ассоциативную память иногда называют ассоциативным процессором, поскольку она не только хранит данные, но производит их обработку.

Как правило, ячейки памяти АЗУ, расположенные в одной строке, рассматриваются как слово. В операции сравнения могут участвовать не все разряды строки ($p_0,...,p_{n-1}$), а только та их часть, которая активизируется управляющим словом, которое называют вектором-маской или просто маской. Например, разряд ячейки памяти переводится в активное состояние, если соответствующий ей разряд маски равен единице, в противном случае этот разряд в операции сравнения не участвует.

АЗУ (рисунок 2.13) подключается к шинам адреса, данных центрального процессора обычным образом (в виде ускорителя для операций поиска). Перед выполнением операции в модуль памяти (МП) вводится

массив данных, для чего он переводится в режим прямоадресуемой памяти. Центральный процессор выставляет адрес ячейки на шину адреса, который используется для выбора (селектирования) соответствующего модуля памяти (если их несколько). Затем адрес дешифрируется логическими схемами АЗУ, указывая выходным сигналом дешифратора в какую ячейку следует осуществить запись. Слово данных по шине данных помещается в адресуемую ячейку по входной шине данных.

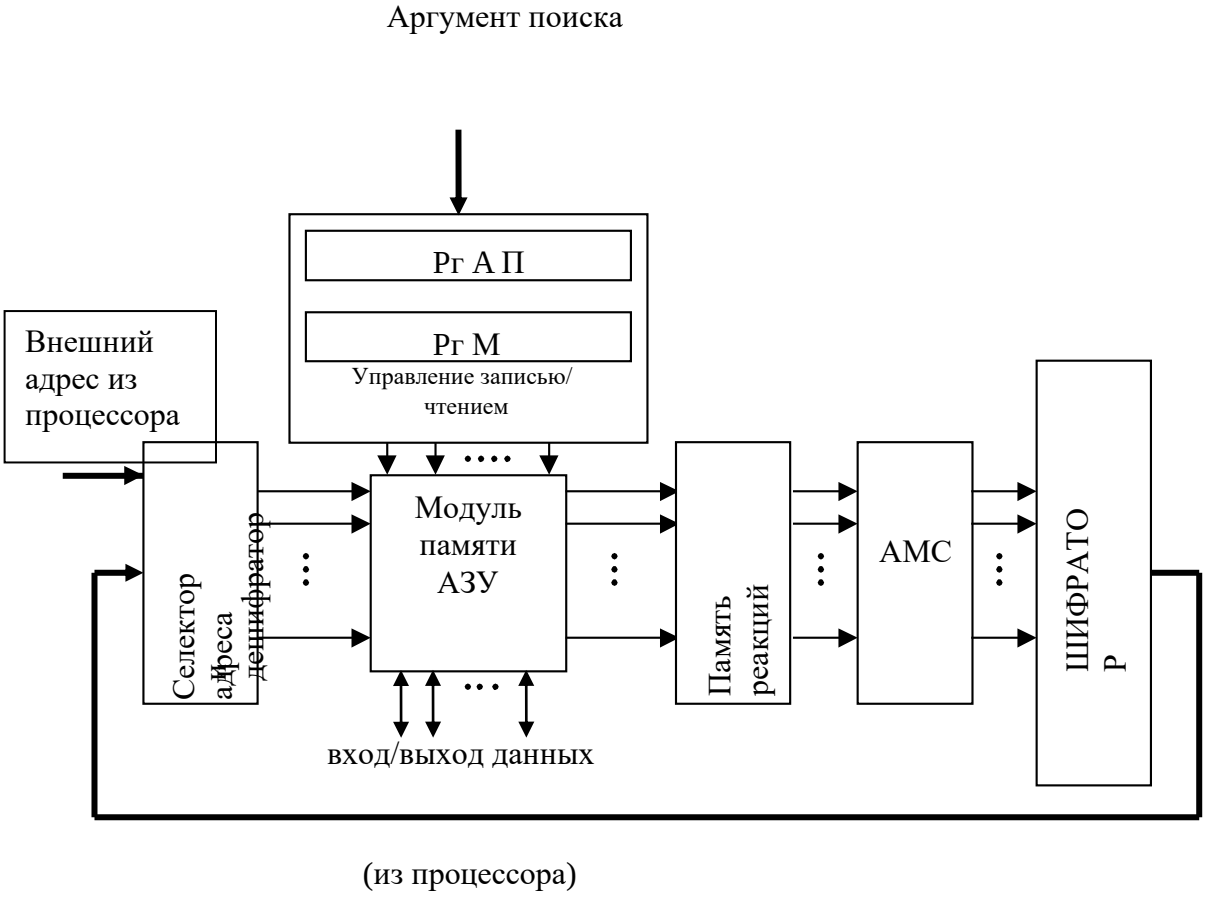


Рис.2.13- Структура параллельного ассоциативного ЗУ

При выборке данных из АЗУ с параллельной организацией в регистр аргумента поиска (РгАП) вводится аргумент поиска, в регистр маски (РгМ) – слово маски. Аргумент поиска и слово маски также формируются центральным процессором. Данные из РгАП и РгМ параллельно поступают в

ячейки памяти АЗУ, выполняется операция сравнения, по окончании которой на выходах тех строк модуля памяти АЗУ, в которых произошло совпадение с заданным аргументом поиска, вырабатываются сигналы опроса. Эти сигналы фиксируются в памяти реакций и в виде унитарного кода подаются в анализатор многократных совпадений (АМС). В АМС выполняется их приоритетный разбор, по результатам которого формируется последовательность выдачи сигналов в шифратор, который соответственно преобразует унитарный код в двоичный код адреса. Этот адрес используется для выборки данных из тех ячеек АЗУ, в которых произошло совпадение.

В принципе сигналы с АМС можно подавать непосредственно на адресные шины модуля памяти АЗУ, что обеспечило бы доступ ко всем "откликнувшимся" ячейкам одновременно. Однако на практике модуль памяти изготавливается из стандартных микросхем памяти с встроенными дешифраторами адреса. Последнее позволяет использовать АЗУ и в режиме адресуемой памяти для ввода массива данных из центрального процессора.

Ввод данных в АЗУ может производиться в «разбег» после предварительного поиска в модуле памяти АЗУ свободных ячеек памяти. С этой целью в ячейке предусматривается специальный бит - бит занятости, который устанавливается в единицу при вводе данных в неё слова данных. Если в модуле памяти АЗУ выявлено несколько свободных строк, ввод данных выполняется по приоритету, определяемому в АМС.

В последовательно-поразрядных или последовательно-побайтных АЗУ (рис. 2. 14) модуль памяти реализуется в виде ортогонального ОЗУ, т. е. с возможностью выборки как по строкам, так и по столбцам. В РгАП вводится слово аргумента поиска, в РгМ – слово маски. Далее выполняется опрос первого столбца модуля памяти АЗУ. Данные поступают на первые входы блока схем сравнения, на вторые входы которого поступает соответствующая часть аргумента поиска из РгАП и часть слова маски из РгМ. Результаты сравнения фиксируются в памяти реакций. Затем выполняется сдвиг

регистров РгАП и РгМ, после чего выполняется опрос второго столбца ячеек памяти.

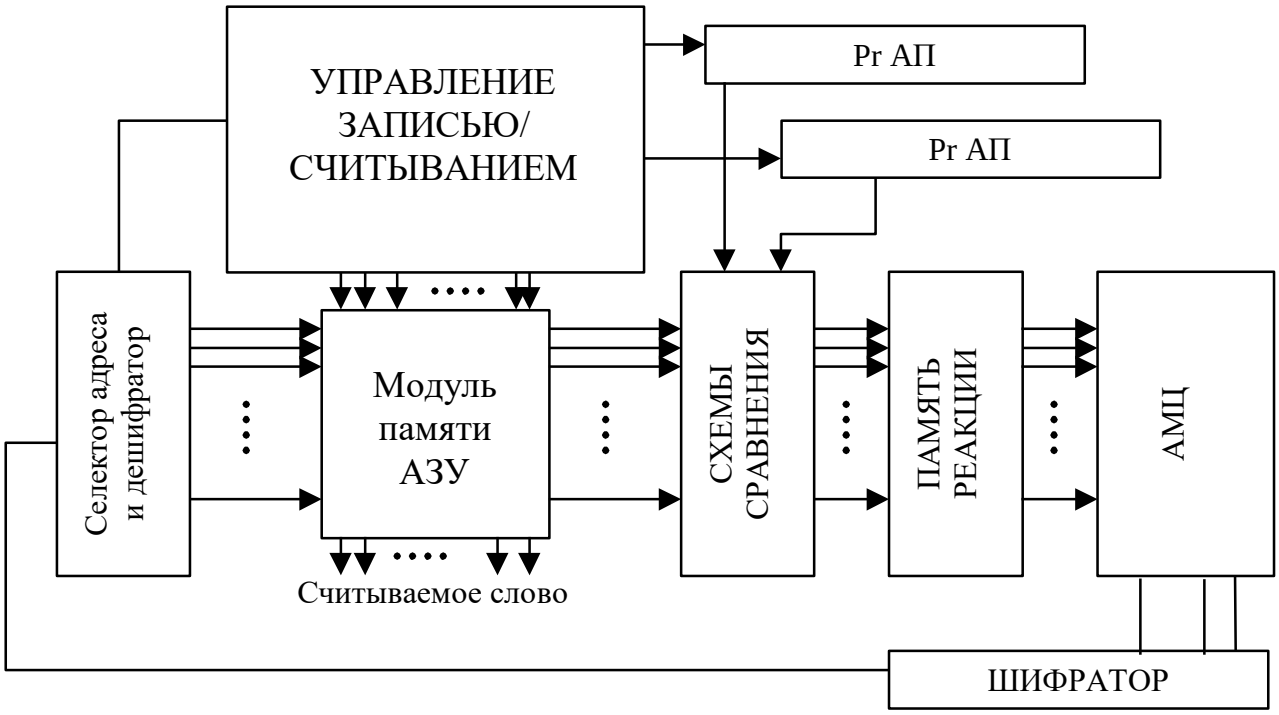


Рис.2.14

Сигналы совпадения фиксируются элементами памяти реакции только в том случае, если в эти же элементы поступали и сигналы о совпадении на предыдущем такте. В случае если на текущем такте сигнала совпадения нет, а на предыдущем он был, то соответствующий элемент памяти реакции фиксирует отсутствие совпадений. Описанный процесс повторяется до тех пор, пока не будет выполнен опрос последнего столбца модуля памяти АЗУ. В итоге в памяти реакций будут зафиксированы сигналы - указатели строк ячеек памяти, в которых произошло совпадение с аргументом поиска. В остальном АЗУ этого типа работает полностью аналогично параллельному АЗУ. Таким образом, параллельные АЗУ независимо от емкости обеспечивают выборку данных, удовлетворяющих заданному аргументу поиска, за один цикл памяти. Однако при этом требуется большой объем схемной логики в каждой ячейке памяти, что приводит к увеличению стоимости устройства. В АЗУ с последовательно-поразрядной организацией

выработка данных обеспечена за *n* циклов памяти, где *n* - число разрядов в строке модуля памяти. Но объем схемной логики для выполнения операций сравнения с аргументом поиска значительно меньше. На практике применяют оба типа АЗУ, причем выбор того или иного типа определяется требованиями быстродействия и стоимости.

Кэш-память

Кэш-память предназначена для уменьшения времени доступа к данным, размещенным в основной (оперативной) памяти (ОП). В отличие от оперативной памяти, где адреса ячеек фиксированы и располагаются в последовательных ячейках, кэш хранит данные в любом порядке, но вместе с их адресом.

В структуру кэш-памяти, как показано на рис.1.18, входят массив данных и справочник. Данные в кэш размещаются строками длиной в несколько слов (обычно 4 - 8). В строке хранится также начальный адрес данных, соответствующий адресу данных в оперативной памяти, и необходимыми управляющими битами (рисунок 1.19). Оперативная память также условно делится на строки размером, равным по величине размеру поля данных кэш. Чтение данных из памяти производится целой строкой за одну транзакцию. Запись в отличие от чтения производится обычно пословно. Для выполнения таких операций контроллер ОП и кэш имеют соответствующие аппаратные средства.

Процессор при выполнении команды обращается в первую очередь к кэш-памяти. Если данные обнаружены в кэш, т. е. адрес, выставленный процессором, обнаружен в кэш, то получен быстрый ответ. При отсутствии в кэш требуемой строки она копируются в кэш из основной памяти. Одновременно требуемое слово процессор из этой строки заносится в процессор (медленный ответ).

Эквивалентное время обращения к памяти t_{on}, определенное по формуле (1.1), равно

$$t_e = t_k + p_{miss} t_{m,} \qquad (1.2)$$

где t_k— время обращения к кэш, t_m— время обращения к основной памяти, p_{miss} — вероятность кэш – промаха, т.е. вероятность того, что требуемая информация отсутствует в кэш. Обычно t_k меньше t_m на порядок или более и при достаточном уменьшении p_{miss} можно достичь t_e приблизительно равным t_k. По мере повышения быстродействия логических элементов повышается быстродействие и процессора. При этом сокращается и время обращения к кэш.

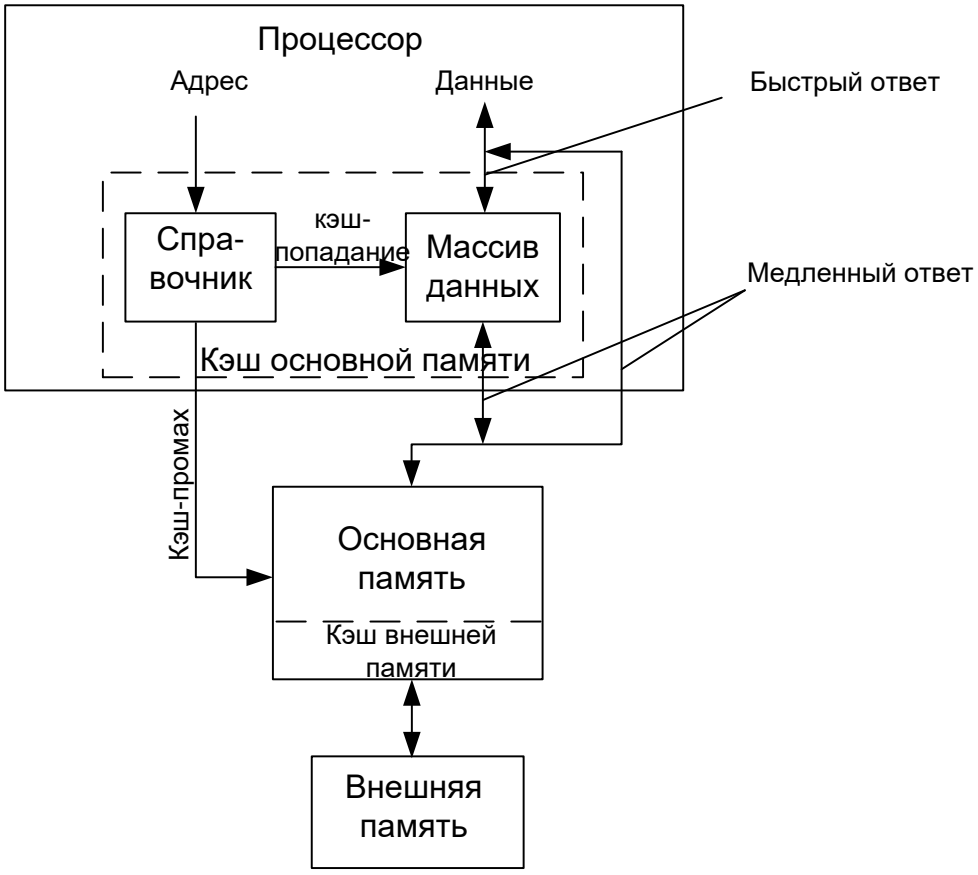


Рис.1.18. Структурная организация и место кэш в подсистеме памяти

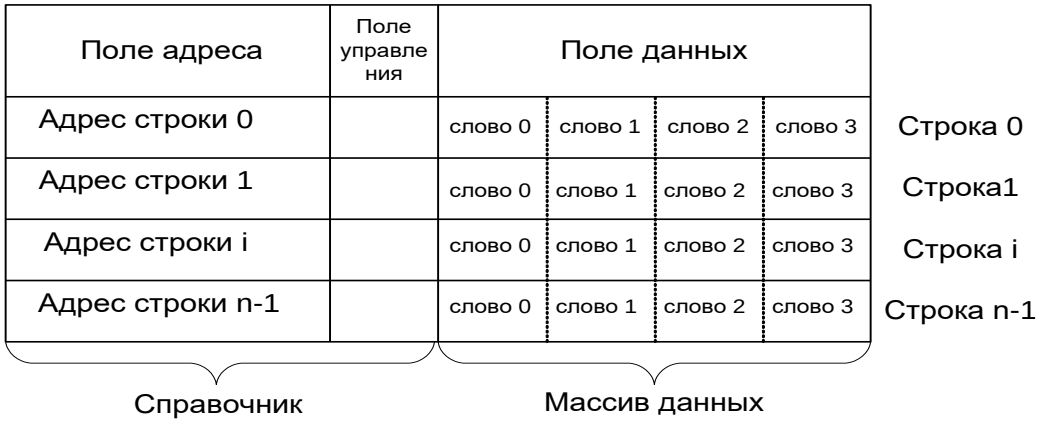


Рисунок 1.19– Информационная структура кэш

Однако в силу постоянно растущих требований к увеличению объема памяти время обращения к основной памяти существенно уменьшить трудно, и разница между t_k и t_m увеличивается, что приводит к снижению эффективности кэш-памяти. Преодолевается эта проблема за счет увеличения емкости кэш-памяти или применения многоуровневой структуры кэш.

Подобный подход к повышению пропускной способности может, например, использоваться для повышения скорости обмена данными между основной памятью и накопителями внешней памяти (ВП). Используя кэш - память емкостью несколько десятков мегабайт, реализованную в виде отдельных микросхем памяти или выделенную в специальной области оперативной памяти (электронный диск), можно увеличить эквивалентную скорость обращения к магнитным дискам примерно в 2—10 раз. Таким образом, кэширование является универсальным методом, пригодным для ускорения доступа к оперативной памяти, к накопителям на дисках и к другим видам запоминающих устройств.

Виртуальную память также можно считать одним из вариантов реализации принципа кэширования данных, при котором оперативная память выступает в роли кэш по отношению к внешней памяти — жесткому диску. И в этом случае кэширование используется не только для того, чтобы уменьшить время доступа к информации, но и для того, чтобы использовать диск для временного хранения неиспользуемых некоторое время программ и данных с

целью освобождения места для активных процессов. В результате наиболее интенсивно используемые программы и данные находятся в оперативной памяти, остальная же информация хранится в более объемной и менее дорогостоящей внешней памяти.

Одна из возможных схем кэширования представлена на рис.1.19. Содержимое кэш-памяти представляет собой совокупность записей обо всех загруженных в нее строках из основной памяти. Каждая запись о строке включает в себя три поля:

- адрес, который эта строка имеет в основной памяти;
- дополнительную информацию, которая используется для реализации алгоритма замещения данных в кэш и обычно включает признак модификации, признак действительности данных и др.
- значение элементов данных (слов);

Первое поле зачастую называют полем тегов, второе – полем управления, третье - полем данных.

При каждом обращении к основной памяти по физическому адресу просматривается содержимое кэш-памяти с целью определения, не находятся ли там нужные данные. Кэш-память не является адресуемой, поэтому поиск нужных данных осуществляется по содержимому, т.е. по взятому из запроса значению адреса слова в оперативной памяти. Далее возможен один из двух вариантов развития событий:

- если строка, содержащая слово данных обнаруживаются в кэш-памяти, т.е. произошло кэш-попадание (cache-hit), слово считывается из нее и результат передается процессору;
- если данные отсутствуют в кэш-памяти, то есть произошел кэш-промах (cache-miss), строка с запрашиваемыми данными считывается из основной памяти, копируется в кэш-память и одновременно с этим нужное слово передается процессору.

Среднее время доступа к данным в системе с кэш-памятью линейно зависит от вероятности попадания в кэш. Очевидно, что использование кэш-памяти имеет смысл только при высокой вероятности кэш-попадания.

Вероятность обнаружения данных в кэш зависит от разных факторов, таких, например, как объем кэш, объем кэшируемой памяти, алгоритм замещения данных в кэш, особенности выполняемой программы, время ее работы, уровень мультипрограммирования и других особенностей вычислительного процесса. Тем не менее, в большинстве реализаций кэш-памяти процент кэш-попадании оказывается весьма высоким — свыше 90 %. Такое высокое значение вероятности нахождения данных в кэш-памяти объясняется наличием у программ и данных объективных свойств, называемых локальностью ссылок, которая подразделяется на пространственную и временную локальность.

Временная локальность заключается в том, что если произошло обращение по некоторому адресу, то следующее обращение по тому же адресу с большой вероятностью произойдет в ближайшее время. Пространственная локальность заключается в том, что если произошло обращение по некоторому адресу, то с высокой степенью вероятности в ближайшее время произойдет обращение к соседним адресам.

Основываясь на свойстве временной локальности, программы и данные, только что считанные из основной памяти, размещают в запоминающем устройстве быстрого доступа, предполагая, что скоро они опять понадобятся. Вначале работы системы, когда кэш-память еще пуста, происходит процедура её заполнения. В этот промежуток времени практически каждый запрос процессора к основной памяти выполняется по следующей схеме: просмотр кэш, констатация промаха, чтение данных из основной памяти, передача результата источнику запроса с одновременным копированием данных в кэш. Затем, по мере заполнения кэш, в полном соответствии со свойством временной локальности возрастает вероятность обращения к данным, которые уже были использованы на предыдущем этапе работы системы, то есть к

данным, которые содержатся в кэш и могут быть считаны значительно быстрее, чем из основной памяти.

Свойство пространственной локальности также используется для увеличения вероятности кэш-попадания: как правило, в кэш-память считывается не одно слово, к которому произошло обращение, а целый блок данных, расположенный в основной памяти в непосредственной близости с данным словом. Поскольку при выполнении программы очень высока вероятность, что команды выбираются из памяти последовательно одна за другой из соседних ячеек, то имеет смысл загружать в кэш-память целый фрагмент программы. Аналогично если программа ведет обработку некоторого массива данных, то ее работу можно ускорить, загрузив в кэш часть или даже весь массив данных. При этом учитывается высокая вероятность того, что значительное число обращений к памяти будет выполняться к адресам массива данных.

Проблема согласования данных

В процессе функционирования вычислительной системы содержимое кэш-памяти постоянно обновляется. Время от времени устаревшие данные из нее вытесняются для освобождения места под новые данные. Реально вытеснение означает либо простое объявление свободной соответствующей области кэш-памяти (сброс бита действительности), если вытесняемые данные за время нахождения в кэш не были изменены, либо в дополнение к этому копирование данных в основную память, если они были модифицированы. Алгоритм замены данных в кэш-памяти существенно влияет на ее эффективность. Алгоритм должен, во-первых, быть максимально быстрым, чтобы не замедлять работу кэш-памяти, а во-вторых, обеспечивать максимально возможную вероятность кэш-попаданий. Поскольку из-за непредсказуемости вычислительного процесса ни один алгоритм замещения данных в кэш-памяти не может гарантировать идеальный результат,

разработчики ограничиваются рациональными решениями, которые, по крайней мере, не сильно замедляют работу кэш.

Наличие в ЭВМ двух копий данных - в основной памяти и в кэш - порождает проблему согласования данных. Если происходит запись в основную память по некоторому адресу, а содержимое этой ячейки находится в кэш, то в результате соответствующая запись в кэш становится недостоверной. Рассмотрим два подхода к решению этой проблемы.

Сквозная запись (write through).

При каждом запросе к основной памяти, в том числе и при записи, просматривается кэш. Если строка с запрашиваемым адресом отсутствует в кэш (для этого просматривают поле тегов), то запись выполняется только в основную память. Если же адрес, по которому выполняется обращение, находится и в кэш, то запись производится одновременно в кэш и основную память. Такая операция по времени равносильна обращению в основную (оперативную) память и не дает большого выигрыша от применения кэш-памяти.

Обратная запись (write back).

Аналогично при возникновении запроса к памяти выполняется просмотр кэш, и если строка с запрашиваемым адресом там отсутствует, то запись выполняется только в основную память. В противном же случае запись производится только в кэш-память, при этом в поле управления делается специальная отметка (признак модификации), которая указывает на то, что при вытеснении этих данных из кэш необходимо переписать их в подходящий момент в основную память, чтобы обновить устаревшее содержимое основной памяти.

Очевидно, что в этом случае в соответствии с принципами временной и пространственной локальности, обращения будут происходить преимущественно к кэш-памяти, а малая их доля упадёт на основную память. Поэтому временная

эффективность метода обратной записи высокая, хотя требует применения более сложных алгоритмов согласования данных. Контроллер кэш-памяти осуществляет копирование модифицированной строки при любых замещениях строки. Модифицированные данные могут выгружаться не только при освобождении места в кэш-памяти для новых данных, но и в тех моментах времени, когда шинный интерфейс не занят процессором или другими устройствами. В некоторых алгоритмах замещения предусматривается первоочередная выгрузка модифицированных, или, как еще говорят, «грязных» данных.

Механизм выполнения запросов в системах с двухуровневой кэш-памятью

Когда выполняется транзакция записи, кэш просматривается только с целью согласования его содержимого и содержимого основной памяти. Если происходит промах, то запросы на запись удовлетворяются только в оперативной памяти, не вызывая никаких изменений в содержимом кэш (рисунок 1.18). В некоторых же реализациях кэш-памяти при отсутствии данных в кэш они копируются туда из основной памяти независимо от того, выполняется запрос на чтение или на запись.

В соответствии с описанной логикой работы кэш-памяти следует, что при возникновении запроса сначала просматривается кэш, а затем, если произошел промах, выполняется обращение к основной памяти. Однако часто реализуется и другая схема работы кэш: поиск в кэш и в основной памяти начинается одновременно, а затем, в зависимости от результата просмотра кэш, операция в основной памяти либо продолжается, либо прерывается.

Согласование (когерентность) данных требует значительных временных затрат. Проблема заключается в том, что при кэш-промахе во время выполнения операции чтения процессор приостанавливается на время, пока данные не будут получены из оперативной памяти. Это время может составлять

несколько десятков или даже сотню или более тактов процессора. Чтобы избежать задержек во многих вычислительных системах используется двухуровневое кэширование (рис.1.20). Кэш первого уровня имеет меньший объем и более высокое быстродействие, чем кэш второго уровня. Кэш второго уровня играет роль основной памяти по отношению к кэш первого уровня.

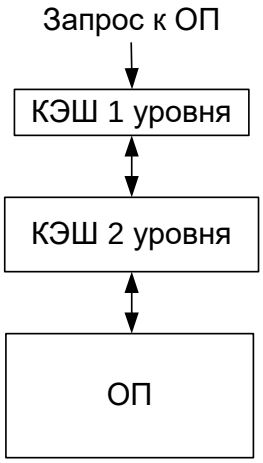


Рис.1.20 Структура памяти с двухуровневым кэш.

Алгоритм выполнения запроса на чтение в системе памяти с двухуровневым кэш следующий. Сначала делается попытка обнаружить данные в кэш первого уровня. Если произошел промах, поиск продолжается в кэш второго уровня. Если же нужные данные отсутствуют и здесь, тогда происходит считывание данных из основной памяти. Очевидно, что время доступа к данным оказывается минимальным, когда кэш-попадание происходит уже на первом уровне, несколько большим - при обнаружении данных на втором уровне. И самое большое время доступа составляет обращение к оперативной памяти, если нужных данных нет ни в том, ни в другом кэш. При считывании данных из оперативной памяти происходит их копирование в кэш второго уровня, а если данные считываются из кэш второго уровня, то они копируются в кэш первого уровня.

При работе такой иерархической организованной памяти необходимо обеспечить непротиворечивость данных на всех уровнях. КЭШ разных уровней могут согласовывать данные разными способами. Пусть, например, кэш первого уровня использует сквозную запись, а кэш второго уровня — обратную запись. (Именно такая комбинация алгоритмов согласования применена в процессоре Pentium при одном из возможных вариантов его работы.)

Если данные обнаружены в кэш первого уровня, то вступает в силу алгоритм сквозной записи: выполняется запись в кэш первого уровня и передается запрос на запись в кэш второго уровня, играющий в данном случае роль основной памяти. Т.е. в этом случае согласование данных между первым и вторым уровнями происходит автоматически. Согласование данных с оперативной памятью происходит на втором уровне кэш, при этом кэш первого уровня остается свободной для выполнения других запросов процессора. Запись в кэш второго уровня в соответствии с алгоритмом обратной записи, принятом на данном уровне, сопровождается установкой признака модификации. Хотя при этом никакой записи в оперативную память не производится, однако в подходящий момент, в соответствии с принятым алгоритмом замещения, измененные данные перемещаются в основную память. Такой способ кэширования существенно повышает производительность вычислительной системы.

Рассмотренные в данном разделе проблемы кэширования охватывают только такой класс систем организации памяти, в котором на каждом уровне имеется одно кэширующее устройство. Существует и другой класс систем памяти, главной отличительной особенностью которого является наличие нескольких кэш одного уровня. Этот вариант характерен для мультипроцессорных, многомашинных и распределенных систем обработки информации.

Одна из существенных проблем, возникающих при управлении кэш, связана с коллективным использованием основной памяти. Когда, как

показано на рис. 1.21, основная память совместно используется несколькими процессорами, каналами обмена с ВЗУ и т. д., возможно возникновение конфликтов между кэш-памятями, между кэш-памятью и основной памятью, поэтому необходимы меры, обеспечивающие когерентность кэш.

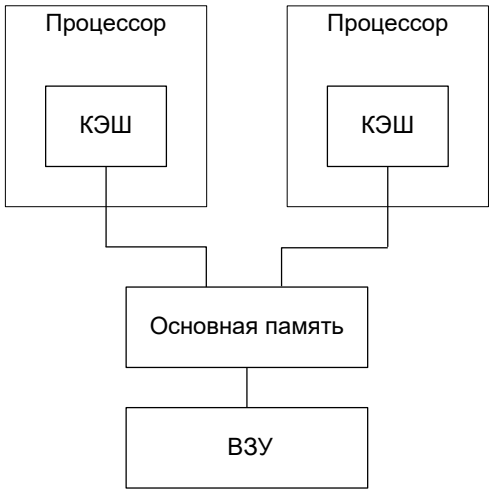


Рис.1.21. Совместное использование основной памяти

Если в мультипроцессорной системе применяется сквозная запись в память, то при записи информации в какую-либо строку её адрес сообщается всем другим процессорам (а не только тому, в кэш которого выполняются изменения данных). В этих других процессорах проверяется, нет ли данной строки в их кэш и, если есть, то в дальнейшем информация этой строки во всех кэш считается недостоверной. При использовании сквозной записи управление производится таким образом, что обновление строки происходит только в кэш одного процессора, а в остальных процессорах эта строка считается недостоверной. При обращении для считывания информации к строке, содержимое которой обновлялось в кэш другого процессора, это обновленное содержимое передается в основную память, затем строка переносится в кэш, к которому имело место обращение для считывания, после чего происходит считывание.

1.1.2. Виртуальная память

Система виртуальной адресации

Виртуальная память представляет собой единое адресное пространство, в котором физическая ограниченность емкости основной памяти скрыта от программиста. Таким образом, для программиста создается видимость произвольной адресации с отсутствием ограничений на емкость используемой памяти, что значительно облегчает программирование. Кроме того, использование виртуальной организации памяти способствует повышению взаимозаменяемости программ между вычислительными системами.

Реально существующую основную память называют физической, а ее адреса - физическими, логическую память - виртуальной, а ее адреса - виртуальными (логическими). Соответствие между физическими и виртуальными адресами устанавливается совместно аппаратными средствами ЭВМ и ее операционной системой. Обычно виртуальное адресное пространство размещается во внешней памяти, например на магнитных дисках. Часть этого пространства, необходимая для выполнения программ в данный момент, копируется в основной памяти.

Для реализации виртуальной памяти необходимо разделить все адресное пространство памяти на части и организовать соответствующий обмен между основной и внешней памятью. При этом память разбивается на страницы или сегменты.

При разбиении на страницы виртуальное и реальное адресные пространства делятся на части фиксированной длины, называемые

страницами.

Рис.1.3. Страничная организация памяти.

Адрес каждой страницы виртуального пространства ставится в соответствие адресу страницы физического адресного пространства. Взаимосвязь между адресами обоих типов устанавливается таблицей преобразования адресов (таблицей страниц), пример которой приведен на рис.1.3.

Перенос страницы виртуального пространства в основную память называется загрузкой страницы (подкачкой страницы в оперативную память), а обратное действие - удалением страницы (откачкой страницы из оперативной памяти).

Отметить достоинства и недостатки!

Сегментация

Сегментацией называется разделение адресного пространства памяти на части (сегменты) по логическим признакам, устанавливаемым программистом. Обычно величина сегмента соответствует объему

программы или подпрограммы и в отличие от страницы имеет переменную длину.

Виртуальный адрес состоит в этом случае из номера сегмента и относительного адреса в пределах сегмента; он преобразуется в физический адрес по таблице сегментов (рис.1.4). Поскольку в каждом сегменте адресное пространство является линейным, виртуальное адресное пространство в целом оказывается двумерным. С учетом того, что сегмент является логической единицей, можно организовывать защиту информации и управление для коллективного использования сегментированной информации.

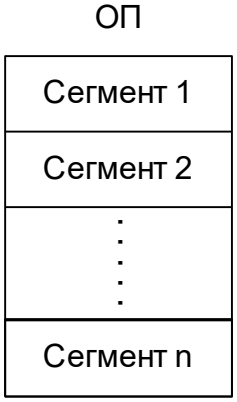


Рис.1.4.Сегментная организация памяти.

Сегментно- страничная организация памяти

Расширение виртуального пространства влечет за собой увеличение таблицы страниц. Одним из способов устранения этого неудобства служит многоуровневое разбиение на страницы. Суть этого разбиения состоит в том, что одномерное виртуальное пространство подразделяется на два уровня - сегментов и страниц, а преобразование виртуального адреса производится по двухуровневой таблице. Это дает возможность обойтись без ведения таблицы неиспользуемых страниц и, следовательно, экономит объем памяти, выделяемый для таблицы страниц. Сегмент в этом случае не является полностью двумерным пространством, но при необходимости в процессе использования его можно сделать двумерным.

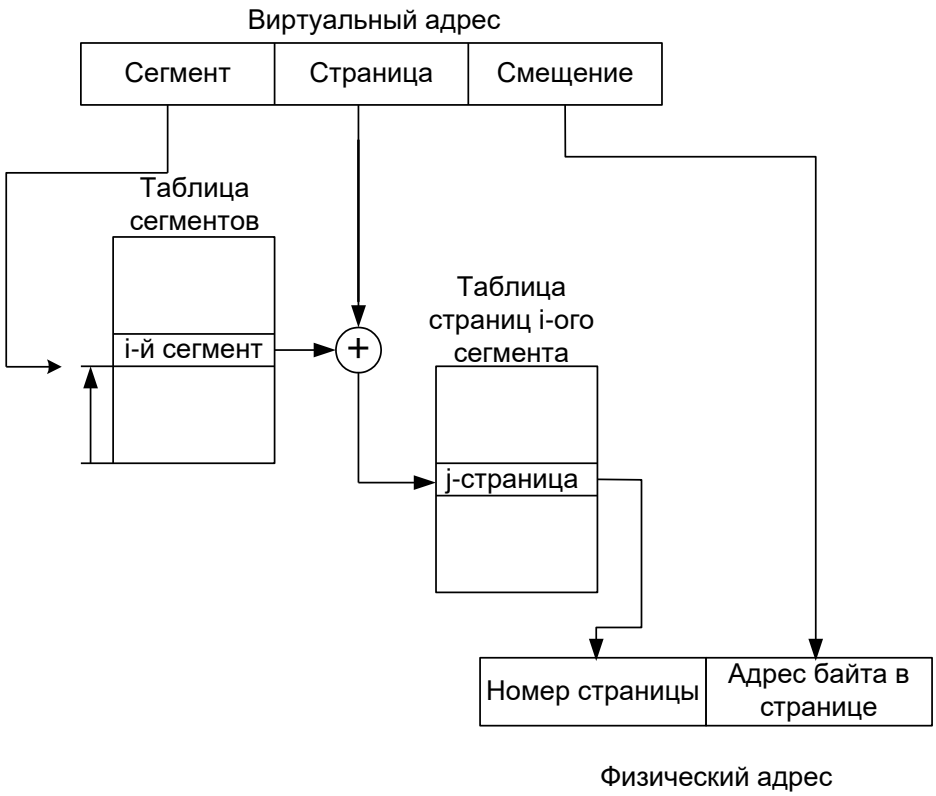


Рис.1.5.Сегментно-страничная организация памяти.

В большинстве вычислительных систем мультипрограммирование ориентировано на параллельную обработку нескольких задач и реализуется посредством системы виртуальной памяти следующими двумя методами. Первый метод основан на разделении виртуального пространства между несколькими задачами. По второму методу для каждой задачи создается отдельное виртуальное пространство адресов. Такая память называется мультиплексной виртуальной памятью.

Для управления мультиплексным виртуальным адресным пространством организуется несколько таблиц преобразования адресов, которые переключаются при переходе от задачи к задаче. Для этого в один из регистров устройства управления процессора вводится указатель, по которому выбирается соответствующая таблица преобразования адресов.

Преимущество этой системы заключается в том, что пространство памяти, используемое каждой задачей, полностью заполняет рамки виртуального пространства. Одновременно обеспечивается высокоэффективная защита памяти, так как никакая задача не может сформировать адрес, относящийся к другой задаче.

Однако время обращения к таблице при преобразовании виртуального адреса в реальный является относительно большим.

Для ускорения этой процедуры на основе использования аппаратных средств разработан так называемый механизм динамического преобразования адресов (Рис.1.6).

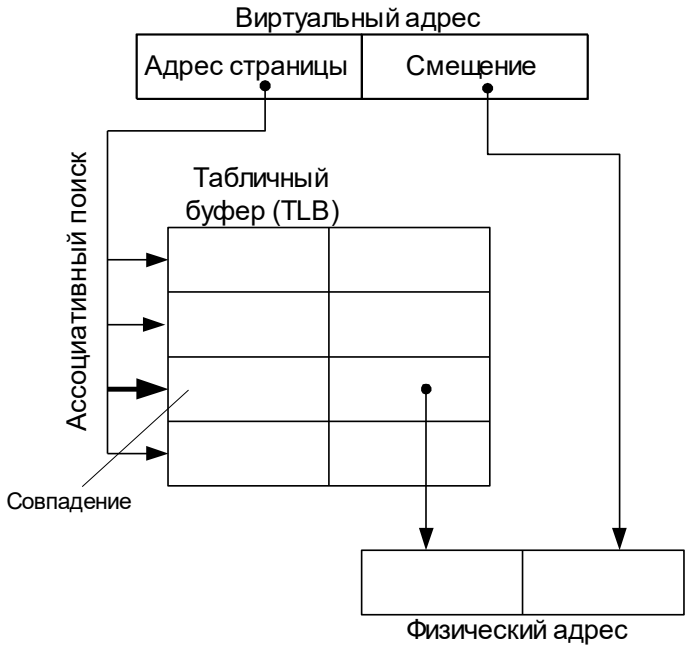


Рис.1.6. Механизм динамического преобразования адресов.

Смысл этого механизма состоит в том, что в ассоциативной памяти заранее записываются номера наиболее часто используемых в данное время страниц номера блоков, соответствующих этим страницам в основной памяти; в ходе преобразования адресов прежде всего проверяется эта ассоциативная память, и если в ней обнаруживаются сведения о

необходимых страницах, эти сведения могут быть сразу же использованы, т. е. сокращается длительность процедуры преобразования адресов.

Подобный буфер для высокоскоростного преобразования адресов называется буфером динамической трансляции адресов виртуальной памяти.

Процедура замены страниц

Когда требуемая страница в основной памяти отсутствует, она переписывается в нее из внешней памяти. Если же в основной памяти не оказывается свободного блока для загрузки страницы, то необходимо удалить какую-либо из страниц, находящихся в ней. Связанные с этим действия называются заменой страниц.

Известны следующие стратегии замены страниц:

1) стратегия FIFO, в соответствии с которой из основной памяти удаляются страницы, раньше других занесенные в нее.

2) стратегия LRU, при использовании которой удаляется та страница, обращение к которой имело место раньше, чем к другим.

3) стратегия WS (Working Set - рабочее множество), в соответствии с которой удаляются страницы, не содержащиеся в так называемом рабочем множестве, т. е. наборе страниц, к которым за определенный истекший интервал времени зафиксировано обращение.

Две из этих стратегий замены страниц - LRU и WS - основаны на предположении, что страницы, использовавшиеся в последний период, будут часто использоваться и впредь. По сравнению с ними реализация стратегии FIFO проще, но эффективность ее относительно ниже. На практике обычно используются стратегии LRU и WS, а также их сочетание и модификации.

Управление распределением основной памяти

Одна из проблем параллельной обработки состоит в том, каким образом распределить блоки основной памяти между всеми программами,

участвующими в этой обработке, В общем случае чем больше объем распределяемой основной памяти, тем реже приходится производить ее перераспределение. Для повышения эффективности выполнения каждой из параллельно обрабатываемых программ необходимо обеспечить их определенным объемом основной памяти. При чрезмерном увеличении уровня мультиплексирования программ уменьшается объем памяти, отводимой каждой из них, повышается частота обмена страниц, что приводит к резкому снижению эффективности всей системы мультипрограммной обработки. Возникает так называемое дробление памяти. Для устранения этого явления управление заметной страниц осуществляется с учетом обеспечения наибольшей эффективности использования центрального процессора. Хотя реализация этого управления занимает время, ситуация в целом улучшается благодаря своевременному сокращению степени мультиплексирования.

3. Процессоры с параллельным выполнением операций

Конвейер команд

Конвейерный принцип выполнения команд относится к MISD (МКОД) архитектурам. Машинные команды (К) выполняются (рисунок 1.23) с помощью ряда последовательных элементарных действий (микроопераций): выборки команды (ВК), дешифрации (декодирования) команды (ДК), формирования адресов операндов и выборки операндов (ВО), исполнения команды (ИК) и запоминания результата (ЗР). В машине с простой структурой аппаратной части выборка следующей машинной команды производится лишь после завершения выполнения предыдущей команды (рис. 1.22а). С другой стороны, в машинах с конвейерной организацией команд, как показано на рис. 1.22б, допустимо одновременное выполнение

нескольких команд путем совмещения во времени выполнения микроопераций этих команд.

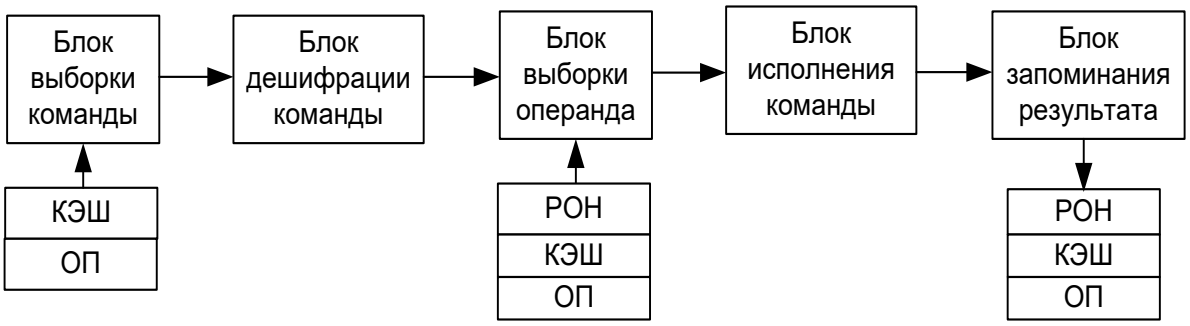


Рисунок 1.22 – Структура конвейерного процессора

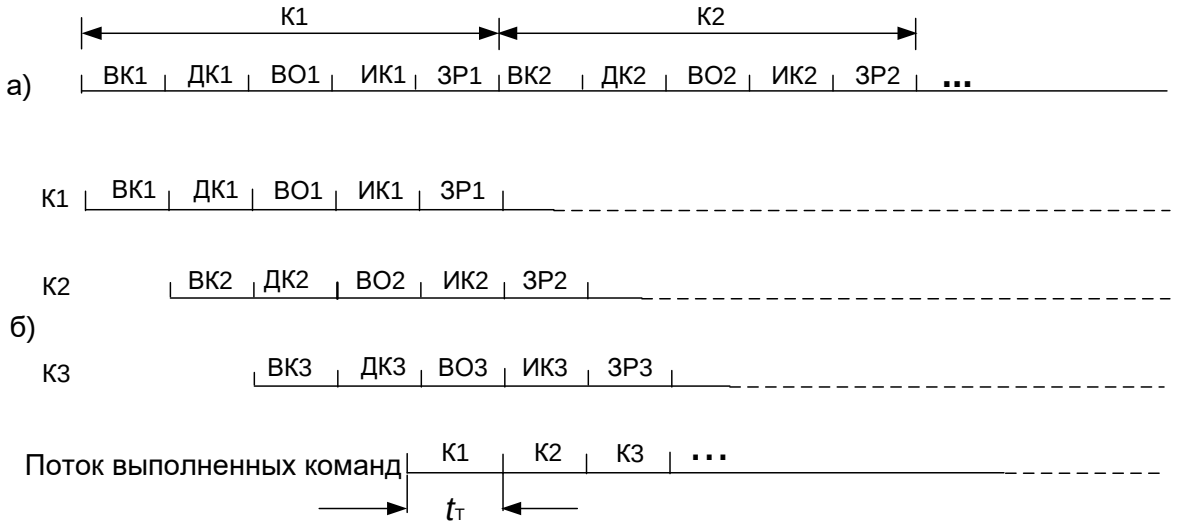


Рис. 1.23. Временная диаграмма работы конвейера микроопераций: а) процессор без конвейера; б) процессор с конвейером.

Если считать, что на выполнение каждого из этапов (шагов) команды затрачивается одинаковое время, например, равное процессорному такту ($t_Т$), то в идеальном случае можно получать результаты операций в каждом процессорном такте.

Конвейерное выполнение команд основано на тех же принципах, что и поточные линии сборки на производстве. Оно имеет максимальную

эффективность, когда продолжительность выполнения всех этапов команд одинакова, бесперебойно подаются команды и данные и на каждом этапе отсутствуют «мертвые» временные зоны, нарушающие непрерывность конвейерной реализации команд. Ситуации, которые приводят к простоям конвейера, называют конфликтами. Существуют конфликты по данным, управлению и структурные конфликты.

К факторам, нарушающим непрерывность конвейера, можно отнести такие ситуации, когда:

1) для выполнения следующей команды требуется результат от предыдущей команды, или предыдущей командой определяется адрес операнда следующей команды (модификация адреса); в этом случае возникает задержка начала выполнения следующей команды, связанная с ожиданием выборки операнда или с преобразованием адресов;

2) операция, реализуемая машинной командой, имеет сложный характер, как, например, операция с плавающей точкой и для ее выполнения требуется много машинных циклов, поэтому последующая команда долго не может достичь стадии выполнения операции;

3) в кэш отсутствуют требуемые данные или команды, поэтому необходимо еще передать их в кэш из основной памяти. При конфликтном обращении к кэш (например, при наложении друг на друга этапов ВК, ВО, ЗР команд, следующих одна за другой) запросы с относительно низкими приоритетами будут находиться в стадии ожидания;

4) при ветвлении программы по результатам проверки условий командой условного перехода команды, находящиеся в процессе конвейерной обработки, остаются невыполненными, и требуется повторная загрузка конвейера, начиная с момента выборки команды условного перехода;

5) возникает прерывание и осуществляется переход к программе его обработки. При этом в командах, находящихся в это время на командном

конвейере остаются незавершенными стадии, на которых прерывается их выполнение, и приходится заново загружать конвейер командами, входящими в программу обработки прерывания.

Конфликты по данным. Для борьбы с конфликтами по данным (1 и 2-й факторы) применяются как программные, так и аппаратные методы. Программные методы ориентированы на устранение самой возможности конфликтов еще на стадии компиляции программы. Оптимизирующий компилятор должен создавать такой объектный код, чтобы между конфликтующими командами находилось достаточное количество нейтральных команд или «холостых» команд типа NOP (команд, не выполняющих никаких действий, но создающих временную задержку).

Фактическое разрешение конфликтов возлагается на аппаратные методы. Конвейерная реализация команд вынудила большинство фирм производителей процессоров перейти на упрощенную систему команд и технологию RISC с ориентацией её на конвейерную обработку. RISC – процессоры характеризуются сравнительной простым устройством управления, унификацией набора и размера команд, а также длительности их выполнения. Все эти факторы положительно сказываются на общем быстродействии процессора и приводят к устранению периодов ожидания в конвейере.

Другим очевидным решением является остановка команды *j* на несколько тактов с тем, чтобы команда *i* успела завершиться или, по крайней мере, миновать ступень конвейера, вызвавшую конфликт. Соответственно задерживаются и команды, следующие в конвейере за *j*-й командой. Данную ситуацию называют «пузырьком» в конвейере. Иногда приостанавливают только команду *j*, не задерживая следующие за ней команды. Это более эффективный прием, но его реализация усложняет конвейер.

Остановки конвейера снижают его эффективность, и разработчики процессоров любыми способами стремятся сократить общее число остановок

или хотя бы их длительность. Поскольку наиболее частые конфликты по данным возникают в результате действия фактора 1, основные усилия тратятся на противодействие именно этому типу конфликтов. Среди известных методов борьбы наибольшее распространение получил прием *ускоренного продвижения информации*. Обычно между двумя соседними ступенями конвейера располагается буферный регистр, через который предшествующая ступень передает результат своей работы на последующую ступень, то есть передача информации возможна лишь между соседними ступенями конвейера. При ускоренном продвижении, когда для выполнения команды требуется операнд, уже вычисленный предыдущей командой, этот операнд может быть получен непосредственно из соответствующего буферного регистра, минуя все промежуточные ступени конвейера. С данной целью в конвейере предусматриваются дополнительные тракты пересылки информации (тракты обхода), снабженные средствами мультиплексирования.

Описанный конфликт обусловлен тем, что одна из команд, например МК2, ожидает записи результата выполнения команды МК1 в РОН. Однако эти данные появляются на выходе АЛУ только по завершении шага ЗР1. Задержку выполнения команды МК2 можно сократить или устранить, предав результаты выполнения команды МК1 непосредственно команде МК2. На рис.1.24 показан тракт обхода, где выбор операнда для следующей операции производится не из РОН, а непосредственно с выхода блока исполнения.

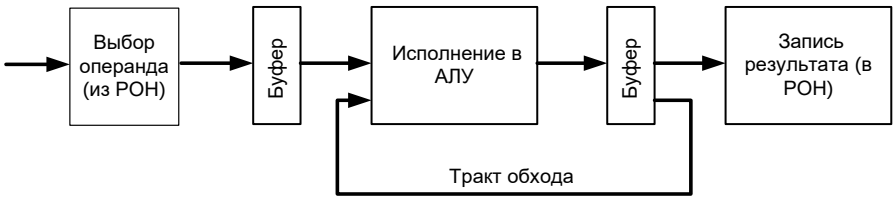


Рисунок 1.24- Ускоренное продвижение операндов в конвейере

Конфликты по управлению. Задача блока выборки команд состоит в обеспечении остальных блоков непрерывным потоком команд. Если этот поток

прерывается, конвейер останавливается. Так происходит в случае промаха при обращении к кэш или выполнении команд перехода (факторы 3,4 и 5). В первом случае блок выборки команды вынужден обратиться к оперативной памяти (если используется один уровень кэш), время доступа к которой значительно больше, чем время обращения к кэш и может занимать несколько десятков тактов. Первый путь решения этой проблемы заключается в применении кэш второго уровня, большего по размеру и с малым временем обращения. Однако из кэш второго уровня данные не могут быть получены за один такт процессора, поэтому в дополнение к этому применяют упреждающую выборку команд. Блок выборки команд извлекает команды ещё до того, как они понадобятся, и помещает их в специальную очередь, откуда они извлекаются диспетчером (дополнительное устройство в составе конвейера). Кроме того, если процессор содержит единственный кэш и для команд и для данных, то параллельно выполняющиеся микрооперации выборки команды и выборки операнда будут часто конфликтовать из-за доступа к кэш. Поэтому в большинстве процессоров используют два первичных кэш: один для хранения команд, другой для хранения операндов.

Существенные проблемы при разработке конвейера обусловлены командами, изменяющими естественный порядок вычислений (факторы 4 и 5). Простейший конвейер эффективно выполняет только линейные программы. Он на этапе (ступени) выборки извлекает команды из последовательных ячеек памяти, используя для этого счетчик команд. Адрес очередной команды в линейной программе формируется автоматически, за счет прибавления к содержимому счетчику команд шага адресации - числа, равного длине текущей команды в байтах. В реальных программах обязательно присутствуют команды передачи управления, изменяющие последовательность вычислений: безусловный и условный переходы, вызов подпрограммы, возврат из подпрограммы и т. п. Выполнение команд перехода приводит к приостановке конвейера на несколько тактов, из-за чего производительность процессора снижается.

Приостановка конвейера связана с выборкой команды из точки перехода (по адресу, указанному в команде перехода). То, что текущая команда относится к командам перехода, становится ясным только после прохождения ступени декодирования (для команд безусловного перехода), или после исполнения команды (для команд условного перехода) то есть спустя 2 такта от момента поступления команды на конвейер или даже выше для длинных конвейеров (с большим числом ступеней). За это время на первые ступени конвейера уже поступят новые команды, извлеченные в предположении, что естественный порядок вычислений не будет нарушен. В случае перехода эти ступени нужно очистить и загрузить в конвейер команду, расположенную по адресу перехода, для чего нужен исполнительный адрес последней. Поскольку в командах перехода обычно указаны лишь способ адресации и адресный код, исполнительный адрес предварительно должен быть вычислен, что и делается па третьей ступени конвейера. Таким образом, реализация перехода в конвейере требует определенных дополнительных операций, выполнение которых равносильно остановке конвейера как минимум на два такта.

Вторая причина нарушения непрерывности работы конвейера имеет отношение только к командам условного перехода. Команды условного перехода в типичной программе могут появляться через каждые 5-10 команд линейного участка и занимают до 20% программного кода. Накладные расходы, связанные с командами перехода, при таком количестве приводят к существенному снижению производительности процессора. Отрицательное влияние команд перехода на производительность можно существенно уменьшить за счет применения специальных методов их обработки.

Буфер предвыборки команд

Конфликты по данным и управлению приводят к приостановке конвейера на один и более тактов. Для устранения этого явления в процессор включают сложные блоки выборки, которые извлекают команды раньше, чем они понадобятся, и помещают их в аппаратно организованную очередь,

называемую буфером предвыборки команд. Чтобы извлекать команды из буфера, необходимо блок дешифрации снабдить устройством, называемым *блоком диспетчеризации,* который извлекает команды, расположенные в начале очереди, и обеспечивает их дешифрацию. Схема конвейера для такой обработки команд приведена на рис. 1.25.

Блок выборки в этом случае должен постоянно формировать основной буфер команд, чтобы уменьшить влияние на работу конвейера различного рода задержек при выборке команд. Кроме того, в блок выборки должен включаются средства распознавания команд перехода. Если останов конвейера вызван конфликтом по данным, блок диспетчеризации должен приостановить выборку команд и их передачу из буфера далее по конвейеру. Блок же выборки может продолжать извлекать команды из памяти и помещать их в буфер. Когда возникает задержка из-за промаха к кэш или выполнения перехода, блок диспетчеризации может продолжать извлекать команды из буфера и передавать их следующим блокам для выполнения.

Когда происходит кэш-промах, блок диспетчеризации продолжает выборку команд из буфера и направляет их на выполнение, пока очередь не опустеет. Тем временем данные считываются из основной памяти или кэш-памяти второго уровня. Если очередь не опустеет до получения новых данных, промах, возникший при обращении к кэш, не отразится на скорости выполнения команд. Очевидно, что эффективность такой технологии конвейерной обработки зависит от величины буфера. Чем больше команд будет находиться в буфере (т.е. длиннее очередь), тем меньше будет величина задержек конвейера.

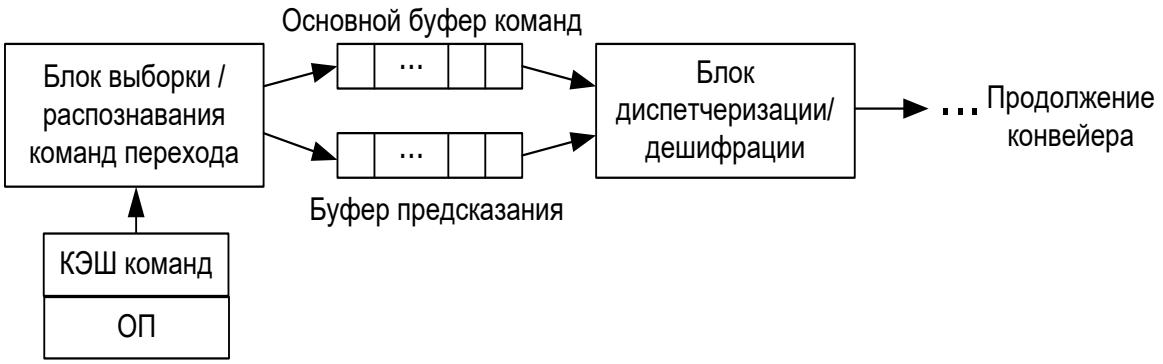


Рисунок 1.25- Включение буферов команд в конвейер

Для обеспечения равномерной работы конвейера при выполнении команд условных переходов в него включают дополнительный буфер. Команды, извлеченные из памяти, анализируется блоком распознавания перехода. При обнаружении команды условного перехода блок распознавания формирует адрес перехода и организует выборку команд в дополнительный буфер предположительной последовательности команд с указанного адреса. Одновременно с этим продолжается выборка и в основной буфер. Далее блок распознавания перехода определяет условие перехода, в зависимости от которого подключает к продолжению конвейера основной или буфер предсказания. Если условие перехода соблюдается, то содержимое основного буфера затем сбрасывается и наоборот.

Предсказание переходов

Предсказание переходов рассматривается как один из наиболее эффективных способов борьбы с конфликтами по управлению. Идея заключается в том, что еще до момента выполнения команды условного перехода делается предположение о том, что с большой вероятностью переход произойдет. Это предположение основано на том, что программы носят циклический характер. После выполнения *i*–й итерации цикла наиболее вероятно будет выполнена (*i*+1) итерация. Если цикл содержит *n* повторений, то вероятность того, что переход к началу цикла состоится, равна (1-1/*n*) и последующие команды будут подаваться

на конвейер в соответствии с предсказанием. При ошибочном предсказании конвейер необходимо очистить начальные ступени конвейера, и приступить к загрузке, начиная с точки перехода, что по эффекту эквивалентно приостановке конвейера. При высокой точности предсказания конвейер функционирует ритмично без остановок и задержек. Исследования показывают, что точность предсказания переходов может составлять более 90%.

Существуют статические и динамические методы предсказания переходов. В современных процессорах применяют те и другие методы или их сочетания. Статические методы предсказания ветвлений являются наиболее простыми. Суть этих методов состоит в том, что различные типы переходов либо выполняются всегда, либо не выполняются никогда. В современных процессорах статические методы используются лишь в том случае, когда невозможно использование динамического предсказания. Статическое предсказание, как правило, не обеспечивает непрерывность работы конвейера при выполнении любой команды перехода. Статическое предсказание часто основывают на результатах компиляции программы.

Рассмотрим выполнение команд безусловного перехода. Из рисунка 1.23 видно, что декодирование команды заканчивается на второй стадии конвейера. Следовательно, блоку выборки команд приходится решать, откуда вызывать следующую команду еще до того, как он узнает, какого типа он только что вызвал. Только в следующем такте он сможет узнать, что получил команду безусловного перехода, а до этого момента времени он уже начал вызывать команды, следующие за безусловным переходом. Эти команду либо надо выполнить и сохранить результат, либо выгрузить, что осуществить достаточно сложно, тем более что 1-2 такта работы конвейера все равно теряются. Кроме того, если загруженным в конвейер командам дается возможность выполниться, то они могут изменить состояние регистров процессора, что приведет к неправильному результату выполнения программы. В процессорах Pentium пошли другим путем. На этапе компиляции производится анализ команд перехода. При их наличии,

компилятор добавляет в программу после команды перехода дополнительное необходимое число команд типа NOP («холостых» команд, не выполняющих никаких действий). После обнаружения команды перехода (на стадии дешифрации или стадии исполнения в зависимости от типа команды: безусловного или условного перехода), в конвейере будут находиться команды NOP, не выполняющие никаких действий. Состояния регистров процессора при их выполнении не претерпят изменений, поэтому выгружать конвейер не потребуется. Такая реализация несколько удлиняет программу, но она оказывается простой.

В других процессорах, например, в SPARC фирмы SUN команда перехода содержит специальный бит предсказания перехода, устанавливаемый компилятором в состояние 0 или 1. Блок выборки команды проверяет этот разряд и выполняет действия в соответствии с указанным в нем предсказанием.

В статических методах особенно плохо дело обстоит с условными переходами. Здесь блок выборки команд узнает, откуда нужно считывать команду, не на этапе декодирования, гораздо позже. Поэтому большинство процессоров прогнозируют, состоится ли условный переход, или нет. Для этого, например, можно предполагать, что все условные переходы назад будут осуществляться, а все условные переходы вперед не будут. Что касается первого предположения, то причина такого выбора кроется в том, что переходы назад часто помещаются в конце цикла. Большинство циклов выполняется много раз, поэтому переход к началу цикла встречается очень часто.

Динамические методы, широко используемые в современных процессорах, подразумевают анализ истории ветвлений. Механизм предсказания переходов выполняет две основные функции — предсказание адреса команды, на которую производится переход (для всех типов команд перехода), и предсказание направления ветвления (для команд условного перехода). Оба предсказания должны быть выполнены заблаговременно —

раньше, чем начнётся декодирование и обработка команды перехода — для того, чтобы выборка новой команды (строки кэш, содержащей новую команду) была произведена без потерь лишних тактов либо с минимальными потерями.

Необходимость предсказания адреса «целевой» команды вызвана тем, что этот адрес может быть извлечён из команды перехода только на финальной стадии декодирования, т.е. с довольно большой задержкой. В современных процессорах для предсказания адреса перехода обычно используют специальную таблицу адресов переходов BTB (Branch Target Buffer). Эта таблица устроена подобно кэш и содержит адреса команд, на которые ранее производились переходы. Например, в процессоре P-III таблица BTB имеет размер 512 элементов и организована в виде 128 наборов с ассоциативностью 4. Если в адресуемой строке кэш есть команды перехода, и если эти команды отрабатывали ранее, то алгоритм предсказания может очень быстро найти адрес целевой команды в таблице BTB и начать считывание строки кэш, содержащей эту команду. Адреса целевых команд помещаются в BTB после выполнения соответствующих команд перехода. В современных процессорах размер таблицы BTB достигает 4096 элементов.

Для предсказания направления условного перехода используется другой механизм, основанный на изучении поведения переходов в программе в процессе её выполнения (сбор статистики). Этот механизм учитывает поведение команды перехода, например, «скорее всего, будет выполнен», «возможно, будет выполнен», «возможно, не будет выполнен», «скорее всего, не будет выполнен». История поведения команды условного перехода записывается в специальных управляющих битах таблицы BTB, обычно называемых битами прогнозирования перехода или «таблицами истории переходов» (Branch History Table, BHT), которые обновляется после выполнения каждого перехода.

Что касается фактора 5, вызываемого прерываниями, то к счастью они случаются не так часто как команды переходов и на производительность конвейера большого влияния не оказывают.

Суперскалярные процессоры.

В конвейерных процессорах наиболее трудоемкой является стадия исполнения операции. Объясняется это тем, что даже в RISC процессорах команды могут иметь разную длину. Например, команды с плавающей точкой содержат большее число микроопераций, чем целочисленные и, следовательно, время выполнения их значительно больше. Поскольку цикл выполнения коротких команд, как указывалось ранее, должен равняться циклу выполнения длинных команд, то это приводит к снижению производительности процессора. С целью повышения производительности конвейер может быть оборудован несколькими исполнительными блоками, чтобы на каждом из них обрабатывалось параллельно несколько команд (рисунок 1.26). Процессоры такого типа называются *суперсколярными.* Суперскалярный процессор является многофункциональным устройством, содержащим множество операционных устройств. Такую архитектуру имеют многие современные высокопроизводительные процессоры.

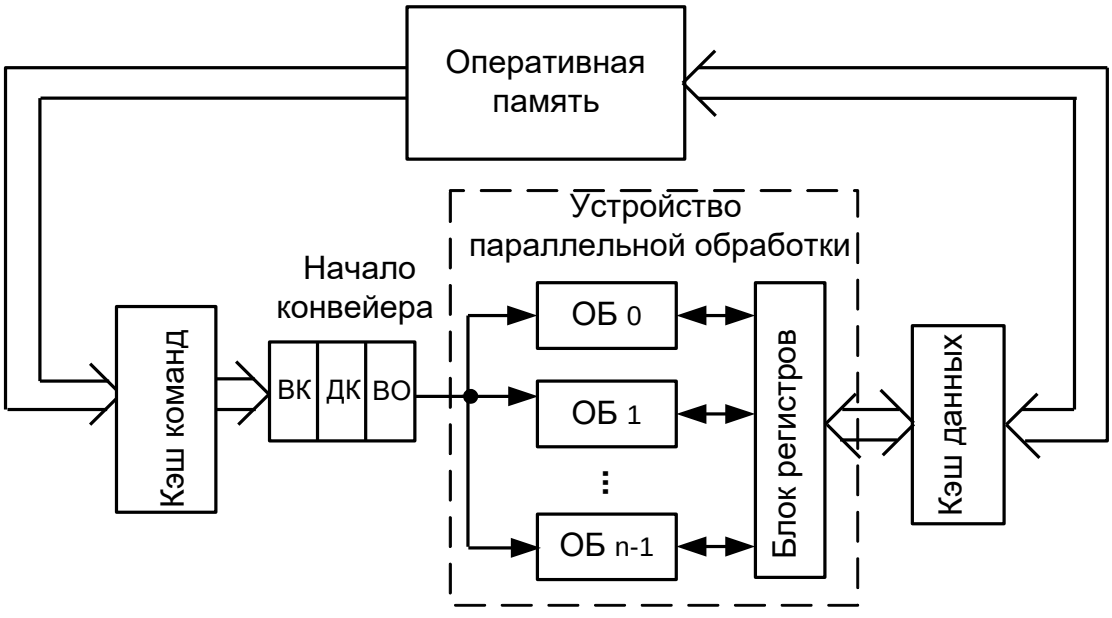


Рис.1.26. Структура суперскалярного процессора

В соответствии с используемым принципом, команды, последовательно формируемые конвейером команд, выполняются свободными операционными блоками (ОБ), причем результаты операций могут выдаваться с очередностью, отличной от очередности команд в командном конвейере. Например, в машине Эльбрус-2 десять операционных блоков (сумматор с плавающей точкой, умножитель, сумматор с фиксированной точкой и др.) параллельно выполняют соответствующие операции, совместно используя одни и те же регистры. Причём в процессе выполнения команд производится проверка занятости необходимых операционных блоков, управление поступлением требуемых данных (операндов) из других операционных блоков и памяти и передача управления операционным блокам, когда для выполнения команд готовы операнды. При невозможности выполнения команды из-за того, что данные не готовы, происходит процесс ожидания до тех пор, пока выполнение команды будет возможно. Таким образом, если длительность операции, задаваемой последующей командой, меньше и связь между ее данными и результатами выполнения предыдущей команды отсутствует, то результат этой операции окажется в регистре раньше, чем результат операции предыдущей команды. В машине Эльбрус-2 новая команда поступает в каждом машинном цикле, тогда как быстродействие операционных блоков позволяет произвести суммирование с фиксированной точкой за три машинных цикла, умножение с плавающей точкой — за 10 машинных циклов, а деление с плавающей точкой — за 29 машинных циклов. Другими словами, выполнение обработки операционным блоком отстает от темпа поступления команд, поэтому введением дополнительных операционных блоков устанавливается баланс между темпом поступления команд и скоростью выполнения операций.

Подобный же принцип управления вычислительным процессом применён в процессорах Pentium, причём в зависимости от семейства число операционных блоков может достигать пяти. Так в процессорах семейства P6 в операционной части конвейера используются два целочисленных блока,

один блок для операций с плавающей точкой, один блок MMX для обработки потока целочисленных данных (Pentium, Pentium II) и один блок SSE для обработки потока чисел с плавающей точкой (Pentium III). В Pentium IV

В суперскалярных процессорах возникает новая проблема, связанная с недостаточностью загрузки операционных блоков. Она вызвана нерегулярностью поступления различных типов команд, необходимых для одновременной загрузки всех исполнительных блоков. Чтобы обеспечить загрузку необходимо иметь возможность выбирать из памяти несколько команд за один такт, причем тип команд должен соответствовать типу операционного блока. Для реализации такого режима выборки применяют широкую шину для связи с кэш-памятью, обеспечивающую выборку двойными или четверными словами за один такт.

На рисунке 1.27 приведен пример процессора с двумя операционными блоками: для операций с фиксированной точкой (целочисленных) и операций с плавающей точкой. Блок выборки команд считывает из кэш по две команды за раз и сохраняет их в буфере. На каждом такте блок диспетчеризации извлекает из очереди и декодирует одну или две команды. Если одна из команд обрабатывает числа с фиксированной точкой, а другая — числа с плавающей точкой, то при отсутствии конфликтов обе команды запускаются на выполнение на одном такте.

Для реализации такого режима применяют оптимизирующие компиляторы, предотвращающие многие конфликты путем переупорядочивания команд до их подачи в конвейер. Например, в компьютерах на основе процессоров Pentium, компилятор по возможности обеспечивает чередование операций с плавающей точкой и операций с фиксированной точкой. Это дает возможность блоку диспетчеризации поддерживать непрерывную работу арифметических устройств с и плавающей фиксированной точкой, что, в конечном счете, обеспечивает максимальную загрузку операционных блоков и повышение производительности. Будем называть согласованным выполнение команд,

если результаты на завершающем этапе работы конвейера появляются в порядке их следования в программе.

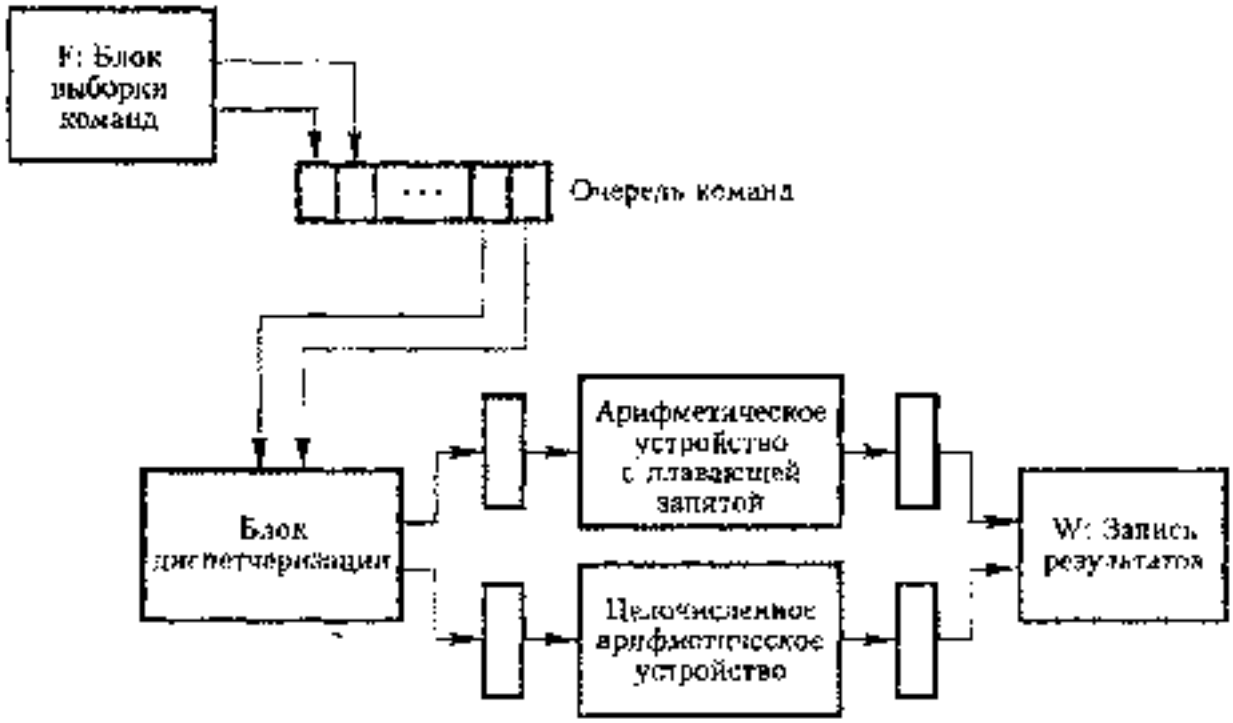


Рис. 1.27. Процессор с двумя операционными блоками.

На рис. 1.28 представлена временная диаграмма конвейера суперскалярного процессора. Пусть К1 и К3 являются командами с плавающей точкой, К2 и К4 – с фиксированной точкой. Для выполнения операции командам К1 и К3 в арифметическом устройстве с плавающей точкой требуются три такта. Устройство с фиксированной точкой выполняет команды К2 и К4 за один такт. Предполагается, что устройство с плавающей запятой организовано как трехступенчатый конвейер. Поэтому на каждом такте оно может принимать новую команду.

Если в ходе выполнения вычислительного процесса не произойдет никаких конфликтов, выполнение команд завершится так, как показано на рис. 1.28.

Рис. 1.28. Временная диаграмма не согласованной работы конвейера команд с двумя операционными блоками

Внеочередное завершение команд

В традиционных процессорах команды завершаются в том порядке, в каком они следуют в программе. В суперскалярных процессорах этот порядок может нарушаться. Короткие команды, следующие в программе после длинных, могут выполниться раньше (см. рис. 1.28). Их выполнение необходимо задержать на некоторое число тактов, чтобы соблюсти порядок выдачи результата, что приведет к необоснованным простоям оборудования и потерям производительности. Как следует из примера на рис. 1.28 выполнение команд К2 и К4 следует задержать на два такта, чтобы обеспечить заданный в программе порядок выполнения.

Однако, чтобы ускорить освобождение операционных блоков и загрузку в них новых команд, необходимо разрешить внеочередное завершение команд. Для этого команды должны выполняться так, как показано на рис. 1.28, но результаты при этом следует сохранять в теневых (временных) регистрах, чтобы позднее пересылать их в основные (постоянные) регистры процессора. Пример такого варианта выполнения программы приведен на рис. 1.29. На такте ЗВР (запись во временный регистр) осуществляется запись в теневой регистр, а на шаге ЗР, завершающем для команды, содержимое теневого регистра переписывается в соответствующий основной регистр. Этот шаг

часто называют *шагом сохранения,* поскольку после него результат выполнения команды уже не может быть отменен.

Теневой регистр некоторое время исполняет роль основного регистра, причем его имя то же, что и у основного регистра. Например, если результат выполнения команды К2 сохраняется в регистре *R*5, то временный регистр, используемый на шаге ЗВР2 интерпретируется как *R*5. Его содержимое передается любым следующим командам, обращающимся к регистру *R*5. Описанная технология называется *переименованием регистров.* Следует обратить внимание на то, что временный регистр применяется только теми командами, которые при естественном порядке выполнения программы следуют за К2. Если регистр *R*5 потребуется команде, предшествующей К2, она получит доступ к реальному регистру *R*5.

Внеочередное завершение команд требует наличия в процессоре специального управляющего блока - *блока сохранения*. Этот блок выбирает следующую команду для сохранения, используя очередь, называемую *буфером реорганизации.* Команды записываются в эту очередь в том порядке, в каком они следуют в программе, по мере диспетчеризации для выполнения. Когда команда достигает начала очереди и завершается, соответствующие результаты пересылаются из теневых регистров в основные. После этого команда удаляется из очереди, а все выделенные ей ресурсы, включая теневые регистры, очищаются. Итак, команда теперь считается покинувшей конвейер.

Описанный алгоритм позволяет выполнять команды в любой последовательности и обеспечивает их выход с конвейера в строгом соответствии с порядком расположения в программе.

Рис. 1.29. Согласованное выполнение команд в программе использованием временных регистров

Преимущества:

- Аппаратные решения обеспечивают:

- автоматическое обнаружение потенциального параллелизма между командами;

- выдачу максимально возможного числа команд, выполняемых параллельно;

- переименование регистров.

- Совместимость на уровне программ:

- если улучшения были внесены в архитектуру без изменения системы команд, то старые программы будут выполняться быстрее из-за наличия параллелизма.

Недостатки:

- Высокая сложность процессора из-за большого объема оборудования, необходимого для реализации дополнительных функций;

- Традиционные ограничения на степень параллелизма на уровне команд, характерные для конвейеров операций;
- Размер окна исполнения (число активных команд, могущих исполняться параллельно) ограничивает возможный присущий программе параллелизм, так как не рассматривается параллельное исполнение команд, находящихся на расстоянии, превышающем размер окна.

Векторные процессоры

Векторные и векторно-конвейерные вычислительные системы

Хотя производительность ВС общего назначения неуклонно возрастает, по-прежнему остаются задачи, требующие существенно большей вычислительной мощности. К таким, прежде всего, следует отнести задачи моделирования реальных процессов и объектов, для которых характерна обработка больших регулярных массивов чисел в форме с плавающей запятой. Такие массивы представляются матрицами и векторами, а алгоритмы их обработки описываются в терминах матричных операций. Как известно, основные матричные операции сводятся к однотипным действиям над парами элементов исходных матриц, которые, чаще всего, можно производить параллельно. В универсальных ВС, ориентированных на скалярные операции, обработка матриц выполняется поэлементно и последовательно. При большой размерности массивов последовательная обработка элементов матриц занимает слишком много времени, что и приводит к неэффективности универсальных ВС для рассматриваемого класса задач. Для обработки массивов требуются вычислительные средства, позволяющие с помощью единой команды производить действие сразу над всеми элементами массивов — средства *векторной обработки.*

Понятие вектора и размещение данных в памяти

В средствах векторной обработки под *вектором* понимается одномерный массив однотипных данных (обычно в форме с плавающей запятой), регулярным образом размещенных в памяти ВС. Если обработке подвергаются многомерные массивы, их также рассматривают как векторы. Такой подход допустим, если учесть, каким образом многомерные массивы хранятся в памяти ВМ. Пусть имеется массив данных *A,* представляющий собой прямоугольную матрицу размерности 4x5 (рис. 13.2).

Рис. 13.2. Прямоугольная матрица данных

При размещении матрицы в памяти все ее элементы заносятся в ячейки с последовательными адресами, причем данные могут быть записаны строка за строкой или столбец за столбцом (рис. 13.3). С учетом такого размещения многомерных массивов в памяти вполне допустимо рассматривать их как векторы и ориентировать соответствующие вычислительные средства на обработку одномерных массивов данных (векторов).

Рис. 13.3. Способы размещения в памяти матрицы 4 × 5

Действия над многомерными массивами имеют свою специфику. Например, в двумерном массиве обработка может вестись как по строкам, так и по столбцам. Это выражается в том, с каким шагом должен меняться адрес очередного выбираемого из памяти элемента. Так, если рассмотренная в примере матрица расположена в памяти построчно, адреса последовательных элементов строки различаются на единицу, а для элементов столбца шаг равен пяти. При размещении матрицы по столбцам единице будет равен шаг по столбцу, а шаг по строке - четырем. В векторной концепции для обозначения шага, с которым элементы вектора извлекаются из памяти, применяют термин *шаг по индексу* (stride).

Еще одной характеристикой вектора является число составляющих его элементов - *длина вектора.*

Понятие векторного процессора

Векторный процессор - это процессор, в котором операндами некоторых команд могут выступать упорядоченные массивы данных - векторы. Векторный процессор может быть реализован в двух вариантах. В первом он представляет собой дополнительный блок к универсальной вычислительной машине (системе). Во втором - векторный процессор - это основа самостоятельной ВС.

Рассмотрим возможные подходы к архитектуре средств векторной обработки. Наиболее распространенные из них сводятся к трем группам:

- конвейерное АЛУ;
- массив АЛУ;
- массив процессорных элементов.

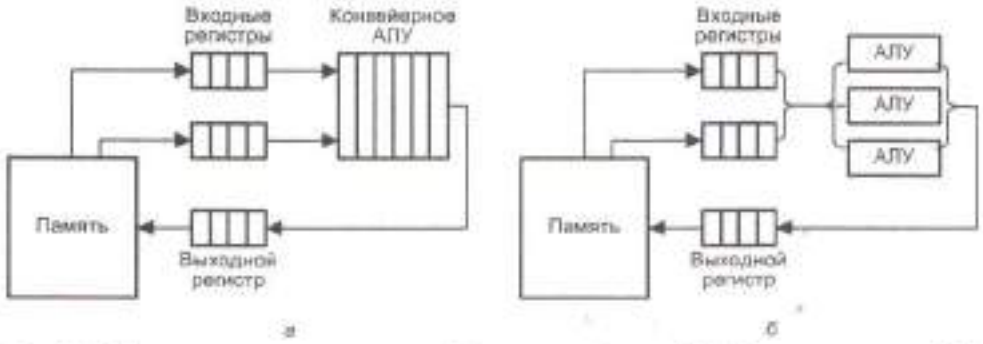


Рис. 13.4. Варианты векторных вычислений: *а* — с конвейерным АЛУ; *б* — с несколькими АЛУ

В варианте с конвейерным АЛУ (рис. 13.4, *а*) обработка элементов векторов производится конвейерным АЛУ для чисел с плавающей запятой (ПЗ). Операции с числами в форме с ПЗ достаточно сложны, но поддаются разбиению на отдельные шаги. Так, сложение двух чисел может быть сведено к четырем этапам [200]:

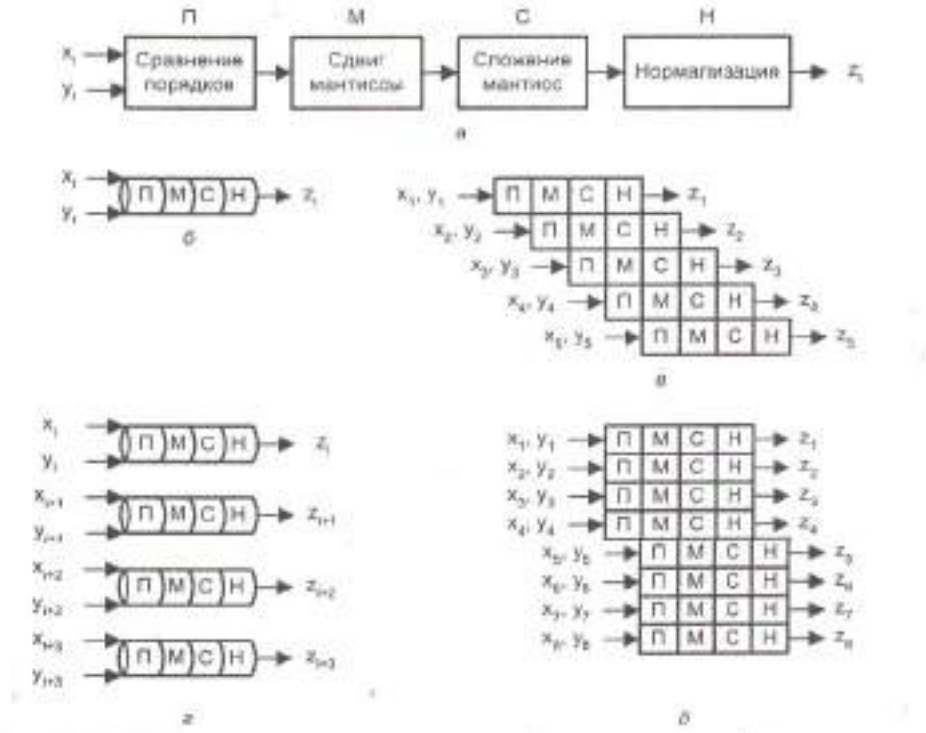


Рис. 13.5. Обработка векторов: *а* — структура арифметического конвейера для чисел с плавающей запятой; *б* — обозначение конвейера; *в* — обработка векторов в конвейерном АЛУ; *г* — параллельная обработка векторов несколькими конвейерными АЛУ; *д* — конвейерная обработка векторов четырьмя АЛУ

Последний вариант - один из случаев многопроцессорной системы, известной как *матричная ВС.* Понятие векторного процессора имеет отношение к двум первым группам, причем, как правило, к первой. Эти две категории иллюстрирует рис. 13.4 [200] сравнению порядков, сдвигу мантиссы меньшего из чисел, сложению мантисс и нормализации

результата (рис. 13.5,*а)*. Каждый этап может быть реализован с помощью отдельной ступени конвейерного АЛУ (рис. 13.5,*б)*. Очередной элемент вектора подается на вход конвейера, как только освобождается первая ступень (рис. 13.5,*в)*. Ясно, что такой вариант вполне годится для обработки векторов.

Одновременные операции над элементами векторов можно проводить и с помощью нескольких параллельно используемых АЛУ, каждое из которых отвечает за одну пару элементов (см. рис. 13.4,*б)*. Такого рода обработка, когда каждое из АЛУ является конвейерным, показана на рис. 13.5,*г*.

Если параллельно используются конвейерные АЛУ, то возможен еще один уровень конвейеризации, что иллюстрирует рис. 13.5, *д*. Вычислительные системы, где реализована эта идея, называют *векторно-конвейерными.* Коммерческие векторно-конвейерные ВС, и состав которых для обеспечения универсальности включен также скалярный процессор, известны как *суперЭВМ.*

Структура векторного процессора

Обобщенная структура векторного процессора приведена на рис. 13.6. На схеме показаны основные узлы процессора, без детализации некоторых связей между ними.

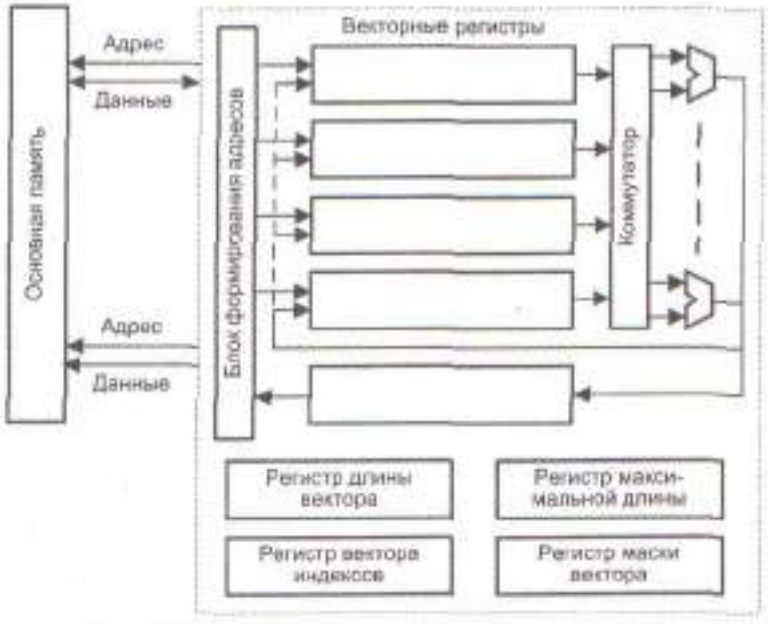


Рис. 13.6. Упрощенная структура векторного процессора

Обработка всех *n* компонентов векторов-операндов задается одной *векторной командой.* Общепринято, что элементы векторов представляются числами в форме с плавающей запятой (ПЗ). АЛУ векторного процессора может быть реализовано в виде единого конвейерного устройства, способного выполнять все предусмотренные операции над числами с ПЗ. Более распространена, однако, иная структура, - в ней АЛУ состоит из отдельных блоков сложения и умножения, а иногда и блока для вычисления обратной величины, когда операция деления *X/Y* реализуется в виде *X(1/Y)*. Каждый ил таких блоков также конвейеризирован.

Кроме того, в состав *векторной вычислительной системы* обычно включают и скалярный процессор, что позволяет параллельно выполнять векторные и скалярные команды.

Для хранения векторов-операндов вместо множества скалярных регистров используют так называемые *векторные регистры,* которые представляют собой совокупность скалярных регистров, объединенных в очередь типа FIFO, способную хранить 50-100 чисел с плавающей запятой. Набор векторных регистров (V_a, V_b, V_c,...) имеется в любом векторным процессоре. Система команд векторного процессора поддерживает работу с векторными регистрами и обязательно включает в себя команды:

- загрузки векторного регистра содержимым последовательных ячеек памяти, указанных адресом первой ячейки этой последовательности;
- выполнения операций над всеми элементами векторов, находящихся в векторных регистрах;
- сохранения содержимого векторного регистра в последовательности ячеек памяти, указанных адресом первой ячейки этой последовательности.

Примером одной из наиболее распространенных операций, возлагаемых на векторный процессор, может служить операция перемножения матриц [161]. Рассмотрим перемножение двух матриц A и B размерности 3x3.

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}.$$

Элементы матрицы результата **C** *связаны с соответствующими элементами исходных матриц* **A** и **B** операцией скалярного произведения:

$$c_{ij} = \sum_{k=1}^{3} a_{ik} \times b_{kj}.$$

Так, элемент c_{11} вычисляется как

$$c_{11} = a_{11} \times b_{11} + a_{12} \times b_{21} + a_{13} \times b_{31}.$$

Это требует трех операций умножения и (после инициализации c_{ij} нулем) трех операций сложения. Общее число умножений и сложений для рассматриваемого примера составляет 9 x 3 = 27. Если рассматривать связанные операции умножения и сложения как одну кумулятивную операцию $c + a$ x b, то для умножения двух матриц nxn необходимо n^3 операций типа «умножение-сложение». Вся процедура сводится к получению n^2 скалярных произведений, каждое из которых является итогом n операций «умножение-сложение», учитывая, что перед вычислением каждого элемента c_{ij} его необходимо обнулить. Таким образом, скалярное произведение состоит изk членов:

$$\mathbf{C} = \mathbf{A}_1\mathbf{B}_1 + \mathbf{A}_2\mathbf{B}_2 + \mathbf{A}_3\mathbf{B}_3 + \mathbf{A}_4\mathbf{B}_4 + \ldots + \mathbf{A}_k\mathbf{B}_k.$$

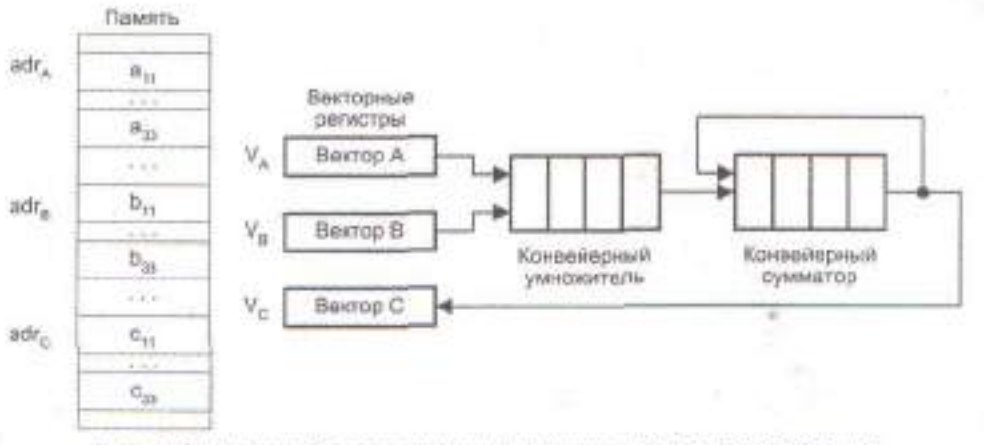


Рис. 13.7. Векторный процессор для вычисления скалярного произведения

Векторный процессор с конвейеризированными блоками обработки для вычисления скалярного произведения показан на рис. 13.7.

Векторы A и B, хранящиеся в памяти начиная с адресов adr_A и adr_B, загружаются в векторные регистры V_A и V_B соответственно. Предполагается, что конвейерные умножитель и сумматор состоят из четырех сегментов, которые вначале инициализируются нулем, поэтому в течение первых восьми циклов, пока оба конвейера не заполнятся, на выходе сумматора будет 0. Пары (A_i, B_i) подаются на вход умножителя и перемножаются в темпе одна пара за цикл. После первых четырех циклов произведения начинают суммироваться с данными, поступающими с выхода сумматора. В течение следующих четырех циклов на вход сумматора поступают суммы произведений из умножителя с нулем. К копну восьмого цикла в сегментах сумматора находятся четыре первых произведения $\mathbf{A_1B_1}$, ..., $\mathbf{A_4B_4}$, а в сегментах умножителя - следующие четыре произведения: A_5B_5,..., A_8B_8. К началу девятого цикла на выходе сумматора будет $\mathbf{A_1B_1}$, а на выходе умножителя – $\mathbf{A_5B_5}$. Таким образом, девятый цикл начнется со сложения в сумматоре $\mathbf{A_1B_1}$ и $\mathbf{A_5B_5}$. Десятый цикл начнется со сложения $\mathbf{A_2B_2}$ + A_6B_6 и т. д. Процесс суммирования в четырех секциях выглядит так:

$$
\begin{aligned}
C = {} & A_1B_1 + A_5B_5 + A_9B_9 + A_{13}B_{13} + \ldots \\
& + A_2B_2 + A_6B_6 + A_{10}B_{10} + A_{14}B_{14} + \ldots \\
& + A_3B_3 + A_7B_7 + A_{11}B_{11} + A_{15}B_{15} + \ldots \\
& + A_4B_4 + A_8B_8 + A_{12}B_{12} + A_{16}B_{16} + \ldots
\end{aligned}
$$

Когда больше не остается членов для сложения, система заносит в умножитель четыре нуля. Конвейер сумматора в своих четырех сегментах при этом будет содержать четыре скалярных произведения, соответствующие четырем суммам, приведенным в четырех строках показанного выше уравнения. Далее четыре частичных суммы складываются для получения окончательного результата.

Программа для вычисления скалярного произведения векторов **A** и **B**, хранящихся в двух областях памяти с начальными адресами adr_A и adr_B, соответственно может выглядеть так:

```
V_load V_A, adr_A
V_load V_B, adr_B
V_multiply V_C, V_A, V_B
```

Первые две векторные команды V_Load загружают векторы из памяти в векторные регистры $V_Л$ и $V_В$. Векторная команда умножения V_multiply вычисляет произведение для всех пар одноименных элементов векторов и записывает полученный вектор в векторный регистр $V_С$.

Важным элементом любого векторного процессора (ВП) является *регистр длины вектора.* Этот регистр определяет, сколько элементов фактически содержит обрабатываемый в данный момент вектор, то есть сколько индивидуальных операций с элементами нужно сделать. В некоторых ВП присутствует также *регистр максимальной длимы вектора,* определяющий максимальное число элементов вектора, которое может быть одновременно обработано аппаратурой процессора. Этот регистр используется при разделении очень длинных векторов на сегменты, длина которых соответствует максимальному числу элементов, обрабатываемых аппаратурой за один прием.

Достаточно часто приходится выполнять такие операции, в которых должны участвовать не все элементы векторов. Векторный процессор обеспечивает данный режим с помощью *регистра маски вектора.* В этом регистре каждому элементу вектора соответствует один бит. Установка бита в единицу разрешает запись соответствующего элемента вектора результата в выходной векторный регистр, а сброс в ноль - запрещает.

Как уже упоминалось, элементы векторов в памяти расположены регулярно, и при выполнении векторных операций достаточно указать значение шага по индексу. Существуют, однако, случаи, когда необходимо обрабатывать только ненулевые элементы векторов. Для поддержки подобных операций в системе команд ВП предусмотрены операции *упаковки /распаковки* (gather/scatter). Операция упаковки формирует вектор, содержащий только ненулевые элементы исходного вектора, а операция распаковки производит обратное преобразование. Обе этих задачи векторный процессор решает с помощью вектора индексов, для хранения которого используется *регистр вектора индексов,* по структуре аналогичный регистру маски. В векторе индексов каждому элементу исходного вектора

соответствует один бит. Нулевое значение бита свидетельствует, что соответствующий элемент исходного вектора равен нулю.

Использование векторных команд окупается благодаря двум качествам. Во-первых, вместо многократной выборки одних и тех же команд достаточно произвести выборку только одной векторной команды, что позволяет сократить издержки за счет устройства управления и уменьшить требования к пропускной способности памяти. Во-вторых, векторная команда обеспечивает процессор упорядоченными данными. Когда инициируется векторная команда, ВС знает, что ей нужно извлечь *n* пар операндов, расположенных в памяти регулярным образом. Таким образом, процессор может указать памяти на необходимость начать извлечение таких пар. Если используется память с чередованием адресов, эти пары могут быть получены со скоростью одной пары за цикл процессора и направлены для обработки в конвейеризированный функциональный блок. При отсутствии чередования адресов или других средств извлечения операндов с высокой скоростью преимущества обработки векторов существенно снижаются.

Структуры типа «память-память» и «регистр-регистр»

Принципиальное различие архитектур векторных процессоров проявляется в том, каким образом осуществляется доступ к операндам. При организации *«память-память»* элементы векторов поочередно извлекаются из памяти и сразу же направляются в функциональный блок. По мере обработки получающиеся элементы вектора результата сразу же заносятся в память. В архитектуре типа *«регистр-регистр»* операнды скачала загружаются в *векторные регистры,* каждый из которых может хранить сегмент вектора, например 64 элемента. Векторная операция реализуется путем извлечения операндов из векторных регистров и занесения результата в векторные регистры.

Преимущество ВП с режимом «память-память» состоит в возможности обработки длинных векторов, в то время как в процессорах типа «регистр-регистр» приходится разбивать длинные векторы на сегменты фиксированной

длины. К сожалению, за Гибкость режима «память-память» приходится расплачиваться относительно большими издержками, известными как *время запуска,* представляющее собой временной интервал между инициализацией команды и моментом, когда первый результат появится на выходе конвейера. Большое время запуска в процессорах типа «память-память» обусловлено скоростью доступа к памяти, которая намного больше скорости доступа к внутреннему регистру. Однако когда конвейер заполнен, результат формируется в каждом цикле. Модель времени работы векторного процессора имеет вид:

$$T = s + \alpha \times N,$$

где s - время запуска, α - константа, зависящая от команды (обычно 1/2,1 или 2) и N- длина вектора.

Архитектура типа «память-память» реализована в векторно-конвейерных вычислительных системах Advanced Scientific Computer фирмы Texas Instruments Inc., семействе вычислительных систем фирмы Control Data Corporation, прежде всего, Star 100, серии Cyber 200 и ВС типа ЕТЛ-10. Все эти вычислительные системы, появились в середине 70-х прошлого века после длительного цикла разработки, но к середине 80-х годов от них отказались. Причиной послужило слишком большое время запуска - порядка 100 циклов процессора. Это означает, что операции с короткими векторами выполняются очень неэффективно, и даже при длине векторов в 100 элементов процессор достигал только половины потенциальной производительности.

В вычислительных системах типа «регистр-регистр» векторы имеют сравнительно небольшую длину (в ВС семейства Cray - 64), но время запуска значительно меньше чем в случае «память-память». Этот тип векторных систем гораздо более эффективен при обработке коротких векторов, но при операциях над длинными векторами векторные регистры должны загружаться сегментами несколько раз. В настоящее время ВП типа «регистр-регистр» доминируют на компьютерном рынке. Это

вычислительные системы фирмы Cray Research Inc., в частности модели Y-MP и C-90. Аналогичный подход заложен в системы фирм Fujitsu, Hitachi и NEC. Время цикла в современных ВП варьируется от 2,5 не (NEC SX-3) до 4,2 не (Cray C90), а производительность, измеренная по тесту LINPACK, лежит в диапазоне от 1000 до 2000 MFLOPS (от 1 до 2 GFLOPS).

Обработка длинных векторов и матриц

Аппаратура векторных процессоров типа «регистр-регистр» ориентирована на об работку векторов, длина которых совпадает с длиной векторных регистров (ВР), поэтому обработка коротких векторов не вызывает проблем - достаточно записать фактическую длину вектора в регистр длины вектора.

Если размер векторов превышает емкость ВР, используется техника разбиения исходного вектора на сегменты одинаковой длины, совпадающей с емкостью векторных регистров (последний сегмент может быть короче), и последовательной обработки полученных сегментов. В английском языке этот прием называют *strip-mining*. Процесс разбиения обычно происходит на стадии компиляции, но в ряде ВП данная процедура производится по ходу вычислений с помощью аппаратных средств на основе информации, хранящейся в регистре максимальной длины вектора.

Ускорение вычислений

Для повышения скорости обработки векторов все функциональные блоки векторных процессоров строятся по конвейерной схеме, причем так, чтобы каждая ступень любого из конвейеров справлялась со своей операцией за один такт (число ступеней в разных функциональных блоках может быть различным), В некоторых векторных ВС, например Cray C90, этот подход несколько усовершенствован - конвейеры во всех функциональных блоках продублированы (рис. 13.8).

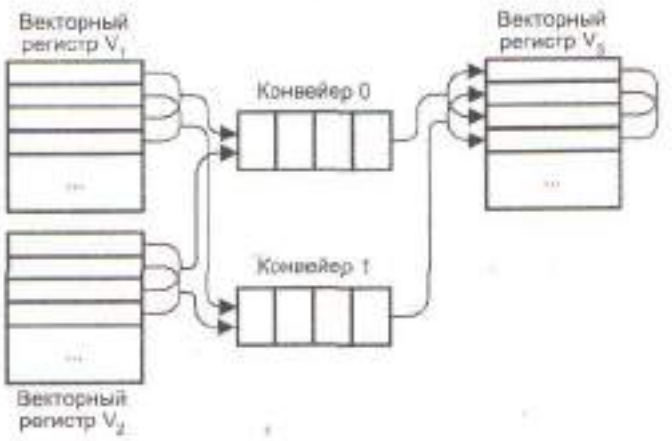


Рис. 13.8. Выполнение векторных операций при двух конвейерах

На конвейер 0 всегда подаются элементы векторов с четными номерами, а на конвейер 1 - с нечетными. В начальный момент на первую ступень конвейера 0 из ВРV$_1$ и V$_2$ поступают нулевые элементы векторов. Одновременно первые элементы векторов из этих регистров подаются на первую ступень конвейера 1. На следующем такте на конвейер 0 подаются вторые элементы из V$_1$, и V$_2$, а на конвейер 2 - третьи элементы и т. д. Аналогично происходит распределение результатов в выходном векторном регистре V$_3$. В итоге функциональный блок при максимальной загрузке в каждом такте выдает не один результат, а два. Добавим, что в скалярных операциях работает только конвейер 0.

Интересной особенностью некоторых ВП типа «регистр-регистр», например ВС фирмы Cray Research Inc., является так называемое *сцепление векторов* (vector chaining или vector linking), когда ВР результата одной векторной операции используется в качестве входного регистра для последующей векторной операции. Для примера рассмотрим последовательность из двух векторных команд, предполагая, что длина векторов равна 64: V$_2$ =V$_o$ x V$_1$, V$_4$ =V$_2$ + V$_3$.

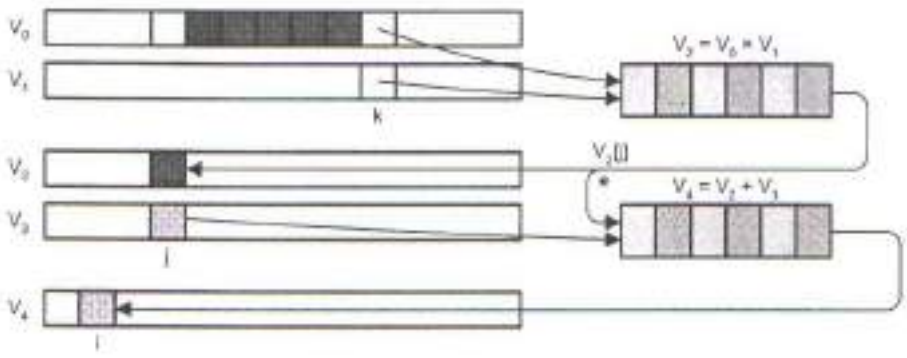


Рис. 1. 3. 9. Сцепление векторов

Результат первой команды служит операндом для второй. Напомним, что поскольку команды являются векторными, первая из них должна послать в конвейерный умножитель до 64 пар чисел. Примерно в середине выполнения команды складывается ситуация, когда несколько начальных элементов вектора $\mathbf{V_2}$ будут уже содержать недавно вычисленные произведения; часть элементов $\mathbf{V_2}$ все еще будет находиться в конвейере, а оставшиеся элементы операндов $\mathbf{V_o}$ и $\mathbf{V_1}$ еще остаются во входных векторных регистрах, ожидая загрузки в конвейер. Такая ситуация показана на рис. 13.9, где элементы векторов $\mathbf{V_o}$ и $\mathbf{V_1}$ находящиеся в конвейерном умножителе**,** имеют темную закраску. В этот момент система извлекает элементы $\mathbf{V_o[}\boldsymbol{k}\mathbf{]}$ и $\mathbf{V_1[}\boldsymbol{k}\mathbf{]}$ с тем, чтобы направить их на первую ступень конвейера, в то время как $\mathbf{V_2[}\boldsymbol{j}\mathbf{]}$ покидает конвейер. Сцепление векторов иллюстрирует линия, обозначенная звездочкой. Одновременно с занесением $\mathbf{V_2[}\boldsymbol{j}\mathbf{]}$ в ВР этот элемент направляется и в конвейерный сумматор, куда также подается и элемент $\mathbf{V_3[}\boldsymbol{j}\mathbf{].}$ Как видно из рисунка, выполнение второй команды может начаться до завершения первой, и поскольку одновременно выполняются две команды, процессор формирует два результата за цикл $\mathbf{V_4[}\boldsymbol{i}\mathbf{]}$ и $\mathbf{V_2[}\boldsymbol{j}\mathbf{]}$ вместо одного. Без сцепления векторов пиковая производительность Сгау-1 была бы 80 MFLOPS (один полный конвейер производит результат каждые 12,5 нс). При сцеплении трех конвейеров теоретический пик производительности - 240 MFLOPS. В принципе сцепление векторов можно реализовать и *в* векторных процессорах типа «память-память», но для этого необходимо

повысить пропускную способность памяти. Без сцепления необходимы три «канала»: два для входных потоков операндов и один - для потока результата. При использовании сцепления требуется обеспечить пять каналов: три входных и два ВЫХОДНЫХ.

Завершая обсуждение векторных и векторно-конвейерных ВС, следует отметить, что с середины 90-х годов прошлого века этот вид ВС стал уступать свои позиции другим более технологичным видам систем. Тем не менее одна из последних разработок корпорации NEC (2002 год) - вычислительная система «Модель Земли» (The Earth Simulator), - являющаяся на сегодняшний момент самой производительной вычислительной системой в классе, по сути представляет собой векторно- конвейерную ВС. Система включает в себя 640 вычислительных узлов по 8 векторных процессоров в каждом. Пиковая производительность суперкомпьютера превышает 40 TFLOPS.

Принцип SIMD используется в современных процессорах путем встраивания в их кристаллы относительно самостоятельных блоков, обеспечивающих технологии MMX (Pentium, Pentium-II) и SSE (Pentium-III и выше), которые значительно ускоряют операции обработки изображений.

Процессоры с длинным командным словом.

Процессоры с длинным командным словом (VLIW) используют параллелизм на уровне команд. В соответствии с этой концепцией, как показано на рис.1.20, сравнительно длинная команда делится на множество полей и каждый операционный блок управляется отдельным полем.

В отличие от суперскалярных процессоров, где возможность распараллеливания операций выясняется в процессе их выполнения, в процессорах данного типа это выяснение происходит в ходе компиляции программы. Компилятор извлекает из программы команды, которые могут быть выполнены параллельно, и из них формируется одна команда.

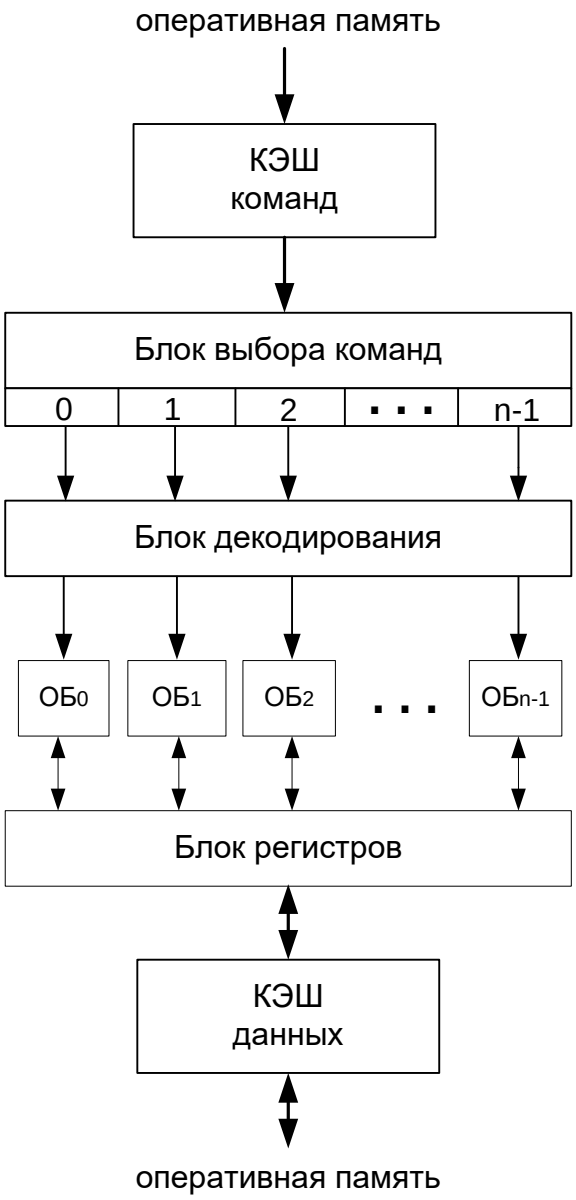


Рис.1.21. Структура VLIW- процессора.

Достоинства VLIW заключаются в следующем. Во-первых, компилятор может более эффектно выявлять зависимости между командами и выбирать параллельно исполняемые команды, чем это делает аппаратура суперскалярного процессора, ограниченная размером окна исполнения. При этом за счет обеспечения степени параллельности, близкой к числу операционных блоков, можно достичь высокой скорости обработки. Во-вторых, VLIW- процессор имеет более простое устройство управления и потенциально может иметь более высокую тактовую частоту.

Однако у VLIW процессоров есть серьезный фактор, снижающий их производительность. Это команды ветвления, зависящие от данных, значения которых становятся известны только в динамике вычислений. Число одновременно выполняющихся команд VLIW-процессора, не может быть очень большим в виду отсутствия у компилятора информации о зависимостях, формируемых динамически, в процессе выполнения.

• Отсутствует оборудование необходимое для выполнения время обнаружения параллелизма.

• Решена проблема окна исполнения, т.к. компилятор может потенциально анализировать всю программу в целях выявления параллельных операций.

Преимущества:

• Простота оборудования:

- Число операционных блоков может быть увеличено без дополнительных сложных аппаратных решений выявления параллелизма, как делается в суперскалярных процессорах.

• Хорошие компиляторы могут обнаружить параллелизм на основе глобального анализа всей программы (без окна выполнения задачи).

Проблемы:

• Чтобы поддерживать все операционные блоки в активном состоянии, требуется большое количество регистров для хранения операндов и результатов.

• Для передачи данных между КЭШем команд и блоком выборки команд, а также между операционными блоками и блоком регистров необходима большая ширина шины.

• Высокая пропускная способность между кэш инструкций и

извлечение устройства. Например, если команда содержит 7 операций, каждая из которых составляет 24 бита, то потребуется 168 бит /команду.

• Большой размер кода, отчасти потому, что неиспользованные операций ⇒ неиспользуемых битов в инструкции слова.

• Incomputability бинарного кода

Например:

Если для новой версии процессора дополнительные Фу вводятся ⇒ число операций

можно выполнять параллельно увеличивается ⇒

Слово изменения инструкции ⇒ старый бинарный код не может будет работать на этом процессоре.

Матричные процессоры

Как показано на рис. 1.21, в архитектуре SIMD (Single Instruction Multiple Data Stream — один поток команд и много потоков данных) команда, выделяемая управляющим устройством, одновременно передается множеству операционных элементов с одинаковой структурой (ПЭ), и все операционные блоки параллельно выполняют одну и ту же операцию. Управляющее устройство разрешает или запрещает выполнение операций на основе информации о состоянии каждого операционного элемента. Информация о состоянии хранится в специальном внутреннем регистре ПЭ. Выполнение операций разрешается только тем процессорным элементам, в которых выполняются определенные условия.

Поскольку поток команд является одиночным, то в случае необходимости условного перехода по результатам проверки выполнения условий заданные операции выполняются только теми элементами, для которых результаты проверки подтверждают выполнение условий, а затем только теми элементами, для которых результаты проверки говорят о том, что условия не выполняются. Следовательно, сначала запрещается

выполнение операций процессорным элементам с отрицательными результатами проверки условий, а затем — элементам, в которых проверяемые условия выполняются.

Рис. 1.22. Архитектура матричного процессора.

Управляющее устройство контролирует взаимосвязь между всеми операционными блоками и управляет обращением к общей памяти. (Пример ILLIAC-IV)

Архитектура IA64

Эта архитектура, известная под названием Intel Architecture-64 (IA-64), полностью "порывает с прошлым". IA-64 не является как 64-разрядным расширением 32-разрядной архитектуры x86 компании Intel, так и переработкой 64-разрядной архитектуры PA-RISC компании HP. IA-64

представляет собой нечто абсолютно новое - передовую архитектуру, использующую длинные слова команд (long instruction words -- LIW), предикаты команд (instruction predication), устранение ветвлений (branch elimination), предварительную загрузку данных (speculative loading) и другие ухищрения для того, чтобы "извлечь больше параллелизма" из кода программ.

Несмотря на то, что Intel и HP обещали добиться обратной совместимости с существующим программным обеспечением, работающим на процессорах архитектур x86 и PA-RISC, они до сих пор не разглашают, каким образом это будет сделано. На самом деле обеспечить такую совместимость совсем не просто; достаточно вспомнить гораздо менее кардинальный переход с 16-разрядной на 32-разрядную архитектуру x86, продолжавшийся 12 лет и до сих пор не завершённый.

По поводу совместимости, стоит заметить, что но в Merced на самом деле существует два режима декодирования команд VLIW и старый CISC. Т.е. программы переключаются в необходимый режим исполнения. В архитектуре x86 были добавлен ряд команд для перехода в новый режим, а также для передачи данных. В IA-64 такие команды есть изначально. Так что теперь ОС будут содержать и 64-х разрядную часть на IA-64 и старую 32-х разрядную.

Правда, переход к архитектуре IA-64 в ближайшее время вряд ли затронет большинство пользователей, поскольку Intel заявила, что Merced разрабатывается для серверов и рабочих станций класса high-end, а не для компьютеров среднего уровня. Фактически, компания заявила, что IA-64 не заменит x86 в ближайшем будущем. Похоже на то, что Intel и другие поставщики продолжат разрабатывать чипы x86.

Перед тем, как углубиться в технические детали, попробуем понять, почему Intel и HP рискнули пойти на столь кардинальные перемены. Причина сводится к следующему: они считают, что как CISC, так и RISC-

архитектуры исчерпали себя.

Разработчики процессоров стремятся создавать чипы, содержащие как можно больше функциональных узлов - что позволяет обрабатывать больше команд параллельно - но одновременно приходится существенно усложнять управляющие цепи для распределения потока команд по обрабатывающим узлам. На данный момент лучшие процессоры не могут выполнять более четырёх команд одновременно, при этом управляющая логика занимает слишком много места на кристалле.

В то же время, последовательная структура кода программ и большая частота ветвлений делают задачу распределения потока команд крайне сложной. Современные процессоры содержат огромное количество управляющих элементов для того, чтобы минимизировать потери производительности, связанные с ветвлениями, и извлечь как можно больше "скрытого параллелизма" из кода программ. Они изменяют порядок команд во время исполнения программы, пытаются предсказать, куда необходимо будет перейти в результате очередного ветвления, и выполняют команды до вычисления условий ветвления. Если путь ветвления предсказан неверно, процессор должен сбросить полученные результаты, очистить конвейеры и загрузить нужные команды, что требует достаточно большого числа тактов. Таким образом, процессор, теоретически выполняющий четыре команды за такт, на деле выполняет менее двух.

Проблему ещё осложняет тот факт, что микросхемы памяти не успевают за тактовой частотой процессоров. Когда Intel разработала архитектуру x86, процессор мог извлекать данные из памяти с такой же скоростью, с какой он их обрабатывал. Сегодня процессор тратит сотни тактов на ожидание загрузки данных из памяти, даже несмотря на наличие большой и быстрой кэш-памяти.

Называют новую LIW-технологию Explicitly Parallel Instruction Computing или EPIC (Вычисления с Явной Параллельностью Инструкций,

где "явной" означает явно указанной при трансляции).

Команды в формате IA-64 упакованы по три в 128-битный пакет для быстрейшей обработки. Каждый содержит шаблон (template) длиной в несколько бит, помещаемый в него компилятором, который указывает процессору, какие из команд могут выполняться параллельно. Теперь процессору не нужно будет анализировать поток команд в процессе выполнения для выявления "скрытого параллелизма". Вместо этого наличие параллелизма определяет компилятор и помещает информацию в код программы. Каждая команда (как для целочисленных вычислений, так и для вычислений с плавающей точкой) содержит три 7-битных поля регистра общего назначения (РОН). Из этого следует, что процессоры архитектуры IA-64 содержат 128 целочисленных РОН и 128 регистров для вычислений с плавающей точкой. Все они доступны программисту и являются регистрами с произвольным доступом (programmer-visible random-access registers). По сравнению с процессорами x86, у которых всего восемь целочисленных РОН и стек глубины 8 для вычислений с плавающей точкой, IA-64 намного "шире" и, соответственно, будет намного реже простаивать из-за "нехватки регистров".

Компиляторы для IA-64 будут использовать технологию "отмеченных команд" (predication) для устранения потерь производительности из-за неправильно предсказанных переходов и необходимости пропуска участков кода после ветвлений. Когда процессор встречает "отмеченное" ветвление в процессе выполнения программы, он начинает одновременно выполнять все ветви. После того, как будет определена "истинная" ветвь, процессор сохраняет необходимые результаты и сбрасывает остальные.

Компиляторы для IA-64 будут также просматривать исходный код с целью поиска команд, использующих данные из памяти. Найдя такую команду, они будут добавлять пару команд - команду предварительной загрузки (speculative loading) и проверки загрузки (speculative check). Во время выполнения программы первая из команд загружает данные в память

до того, как они понадобятся программе. Вторая команда проверяет, успешно ли произошла загрузка, перед тем, как разрешить программе использовать эти данные. Предварительная загрузка позволяет уменьшить потери производительности из-за задержек при доступе к памяти, а также повысить параллелизм.

Из всего вышесказанного следует, что компиляторы для процессоров архитектуры IA-64 должны быть намного "умнее" и лучше знать микроархитектуру процессора, код для которого они вырабатывают. Существующие чипы, в том числе и RISC-процессоры, производят гораздо больше оптимизации на этапе выполнения программ, даже при использовании оптимизирующих компиляторов. IA-64 перекладывает практически всю работу по оптимизации потока команд на компилятор. Таким образом, программы, скомпилированные для одного поколения процессоров архитектуры IA-64, на процессорах следующего поколения без перекомпиляции могут выполняться неэффективно. Это ставит перед поставщиками нелёгкую задачу по выпуску нескольких версий исполняемых файлов для достижения максимальной производительности.

Другим не очень приятным следствием будет увеличение размеров кода, так как команды IA-64 длиннее, чем 32-битные RISC-команды (порядка 40 бит). Компиляция при этом будет занимать больше времени, поскольку IA-64, как уже было сказано, требует от компилятора гораздо больше действий. Intel и HP заявили, что уже работают совместно с поставщиками средств разработки над переработкой этих программных продуктов.

Технология "отмеченных команд" является наиболее характерным примером "дополнительной ноши", перекладываемой на компиляторы. Эта технология является центральной для устранения ветвлений и управления параллельным выполнением команд.

Обычно компилятор транслирует оператор ветвления (например, IF-THEN-ELSE) в блоки машинного кода, расположенные последовательно в

потоке. В зависимости от условий ветвления процессор выполняет один из этих блоков и перескакивает через остальные. Современные процессоры стараются предсказать результат вычисления условий ветвления и предварительно выполняют предсказанный блок. При этом в случае ошибки много тактов тратится впустую. Сами блоки зачастую весьма малы - две или три команды, - а ветвления встречаются в коде в среднем каждые шесть команд. Такая структура кода делает крайне сложным его параллельное выполнение.

Когда компилятор для IA-64 находит оператор ветвления в исходном коде, он исследует ветвление, определяя, стоит ли его "отмечать". Если такое решение принято, компилятор помечает все команды, относящиеся к одному пути ветвления, уникальным идентификатором, называемым предикатом (predicate). Например, путь, соответствующий значению условия ветвления TRUE, помечается предикатом P1, а каждая команда пути, соответствующего значению условия ветвления FALSE - предикатом P2. Система команд IA-64 определяет для каждой команды 6-битное поле для хранения этого предиката. Таким образом, одновременно могут быть использованы 64 различных предиката. После того, как команды "отмечены", компилятор определяет, какие из них могут выполняться параллельно. Это опять требует от компилятора знания архитектуры конкретного процессора, поскольку различные чипы архитектуры IA-64 могут иметь различное число и тип функциональных узлов. Кроме того, компилятор, естественно, должен учитывать зависимости в данных (две команды, одна из которых использует результат другой, не могут выполняться параллельно). Поскольку каждый путь ветвления заведомо не зависит от других, какое-то "количество параллелизма" почти всегда будет найдено.

Заметим, что не все ветвления могут быть отмечены: так, использование динамических методов вызова приводит к тому, что до этапа выполнения невозможно определить, возникнет ли исключение. В других случаях применение этой технологии может привести к тому, что будет

затрачено больше тактов, чем сэкономлено.

После этого компилятор транслирует исходный код в машинный и упаковывает команды в 128-битные пакеты. Шаблон пакета (bundle's template field) указывает не только на то, какие команды в пакете могут выполняться независимо, но и какие команды из следующего пакета могут выполняться параллельно. Команды в пакетах не обязательно должны быть расположены в том же порядке, что и в машинном коде, и могут принадлежать к различным путям ветвления. Компилятор может также помещать в один пакет зависимые и независимые команды, поскольку возможность параллельного выполнения определяется шаблоном пакета. В отличие от некоторых ранее существовавших архитектур со сверхдлинными словами команд (VLIW), IA-64 не добавляет команд "нет операции" (NOPS) для дополнения пакетов.

Во время выполнения программы IA-64 просматривает шаблоны, выбирает взаимно независимые команды и распределяет их по функциональным узлам. После этого производится распределение зависимых команд. Когда процессор обнаруживает "отмеченное" ветвление, вместо попытки предсказать значение условия ветвления и перехода к блоку, соответствующему предсказанному пути, процессор начинает параллельно выполнять блоки, соответствующие всем возможным путям ветвления. Таким образом, на машинном уровне ветвления нет.

Разумеется, в какой-то момент процессор наконец вычислит значение условия ветвления в нашем операторе IF-THEN-ELSE. Предположим, оно равно TRUE, следовательно, правильный путь отмечен предикатом P1. 6-битному полю предиката соответствует набор из 64 предикатных регистров (predicate registers) P0-P63 длиной 1 бит. Процессор записывает 1 в регистр P1 и 0 во все остальные.

К этому времени процессор, возможно, уже выполнил некоторое количество команд, соответствующих обоим возможным путям, но до сих

пор не сохранил результат. Перед тем, как сделать это, процессор проверяет соответствующий предикатный регистр. Если в нём 1 - команда верна и процессор завершает её выполнение и сохраняет результат. Если 0 - результат сбрасывается.

Технология "отмеченных команд" существенно снижает негативное влияние ветвлений на машинном уровне. В то же время, если компилятор не "отметил" ветвление, IA-64 действует практически так же, как и современные процессоры: пытается предсказать путь ветвления и т.д. Испытания показали, что описанная технология позволяет устранить более половины ветвлений в типичной программе, и, следовательно, уменьшить более чем в два раза число возможных ошибок в предсказаниях.

Другой ключевой особенностью IA-64 является предварительная загрузка данных. Она позволяет не только загружать данные из памяти до того, как они понадобятся программе, но и генерировать исключение только в случае, если загрузка прошла неудачно. Цель предварительной загрузки - разделить собственно загрузку и использование данных, что позволяет избежать простоя процессора. Как и в технологии "отмеченных команд" здесь также сочетается оптимизация на этапе компиляции и на этапе выполнения.

Сначала компилятор просматривает код программы, определяя команды, использующие данные из памяти. Везде, где это возможно, добавляется команда предварительной загрузки на достаточно большом расстоянии перед командой, использующей данные и команда проверки загрузки непосредственно перед командой, использующей данные.

На этапе выполнения процессор сначала обнаруживает команду предварительной загрузки и, соответственно, пытается загрузить данные из памяти. Иногда попытка оказывается неудачной - например, команда, требующая данные, находится после ветвления, условия которого ещё не вычислены. "Обычный" процессор тут же генерирует исключение. IA-64

откладывает генерацию исключения до того момента, когда встретит соответствующую команду проверки загрузки. Но к этому времени условия ветвления, вызывавшего исключение, уже будут вычислены. Если команда, инициировавшая предварительную загрузку, относится к неверному пути, загрузка признается неудачной и генерируется исключение. Если же путь верен, то исключение вообще не генерируется. Таким образом, предварительная загрузка в архитектуре IA-64 работает аналогично структуре типа TRY-CATCH.

Возможность располагать команду предварительной загрузки до ветвления очень существенна, так как позволяет загружать данные задолго до момента использования (напомню, что в среднем каждая шестая команда является командой ветвления).

Потоковые процессоры

Потоковыми называют процессоры, в основе работы которых лежит принцип обработки многих данных с помощью одной команды. Согласно классификации Флинна, они принадлежат к SIMD (single instruction stream / multiple data stream) архитектуре. Технология SIMD позволяет выполнять одно и то же действие, например, вычитание и сложение, над несколькими наборами чисел одновременно. SIMD-операции для чисел двойной точности с плавающей запятой ускоряют работу ресурсоемких приложений для создания контента, трехмерного рендеринга, финансовых расчетов и научных задач. Кроме того, усовершенствованы возможности 64-разрядной технологии MMX (целочисленных SIMD-команд); эта технология распространена на 128-разрядные числа, что позволяет ускорить обработку видео, речи, шифрование, обработку изображений и фотографий. Потоковый процессор повышает общую производительность, что особенно важно при работе с 3D-графическими объектами.

Может быть отдельный потоковый процессор (Single-streaming processor — SSP) и многопотоковый процессор (Multi-Streaming Processor – MSP).

Ярким представителем потоковых процессоров является семейство процессоров Intel, начиная с Pentium III, в основе работы которых лежит технология Streaming SIMD Extensions (SSE, потоковая обработка по принципу «одна команда – много данных»). Эта

технология позволяет выполнять такие сложные и необходимые в век Internet задачи как обработка речи, кодирование и декодирование видео- и аудиоданных, разработка трехмерной графики и обработка изображений.

Представителями класса SIMD считаются матрицы процессоров: ILLIAC IV, ICL DAP, Goodyear Aerospace MPP, Connection Machine 1 и т.п. В таких системах единое управляющее устройство контролирует множество процессорных элементов. Каждый процессорный элемент получает от устройства управления в каждый фиксированный момент времени одинаковую команду и выполняет ее над своими локальными данными.

Другими представителями SIMD-класса являются векторные процессоры, в основе которых лежит векторная обработка данных. Векторная обработка увеличивает производительность процессора за счет того, что обработка целого набора данных (вектора) производится одной командой. Векторные компьютеры манипулируют массивами сходных данных подобно тому, как скалярные машины обрабатывают отдельные элементы таких массивов. В этом случае каждый элемент вектора надо рассматривать как отдельный элемент потока данных. При работе в векторном режиме векторные процессоры обрабатывают данные практически параллельно, что делает их в несколько раз более быстрыми, чем при работе в скалярном режиме. Максимальная скорость передачи данных в векторном формате может составлять 64 Гбайт/с, что на 2 порядка быстрее, чем в скалярных машинах. Примерами систем подобного типа являются, например, процессоры фирм NEC и Hitachi.

Процессор Intel Pentium 4

Основные особенности процессора Pentium 4 связаны с его микроархитектурой. *Микроархитектура процессора* определяет реализацию его внутренней структуры, принципы выполнения поступающих команд, способы размещения и обработки данных. Новая микроархитектура процессора Pentium 4, получившая название NetBurst (пакетно-сетевая), ориентирована на эффективную работу с Интернет-приложениями. Необходимо отметить, что в микроархитектуре NetBurst реализованы многие принципы, использованные в предыдущей модели Pentium III (микроархитектура P6). Характерными чертами этой микроархитектуры являются:

- гарвардская структура с разделением потоков команд и данных;
- суперскалярная архитектура, обеспечивающая одновременное выполнение нескольких команд в параллельно работающих исполнительных устройствах;
- динамическое изменение последовательности команд (выполнение команд с опережением — спекулятивное выполнение);
- конвейерное исполнение команд;
- предсказание направления ветвлений.

Практическая реализация данных принципов в структуре процессора Pentium 4 имеет ряд существенных особенностей (рис. 4).

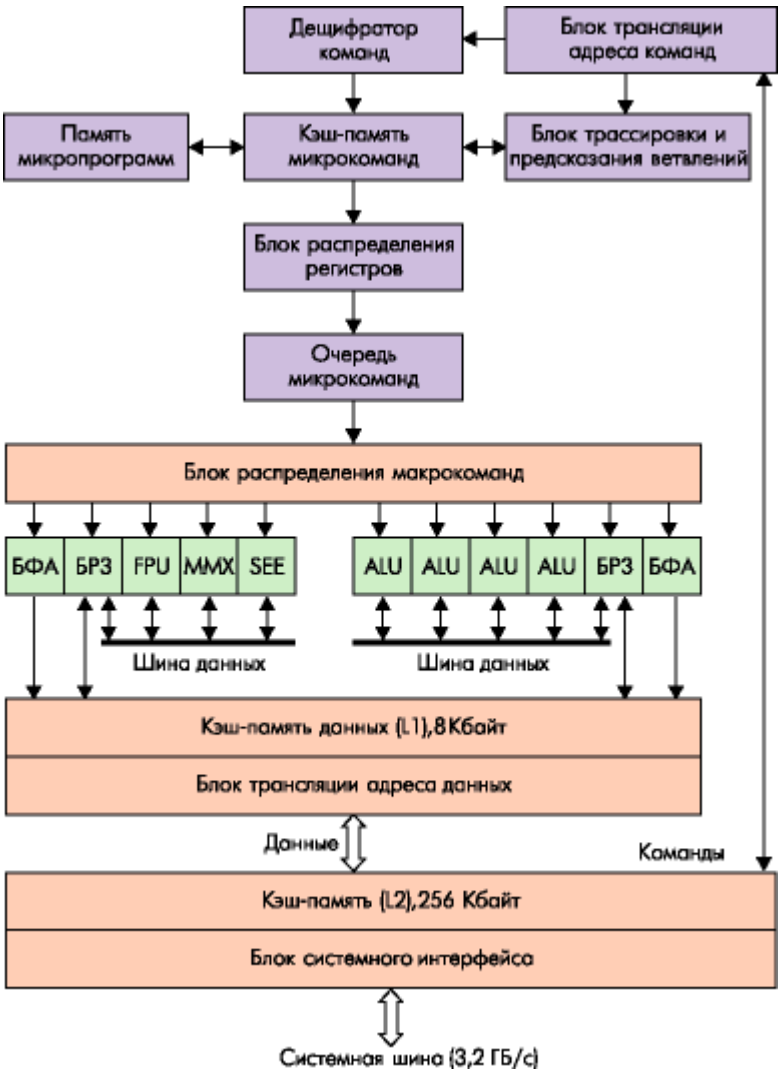


Рис. 4. Обобщенная структура Pentium 4

Гарвардская внутренняя структура реализуется путём разделения потоков команд и данных, поступающих от системной шины через блок внешнего интерфейса и размещённую на кристалле процессора общую кэш-память 2-го уровня (L2- Advanced Transfer Cache) ёмкостью 256 Кбайт. Такое размещение позволяет сократить время выборки команд. Так как Pentium 4 рассчитан на обработку потоковых данных, скорость работы L2-кеша для него является одним из ключевых моментов. Поэтому, Intel увеличил пропускную способность кеша второго уровня в Pentium 4 в два раза. Это усовершенствование было сделано благодаря передаче данных из L2-кеша в каждом процессорном такте. Пропускная способность L2-кеша Pentium 4, работающего с частотой 1.4 ГГц имеет 44.8 Мбайт/с. L2 кэш имеет восемь областей ассоциативности и строки длиной 128 байт.

Блок внешнего интерфейса реализует обмен процессора с системной шиной, к которой подключается память, контроллеры ввода/вывода и другие активные устройства системы. Обмен по системной шине осуществляется с помощью 64-разрядной двунаправленной шины данных, 41-разрядной шины адреса, обеспечивающей адресацию до 64 Гбайт внешней памяти.

Дешифратор команд работает вместе с памятью микропрограмм, формируя последовательность микрокоманд, обеспечивающих выполнение поступивших команд. Декодированные команды загружаются в кэш-память микрокоманд, откуда они выбираются для исполнения. Вместо обычного L1 кеша, который в более ранних процессорах был разделен на область команд и область данных в Pentium 4 применен новый подход. Команды в L1 кэше не сохраняются, он предназначен теперь только для данных. Для кэширования инструкций теперь используется специальный кэш трассировки (Trace Cache), однако по сравнению с обычным L1-кешем он имеет много преимуществ, направленных на минимизацию простоев процессора при выполнении неправильных предсказаний переходов. В кэш трассировки сохраняются уже декодированные инструкции, т.е. в нем хранятся не классические x86 инструкции, а так называемые микрокоманды, более простые операции которыми непосредственно оперирует процессорное ядро. Сохранение в кэш трассировки микроопераций позволяет избежать повторного декодирования x86 инструкций при повторном выполнении того же участка программы или при неправильном предсказании переходов. Итак, после заполнения кэш трассировки практически любая команда будет храниться в ней в декодированом виде. Поэтому при поступлении очередной команды блок трассировки выбирает из этой кэш-памяти необходимые микрокоманды, обеспечивающие её выполнение.

Микрооперации в кэш трассировки сохраняются именно в том порядке, в каком они выполняются. Если в потоке команд оказывается команда условного перехода (ветвления программы), то включается механизм

предсказания ветвления, который формирует адрес следующей выбираемой команды до того, как будет определено условие выполнения перехода. Вероятность того, что переходы предсказываются неправильно, достаточно мала для того, чтобы отказаться от очевидного выигрыша, получаемого путем отказа от повторных декодирований и предсказаний переходов. Кэш-память может хранить до 12000 микрокоманд. После формирования потоков микрокоманд производится выделение регистров, необходимых для выполнения декодированных команд. Эта процедура реализуется блоком распределения регистров, который выделяет для каждого указанного в команде логического регистра (регистра целочисленных операндов EAX, ECX и других, регистра операндов с плавающей точкой ST0-ST7 или регистра блоков MMX, SSE, рис. 2) один из 128 физических регистров, входящих в состав блоков регистров замещения (БРЗ). Эта процедура позволяет выполнять команды, использующие одни и те же логические регистры, одновременно или с изменением их последовательности. Выбранные микрокоманды размещаются в очереди микрокоманд. В ней содержатся микрокоманды, реализующие выполнение 126 поступивших и декодированных команд, которые затем направляются в исполнительные устройства по мере готовности операндов. Отметим, что в процессорах Pentium III в очереди находятся микрокоманды для 40 поступивших команд. Значительное увеличение числа команд, стоящих в очереди, позволяет более эффективно организовать поток их исполнения, изменяя последовательность выполнения команд и выделяя команды, которые могут выполняться параллельно. Эти функции реализует блок распределения микрокоманд. Он выбирает микрокоманды из очереди не в порядке их поступления, а по мере готовности соответствующих операндов и исполнительных устройств. В результате команды, поступившие позже, могут быть выполнены до ранее выбранных команд (внеочередное выполнение). При этом реализуется одновременное выполнение нескольких микрокоманд (команд) в параллельно работающих исполнительных устройствах. Таким образом,

естественный порядок следования команд нарушается, чтобы обеспечить более полную загрузку параллельно включенных исполнительных устройств и повысить производительность процессора.

Архитектура многопроцессорных систем

Классификация многопроцессорных систем

Многопроцессорные системы по классификации Флинна относятся к архитектурам типа MIMD(Multiple Instruction Multipl Data) с множественным потоком команд при множественном потоке данных (МКМД). В многопроцессорной системе каждый процессор выполняет свою программу достаточно независимо от других процессоров. Процессоры в ходе решения общей задачи должны связываться друг с другом в соответствии с графом взаимодействия её параллельных ветвей. Это вызывает необходимость более подробно производить классификацию систем типа MIMD.

В многопроцессорных системах с общей памятью (сильносвязанных) имеется память данных и команд, доступная всем процессорам. С общей памятью процессоры связываются с помощью коммуникационной среды, основой которой может быть либо общая шина (ОШ), либо множество шин (МШ), либо перекрёстный коммутатор (ПК).

В противоположность этому варианту в слабосвязанных многопроцессорных системах (машинах с распределённой памятью) вся память разделена между процессорами и каждый блок памяти доступен только локальному процессору. Память и процессор образуют фактически независимые вычислительные модули (вычислительные узлы), которые связываются между собой при помощи высокоскоростной сети обмена с коммутацией сообщений.

Сообщение – это блок информации, сформированный процессом-отправителем таким образом, чтобы он был понятен процессу- получателю. Сообщение состоит из заголовка фиксированной длины и набора данных определённого типа обычно переменной длины. В заголовок, как правило, включают следующую информацию:

- адрес - это поле, предназначенное для идентификации процессоров (вычислительных узлов), участвующих в процедуре обмена. Адрес процессора или вычислительного узла является уникальным и состоит из двух частей - адреса процессора - отправителя и адреса процессора - получателя;
- управляющие поля, в которые могут входить символы синхронизации, отмечающие начало и конец передаваемого блока (кадра) данных, символы, обозначающие тип данных, длину передаваемых данных и др.

В качестве топологической модели коммуникационной среды применяют линейные или кольцевые моноканалы, звездообразную конфигурацию, плоскую решётку, n- мерный тор, либо n- кубическую (гиперкубическую) сеть.

Базовой моделью вычислений на MIMD-системах является совокупность независимых процессов, время от времени обращающихся к разделяемым данным. Основана она на распределенных вычислениях, в которых программа делится на довольно большое число параллельных задач.

В настоящее время всё большее внимание разработчики проявляют к архитектурам типа MIMD. Это находит объяснение главным образом существованием двух факторов:

1) в архитектурах MIMD используются высокотехнологичные, дешёвые, выпускаемые массово микропроцессоры, что позволяет оптимизировать соотношение стоимость/производительность;

2) архитектура MIMD дает большую гибкость и при наличии соответствующей поддержки со стороны аппаратных средств и программного обеспечения, поскольку может работать и как однопрограммная система, обеспечивая высокопроизводительную обработку данных для одной прикладной задачи, и как многопрограммная, выполняющая множество задач параллельно, а также как некоторая комбинация этих возможностей.

Одной из отличительных особенностей многопроцессорной вычислительной системы является коммуникационная среда, с помощью которой процессоры соединяются друг с другом или с памятью. Топология коммуникационной среды настолько важна для многопроцессорной системы, что многие характеристики производительности и другие оценки выражаются отношением времени обработки к времени обмена, которые в общем случае зависят от алгоритмов решаемых задач и порождаемых ими вычислительных процессов.

Существуют две основные модели межпроцессорного обмена: одна основана на передаче сообщений, другая - на использовании общей памяти.

В многопроцессорных системах с общей памятью один процессор осуществляет запись в конкретную ячейку, а другой процессор производит считывание из этой ячейки памяти. Чтобы обеспечить согласованность данных и синхронизацию процессов, обмен часто реализуется по принципу взаимно исключающего доступа к общей памяти методом "почтового ящика".

В архитектурах с распределённой памятью непосредственное разделение памяти невозможно. Вместо этого процессоры получают доступ к совместно используемым данным посредством передачи сообщений по сети

обмена. Эффективность схемы коммуникаций зависит от протоколов обмена, каналов обмена и пропускной способности памяти. Такие системы часто называют системами с передачей сообщений.

Каждый из этих механизмов обмена имеет свои преимущества. Для обмена в общей памяти это включает:

- совместимость с хорошо понятными и используемыми в однопроцессорных системах механизмами взаимодействия процессора с основной памятью;
- простота программирования, особенно это заметно в тех случаях, когда процедуры обмена между процессорами сложные или динамически меняются во время выполнения. Подобные преимущества упрощают конструирование компилятора;
- более низкая задержка обмена и лучшее использование полосы пропускания при обмене малыми порциями данных;
- возможность использования аппаратно управляемого кэширования для снижения частоты удаленного обмена, допускающая кэширование всех данных как разделяемых, так и неразделяемых.

Основные преимущества обмена с помощью передачи сообщений являются:

- аппаратура может быть более простой, особенно по сравнению с моделью разделяемой памяти, которая поддерживает масштабируемую когерентность кэш-памяти;
- процедуры обмена понятны, принуждают программистов (или компиляторы) уделять внимание обмену, который обычно имеет высокую, связанную с ним, стоимость.

Часто, в системах с общей памятью затраты времени на обмен не учитываются, так как проблемы обмена в значительной степени скрыты от программиста. Однако накладные расходы на обмен в этих системах имеются и определяются в основном конфликтами при доступе процессоров и других устройств к общим шинам и блокам основной памяти. Чем больше процессоров добавляется в систему, тем больше процессов соперничают при использовании одних и тех же данных и шины, что может привести к значительным задержкам хода вычислительного процесса и, следовательно, к потерям общей производительности. Причём с увеличением числа процессоров в системе возможно появление эффекта насыщения при котором рост числа процессоров не приведёт к росту производительности, и даже наоборот к её падению. Модель системы с общей памятью очень удобна для программирования, поскольку пользователь практически не задумывается о процедурах распараллеливания (они большей частью ложаться на

компилятор) и процедурах взаимодействия параллельных процессов (они производятся аппаратными средствами посредством общей памяти).

В сетях с коммутацией сообщений по мере возрастания требований к обмену следует учитывать возможность перегрузки сети. Здесь межпроцессорный обмен связывает сетевые ресурсы: каналы, процессоры, буферы сообщений. Объем передаваемой информации может быть сокращен за счет тщательного разбиения задачи на параллельные ветви и тщательного диспетчирования процесса их исполнения.

Таким образом, существующие MIMD-системы распадаются на два основных класса в зависимости от количества объединяемых процессоров, которое определяет и способ организации памяти и методику их межсоединений.

К первому классу относятся системы с разделяемой (общей) основной памятью (Shared Memory multiProcessing, SMP), объединяющие до нескольких (2-16) процессоров, число которых зависит от типа применяемой коммуникационной среды. Сравнительно небольшое количество процессоров в таких системах позволяет иметь одну централизованную общую память и зачастую объединить процессоры и память с помощью лишь одной шины. При наличии у процессоров кэш-памяти достаточного объема высокопроизводительная шина и общая память могут удовлетворить обращения к памяти, поступающие от нескольких процессоров.

Поскольку имеется единственная память с одним и тем же временем доступа, эти системы называют симметричными, а иногда - UMA (Uniform Memory Access). Симметричная архитектура предполагает однородность процессоров и единообразную схему их включения в многопроцессорную систему. Такой способ организации со сравнительно небольшой разделяемой памятью в настоящее время является наиболее популярным. Структура подобной системы представлена на рис. 2.1.

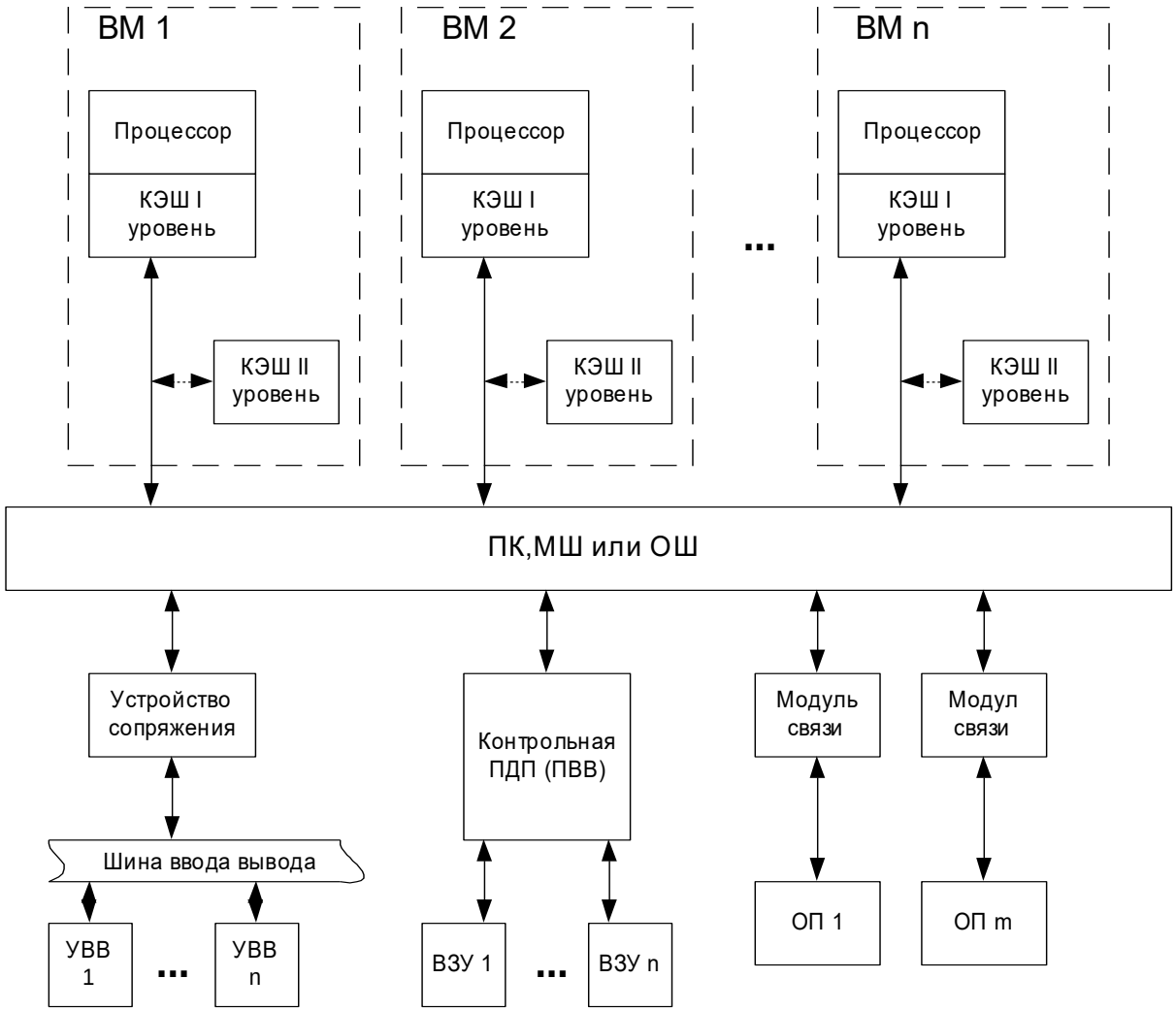


Рис. 2.1. Архитектура многопроцессорной системы с разделяемой (общей) памятью.

Второй класс составляют крупномасштабные вычислительные системы с распределенной памятью. **Подобные ВС получили название систем с массовым параллелизмом (Mass-Parallel Processing, MPP**). Для того чтобы подключать в систему большое количество процессоров необходимо физически разделять основную память и распределять её между ними. В противном случае пропускной способности памяти просто может не хватить для удовлетворения запросов, поступающих от очень большого числа процессоров. Естественно при таком подходе также требуется реализовать связь процессоров между собой. На рис. 2.2 показана структура такой системы.

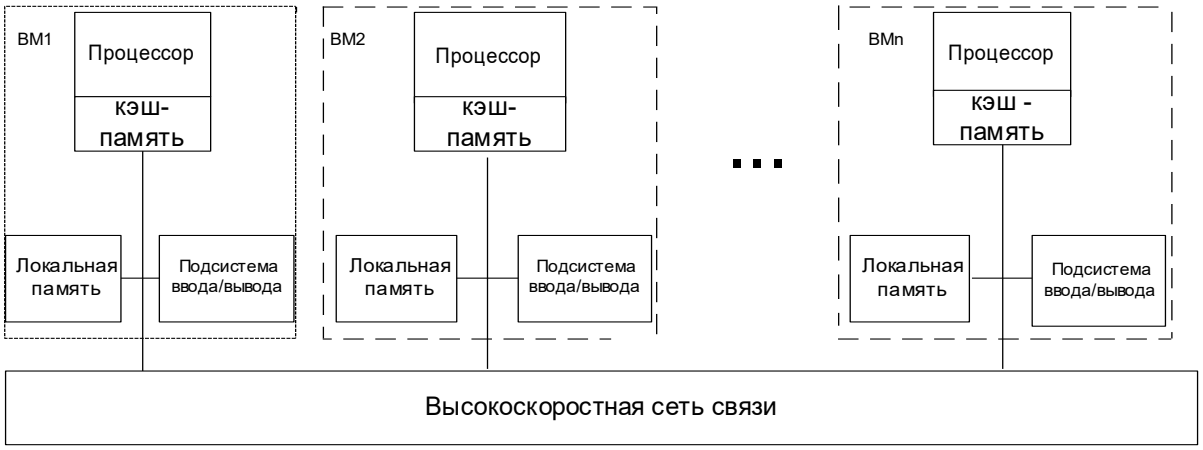


Рис. 2.2. Архитектура многопроцессорной системы с распределенной памятью.

Адресное пространство в таких системах состоит из отдельных адресных пространств, которые логически не связаны, и доступ к которым не может быть осуществлен аппаратно другим процессором. Фактически каждый модуль процессор-память представляет собой отдельный компьютер, поэтому такая структура в какой-то степени приближена к многопроцессорным системам.

MPP - система менее эффективна с точки зрения пользователя из-за усложнённой процедуры программирования, которая связана с применением специальных коммуникационных библиотек для организации взаимодействия между вычислительными узлами (процессами). Необходимость же реализации модели распределенной памяти объясняется тем, что масштабируемость (способность системы к наращиванию числа процессоров) систем с общей памятью ограничена пропускной способностью памяти и коммуникационной среды.

Вообще распределение памяти между отдельными узлами системы имеет два главных преимущества. Во-первых, это эффективный с точки зрения стоимости способ увеличения пропускной способности памяти, поскольку большинство обращений могут выполняться параллельно к локальной памяти в каждом узле. Во- вторых уменьшается задержка обращения к локальной памяти из-за отсутствия конфликтов при доступе к ней. Поэтому совершенно естественно появление промежуточного класса систем, объединяющего достоинства первого и второго классов. Память в таких системах распределена по вычислительным узлам и одновременно является доступной для всех процессоров. Такие ВС называются системами с распределенной разделяемой (общей) памятью (DSM - Distributed Shared Memory), а иногда NUMA (Non-Uniform Memory Access), поскольку время

доступа зависит от расположения данных в подсистеме памяти (Рис.2.3). Если данные находятся в локальной памяти местного вычислительного узла, то время доступа к ним минимально, если в локальной памяти удалённого вычислительного узла, то время доступа увеличивается в несколько раз.

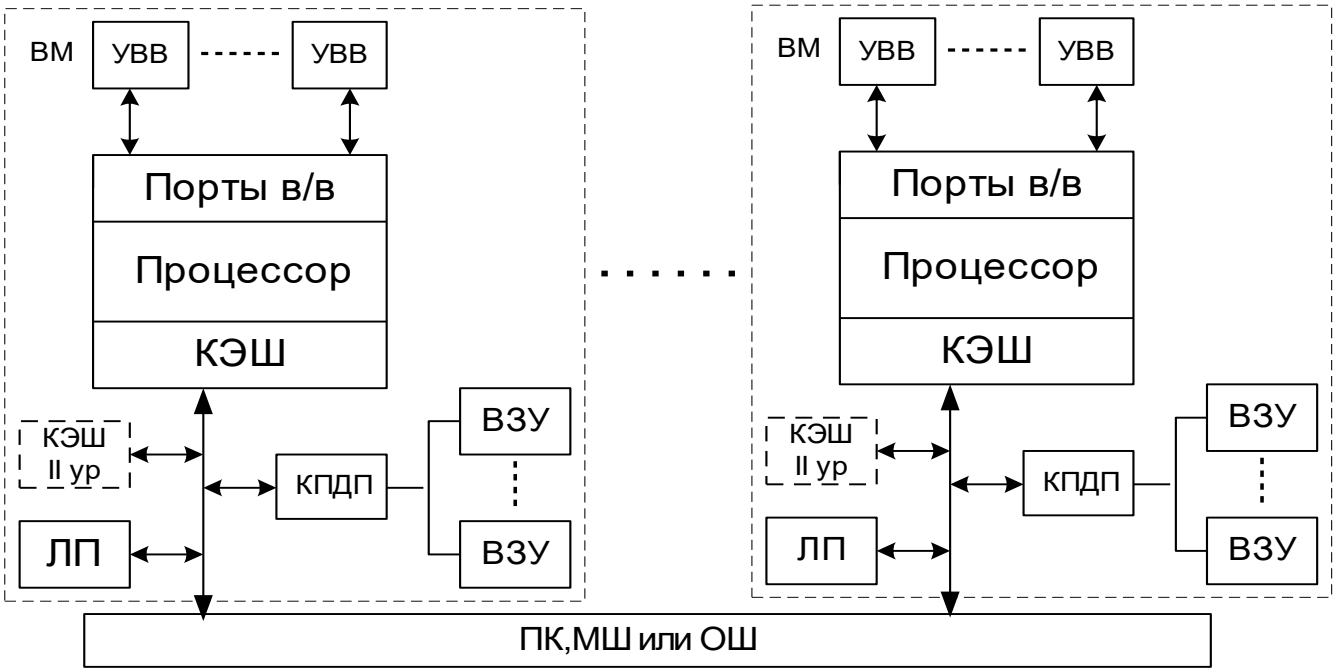


Рис.2.3. Архитектура многопроцессорной системы с распределённой разделяемой памятью.

Хотя структурно DSM и MPP-системы сходны, однако технически они реализуются по-разному. В DSM-системах физически отдельные, распределённые по вычислительным узлам, устройства памяти могут представляться логически как единое адресное пространство, что означает, что любой процессор может выполнять обращения к любым ячейкам памяти, предполагая, что он имеет соответствующие права доступа. Коммуникационная же среда и вовсе другая, так как она полностью соответствует структурам SMP-систем. Поскольку, в связи с принципами локальности, вычислительный процесс в основном развивается внутри вычислительного узла и редко обращается к удалённой памяти, что резко снижает объём передаваемых данных по коммуникационной среде, то масштабируемость таких систем повышается по сравнению с SMP-системами, и число процессоров может достигать 32-х.

Что касается устройств ввода/вывода и внешних запоминающих устройств, то они также как и память, либо распределяются по узлам, либо находятся в общем пользовании.

2.2 Организация коммуникационной среды в системах с разделяемой памятью.

Коммуникационную среду, реализующую множество соединений между процессорами или вычислительными узлами в многопроцессорных системах, называют коммутатором. По способу реализации различают коммутаторы с временной и пространственной коммутацией.

При временной коммутации передача информации осуществляется методами разделения времени или мультиплексирования. Такой коммутатор называется в простейшем случае общей шиной (ОШ). Структура многопроцессорной системы с общей шиной представлена на рис. 2.4.

В каждый момент времени ОШ способна, передавать лишь одно сообщение, т. е. она представляет собой разделенную во времени шину. Это говорит о возможности возникновения конфликтных ситуаций тогда, когда нескольким модулям одновременно необходимо связаться со своими абонентами - вычислительными узлами. С целью разрешения конфликтов

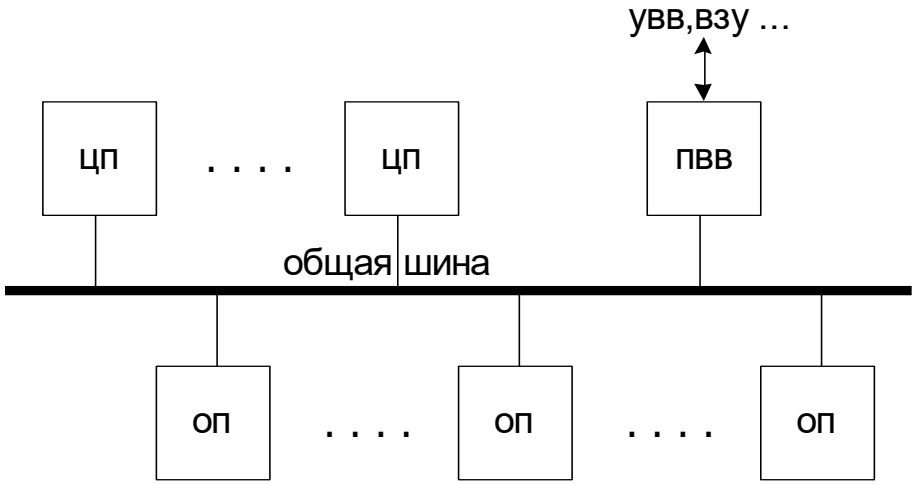


Рис.2.4. Структура системы с коммуникационной средой на основе общей шины.

Всем абонентам, могут быть назначены определенные приоритеты. Свободная общая шина удовлетворяет запросы абонента (процессора, периферийных устройств и др.) с наивысшим приоритетом.

Процедура занятия ОШ абонентами, являющимися источниками информации, заключается в их "борьбе" за ОШ. Достоинством коммутаторов с ОШ является простота организаций и гибкость (простое добавление и изъятие модулей). Очевидно, что такой коммутатор не может обеспечить

высокую пропускную способность. Для всех потоков информации здесь только один путь, поэтому временные задержки при передаче данных значительны. Из-за этого оказываются низкими общая производительность, поскольку различные пары абонентов не могут работать одновременно, и надежность системы, так как отказ в единственном пути передачи информации ведет к отказу всей системы. Пространственную коммутацию осуществляет перекрестный коммутатор (ПК). Он представляет собой многополюсник с M входами и N выходами, допускающими одновременное установление любого количества соединений между заданными входами и выходами.

Структуру системы с перекрестным коммутатором можно представить так как показано на рис. 2.5. Между любыми двумя абонентами здесь устанавливается физический контакт на все время передачи информации. При этом возможные конфликты между модулями разрешаются в логических схемах коммутационной матрицы, что существенно усложняет аппаратуру коммутаторов. В отличие от систем с ОШ, где во-первых коммутационного оборудования много меньше, а во-вторых эти функции могут быть возложены на вычислительные узлы или на центральный арбитр шины.

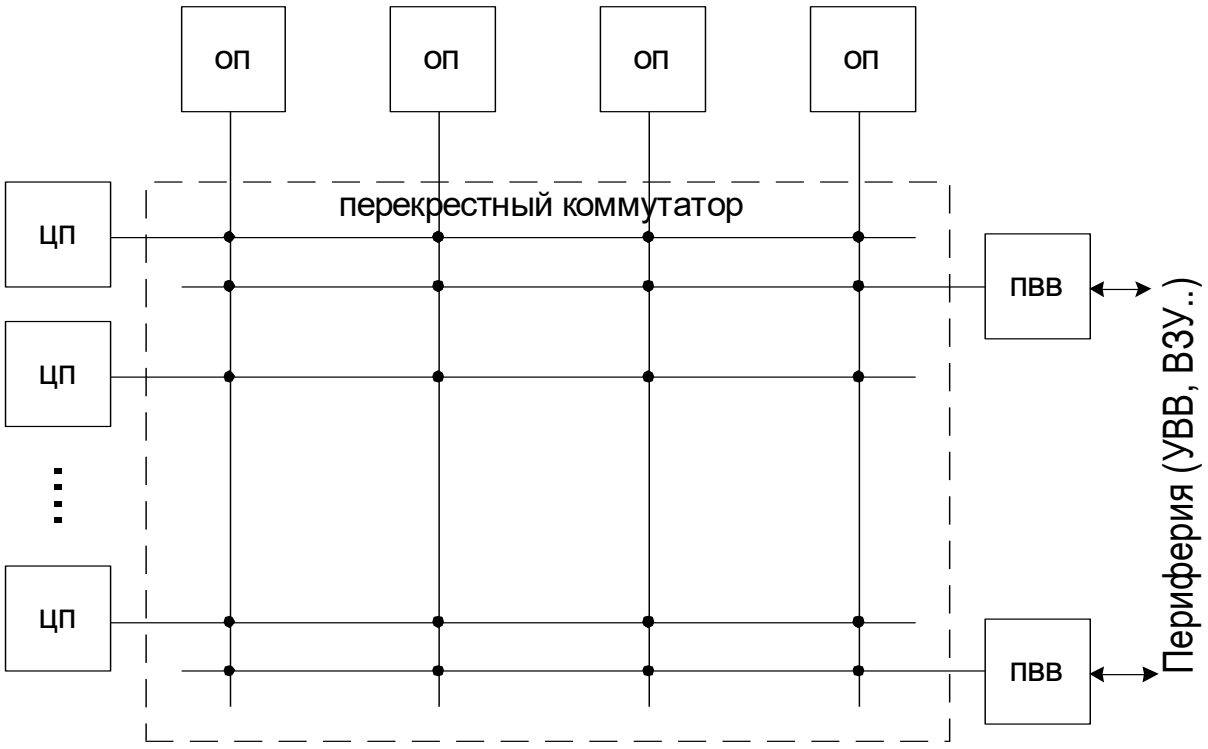


Рис.2.5. Многопроцессорная система с коммуникационной средой на основе перекрёстного коммутатора.

У перекрестного коммутатора в противоположность общей шине имеется больше достоинств, но есть и существенные недостатки, к которым относятся сложность внутренних связей, ведущая к усложнению аппаратуры

и плохая масштабируемость, ограниченная числом входов и выходов коммутатора.

Достоинствами ПК являются: возможность установления нескольких одновременных путей передачи информации; обеспечение большей производительности и надежности многопроцессорной системы.

Более экономичной схемой пространственной коммутации по сравнению с ПК является схема, использующая многовходовую память (рис. 2.6).

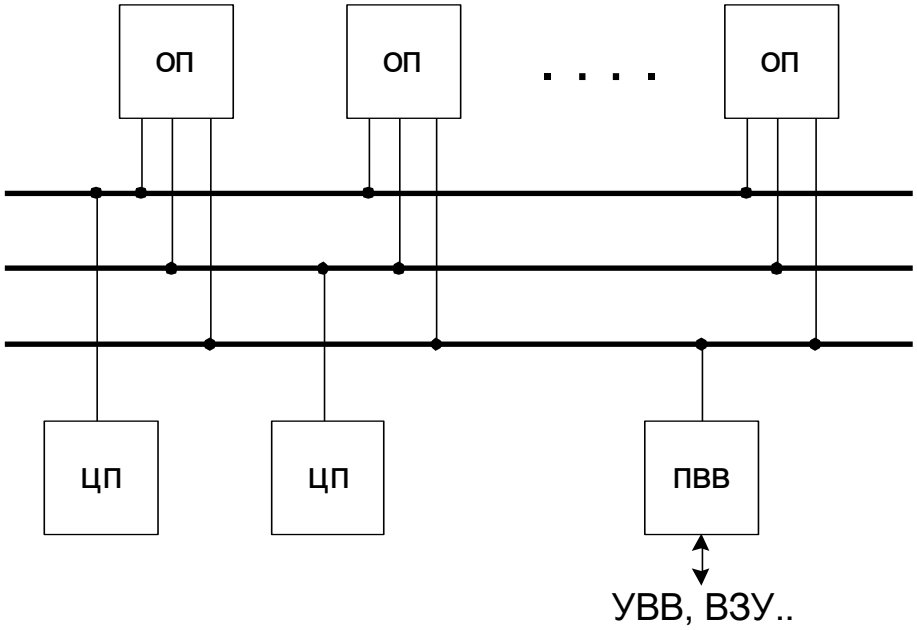


Рис.2.6. Многопроцессорная система с многошинной коммуникационной средой.

Здесь имеет место меньшее количество точек, в которых нужно разрешать конфликты. Максимально возможная конфигурация системы (количество подключаемых процессоров) ограничивается числом входов блоков памяти.

2.2.1. Организация памяти в системах с ОШ.

По способу организации памяти ММПВС с ОШ делятся на два типа: 1) с общей памятью; 2) с локальной памятью.

Структура ММПВС с общей памятью приведена на рис. 2. 13. При такой организации каждый процессор имеет доступ к любому модулю памяти и периферийному устройству. Следовательно, любому МП системы доступны любые программные модули, в том числе и программы операционной системы, которые осуществляют общее управление всеми аппаратными и программными средствами системы и обеспечивают взаимодействие между процессами, реализуемыми в микропроцессорах.

Для ММПВС с общей памятью более всего подходит для решения сильносвязанных задач, общие данные которых могут быть размещены в одном модуле памяти. При этом достигается значительная экономия памяти. Однако высокой производительности при увеличении числа процессоров здесь достичь не удается, так как за каждой командой и каждым словом каждому процессору требуется обращение в общую память.

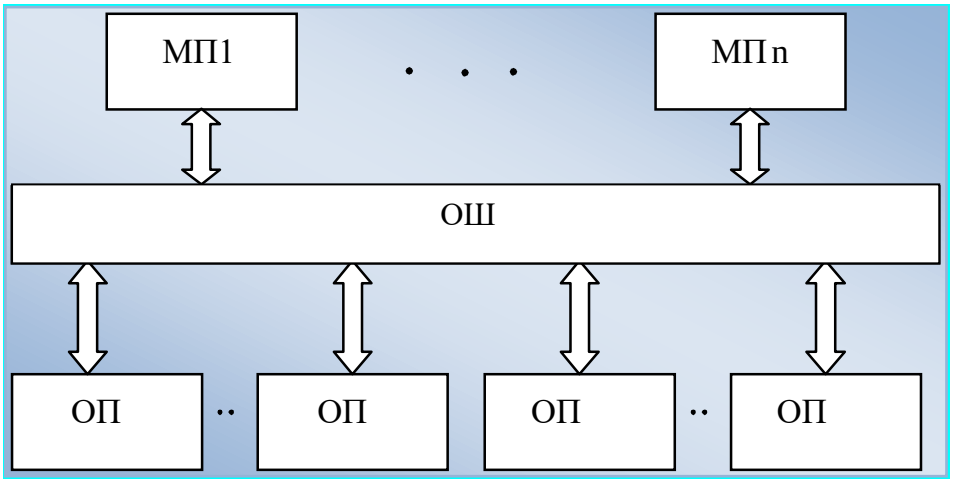


Рис.2.13

Потоки информации, проходящие по общей шине, имеют большую интенсивность, которая будет расти с увеличением числа процессоров в системе. Это приводит к многочисленным конфликтным ситуациям в общей шине. Наращиваемость таких МПВС ограничивает 3...5 процессорами.

МПВС с локальной памятью (ЛП) (рис.2.14) используются для решения слабосвязанных задач. При этом часть программ и данных дублируется, либо характер задачи таков, что она разбивается на ряд подзадач с малым взаимообменом. Тогда каждый вычислительный модуль (ВМ) в своей работе использует главным образом ограниченное подмножество данных, находящихся в его локальной памяти, и крайне редко обращается к остальным модулям памяти за другими данными. Для межпроцессорного обмена данными и хранения данных системного значения выделяется небольшая общая память, которая в виде общих полей может быть размещена непосредственно в локальной памяти (рис.5.14,а) или в виде отдельного блока на общей шине (рис.2.14,б).

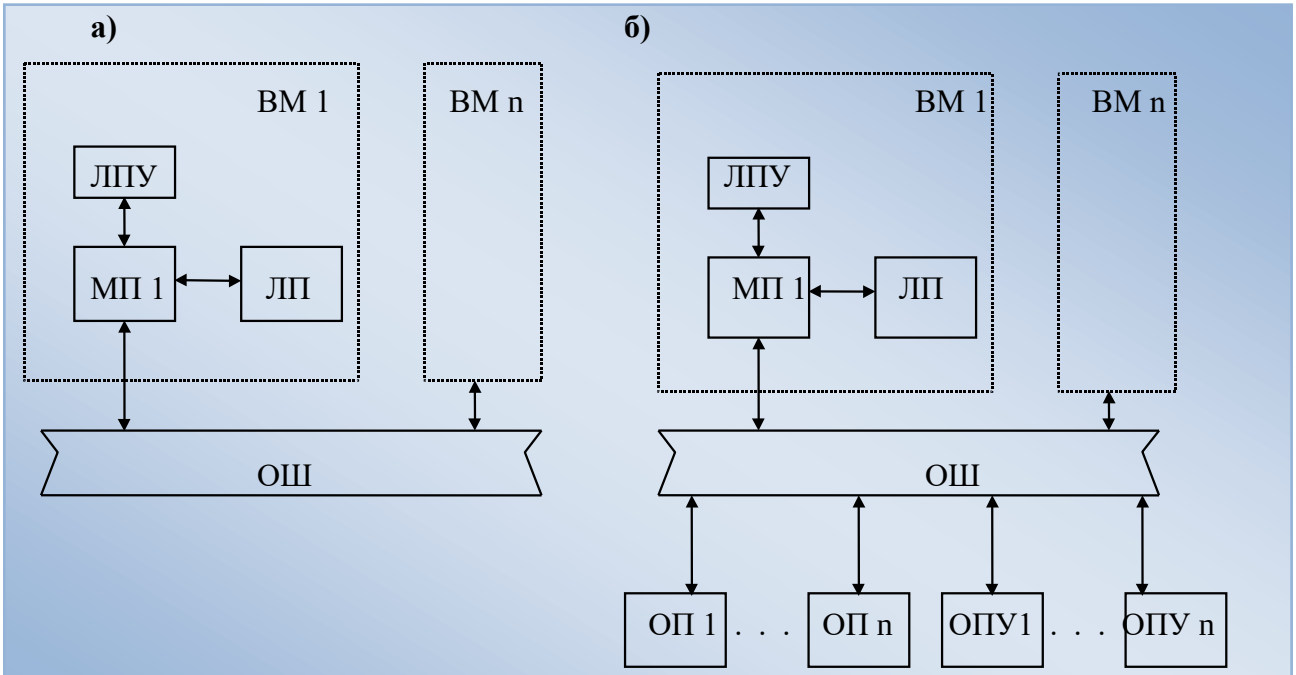


Рис. 2. 14

Интенсивность потока на общей шине снижается по сравнению с интенсивностью потока в ранее рассмотренной структуре, поэтому достижимая производительность при такой организации более высока. Однако дублирование программ и данных приводит к увеличению затрат памяти.

2.2.1 Управление шиной

Организация интерфейса межмодульной связи характеризуется следующими признаками: способом управления общей шиной; способом связи модулей в процессе обмена; дисциплиной обслуживания общей шиной.

По способу управления общей шиной МПВС с ОШ подразделяется на системы с централизованным и децентрализованным управлением. В первом случае выделяется центр, в функции которого входит управление занятием общей шины и обслуживание заявок в соответствии с принятой дисциплиной обслуживания. Во втором способе эти функции распределяются по активным устройствам. Под активными устройствами понимают модули, способные генерировать заявки. Ими могут быть микропроцессоры или быстродействующие ПУ. К пассивным относятся модули, способные обслуживать заявки (оперативная память, УВВ).

Возможны два способа связи модулей в процессе межмодульного обмена по общей шине: без буферизации и с буферизацией сообщений. В первом случае ОШ представляется активному модулю на весь период взаимодействия активного и пассивного модулей. Например, при взаимодействии МП и ОП их соединение производится на время, равное

циклу памяти. За этот промежуток времени по ОШ передается слово данных и сопровождающая его адресная и управляющая информация.

В системах с буферизацией сообщений канал связи между активными и пассивными модулями образуется на короткий промежуток времени, много меньший времени цикла пассивного модуля. Передаваемая информация запоминается в буферных регистрах интерфейсных схем сопряжения ОШ с микропроцессорами. По окончании передачи одного слова данных, ОШ освобождается и предоставляется другому запрашивающему модулю. Такой способ связи обладает высокой скоростью передачи, однако аппаратурные затраты довольно велики.

Для разрешения конфликтных ситуаций при одновременном запросе ОШ несколькими источниками используются дисциплины обслуживания с приоритетами и без приоритетов.

Последняя представляет собой обычный метод циклического обслуживания. При организации приоритетного доступа возможны два варианта: с фиксированными приоритетами и с динамическими.

Первый вариант проще, однако имеет тот недостаток, что модули, имеющие больший приоритет, будут занимать ОШ чаще, чем модули, имеющие низкий приоритет. Вследствие этого вторые будут простаивать. Это явление называют "перекосом" системы. Перекос приводит к потере производительности.

Второй вариант сложнее, требует больших аппаратных затрат. Кроме того, вносятся временные задержки, связанные с передачей приоритетов от одного модуля к другому.

Рассмотрим подробнее организацию интерфейса межмодульной связи в части управления занятием общей шины.

Последовательное управление занятием ОШ, или шлейфовое управление (рис.2.15). Здесь имеют место две линии управления и одна шина. Линия 1 не содержит какой-либо аппаратуры в местах подключения модулей, ее называют линией "блокировки". Линия 2 проходит последовательно через аппаратуру модулей. Шину, используемую для обмена данными, называют ОШ. Будем считать, что высокий потенциал на линии 1 свидетельствует о не занятости ОШ и наоборот.

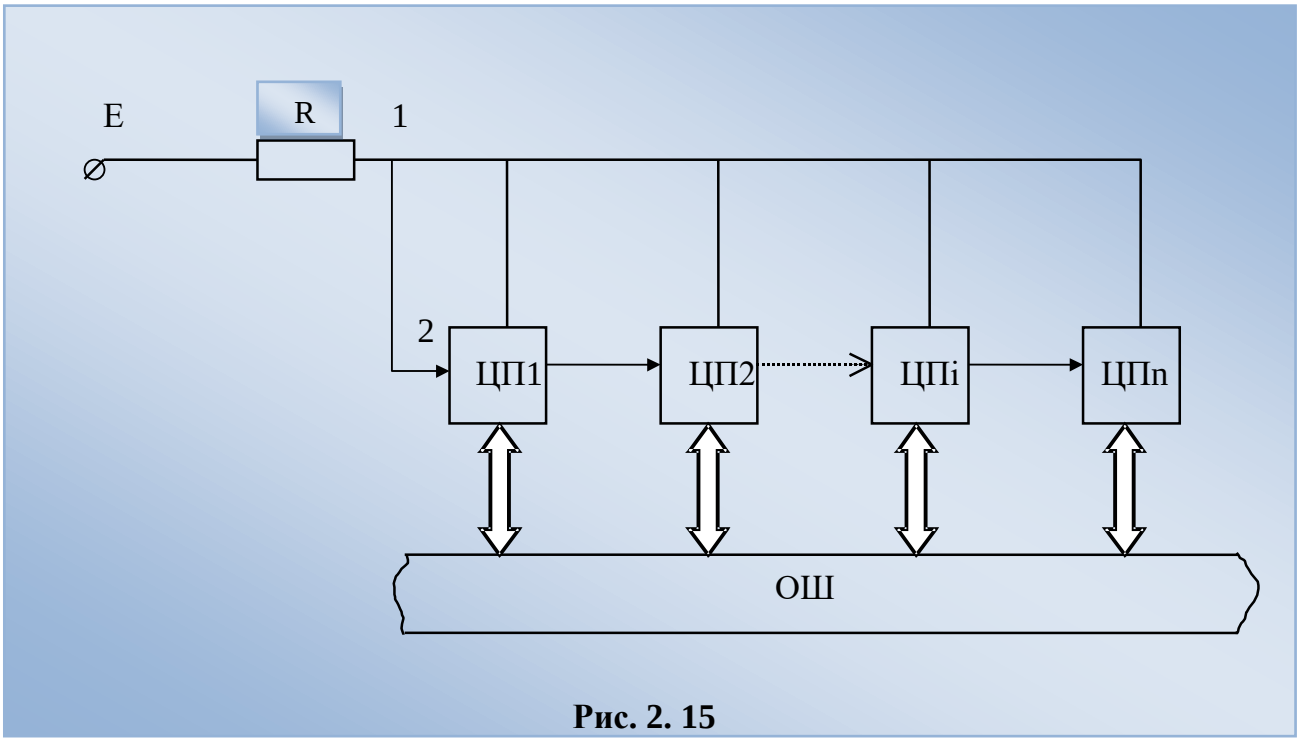


Рис. 2. 15

Если несколько модулей одновременно пытаются занять ОШ, то займет ее крайний левый по схеме. Процесс занятия происходит следующим образом. Если модуль Mi, требующий ОШ, обнаруживает высокий потенциал на линии 1, то он устанавливает на ней низкий потенциал и одновременно выставляет высокий потенциал на линии 2, выходящей из него, после чего Mi ожидает низкий потенциал на своем входе. Если ни один из модулей Mi,..., Mi-1 одновременно с Mi не запрашивал ОШ, то низкий потенциал, установленный в линии1 распространяется до модуля Mi. Совпадение низких потенциалов, поступающих из линий 1 и 2, является разрешающим сигналом для занятий ОШ. Таким образом, рассмотренный способ управлением является децентрализованным. Доступ к ОШ производится с помощью приоритетных схем. Приоритеты модулей фиксированы, причем высшим приоритетом обладает модуль с меньшим номером (крайний левый по схеме).

Достоинство такой структуры - малые аппаратные затраты. Недостаток - большое время занятия ОШ. Для модуля M_n оно максимально и составляет

$$t_3^{max}=n*(t_I+t_L)$$

где t_I - задержка в интерфейсном оборудовании модуля; t_L - время распространения сигнала вдоль длинной линии.

Минимальной время задержки у модуля M_1

$$t_3^{min}= t_I +t_L$$

Среднее время задержки составит:

$$t_З^{ср}=(n+1)*(t_I+t_L)/2$$

Параллельное управление занятием ОШ. Уменьшение времени занятия ОШ достигается другим способом управления радиальным соединением модулей с арбитром, представляющим собой приоритетное устройство управления (рис.2.16). Здесь также имеется линия 1 - "свободно-занято", которая гарантирует монопольное владение ОШ один из модулей. Так же, как и в предыдущем случае, будем считать, что высокий потенциал соответствует не занятости ОШ. Модули, требующие ОШ, опрашивают линию 1. Если на ней обнаруживается высокий потенциал, то они посылают запросы в арбитр по индивидуальным линиям запроса ЗП1,... ЗПn. Модуль, имеющий наивысший приоритет, получит сигнал подтверждения запроса (ПЗП), который является разрешающим сигналом для занятия ОШ.

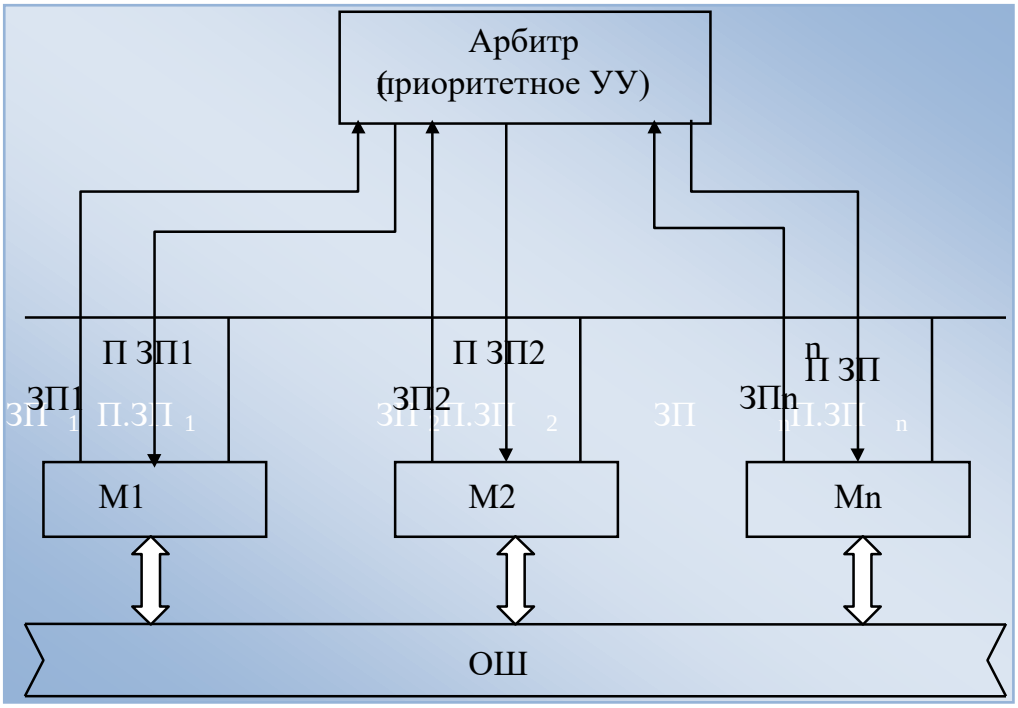


Рис.2.16.

Время занятия одинаково для всех модулей и определяется задержкой в схемах арбитра и в линиях "запрос-ответ":

$$t_З=t_A+2t_L$$

где t_A - время задержки в схемах арбитра.

Такая структура управления является централизованной. Аппаратные затраты здесь значительные (множество линий связи, сложный арбитр), однако обеспечивается малое время занятия.

Способы управления зажатием ОШ, подобные рассмотренным, были разработаны фирмой Intel для организации МПВС на микропроцессорах.

Они получили название последовательной и параллельной Multibus. Отличие их от рассмотренных заключается в том, что в Multibus имеется третья линия Т (параллельная линии 1), по которой передаются тактовые синхроимпульсы от синхрогенератора. Модули, которым необходимо занять ОШ, займут ее только во время отрицательного фронта тактового импульса. Интерфейс Multibus стал в настоящее время практически стандартом. Все зарубежные фирмы, создающие ММПВС с ОШ, выпускают совместимые с ним модули.

Последовательно-параллельное управление занятием ОШ. Последовательно-параллельный (шлейфово-радиальный) принцип управления занятием ОШ используется в интерфейсе *Unibus* (рис.2.17)

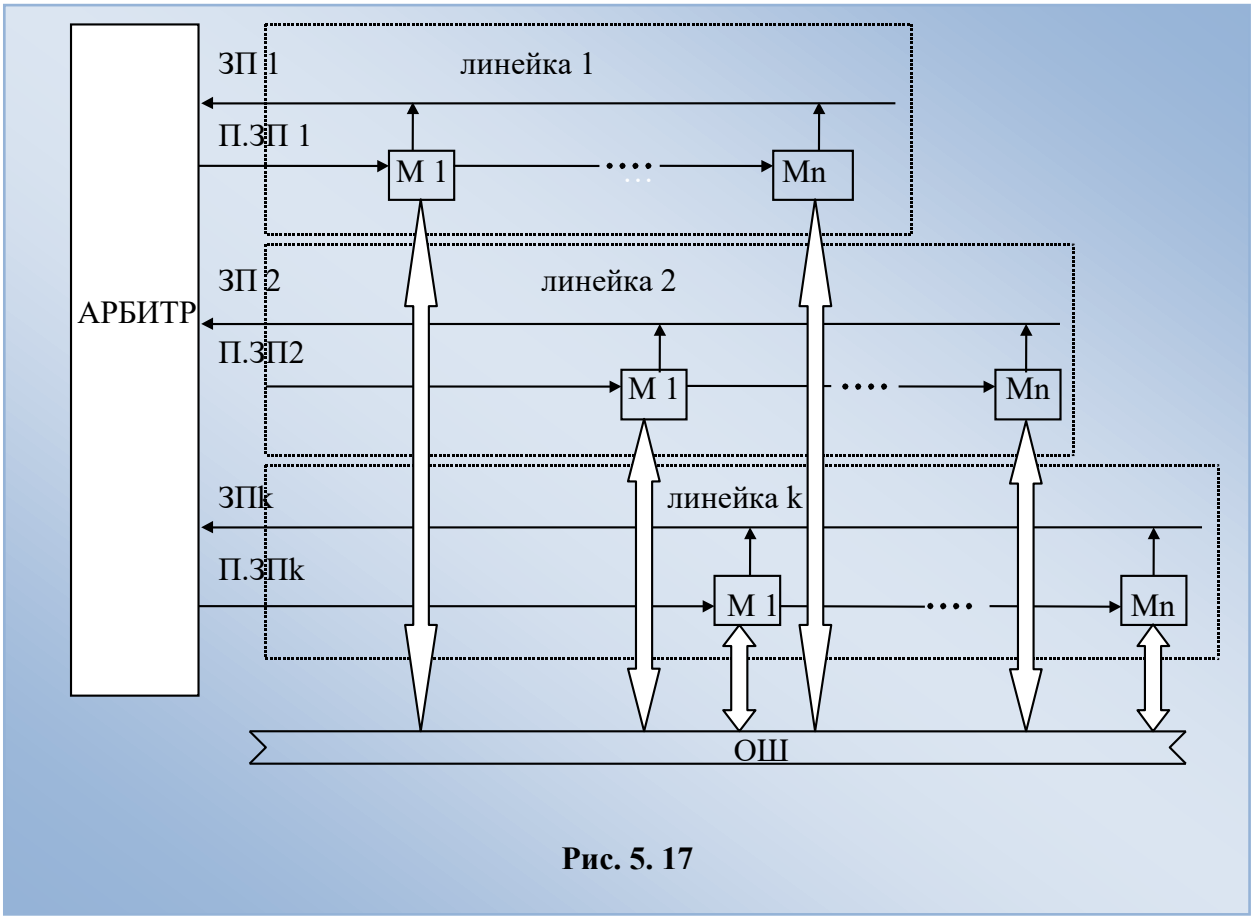


Рис. 5. 17

Такой интерфейс может быть использован и для организации ММПВС с ОШ. Все модули системы группируются в линейки. Каждая линейка имеет линию запроса ЗП и подтверждения запроса ПЗП. Линия ПЗП проходит последовательно все устройства линейки. Запросы ЗП1,...,ЗПn на занятие ОШ удовлетворяются согласно присвоенным им приоритетам в центральном арбитре (приоритетный уровень 1). Внутри линеек приоритеты определяются последовательной схемой управления занятием ОШ (приоритетный уровень 2).

Циклическое обслуживание занятием ОШ. Рассмотрим одну из бесприоритетных дисциплин управления занятием ОШ на примере циклического обслуживания (рис. 5. 18). Такое управление часто

используется в различных системах из-за своей простоты. Управление заключается в том, что ОШ последовательно предоставляется модулям на некоторый промежуток времени, достаточный для выполнения операции обмена. После того, как будут обслужены все модули, начинается новый цикл.

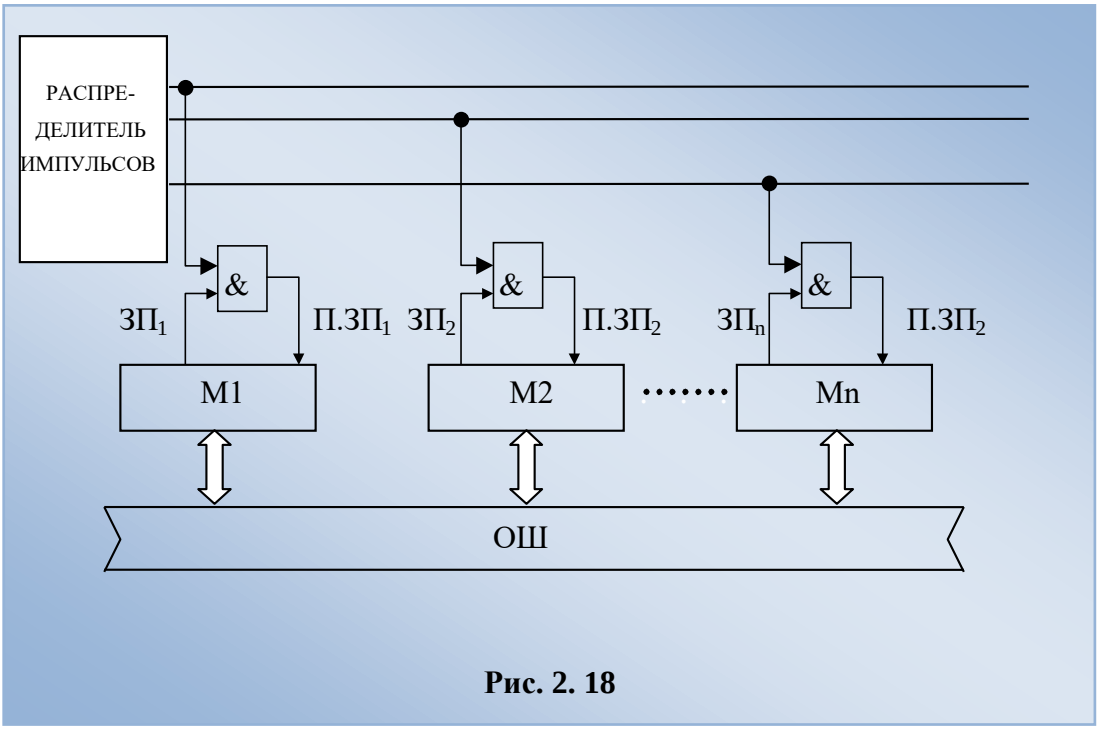


Рис. 2. 18

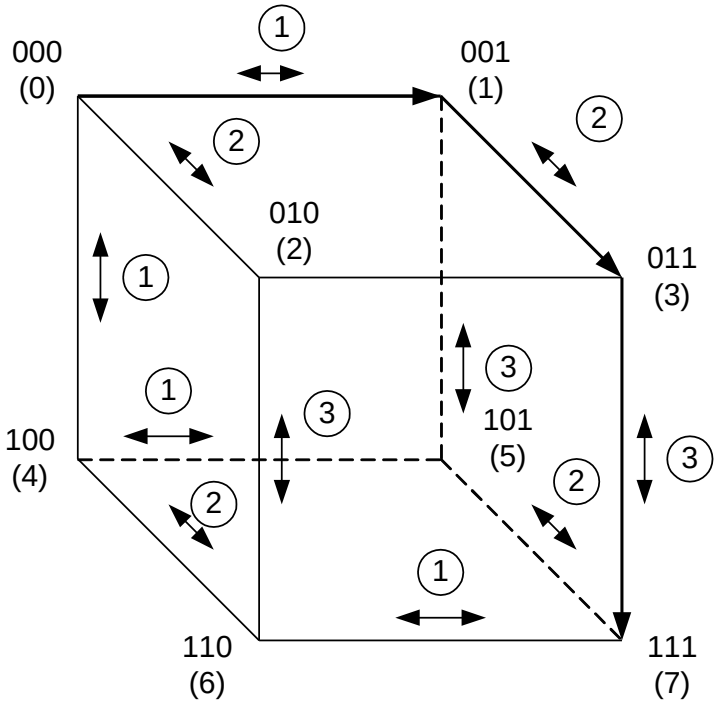
Функционирование схемы заключается в следующем. Если модуль требует ОШ, то посылает запрос на ее занятие и ожидает до тех пор, пока не наступит его очередь обслуживания. Если запрос отсутствует, то модуль не подключается к ОШ даже в момент ее предоставления и продолжает самостоятельную работу. Время занятия может быть большим, если запросы не совпадают по времени с моментами принудительного предоставления ОШ. Число линий управления сравнительно мало и составляет $\log_2 n$, где n - число модулей, подключенных к ОШ.

2.3. Организация коммуникационной среды в системах с распределённой памятью.

К настоящему времени разработано множество схем соединения, осуществляющих связь между произвольными процессорами с помощью коммуникационной среды из многоступенчатых переключателей, размещенных в log2 *n* ступенях. В качестве примеров этих схем можно назвать сеть Омега и *n*-кубическую (гиперкубическую) сеть.

Принцип действия п-кубической сети отражен на рис. 2.7, *a*, а пример его структурной реализации на многоступенчатом переключателе показан на рис. 2.7,*b*. На рисунке представлена система соединения восьми процессоров. Как видно из рис. 2.7, *a*, всем процессорам присваиваются двоичные номера *(Xm-1…X1Xo).* Процессор с номером *(Xm-1 ... X1Xo)* соединяется с *m*

процессорами, для номеров которых расстояние Хемминга равняется 1. В примере на рис. 2.7, *а* процессор (000) может быть соединен с каждым из процессоров с номерами (001), (010) и (100), а процессор (010) - с любым из процессоров с номерами (011), (000) и (110).



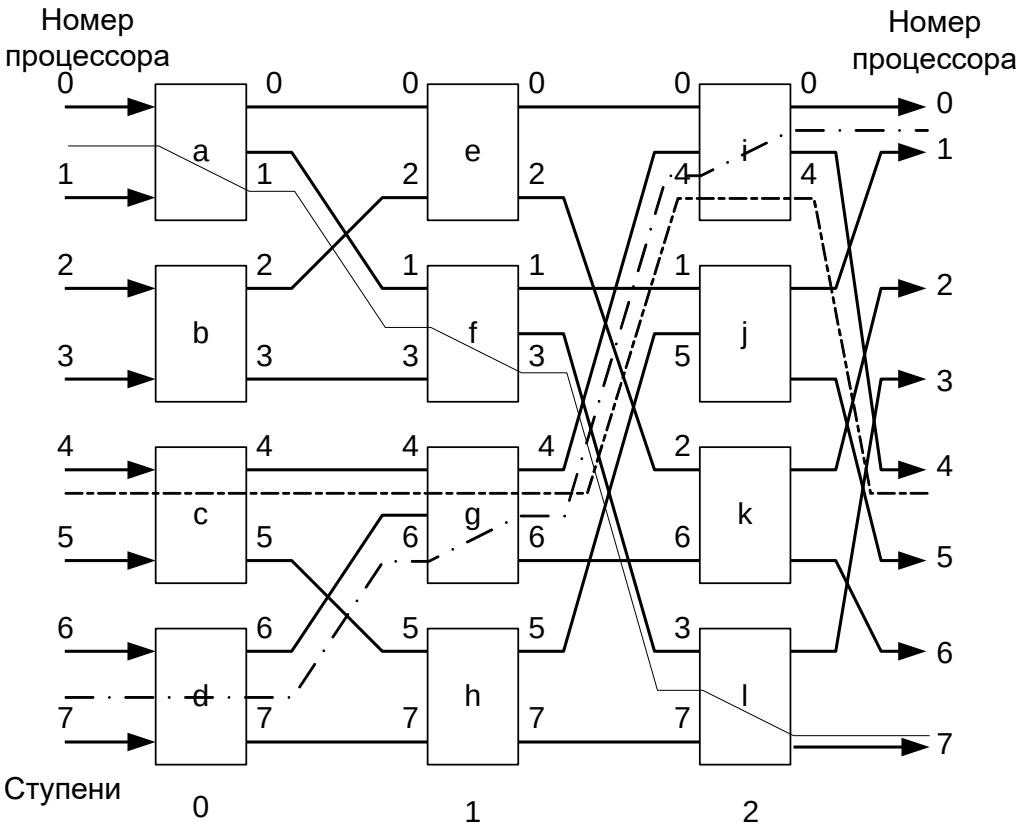


Рис.2.7. Коммуникационная среда на основе гиперкубической сети.

Рассмотрим теперь передачу данных от исходного процессора *(Xm-1... X1Xo)* в процессор-адресат *(Ym-1 ... Y1Y0).* Сначала сравниваются биты Xo и Y0 младших разрядов номеров процессоров. При *Xo = Y0* данные из процессора *(Xm-1 ... X1Xo*), передаются в процессор (*Xm-1 ... X1Xo)* (на рис. 2.7,*a* это обозначено знаком 1 в кружке). При *Xo = Yo* данные не передаются. Далее, в номерах процессоров(*Xm-1 ... X1Xo)* и *(Ym-1…Y1Y0)* сравниваются биты *X1* и *Y1* и осуществляется аналогичная процедура (на рис. 2.7,*a* обозначена индексом 2 в кружке). Этот процесс повторяется со всеми парами битов до самого старшего разряда; в результате *m*-кратного повторения описанной процедуры данные поступают в процессор *(Ym-1 ...Y1Y0).* Путь передачи данных от процессора (000) в процессор (111) на рис. 2.7,*a* показан утолщенной линией. Рассмотрим реализацию данного способа с использованием многоступенчатого переключателя с *m* ступенями (log2 *n* ступеней).

Как видно из рис. 2.7,*b* , на каждой ступени размещается *n*/2 переключателей. На каждый из переключателей ступени 0 поступают данные от пары процессоров с двоичными номерами, различающимися значениями младших разрядов (например, (000) и (001), (010) и (Oil) и т. д.). В общем случае на каждый из переключателей ступени подаются данные от двух процессоров с номерами, различающимися значением *i*-го бита (например, на ступени 1 это (000) и (010), (001) и (Oil) и т.д. Для переключателей ступени *i* сравниваются бит *Xi* номера процессора-источника и бит *Yi* номера процессора-получателя; при *Xi=Yi* переключатель устанавливается в режим прямой связи, а при *Xi* не равным *Yi* - в режим диагональной связи. Таким образом, реализуется отмеченный ранее принцип *n*-кубической коммутации («прямая связь», когда данные не передаются, и «диагональная связь», когда данные передаются). Например, при передаче данных из процессора (000) в процессор (111) управление переключателями производится в соответствии с пунктирной линией передачи на рис. 2.7,*b*. Поскольку для всех переключателей управление их режимов производится на основании сравнения *Xi* и *Yi,* необходимо снабжать данные заголовком, представляющим собой поразрядную сумму *X(Xm-1 ... X1Xo)* и *Y{Ym-1...Y1Yo)* по модулю 2. Со стороны процессора - источника вычисляется *Zi = Xi + Yi,* где (+ - исключающее ИЛИ), и каждый переключатель при *Zi* = 0 устанавливается в режим прямой связи, а при *Zi* = 1 - в режим диагональной связи. Длина заголовка данных не превышает *m* бит.

При такой многоступенчатой коммутации возможна связь между любыми процессорами. Однако при одновременной организации связи между более чем двумя процессорами, как показано на рис. 2.7,*b* штрихпунктирной линией, возможны конфликты переключателей (блокирование). На рис. 2.7,*b* показана ситуация, когда одновременно передаются данные от процессора 4 к процессору 0 и от процессора 6 к процессору 4. При возникновении конфликтов данные запоминаются в буфере переключателя и ожидают своей очереди на передачу.

2.4. Когерентность кэш- памяти в SMP- системах.

Требования, предъявляемые современными процессорами к полосе пропускания памяти можно существенно сократить путем применения больших многоуровневых кэшей. Тогда, если эти требования снижаются, то несколько процессоров смогут разделять доступ к одной и той же памяти. Начиная с 1980 года эта идея, подкрепленная широким распространением микропроцессоров, стимулировала многих разработчиков на создание небольших мультипроцессоров, в которых несколько процессоров разделяют одну физическую память, соединенную с ними с помощью разделяемой шины. Из-за малого размера процессоров и заметного сокращения требуемой полосы пропускания шины, достигнутого за счет возможности реализации достаточно большой кэш-памяти, такие машины стали исключительно эффективными по стоимости. В первых разработках подобного рода машин удавалось разместить весь процессор и кэш на одной плате, которая затем вставлялась в заднюю панель, с помощью которой реализовывалась шинная архитектура. Современные конструкции позволяют разместить до четырех процессоров на одной плате.

В такой машине кэши могут содержать как разделяемые, так и частные данные. Частные данные - это данные, которые используются одним процессором, в то время как разделяемые данные используются многими процессорами, по существу обеспечивая обмен между ними. Когда кэшируется элемент частных данных, их значение переносится в кэш для сокращения среднего времени доступа, а также требуемой полосы пропускания. Поскольку никакой другой процессор не использует эти данные, этот процесс идентичен процессу для однопроцессорной машины с кэш-памятью. Если кэшируются разделяемые данные, то разделяемое значение реплицируется и может содержаться в нескольких кэшах. Кроме сокращения задержки доступа и требуемой полосы пропускания такая репликация данных способствует также общему сокращению количества обменов. Однако кэширование разделяемых данных вызывает новую проблему: когерентность кэш-памяти.

Проблема, о которой идет речь, возникает из-за того, что значение элемента данных в памяти, хранящееся в двух разных процессорах, доступно этим процессорам только через их индивидуальные кэши. На рис. 2.8 показан простой пример, иллюстрирующий эту проблему.

Проблема когерентности памяти для мультипроцессоров и устройств ввода/вывода имеет много аспектов. Обычно в малых мультипроцессорах используется аппаратный механизм, называемый протоколом, позволяющий решить эту проблему. Такие протоколы называются протоколами когерентности кэш-памяти. Существуют два класса таких протоколов:

1. Протоколы на основе справочника (directory based). Информация о состоянии блока физической памяти содержится только в одном месте, называемом справочником (физически справочник может быть распределен по узлам системы). Этот подход будет рассмотрен в разд. 10.3.

2. Протоколы наблюдения (snooping). Каждый кэш, который содержит копию данных некоторого блока физической памяти, имеет также соответствующую копию служебной информации о его состоянии. Централизованная система записей отсутствует. Обычно кэши расположены на общей (разделяемой) шине и контроллеры всех кэшей наблюдают за шиной (просматривают ее) для определения того, не содержат ли они копию соответствующего блока.

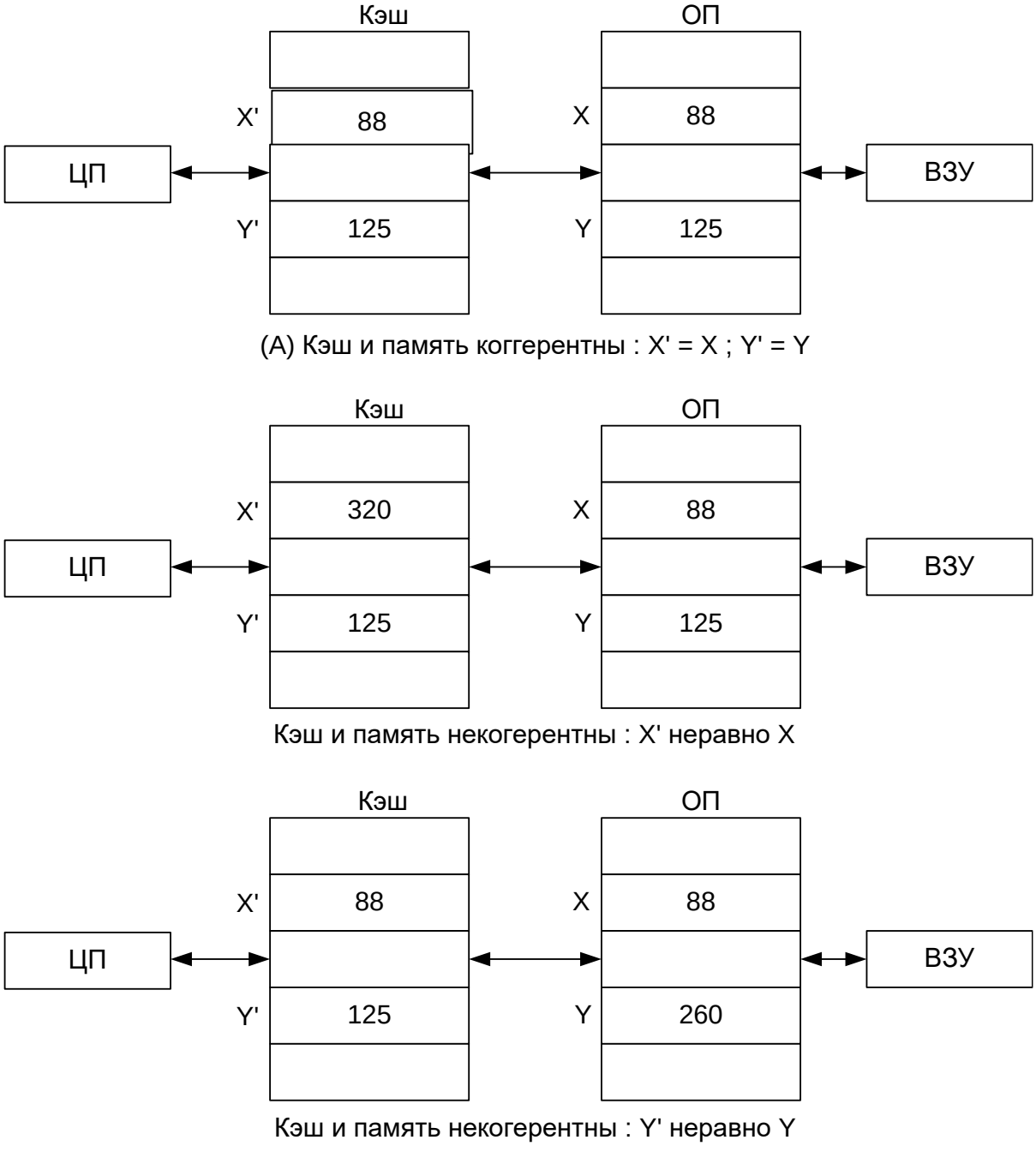


Рис. 2.8. Иллюстрация проблемы когерентности кэш-памяти

а) Когерентное состояние кэша и основной памяти.

б) Предполагается использование кэш-памяти с обратной записью, когда ЦП записывает значение 320 в ячейку X. В результате X’ содержит новое значение, а в основной памяти осталось старое значение 88. При попытке вывода X из памяти будет получено старое значение.

в) Внешняя память вводит в ячейку памяти Y новое значение 260, а в кэш-памяти осталось старое значение Y.

В мультипроцессорных системах, использующих процессоры с кэш-памятью, подсоединенные к централизованной общей памяти, протоколы наблюдения приобрели популярность, поскольку для опроса состояния кэшей они могут использовать заранее существующее физическое соединение - шину памяти.

Неформально, проблема когерентности памяти состоит в необходимости гарантировать, что любое считывание элемента данных возвращает последнее по времени записанное в него значение. Это определение не совсем корректно, поскольку невозможно требовать, чтобы операция считывания мгновенно видела значение, записанное в этот элемент данных некоторым другим процессором. Если, например, операция записи на одном процессоре предшествует операции чтения той же ячейки на другом процессоре в пределах очень короткого интервала времени, то невозможно гарантировать, что чтение вернет записанное значение данных, поскольку в этот момент времени записываемые данные могут даже не покинуть процессор. Вопрос о том, когда точно записываемое значение должно быть доступно процессору, выполняющему чтение, определяется выбранной моделью согласованного (непротиворечивого) состояния памяти и связан с реализацией синхронизации параллельных вычислений. Поэтому с целью упрощения предположим, что мы требуем только, чтобы записанное операцией записи значение было доступно операции чтения, возникшей немного позже записи и что операции записи данного процессора всегда видны в порядке их выполнения.

С этим простым определением согласованного состояния памяти мы можем гарантировать когерентность путем обеспечения двух свойств:

1. Операция чтения ячейки памяти одним процессором, которая следует за операцией записи в ту же ячейку памяти другим процессором получит записанное значение, если операции чтения и записи достаточно отделены друг от друга по времени.

2. Операции записи в одну и ту же ячейку памяти выполняются строго последовательно (иногда говорят, что они сериализованы): это означает, что две подряд идущие операции записи в одну и ту же ячейку памяти будут наблюдаться другими процессорами именно в том порядке, в

котором они появляются в программе процессора, выполняющего эти операции записи.

Первое свойство очевидно связано с определением когерентного (согласованного) состояния памяти: если бы процессор всегда бы считывал только старое значение данных, мы сказали бы, что память некогерентна.

Необходимость строго последовательного выполнения операций записи является более тонким, но также очень важным свойством. Представим себе, что строго последовательное выполнение операций записи не соблюдается. Тогда процессор P1 может записать данные в ячейку, а затем в эту ячейку выполнит запись процессор P2. Строго последовательное выполнение операций записи гарантирует два важных следствия для этой последовательности операций записи. Во-первых, оно гарантирует, что каждый процессор в машине в некоторый момент времени будет наблюдать запись, выполняемую процессором P2. Если последовательность операций записи не соблюдается, то может возникнуть ситуация, когда какой-нибудь процессор будет наблюдать сначала операцию записи процессора P2, а затем операцию записи процессора P1, и будет хранить это записанное P1 значение неограниченно долго. Более тонкая проблема возникает с поддержанием разумной модели порядка выполнения программ и когерентности памяти для пользователя: представьте, что третий процессор постоянно читает ту же самую ячейку памяти, в которую записывают процессоры P1 и P2; он должен наблюдать сначала значение, записанное P1, а затем значение, записанное P2. Возможно он никогда не сможет увидеть значения, записанного P1, поскольку запись от P2 возникла раньше чтения. Если он даже видит значение, записанное P1, он должен видеть значение, записанное P2, при последующем чтении. Подобным образом любой другой процессор, который может наблюдать за значениями, записываемыми как P1, так и P2, должен наблюдать идентичное поведение. Простейший способ добиться таких свойств заключается в строгом соблюдении порядка операций записи, чтобы все записи в одну и ту же ячейку могли наблюдаться в том же самом порядке. Это свойство называется последовательным выполнением (сериализацией) операций записи (write serialization). Вопрос о том, когда процессор должен увидеть значение, записанное другим процессором достаточно сложен и имеет заметное воздействие на производительность, особенно в больших машинах.

Альтернативные протоколы

Имеются две методики поддержания описанной выше когерентности. Один из методов заключается в том, чтобы гарантировать, что процессор должен получить исключительные права доступа к элементу данных перед выполнением записи в этот элемент данных. Этот тип протоколов называется протоколом записи с аннулированием (write ivalidate protocol), поскольку при выполнении записи он аннулирует другие копии. Это наиболее часто используемый протокол как в схемах на основе справочников, так и в схемах наблюдения. Исключительное право доступа гарантирует, что во время выполнения записи не существует никаких других копий элемента данных, в которые можно писать или из которых можно читать: все другие кэшированные копии элемента данных аннулированы. Чтобы увидеть, как такой протокол обеспечивает когерентность, рассмотрим операцию записи, вслед за которой следует операция чтения другим процессором. Поскольку запись требует исключительного права доступа, любая копия, поддерживаемая читающим процессором должна быть аннулирована (в соответствии с названием протокола). Таким образом, когда возникает операция чтения, произойдет промах кэш-памяти, который вынуждает выполнить выборку новой копии данных. Для выполнения операции записи мы можем потребовать, чтобы процессор имел достоверную (valid) копию данных в своей кэш-памяти прежде, чем выполнять в нее запись. Таким образом, если оба процессора попытаются записать в один и тот же элемент данных одновременно, один из них выиграет состязание у второго (мы вскоре увидим, как принять решение, кто из них выиграет) и вызывает аннулирование его копии. Другой процессор для завершения своей операции записи должен сначала получить новую копию данных, которая теперь уже должна содержать обновленное значение.

Альтернативой протоколу записи с аннулированием является обновление всех копий элемента данных в случае записи в этот элемент данных. Этот тип протокола называется протоколом записи с обновлением (write update protocol) или протоколом записи с трансляцией (write broadcast protocol). Обычно в этом протоколе для снижения требований к полосе пропускания полезно отслеживать, является ли слово в кэш-памяти разделяемым объектом, или нет, а именно, содержится ли оно в других кэшах. Если нет, то нет никакой необходимости обновлять другой кэш или транслировать в него обновленные данные.

Разница в производительности между протоколами записи с обновлением и с аннулированием определяется тремя характеристиками:

1. Несколько последовательных операций записи в одно и то же слово, не перемежающихся операциями чтения, требуют нескольких операций трансляции при использовании протокола записи с обновлением, но только одной начальной операции аннулирования при использовании протокола записи с аннулированием.

2. При наличии многословных блоков в кэш-памяти каждое слово, записываемое в блок кэша, требует трансляции при использовании протокола записи с обновлением, в то время как только первая запись в любое слово блока нуждается в генерации операции аннулирования при использовании протокола записи с аннулированием. Протокол записи с аннулированием работает на уровне блоков кэш-памяти, в то время как протокол записи с обновлением должен работать на уровне отдельных слов (или байтов, если выполняется запись байта).

3. Задержка между записью слова в одном процессоре и чтением записанного значения другим процессором обычно меньше при использовании схемы записи с обновлением, поскольку записанные данные немедленно транслируются в процессор, выполняющий чтение (предполагается, что этот процессор имеет копию данных). Для сравнения, при использовании протокола записи с аннулированием в процессоре, выполняющим чтение, сначала произойдет аннулирование его копии, затем будет производиться чтение данных и его приостановка до тех пор, пока обновленная копия блока не станет доступной и не вернется в процессор.

Эти две схемы во многом похожи на схемы работы кэш-памяти со сквозной записью и с записью с обратным копированием. Также как и схема задержанной записи с обратным копированием требует меньшей полосы пропускания памяти, так как она использует преимущества операций над целым блоком, протокол записи с аннулированием обычно требует менее тяжелого трафика, чем протокол записи с обновлением, поскольку несколько записей в один и тот же блок кэш-памяти не требуют трансляции каждой записи. При сквозной записи память обновляется почти мгновенно после записи (возможно с некоторой задержкой в буфере записи). Подобным образом при использовании протокола записи с обновлением другие копии обновляются так быстро, насколько это возможно. Наиболее важное отличие в производительности протоколов записи с аннулированием и с обновлением связано с характеристиками прикладных программ и с выбором размера блока.

Основы реализации

Ключевым моментом реализации в многопроцессорных системах с небольшим числом процессоров как схемы записи с аннулированием, так и схемы записи с обновлением данных, является использование для выполнения этих операций механизма шины. Для выполнения операции обновления или аннулирования процессор просто захватывает шину и

транслирует по ней адрес, по которому должно производиться обновление или аннулирование данных. Все процессоры непрерывно наблюдают за шиной, контролируя появляющиеся на ней адреса. Процессоры проверяют не находится ли в их кэш-памяти адрес, появившийся на шине. Если это так, то соответствующие данные в кэше либо аннулируются, либо обновляются в зависимости от используемого протокола. Последовательный порядок обращений, присущий шине, обеспечивает также строго последовательное выполнение операций записи, поскольку когда два процессора конкурируют за выполнение записи в одну и ту же ячейку, один из них должен получить доступ к шине раньше другого. Один процессор, получив доступ к шине, вызовет необходимость обновления или аннулирования копий в других процессорах. В любом случае, все записи будут выполняться строго последовательно. Один из выводов, который следует сделать из анализа этой схемы заключается в том, что запись в разделяемый элемент данных не может закончиться до тех пор, пока она не захватит доступ к шине.

В дополнение к аннулированию или обновлению соответствующих копий блока кэш-памяти, в который производилась запись, мы должны также разместить элемент данных, если при записи происходит промах кэш-памяти. В кэш-памяти со сквозной записью последнее значение элемента данных найти легко, поскольку все записываемые данные всегда посылаются также и в память, из которой последнее записанное значение элемента данных может быть выбрано (наличие буферов записи может привести к некоторому усложнению).

Однако для кэш-памяти с обратным копированием задача нахождения последнего значения элемента данных сложнее, поскольку это значение скорее всего находится в кэше, а не в памяти. В этом случае используется та же самая схема наблюдения, что и при записи: каждый процессор наблюдает и контролирует адреса, помещаемые на шину. Если процессор обнаруживает, что он имеет модифицированную ("грязную") копию блока кэш-памяти, то именно он должен обеспечить пересылку этого блока в ответ на запрос чтения и вызвать отмену обращения к основной памяти. Поскольку кэши с обратным копированием предъявляют меньшие требования к полосе пропускания памяти, они намного предпочтительнее в мультипроцессорах, несмотря на некоторое увеличение сложности. Поэтому далее мы рассмотрим вопросы реализации кэш-памяти с обратным копированием.

Для реализации процесса наблюдения могут быть использованы обычные теги кэша. Более того, упоминавшийся ранее бит достоверности (valid bit), позволяет легко реализовать аннулирование. Промахи операций чтения, вызванные либо аннулированием, либо каким-нибудь другим событием, также не сложны для понимания, поскольку они просто основаны на возможности наблюдения. Для операций записи мы хотели бы также знать, имеются ли другие кэшированные копии блока, поскольку в случае

отсутствия таких копий, запись можно не посылать на шину, что сокращает время на выполнение записи, а также требуемую полосу пропускания.

Чтобы отследить, является ли блок разделяемым, мы можем ввести дополнительный бит состояния (shared), связанный с каждым блоком, точно также как это делалось для битов достоверности (valid) и модификации (modified или dirty) блока. Добавив бит состояния, определяющий является ли блок разделяемым, мы можем решить вопрос о том, должна ли запись генерировать операцию аннулирования в протоколе с аннулированием, или операцию трансляции при использовании протокола с обновлением. Если происходит запись в блок, находящийся в состоянии "разделяемый" при использовании протокола записи с аннулированием, кэш формирует на шине операцию аннулирования и помечает блок как частный (private). Никаких последующих операций аннулирования этого блока данный процессор посылать больше не будет. Процессор с исключительной (exclusive) копией блока кэш-памяти обычно называется "владельцем" (owner) блока кэш-памяти.

При использовании протокола записи с обновлением, если блок находится в состоянии "разделяемый", то каждая запись в этот блок должна транслироваться. В случае протокола с аннулированием, когда посылается операция аннулирования, состояние блока меняется с "разделяемый" на "неразделяемый" (или "частный"). Позже, если другой процессор запросит этот блок, состояние снова должно измениться на "разделяемый". Поскольку наш наблюдающий кэш видит также все промахи, он знает, когда этот блок кэша запрашивается другим процессором, и его состояние должно стать "разделяемый".

Поскольку любая транзакция на шине контролирует адресные теги кэша, потенциально это может приводить к конфликтам с обращениями к кэшу со стороны процессора. Число таких потенциальных конфликтов можно снизить применением одного из двух методов: дублированием тегов, или использованием многоуровневых кэшей с "охватом" (inclusion), в которых уровни, находящиеся ближе к процессору являются поднабором уровней, находящихся дальше от него. Если теги дублируются, то обращения процессора и наблюдение за шиной могут выполняться параллельно. Конечно, если при обращении процессора происходит промах, он должен будет выполнять арбитраж с механизмом наблюдения для обновления обоих наборов тегов. Точно также, если механизм наблюдения за шиной находит совпадающий тег, ему будет нужно проводить арбитраж и обращаться к обоим наборам тегов кэша (для выполнения аннулирования или обновления бита "разделяемый"), возможно также и к массиву данных в кэше, для нахождения копии блока. Таким образом, при использовании схемы дублирования тегов процессор должен приостановиться только в том случае, если он выполняет обращение к кэшу в тот же самый момент времени, когда

механизм наблюдения обнаружил копию в кэше. Более того, активность механизма наблюдения задерживается только когда кэш имеет дело с промахом.

Наименование	Тип протокола	Стратегия записи в память	Уникальные свойства Применение
Одиночная запись	Запись с аннулированием	Обратное копирование при первой записи	Первый описанный в литературе протокол наблюдения -
Synapse N+1	Запись с аннулированием	Обратное копирование	Точное состояние, где "владельцем является память" Машины Synapse Первые машины с когерентной кэш-памятью
Berkely	Запись с аннулированием	Обратное копирование	Состояние "разделяемый" Машина SPUR университета Berkely
Illinois	Запись с аннулированием	Обратное копирование	Состояние "приватный"; может передавать данные из любого кэша Серии Power и Challenge компании Silicon Graphics

"Firefly"	Запись с трансляцией	Обратное копирование для "приватных" блоков и сквозная запись для "разделяемых"	Обновление памяти во время трансляции SPARC center 2000

Рис. 2.9. Примеры протоколов наблюдения

Если процессор использует многоуровневый кэш со свойствами охвата, тогда каждая строка в основном кэше имеется и во вторичном кэше. Таким образом, активность по наблюдению может быть связана с кэшем второго уровня, в то время как большинство активностей процессора могут быть связаны с первичным кэшем. Если механизм наблюдения получает попадание во вторичный кэш, тогда он должен выполнять арбитраж за первичный кэш, чтобы обновить состояние и возможно найти данные, что обычно будет приводить к приостановке процессора. Такое решение было принято во многих современных системах, поскольку многоуровневый кэш позволяет существенно снизить требований к полосе пропускания. Иногда может быть даже полезно дублировать теги во вторичном кэше, чтобы еще больше сократить количество конфликтов между активностями процессора и механизма наблюдения.

В реальных системах существует много вариаций схем когерентности кэша, в зависимости от того используется ли схема на основе аннулирования или обновления, построена ли кэш-память на принципах сквозной или обратной записи, когда происходит обновление, а также имеет ли место состояние "владения" и как оно реализуется. На рис. 2.9 представлены несколько протоколов с наблюдением и некоторые машины, которые используют эти протоколы.

2.5. Когерентность кэш- памяти в MPP-системах.

Существуют два различных способа построения крупномасштабных систем с распределенной памятью. Простейший способ заключается в том, чтобы исключить аппаратные механизмы, обеспечивающие когерентность кэш-памяти, и сосредоточить внимание на создании масштабируемой системы памяти. Несколько компаний разработали такого типа машины.

Наиболее известным примером такой системы является компьютер T3D компании Cray Research. В этих машинах память распределяется между узлами (процессорными элементами) и все узлы соединяются между собой посредством того или иного типа сети. Доступ к памяти может быть локальным или удаленным. Специальные контроллеры, размещаемые в узлах сети, могут на основе анализа адреса обращения принять решение о том, находятся ли требуемые данные в локальной памяти данного узла, или размещаются в памяти удаленного узла. В последнем случае контроллеру удаленной памяти посылается сообщение для обращения к требуемым данным.

Чтобы обойти проблемы когерентности, разделяемые (общие) данные не кэшируются. Конечно, с помощью программного обеспечения можно реализовать некоторую схему кэширования разделяемых данных путем их копирования из общего адресного пространства в локальную память конкретного узла. В этом случае когерентностью памяти также будет управлять программное обеспечение. Преимуществом такого подхода является практически минимально необходимая поддержка со стороны аппаратуры, хотя наличие, например, таких возможностей как блочное (групповое) копирование данных было бы весьма полезным. Недостатком такой организации является то, что механизмы программной поддержки когерентности подобного рода кэш-памяти компилятором весьма ограничены. Существующая в настоящее время методика в основном подходит для программ с хорошо структурированным параллелизмом на уровне программного цикла.

Машины с архитектурой, подобной Cray T3D, называют процессорами (машинами) с массовым параллелизмом (MPP Massively Parallel Processor). К машинам с массовым параллелизмом предъявляются взаимно исключающие требования. Чем больше объем устройства, тем большее число процессоров можно расположить в нем, тем длиннее каналы передачи управления и данных, а значит и меньше тактовая частота. Происшедшее возрастание нормы массивности для больших машин до 512 и даже 64К процессоров обусловлено не ростом размеров машины, а повышением степени интеграции схем, позволившей за последние годы резко повысить плотность размещения элементов в устройствах. Топология сети обмена между процессорами в такого рода системах может быть различной. На рис. 2.10 приведены характеристики сети обмена для некоторых коммерческих MPP.

В частности, чтобы предотвратить появление узкого горла в системе, связанного с единым справочником, можно распределить части этого справочника вместе с устройствами распределенной локальной памяти. Таким образом можно добиться того, что обращения к разным справочникам (частям единого справочника) могут выполняться параллельно, точно также как обращения к локальной памяти в распределенной памяти могут

выполняться параллельно, существенно увеличивая общую полосу пропускания памяти. В распределенном справочнике сохраняется главное свойство подобных схем, заключающееся в том, что состояние любого разделяемого блока данных всегда находится во вполне определенном известном месте. На рис.2.11 показан общий вид подобного рода системы с распределенной памятью.

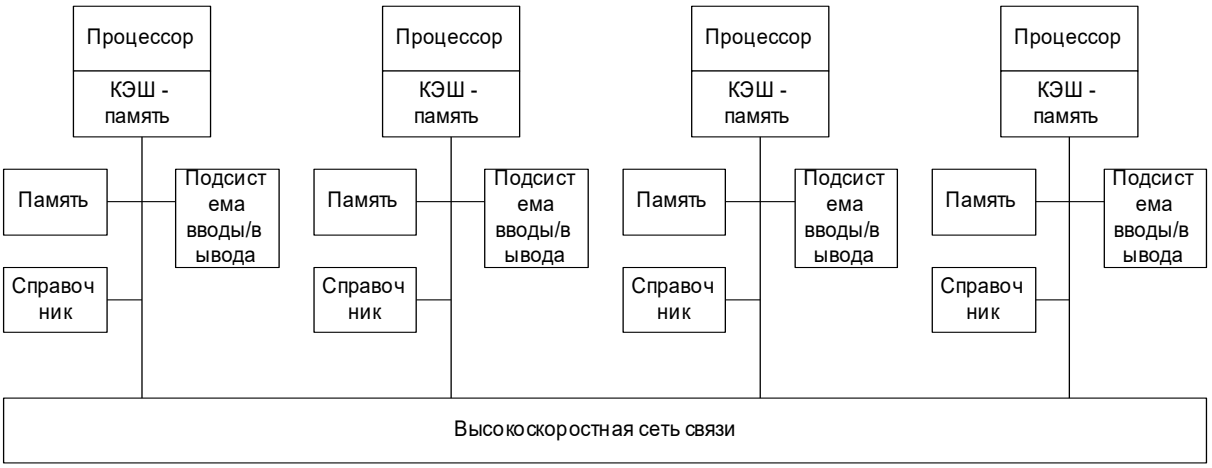


Рис. 2.11. Архитектура системы с распределенной памятью и распределенными по узлам справочниками.

2.6. Организация прерываний в мультипроцессорных системах

Рассмотрим реализацию прерываний в наиболее простых симметричных многопроцессорных системах, в которых используется несколько процессоров, объединенных общей шиной. Каждый процессор выполняет свою задачу, задаваемую операционной системой (ОС). При этом процессоры совместно используют общие ресурсы системы (память, внешние устройства), обращение к которым регулируется ОС. В каждый момент времени один из процессоров является ведущим (master) - только он имеет доступ к системной шине. Другие процессоры в случае необходимости обращения к шине выдают соответствующий запрос. Эти запросы анализируются специальным устройством - арбитром шины, который работает под управлением ОС. В соответствии с определенным алгоритмом арбитр предоставляет доступ к шине одному из запросивших процессоров, который становится таким образом ведущим. Поддержку функционирования таких мультипроцессорных систем обеспечивает ряд современных ОС (Windows NT, Novell NetWare и другие). Чаще всего симметричные мультипроцессорные системы содержат два или четыре процессора.

Характерным примером является система прерываний, реализованная в процессорах фирмы Intel. Так, например, процессоры семейства P6(PentiumII, PentiumIII, Celeron и др.) имеют ряд средств для поддержки работы

мультипроцессорных систем, обеспечивая для процессоров взаимный доступ к содержимому внутренней кэш-памяти данных (снупинг), возможность блокировки доступа к шине при выполнении ряда процедур и другие возможности. Различные модели этого семейства позволяют организовать эффективную работу двух- или четырехпроцессорных систем.

Одной из наиболее серьезных проблем при реализации мультипроцессорных систем является организация обслуживания внешних (аппаратных) прерываний. Классическая организация обслуживания с помощью контроллера прерываний, подающего сигнал запроса INTR и формирующего код команды INT n, с реализацией процессором цикла подтверждения прерывания ориентирована на использование в однопроцессорной системе. Для обеспечения функционирования мультипроцессорных систем в процессоры семейства P6 введен программируемый контроллер прерываний с расширенными возможностями APIC (APIC – Advanced Programmable Interrupt Controller)

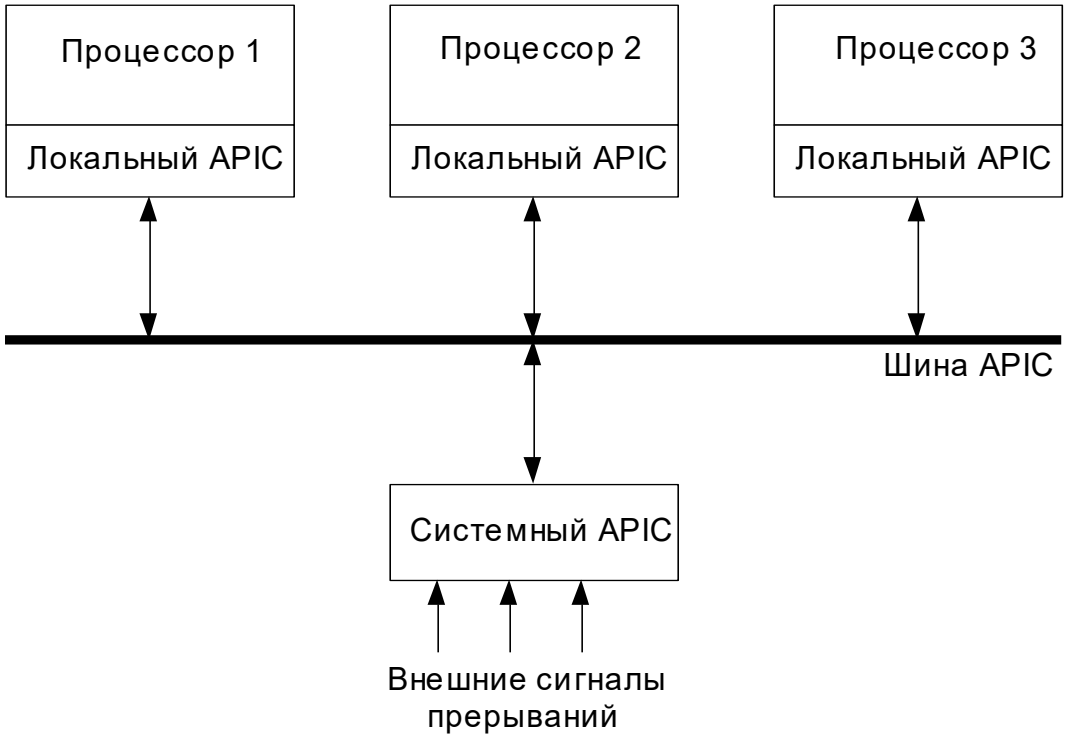


Рис.2.12. Схема прерываний в многопроцессорной системе.

Внутренние контроллеры прерываний связаны между собой по специальной APIC-шине (рис.2.12). Общие внешние запросы прерываний поступают на системный APIC - контроллер, который реализован в виде отдельной микросхемы, разработанной и поставляемой компанией Intel. Каждый из процессоров содержит локальный APIC, имеющий две входных лини LINT 0, LINT 1, на которые поступают локальные запросы прерывания, обслуживаемые только данным процессором. При работе в однопроцессорной системе APIC отключается, и выводы LINT 1-0 используются для подачи запросов немаскируемого NMI и маскируемого INTR прерываний.

Общие запросы прерывания поступают на системный APIC, который после их анализа выдает соответствующие послания на внутреннюю APIC - шину. Эта шина содержит три линии, на одну из которых (PICCLK) выдается синхросигнал, а две других (PICD 1-0) служат для последовательного обмена информацией в процессе организации обслуживания поступивших запросов. При этом для внешних устройств, формирующих запросы прерывания, мультипроцессорная система выглядит как один процессор, а процедура обслуживания запросов соответствует процедуре, выполняемой серийным контроллером прерываний Intel 8259A, который широко используется в современных системах.

2.7. Организация межпроцессорного обмена в системах с разделяемой памятью и общей шиной

Рассмотрим принципы организации обмена между вычислительными модулями на примере многопроцессорной системы с разделяемой памятью, использующей в качестве коммуникационной среды общую шину.

2.7.1. Структура системы с разделяемой памятью

Структура многопроцессорной системы с общей шиной и разделяемой (общей) памятью (рис.2.13) содержит n вычислительных модулей $ВМ_1,...,ВМ_n$, m модулей оперативной памяти $МОП_1,...,МОП_m$, работающих под управлением контроллера. В составе вычислительных модулей находится центральный процессор (ЦП) и шинный интерфейс, сопрягающий его с общей шиной.

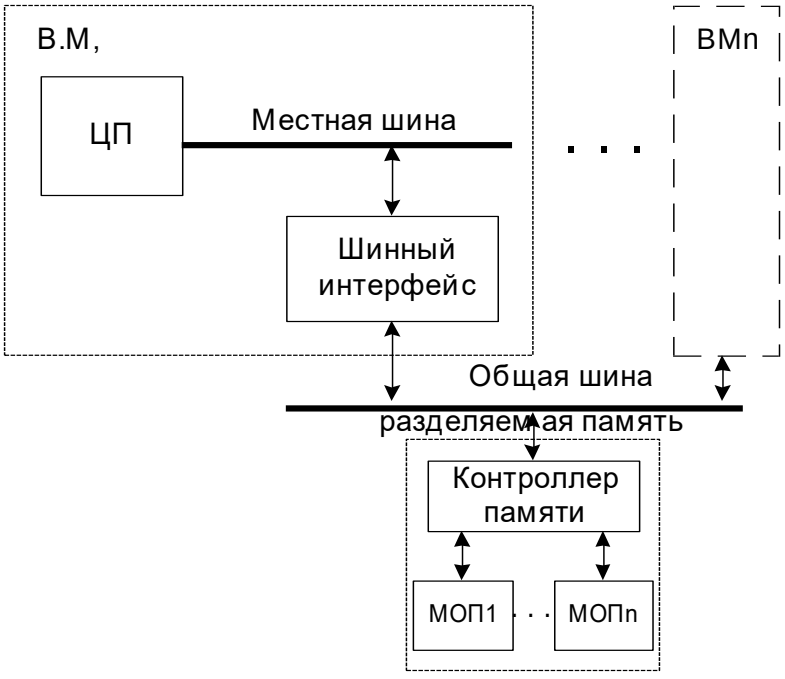


Рис.2.13. Структура системы с общей шиной и разделяемой памятью

Функции шинного интерфейса следующие:

- управление занятием/освобождением общей шины;
- управление обменом между вычислительным модулем и общей памятью;
- распознавание собственного адреса, выставленного на общую шину контроллером разделяемой памяти или другим устройством;
- организация внешних прерываний от запросов из системного контролера прерываний;
- буферизация передаваемых данных.

Каждому вычислительному модулю присваивают уникальный номер, который будем называть собственным адресом. Протоколом обмена предусматривается распознавание собственного адреса, когда сообщение передаётся извне в данный вычислительный модуль. Для этого шинный интерфейс постоянно отслеживает код присвоенного ему адреса. Как только он обнаружит собственный адрес, производится коммутация вычислительного модуля на общую шину и осуществляется приём данных с ОШ в приемный буфер интерфейса или в процессор.

Будем считать, что обмен процессор - память может производиться пословно или группами слов. Если применяется обмен без буферизации (в шинном интерфейсе и контроллере памяти отсутствуют буферы данных), то используется обычный машинный цикл (его ещё называют циклом шины) чтения или записи в память. По окончании цикла шины память освобождается и может быть передана в распоряжение другому вычислительному модулю.

Время обмена составит $t_B = \tau + t_{MC}$, где τ - время занятия общей шины; $t_{мс}$ - машинный цикл процессора, связанный с обращением к памяти. Время занятия τ зависит от способа управления общей шиной. Цикл обращения к памяти, в свою очередь, зависит от типа процессора и не зависит от быстродействия модуля памяти.

Из этого следует, что при способе обмена без буферизации сообщений в системе можно иметь только один модуль общей памяти, т.к. параллельная работа нескольких модулей невозможна. Это обстоятельство не позволяет достичь высокой степени параллельности выполнения работ из-за частых конфликтов, которые будут возникать между вычислительными модулями в борьбе за доступ к общей памяти. В связи с этим потенциальная производительность таких систем низкая.

Чтобы обеспечить параллельность работы модулей памяти, необходимо значительно увеличить пропускную способность общей шины и общей(разделяемой) памяти.

Первое достигается за счёт сокращения цикла шины, связанного с записью (считыванием) данных в память. Для этого передаваемые сообщения

буферизуются в быстрых регистрах шинного интерфейса. В режиме записи процессор инициирует обмен, передав в буферную память шинного интерфейса адрес ячейки памяти, управляющую информацию и данные. Контроллер шинного интерфейса самостоятельно, после получения доступа к общей шине, быстро передаёт всю информацию в буфер общей памяти. Для этих целей в контроллер разделяемой памяти так же как и в устройство шинного интерфейса вычислительного модуля включают быстродействующий буфер небольшого объёма. В этом случае цикл шины может быть значительно меньше цикла памяти. Следовательно, в течение цикла памяти возможна передача нескольких слов по общей шине. Поэтому число модулей памяти может быть увеличено во столько раз, во сколько цикл памяти больше цикла шины. Такой способ обмена называют с буферизацией передаваемых данных.

Для повышения пропускной способности разделяемой памяти применяют её расслоение на ряд независимых модулей МОП1-МОПn (см. рис.3.9) с использованием чередования адресов.

Время записи составит $t_W = \tau + t_{BF}$, где t_{BF} - время обращения к буферному регистру.

При чтении данных процессором из разделяемой памяти в режиме с буферизацией потребуется уже два, но таких же коротких, как и при записи, цикла шины. Вычислительный модуль, требующий данные из общей памяти, выставляет собственный адрес, адрес модуля памяти, адрес ячейки памяти внутри модуля и необходимые управляющие сигналы. Эта информация запоминается в быстродействующей буферной памяти шинного интерфейса. Контроллер шинного интерфейса самостоятельно организует их отправку адресату, т.е. контроллеру разделяемой памяти. Контроллер производит выборку операнда из адресуемого модуля памяти и в свою очередь формирует кадр обмена с вычислительным модулем-адресантом. В него включают адрес вычислительного модуля- адресанта(берут его из сообщения, которое было направлено в память) и прочитанные из памяти данные .

Время, необходимое для чтения данных из общей памяти, составит $t_R = 2(\tau + t_{BF})$. Среднее время обмена вычислительного модуля за время выполнения программы, ели в ней содержится h команд записи и g команд чтения, определится выражением

$$t_{обм} = \frac{ht_W + gt_R}{h+g}.$$

Подставив в (3.25) выражения для t_W и t_R, получим:

$$t_B = \tau + t_{BF} + \frac{g(\tau + t_{RF})}{h+g}.$$

При таком способе обмена совмещается работа процессора, памяти и интерфейсного оборудования, улучшаются возможности масштабирования и, как следствие, может быть увеличена общая производительность системы.

Пропускную способность общей шины может варьироваться, поскольку она зависит от ширины шины, которая определяется величиной, обратной числу циклов шины, необходимых для передачи одного слова (байта) данных.

Если обозначить ширину шины буквой *b*, то при *b*=1 число линий в шине равно числу разрядов в информационной части передаваемого сообщения. Для небольших многопроцессорных систем, как правило, *b<1*. В современных крупных многопроцессорных системах b>>1, где разрядность только системной шины данных составляет 128 или более разрядов.

От ширины шины зависят характеристики межмодульного интерфейса. Чем меньше ширина шины, тем меньше затраты на изготовление интерфейсного оборудования, однако ниже пропускная способность, и следовательно, потенциальная производительность и масштабируемость системы.

2.7.2 Структура системы с разделяемой распределённой памятью

Структура системы представлена на рис.2.14 и содержит в своём составе процессор, локальную память, подключённую на местную шину и шинный интерфейс

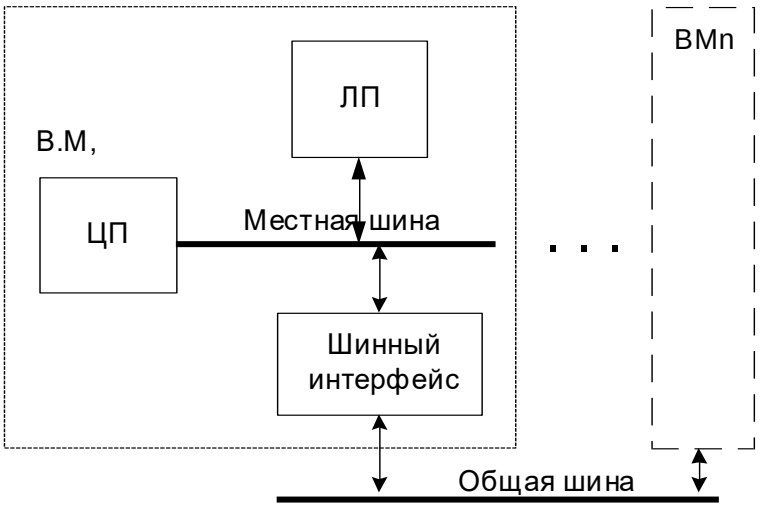


Рис.2.14. Структура системы с разделяемой распределённой памятью

Функции шинного интерфейса зависят от способа организации управления системой и могут включать:

- организацию доступа к памяти удаленного вычислительного модуля;
- управление занятием/освобождением общей шины;
- организацию внешних прерываний;

- управление локальной памятью при доступе к ней со стороны удалённого вычислительного модуля;
- распознавание собственного адреса, выставленного на общую шину удалённым вычислительным модулем;
- буферизацию передаваемых данных.

Коммутация адреса, данных и управляющей информации может производиться с буферизацией или без неё, в зависимости от принятого способа связи между вычислительными модулями. Коммутаторы шинного интерфейса должны обеспечивать режим высокого выходного сопротивления для отключения вычислительного модуля от общей шины на период местных обращений процессора в локальную память. Освобождённая ОШ может быть предоставлена для осуществления связи другим вычислительным модулям.

Обращения вычислительных модулей в удалённую память может производится либо прямым доступом, либо некоторым другим способом, например, по прерыванию. Для организации обращения в удалённую память вычислительный модуль-источник выставляет на общую шину адрес вычислительного модуля - приёмника, адрес ячейки локальной памяти удалённого вычислительного модуля, данные и управляющую информацию, т.е. формирует кадр обмена. Шинный интерфейс вычислительного модуля - приёмника, обнаружив собственный адрес, вырабатывает сигнал ЗАПРОС ПРЯМОГО ДОСТУПА или ЗАПРОС ПРЕРЫВАНИЯ, с помощью которого приостанавливается работа местного процессора на период времени обращения к его локальной памяти внешнего процессора. По сигналу ПОДТВЕРЖДЕНИЕ ДОСТУПА шинный интерфейс производит коммутацию общей шины на местную. По окончании обмена вычислительный модуль - приёмник возвращается в исходное состояние, которое предшествовало внешнему обращению. Такой способ организации вычислительных модулей возможно использование способов связи как с буферизацией, так и без буферизации данных. В обоих случаях могут возникать конфликты за доступ к разделяемой памяти при одновременном запросе её со стороны удалённого и местного процессоров. Разрешение конфликтных ситуаций может производится схемой приоритетов, причём высший приоритет должны иметь запросы, поступающие из общей шины, чтобы не загружать её дополнительной передачей информации о состоянии памяти удалённого вычислительного модуля. Это требование связано с обеспечением высокой пропускной способности общей шины, являющейся «узким местом» системы.

Чтобы минимизировать конфликты за доступ к локальной памяти со стороны местного и удалёного процессоров, её выполняют с расслоением. Ещё большей минимизации конфликтов достигают применением в вычислительных модулях кэш – памяти разных уровней. На рисунке 2.15 представлен вариант построения вычислительного модуля с кэш-памятью.

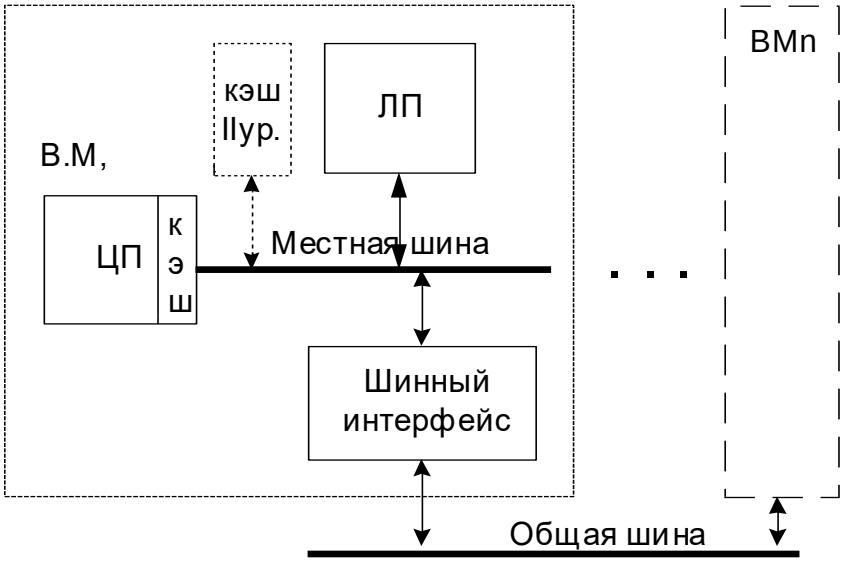


Рис.2.15. Структура вычислительного модуля с кэш-памятью для систем с разделяемой распределённой памятью

Поскольку, в соответствии с принципами временной и пространственной локальности, основная доля обращений при обработке программ вычислительным модулем будет приходиться на кэш-память, то загрузка локальной памяти сокращается. Отсюда следует уменьшение препятствий при доступе к локальной памяти со стороны удалённых вычислительных модулей. Этот же подход применим и к структурам с разделяемой памятью (рис.2.16).

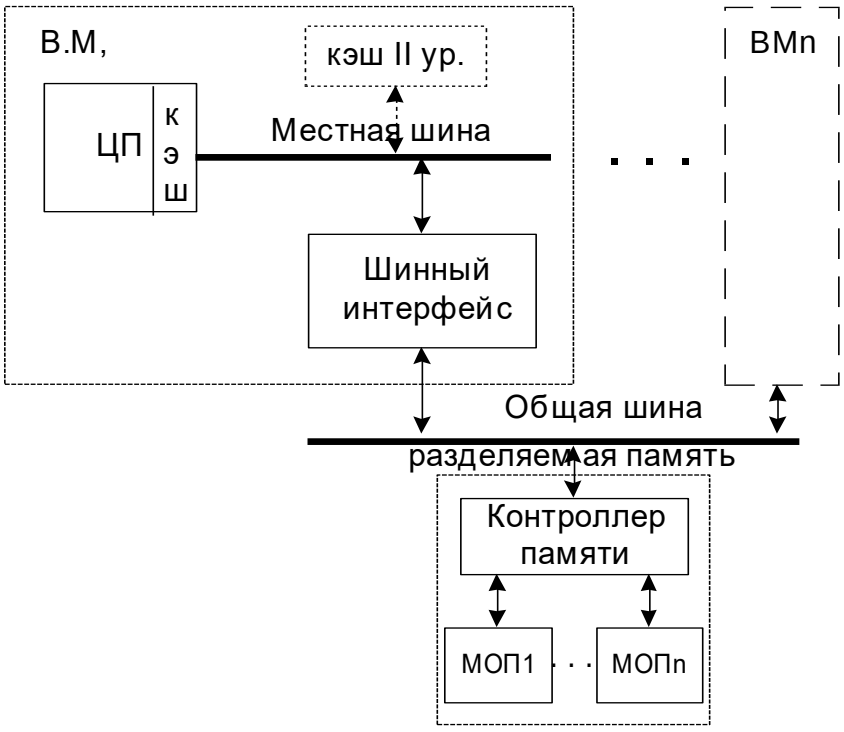


Рис. 2.16. Структура вычислительного модуля с кэш-памятью для систем с разделяемой памятью

Мультипроцессорные операционные системы

Перейдем теперь от обсуждения аппаратного обеспечения мультипроцессоров к рассмотрению программного обеспечения, в частности к обсуждению мультипроцессорных операционных систем. Возможны различные варианты организации данных систем. Ниже будут рассмотрены три из них.

1) Операционные системы с раздельным выполнением заданий

Простейший способ организации мультипроцессорных операционных систем состоит в том, чтобы статически разделить оперативную память по числу центральных процессоров и дать каждому центральному процессору свою собственную память с собственной копией операционной системы. В результате *n* центральных процессоров будут работать как *n* независимых компьютеров. В качестве очевидного варианта оптимизации можно позволить всем центральным процессорам совместно использовать код операционной системы и хранить только индивидуальные копии данных (рис. 8.7). Квадратики, помеченные словом Data, означают приватные данные операционной системы для каждого центрального процессора.

Рис.8.7. Разделение памяти мультипроцессора между четырьмя центральными процессорами с общей копией кода операционной системы

Тем не менее такая схема лучше, чем *n* независимых компьютеров, так как она позволяет всем машинам совместно использовать набор дисков и других устройств ввода-вывода, а также обеспечивает гибкое совместное использование памяти. Например, если требуется запустить большую программу, одному из центральных процессоров может быть выделена большая порция памяти на время выполнения этой программы. Кроме того,

процессы могут эффективно общаться друг с другом, если одному процессу будет позволено писать данные в память, а другой процесс будет их считывать в этом месте. Но с точки зрения операционной системы наличие операционной системы у каждого центрального процессора является крайне примитивным подходом.

Следует отметить четыре аспекта данной схемы, возможно, не являющихся очевидными. Во-первых, когда процесс обращается к системному вызову, системный вызов перехватывается и обрабатывается его собственным центральным процессором при помощи структур данных в таблицах операционной системы.

Во-вторых, поскольку у каждой операционной системы есть свои собственные таблицы, у нее есть также и свой набор процессов, которые она сама планирует совместного использования процессов нет. Если пользователь регистрируется на центральном процессоре 1, то все его процессы работают на центральном процессоре 1. В результате может случиться так, что центральный процессор 1 окажется загружен работой, тогда как центральный процессор 2 будет простаивать.

В-третьих, совместного использования страниц также нет. Может случиться так, что у центрального процессора 2 много свободных страниц, в то время как цен- тральный процессор 1 будет постоянно заниматься свопингом. И нет никакого способа занять свободные страницы у соседнего процессора, так как выделение памяти статически фиксировано.

В-четвертых, и это хуже всего, если операционная система поддерживает буферный кэш недавно использованных дисковых блоков, то каждая операционная система будет выполнять это независимо от остальных. Таким образом, может случиться так, что некоторый блок диска будет присутствовать в нескольких буферах одновременно, причем в нескольких буферах сразу он может оказаться модифицированным, что приведет к порче данных на диске. Единственный способ избежать этого заключается в полном отказе от блочного кэша, что значительно снизит производительность системы.

Мультипроцессоры типа «ведущий-ведомый»

По причине приведенных выше соображений такая модель теперь используется редко, хотя она применялась на заре эпохи мультипроцессоров, когда ставилась цель просто перенести существующие операционные системы на какой-либо новый мультипроцессор как можно быстрее. Вторая модель показана на рис. 8.8. Здесь используется всего одна копия операционной системы, находящаяся на центральном процессоре 1 и отсутствующая на других центральных процессорах, системные вызовы перенаправляются для обработки на центральный процессор Центральный процессор 1 может также выполнять процессы пользователя, если у него будет оставаться для этого время. Такая схема называется «хозяин-подчиненный», так как центральный процессор 1 является - хозяином», то есть ведущим, а все остальные процессоры — подчиненными, или ведомыми.

Рис.8.8. Модель мультипроцессора «хозяин-подчинённый»

Модель мультипроцессора «хозяин-подчиненный» позволяет решить большинство проблем первой модели. В этой модели используется единая структура данных (например, один общий список или набор приоритетных списков), учитывающая готовые процессы. Когда центральный процессор переходит в состояние простоя, он запрашивает у операционной системы процесс, который можно обрабатывать, и при наличии готовых процессов операционная система назначает этому процессору процесс. Поэтому при такой организации никогда не может служиться так, что один центральный процессор будет простаивать, в то время как другой центральный процессор перегружен. Страницы памяти могут динамичес**ки** предоставляться всем процессам. Кроме того, в такой системе есть всего один «общий буферный кэш блочных устройств, поэтому дискам не грозит порча данных, как в предыдущей модели при попытке использования блочного кэша. Недостаток этой модели состоит в том, что при большом количестве центральных процессоров хозяин может стать узким местом системы. Ведь ему приходится обрабатывать все системные вызовы от всех центральных процессоров. Например, если обработка системных вызовов занимает 10 % времени, тогда 10 центральных .процессоров завалят хозяина работой, а при 20 центральных процессорах хозяин уже не будет успевать их обрабатывать, и система начнет простаивать. Следовательно, такая модель проста и работоспособна для небольших мультипроцессоров, но на больших она работать не может.

Операционные системы с симметричной обработкой

Третья модель, представляющая собой симметричные мультипроцессоры (SMP, Symmetric Multiprocessor), позволяет устранить перекос предыдущей модели. Как и в предыдущей схеме, в памяти находится всего одна копия операционной системы, но выполнять ее может любой процессор. При системном вызове на центральном процессоре, обратившемся к системе с системным вызовом, происходит прерывание с переходом в режим ядра и обработкой системного вызова. Модель симметричного мультипроцессора показана на рис. 8.9. Эта модель обеспечивает динамический баланс процессов и памяти, поскольку имеется всего один набор таблиц операционной системы. Она также позволяет

избежать простоя системы, связанного с перегрузкой ведущего центрального процессора («хозяина»), так как в ней нет ведущего центрального процессора и все же данная модель имеет собственные проблемы. В частности, если код операционной системы будет выполняться одновременно на двух или более центральных процессорах, произойдет катастрофа. Представьте себе два центральных процессора, одновременно берущих один и тот же процесс для запуска или запрашивающих одну и ту же свободную страницу памяти. Простейший способ разрешения подобных проблем заключается в связывании мьютекса (то есть блокировки) с операционной системой, в результате чего вся система превращается в одну большую критическую область. Когда центральный процессор хочет выполнять код операционной системы, он должен сначала получить мьютекс. Если мьютекс; блокирован, процессор вынужден ждать. Таким образом, любой центральный процессор может выполнить код операционной системы, но в каждый момент в ни только один из них будет делать это.

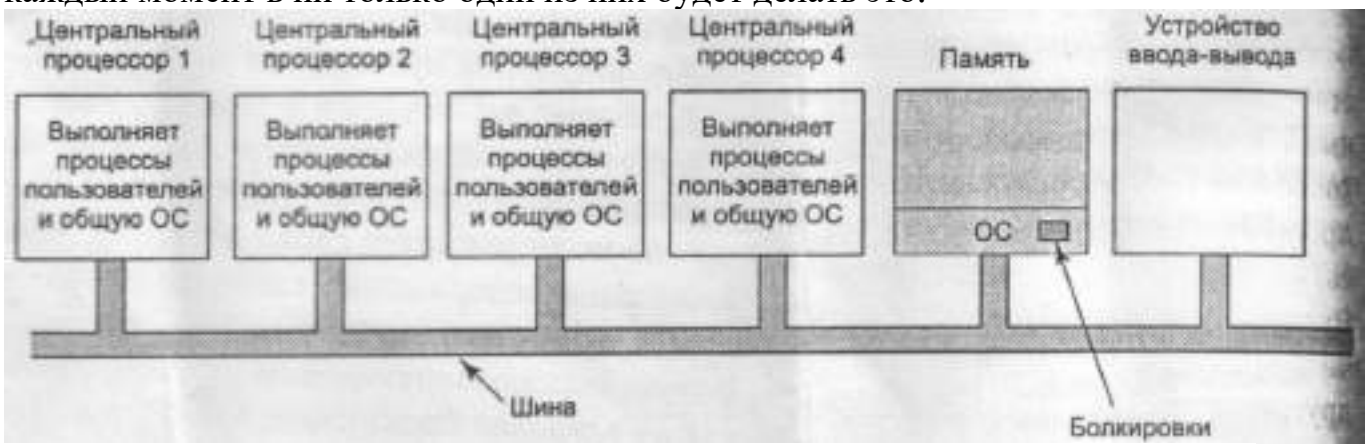


Рис.8.9. Модель симметричного мультипроцессора

Такая модель работает, но она практически так же плоха, как *и* модель «хозяин-подчиненный». Опять же предположим, что 10 % всего процессорного времени расходуется на выполнение самой операционной системы. При 20 центральных процессорах большинство процессоров будут подолгу стоять в длинных очередях, ожидая разрешения доступа к мьютексу. К счастью, такую ситуацию легко исправить. Многие части операционной системы независимы друг от друга. Например, один центральный процессор может заниматься планированием, в то время как другой центральный процессор будет выполнять обращение к файловой системе, а третий обрабатывать страничное прерывание.

Такое наблюдение приводит к расщеплению операционной системы на независимые критические области, не взаимодействующие друг с другом. У критической области есть свой мьютекс, поэтом только один центральный процессор может выполнять ее в каждый момент времени. Таким образом можно достичь большей степени распараллеливания. Однако может случиться, что некоторые таблицы, например таблица процессов, используются в нескольких критических областях. Так, таблица процессов требуется для планирования, но также для выполнения системного вызова

fork и для обработки сигналов. Для каждой таблицы, использующейся несколькими критическими областями, требуется свой собственный мьютекс. Итак, в каждый момент времени каждая критическая область может выполняться только одним центральным процессором, а также к каждой критической таблице может быть предоставлен доступ только полному центральному процессору.

Подобная организация используется в большинстве современных мультипроцессоров. Сложность написания операционной системы для такой машины заключается не в том, что программный код сильно отличается от обычной операционной системы. Нет. Самым сложным является расщепление операционной системы на критические области, которые могут выполняться параллельно на разных центральных процессорах, не мешая друг другу даже косвенно. Кроме того, каждая таблица, используемая двумя и более критическими областями, должна быть отдельно защищена мьютексом, а все программы, пользующиеся этой таблицей, должны корректно использовать мьютекс.

Кроме того, следует уделить особое внимание вопросу избежания взаимоблокировок. Если двум критическим областям одновременно потребуются таблица *A* и таблица *B* и они затребуют эти таблицы в разном порядке, то рано или поздно возникнет взаимоблокировка, причину появления которой будет очень трудно определить. Теоретически всем таблицам можно поставить в соответствие целые числа и потребовать от всех критических областей запрашивать эти таблицы по порядку номеров. Такая стратегия позволяет избежать взаимоблокировок, но она требует от программиста детального исследования того, какие таблицы потребуются каждой критической области, чтобы запросить их в правильном порядке.

При изменении программы со временем критической области может потребоваться новая таблица, которая не была нужна ранее. Если изменением программы занимается новый программист, не понимающий всей логики системы, он может поддаться искушению просто захватить мьютекс в тот момент, когда требуется таблица, и отпустить его, когда таблица более не нужна. Однако именно такие простые и кажущиеся разумными действия могут привести к взаимоблокировке, что с точки зрения пользователя выглядит как зависание системы. Очень не просто грамотно спроектировать систему, но поддерживать ее в правильном состоянии в течение нескольких лет и при этом совершенствовать систему меняющимся штатом программистов крайне трудно.

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ ИНСТРУМЕНТЫ РАЗВЕРТЫВАНИЯ И ПОДДЕРЖАНИЯ ЖИЗНЕННОГО ЦИКЛА ИНТЕЛЛЕКТУАЛЬНЫХ МОДЕЛЕЙ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

Содержание

Лабораторная работа №1

«Знакомство с Docker»

1.1. Цель работы

Цель работы: Изучение контейнеризации приложений на примере Docker. Изучение основных возможностей Docker.

1.2. Теоретический материал

Контейнеризация (виртуализация на уровне операционной системы, контейнерная виртуализация, зонная виртуализация) — метод виртуализации, при котором ядро операционной системы поддерживает несколько изолированных экземпляров пространства пользователя вместо одного. Эти экземпляры (обычно называемые контейнерами) с точки зрения пользователя полностью идентичны отдельному экземпляру операционной системы. Ядро обеспечивает полную изолированность контейнеров, поэтому программы из разных контейнеров не могут воздействовать друг на друга.

Контейнеризация стала стандартной практикой для компаний, занимающихся разработкой и сопровождением программных продуктов так как обладает рядом преимуществ, благодаря которым облегчается задача масштабирования, доставки продукта, обеспечения безопасного запуска приложений и развёртывания экземпляров приложений на одной машине, требующих разные версии одних и тех же библиотек и конфигурационных файлов (ад зависимостей). Преимущества контейнеризации представлены далее:

— *Изолированность*. Несмотря на совместное использование ядра, приложение в контейнере исполняется изолированно от других частей системы и приложений, при этом приложения вне контейнера так же не оказывают влияние на содержимое контейнера. Конфигурационные файлы и все

необходимые библиотеки хранятся внутри контейнера и позволяет на одной машине запустить приложения с разной версией библиотек.

— *Легковесность.* Данную характеристику стоит ставить в сравнение с образами виртуальных машин с предустановленной ОС. В отличии от образов виртуальных машин, контейнер потребляет гораздо меньше ресурсов физической машины-хоста. Так же файл контейнера имеет меньше размер, чем образ виртуальной машины с тем же приложением, благодаря чему облегчается передача приложения клиенту.

— *Переносимость*. Является следствием свойств «Легковесность» и «Изолированность». Так как контейнер сам по себе имеет относительно небольшой размер, а также необходимые компоненты для запуска приложения находятся внутри контейнера, то перенос контейнера с машины на машину не составляют особого труда и каких-либо подготовок кроме установки соответствующего ПО для управления контейнером.

— *Автоматизация.* Использование контейнеров позволяет строить полностью автоматизированные цепочки непрерывной интеграции-развёртывания приложений (CI/CD), в которых «ручной» частью будет в основном написание кода. Сборка приложение, тестирование, сборка контейнера, доставка контейнера до целевой машины клиента, запуск приложения может быть возложена на соответствующие программные комплексы CI/CD.

Docker — программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации. Позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть перенесён на любую Linux-систему с поддержкой cgroups в ядре, а также предоставляет среду по управлению контейнерами. Существуют версии Docker для Windows 10, однако на момент составления теоретического материала, данные продукты имеют не

достаточный уровень стабильности. По этой причине в теоретическом материале описывается принцип работы Docker на ОС семейства Linux.

Docker вводит свои термины, необходимые для понимания материала:

— Docker platform (Docker-платформа) - программа, обеспечивающая возможность упаковки и запуска приложения в контейнере на любом Linux сервере. Она собирает код и зависимости. Благодаря хорошей мобильности и воспроизводимости это упрощает масштабирование.

— Docker Engine (Docker-движок) - это приложение, которое следует архитектуре клиент-сервер. Он установлен на хост-машине.

— Docker-клиент — основной способ взаимодействия с Docker'ом. При использовании Docker Command Line Interface (CLI) вы просто вводите в терминал нужную команду, которая обычно начинается со слова docker. Затем Docker-клиент использует Docker API для отправки команды на Docker Daemon.

— Docker Daemon — Docker-сервер, отвечающий на Docker API запросы. Он управляет образами, контейнерами, Docker-сетями и тома Docker.

— Docker тома (Volumes) — предпочтительный механизм для хранения используемых и генерируемых вашим приложением данных.

— Docker-реестр — удалённое место, где содержатся все Docker-образы. Есть возможность загружать туда образы или, наоборот, скачиваете их оттуда. Можно использовать как свой собственный реестр, так и реестр любого провайдера. Наиболее часто используемый является Docker Hub.

— Docker-репозиторий — собрание Docker-образов с одинаковыми названиями, но с разными тегами. Тег — это идентификатор Docker-образа. Обычно в репозитории содержатся разные версии одного и того же образа. В реестр можно загружать как целый репозиторий, так и один образ.

— Docker-image (Docker-образ) - является шаблоном только для чтения с набором инструкций для создания контейнера. К примеру, образ

может состоять из операционной системы Ubuntu, веб-сервера Apache и вашего веб-приложения. Вы можете собрать образ с нуля, обновить или загрузить и использовать образы, созданные другими пользователями. Образ может быть самодостаточным или являться дополнением другого образа. Образ Docker описывается в специальном текстовом файле с именем Dockerfile.

— Dockerfile (Docker-файл) — инструкции для Docker по настройке и запуску приложений. В Docker-файле находится описание базового образа, на котором построен контейнер. С помощью дополнительных слоёв в Docker-файле можно добавить необходимое ПО.

— Контейнер — это компонента работы. Контейнер создается из Docker-образа при запуске командой «docker run image_name». Контейнеры похожи на директории. В контейнерах содержится все, что нужно для работы приложения. Каждый контейнер создается из образа. Контейнеры могут быть созданы, запущены, остановлены, перенесены или удалены. Каждый контейнер изолирован и является безопасной платформой для приложения.

При вызове команды «docker build» запускает процедура сборки образа. Во время сборки будет использован Dockerfile с указанием основных действий при развертывании контейнера, по умолчанию считается, что Докер-файл расположен в рабочей директории, однако можно задать иной путь с помощью ключа -f.

В Докер-файле заносятся специальные команды для постройки образа. Команды записываются в верхнем регистре.

Далее перечислены основные инструкции Докер-файла:

— FROM — задаёт родительский (главный) образ;

— LABEL — добавляет метаданные для образа. Хорошее место для размещения информации об авторе;

— ENV — создаёт переменную окружения;

— RUN — запускает команды, создаёт слой образа. Используется для установки пакетов и библиотек внутри контейнера;

— COPY — копирует файлы и директории в контейнер;

— ADD — делает всё то же, что и инструкция COPY. Но ещё может распаковывать локальные .tar файлы;

— CMD — указывает команду и аргументы для выполнения внутри контейнера. Параметры могут быть переопределены. Использоваться может только одна инструкция CMD;

— WORKDIR — устанавливает рабочую директорию для инструкции CMD и ENTRYPOINT;

— ARG — определяет переменную для передачи Docker'у во время сборки;

— ENTRYPOINT — предоставляет команды и аргументы для выполняющегося контейнера. Суть его несколько отличается от CMD, о чём мы поговорим ниже;

— EXPOSE — открывает порт;

— VOLUME — создаёт точку подключения директории для добавления и хранения постоянных данных.

Инструкции Dockerfile обрабатываются сверху вниз. Каждая инструкция добавляет новый слой в образ и коммитит изменения. Docker исполняет инструкции, следуя процессу:

— Запуск контейнера из образа

— Исполнение инструкции и внесение изменений в контейнер

— Запуск эквивалента docker commit для записи изменений в новый слой образа

— Запуск нового контейнера из нового образа

— Исполнение следующей инструкции в файле и повторение шагов процесса.

Это означает, что если исполнение Dockerfile остановится по какой-то причине (например инструкция не сможет завершиться), то можно использовать образ до этой стадии. Это очень полезно при отладке: есть возможность запустить контейнер из образа интерактивно и узнать, почему инструкция не выполнилась, используя последний созданный образ.

Далее представлены основные команды командной строки для управления Docker и Docker-контейнерами (большинство команд выполнятся только при наличии прав администратора, при использование ОС Linux необходимо вводить префиксную команду sudo) :

— docker search [имя образа] – поиск образа в репозитории;

— docker pull [имя образа] – выгрузить образ из репозитория (при использовании низкоскоростного лимитного подключения стоит учесть, что некоторые образы имеют достаточно большой размер);

— docker push [имя образа] – загрузить образ в репозиторий;

— docker create -t -i [имя образа] --name [имя контейнера] – создать контейнер;

— docker run --name [имя контейнера] -d [имя образа] – создать и запустить контейнер;

— docker rm [имя контейнера] – удалить контейнер;

— docker update [опции] [имя контейнера] – изменить конфигурацию потребляемых ресурсов контейнера;

— docker start [имя контейнера] – запустить построенный контейнер;

— docker stop [имя контейнера] – остановить построенный контейнер;

— docker restart [имя контейнера] – перезапуск контейнера;

— docker pause [имя контейнера] – остановить все процессы внутри контейнера;

— docker unpause [имя контейнера] – снять с паузы ранее приостановленный контейнер;

— docker kill [имя контейнера] – отправка сигнала SIGKILL (используется при зависании контейнера, и команда docker stop/restart перестают работать);

— docker ps – вывести список работающих контейнеров;

— docker logs [имя контейнера] – просмотр журнала контейнера;

— docker events [имя контейнера] – просмотр событий внутри контейнера;

— docker attach [имя контейнера] – подключиться к потоку ввода/вывода/ошибок контейнера;

— docker exec -it [имя контейнера] bash – подключиться и запустить терминал контейнера;

— docker stats [имя контейнера] – просмотр потребляемых контейнером ресурсов;

— docker top [имя контейнера] – просмотр выполняемых процессов внутри контейнера;

— docker port [имя контейнера] – просмотр открытых портов контейнера;

— docker images – просмотреть список доступных образов;

— docker build . – построить образ по параметрам по умолчанию (считается, что DockerFile находится в той же директории, из которой была вызвана команда);

— docker rmi [имя образа] – удалить образ;

1.2.1 Инструкция по установке Docker в ОС Ubuntu.

В случае потери актуальности инструкции стоит перейти по ссылке, указанной в списке источников.

Для установки Docker требуется 64-разрядный дистрибьютив Linux на базе Ubuntu 18.04 или выше.

1) Требуется удалить старые версии docker, если они присутствуют, командой *sudo apt-get remove docker docker-engine docker.io containerd runc*

2) Далее необходимо обновить индексные файлы пакетов командой *sudo apt-get update*

3) Нужно установить пакеты, требуемые для установки и работы docker командой *sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common*

4) Добавить официальный GPG-ключ командой *curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –*

5) Полученный ключ необходимо проверить на валидность, выполнив команду *sudo apt-key fingerprint 0EBFCD88*

После выполнения команды в разделе *pub* должен высветиться опечаток ключа с значением ***9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88***

6) Добавить репозиторий Docker командой

*sudo add-apt-repository *
"deb [arch=amd64] https://download.docker.com/linux/ubuntu
*$(lsb_release -cs) *
stable"

7) Повторно обновить индексные файлы и установить Docker командой *sudo apt-get update && sudo apt-get install docker-ce docker-ce-cli containerd.io*

8) Далее необходимо проверить работу установленного Docker, запустив спец.контейнер командой *sudo docker run hello-world*

Результатом работы является сообщение об удачном запуске контейнера.

1.3. Задание на лабораторную работу

1. Запуск простого контейнера с HTTP-сервером nginx:

1.1. Проверить наличие готового образа на ПК с помощью команды [sudo] docker images, искомый образ имеет имя nginx. В случае отсутствия образа его необходимо выгрузить из репозитория командой [sudo] docker pull nginx.

1.2. Построить и запустить контейнер с именем lb1nginx, перебросив соединение на 8080 порт командой docker run -dit --name lb1nginx -p 8080:80 nginx.

1.3. Удостовериться в успешном запуске HTTP-сервера, перейдя в браузер по адресу localhost:8080. При переходе должна загрузиться окно приветствия nginx.

2. Создание пользовательского образа, наследуемого образа с HTTP-сервером Apache 2.4:

2.1. У вас должен быть Dockerfile, содержащий строки для настройки сервера Apache:

```
FROM httpd:2.4
RUN sed -i \
    -e 's/^Listen 80$/Listen 8085/' \
    conf/httpd.conf && \
  echo "Apache is work" > /usr/local/apache2/htdocs/index.html
EXPOSE 8085
```

Расположите Dockerfile в удобное для вас местоположение и выполните команду [sudo] docker build -t lb1_img [путь к Dockerfile]. После выполнения команды проверьте наличие нового образа командой docker images.

2.2. Постройте и запустите контейнер командой [sudo] docker run -dit --name lb1httpd -p 8085:8085 lb1_img.

2.3. Повторить пункт 1.3, заменив адрес на localhost:8085.

3. Мониторинг ресурсов контейнеров:

3.1. Проверить, что на данный момент работают оба контейнера, иначе запустить недостающий командой [sudo] docker start.

3.2. Посмотреть потребляемые контейнерами ресурсы командой [sudo] docker stats.

4. 1.4. Вопросы для самопроверки

1. Что такое контейнеризация?
2. Какими характеристиками обладает контейнеризация?
3. Что такое Docker?
4. Что такое Docker-образ и Docker-контейнер?
5. Как работает сборка образа через Dockerfile?

1.5. Список использованных источников

1. Милл И., Сейерс Э. Docker на практике //М.: ДМК Пресс. – 2020. –Т.200.

2. Моуэт Э. Использование Docker. //М.: ДМК Пресс. – 2020. – Т.200.

3. Кочер П. Микросервисы и контейнеры Docker //М.: ДМК Пресс. – 2019. – Т.200.

4. Get Docker Engine - Community for Ubuntu [Электронный ресурс]. https://docs.docker.com/install/linux/docker-ce/ubuntu/ (дата обращения: 05.04.2020).

Лабораторная работа №2
«Управление многомодульным проектом при помощи Docker Compose»

2.1. Цель работы

Цель работы: Изучение инструментальных возможностей Docker Compose. Разработка и управление многомодульным проектом с использованием контейнеризации и Docker Compose.

2.2. Теоретический материал

2.2.1. Общие сведения

Большинство WEB-приложений представляют собой связанные друг с другом набор модулей (сервисов). Модули так же могут иметь либо монолитную, либо микросервисную архитектуру. Однако в большинстве случаев WEB-приложений состоит из Front-end и Back-end программного сервера и к ним применима контейнеризация.

Проблема многомодульных проектов заключается в высокой времязатрате развертывания каждого контейнера по отдельности, а также в сложной схеме зависимостей одних контейнеров от других.

Docker Compose — это инструментальное средство, входящее в состав Docker. Оно предназначено для решения задач, связанных с развёртыванием сложных многомодульных проектов. Docker Compose по своему поведению похож на Docker тем лишь исключением, что оперирует не одним контейнером, а группой.

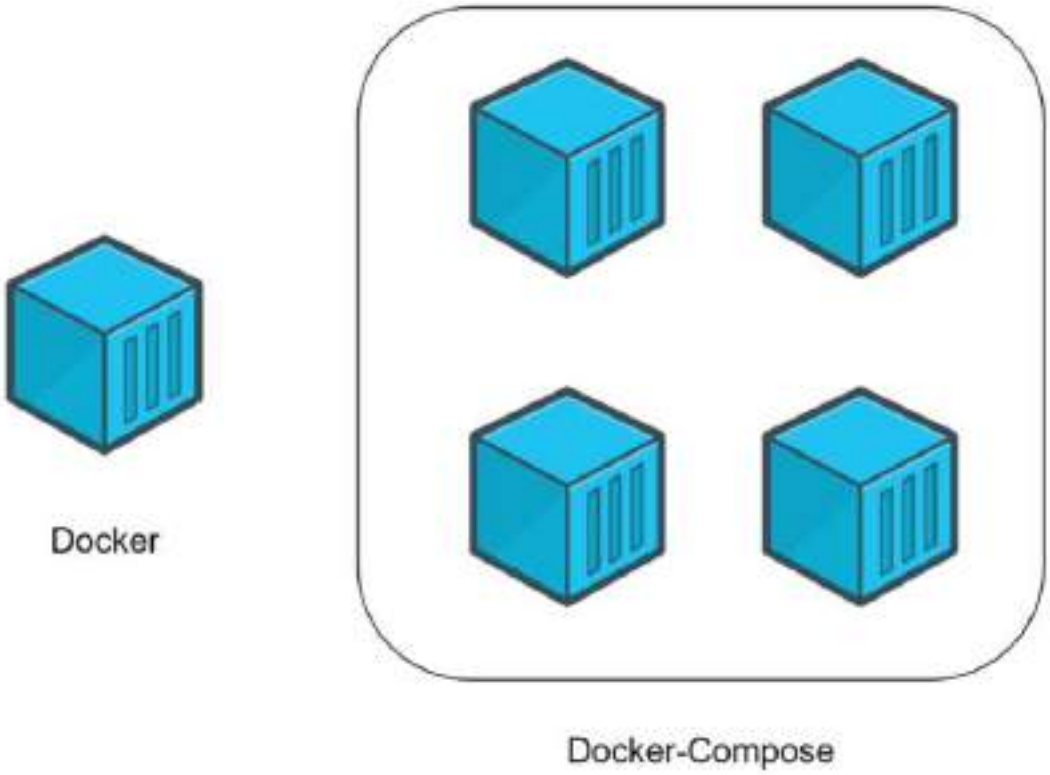


Рисунок 2.1 - Docker (отдельный контейнер) и Docker Compose (несколько контейнеров).

Для управления контейнерами Docker Compose использует файл docker-compose.yml, где описываются все сервисы и их зависимости, а также параметры для «переброса» портов, выполнения команд внутри контейнеров, путь к альтернативным Docker-файлам, аргументы среды и т.д.

Описание docker-compose.yml начинается с указания версии диалекта команд файла. Далее идет ключевое слово services, где перечисляются параметры сервисов (контейнеров), далее опционально может присутствовать networks для конфигурирования Docker-Net и так же опционально конфигурации ограничений на потребление аппаратных ресурсов ЭВМ.

Далее перечислены основные команды Docker Compose:

— docker-compose build – сборка контейнеров по docker-compose.yml;

— docker-compose config – проверка и вывод на экран содержимое docker-compose.yml;

— docker-compose up – запустить построенные контейнеры;

— docker-compose down – остановить и удалить контейнеры;

— docker-compose logs -f [имя сервиса] – просмотр журнала сервиса;

— docker-compose exec [имя сервиса] [команда] – выполнить команду в выбранном контейнере.

Остальные команды Docker Compose очень сильно похожи на команды Docker-CLI.

2.2.2. Создание простейшего приложения.

Далее приведен пример работы простейшего приложения, состоящий из RESTful-сервера на Spring и простого клиента в виде скрипта на Python 3, сервер и клиент буду помещены каждый в свой контейнер.

Клиент должен через GET-запрос получить от сервера сообщение в виде строки текста и вывести ее на экран.

Далее представлено пошаговое выполнение данной задачи:

1. **Создание проекта на Spring**. Для упрощения этой задачи существует сервис Spring Initializr, который сгенерирует первичную структуру проекта и позволит быстро добавить необходимые зависимости проекта. Проект будет использовать Maven для управления сборкой проекта, Java 8, из зависимостей только Spring Web.

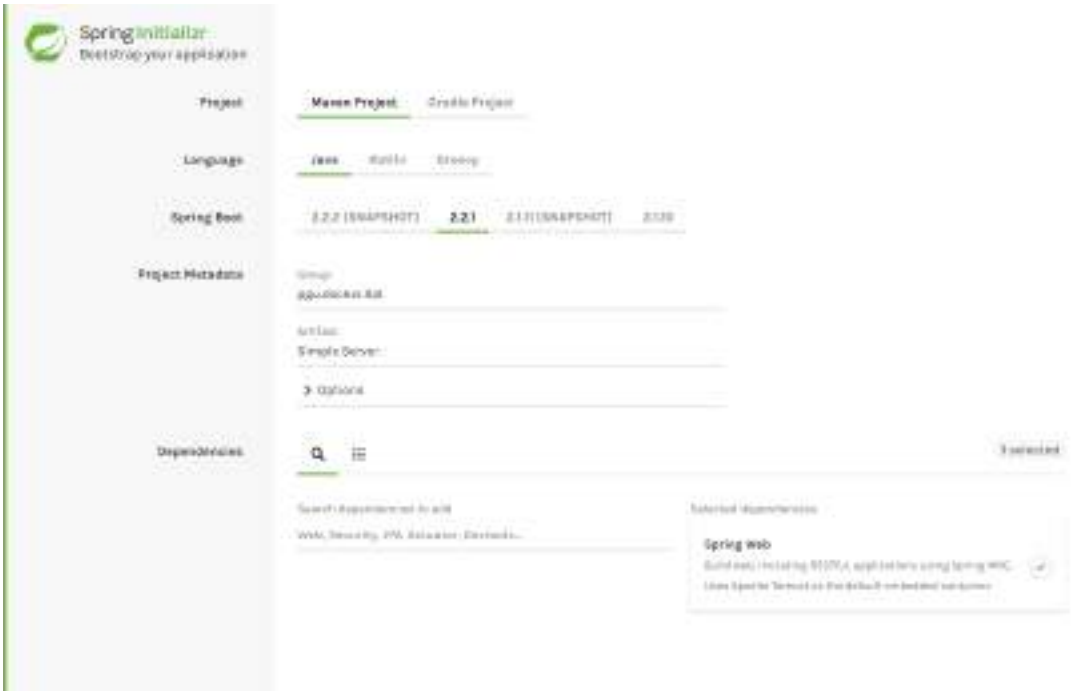


Рисунок 2.2 – Экран Spring Initializr.

2. **Конфигурирование и создание REST-контроллера.** Полученный на первом шаге архив проекта необходимо распаковать в удобную для работы папку**.** Далее необходимо перейти по пути src\main\resources и открыть файл application.properties, где вводим строчки:

server.servlet.context-path=/lb2

server.port=8086

После необходимо создать класс, описывающий работу REST-контроллера. Класс помечается аннотациями @RestController и @RequestMapping("/app") перед своим объявлением. Единственный метод класса помечается аннотацией @GetMapping и возвращает текст "Hello from Spring Simple Server". Проверить работу контроллера можно, запустив сервер командой mvnw spring-boot:run (если в систему установлен Maven, то просто mvn spring-boot:run) и перейдя по адресу http://localhost:8086/lb2/app. В результате в окне браузера должен отобразиться заявленный текст.

3. **Конфигурирование Dockerfile для сервера.** Прежде чем начать конфигурировать Docker-файл, необходимо создать jar-файл с разработанным

сервером, для этого вводится команда mvnw package, файл появится в папке target в корне проекта. В данном примере файл имеет имя server-0.0.1-SNAPSHOT.jar. В корне проекта создается Dockerfile с следующим содержимом:

```
FROM openjdk:8-jdk-alpine
EXPOSE 8086
VOLUME /tmp
ARG JAR_FILE=target/server-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} server-0.0.1-SNAPSHOT.jar
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","server-0.0.1-SNAPSHOT.jar"]
```

4. **Создание клиентского скрипта, конфигурирование Dockerfile.** В той же корневой папке, в которой находится папка с проектом сервера, создается папка для хранения кода клиентского скрипта. Создается файл client.py с следующим содержимом:

```
import urllib.request


fp = urllib.request.urlopen("http://localhost:8086/lb2/app")
content = fp.read()
print(content)
fp.close()
```

Далее в корне проекта клиента создается Dockerfile с тривиальным содержимом:

```
FROM python:3
WORKDIR client/
ADD client/client.py ./
```

5. **Конфигурирование docker-compose.yml, запуск приложения.** В корневой папке 2-х проектов создается файл docker-compose.yml, в котором мы указываем:

— Пути к Dockerfile созданных проектов;

— Для сервера проброс портов;

— Для клиента команду запуска скрипта, установку зависимости от сервера и режим соединения.

Содержимое файла следующее:

```
version: "3"

services:

  server:
    build: srv/lb2serv/
    ports:
      - 8086:8086

  client:
    build: cln/lb2client
    command: python3 client.py
    network_mode: host
    depends_on:
      - server
```

6. **Сборка контейнеров и просмотр результат.** Далее запускаем команду docker-compose build, начнется процедура сборки контейнеров с последующей закачкой необходимых Docker-образов. После успешного

завершения сборки вводится команда docker-compose up, после чего запустятся контейнеры. В терминал будет выводиться выходной поток данных с контейнеров с пометкой названия контейнера-источника. При сборке контейнер с сервером получил имя server_1, когда как контейнер с клиентом получил имя client_1. На рисунке 2.3 видно, что после запуска контейнера server_1, контейнер с клиентом вывел на экран текст, полученный от сервера, после чего скрипт завершился и контейнер клиента остановился.

Рисунок 2.3 – Экран вывода работы 2-х контейнеров.

2.3. Задание на лабораторную работу

Разработать простой клиент-серверное приложение, выполняющее обмен данными между собой. Клиентская программа и программный сервер должны работать внутри контейнеров. Для постройки и запуска контейнеров необходимо использовать Docker Compose.

2.4. Вопросы для самопроверки

1. Для чего используется Docker Compose?
2. Для чего используется docker-compose.yml?

3. Как выглядит структура docker-compose.yml?

2.5. Список использованных источников

1. Милл И., Сейерс Э. Docker на практике //М.: ДМК Пресс. – 2020. – Т.200.

2. Моуэт Э. Использование Docker. //М.: ДМК Пресс. – 2020. – Т.200.

3. Руководство по Docker Compose для начинающих [Электронный ресурс]. URL: https://habr.com/ru/company/ruvds/blog/450312/ (дата обращения: 14.03.2020).

4. Overview of Docker Compose [Электронный ресурс]. https://docs.docker.com/compose/ (дата обращения: 14.03.2020).

Лабораторная работа №3
«Создание кластера Kubernetes»

3.1. Цель работы

Цель работы: Изучение принципов работы Kubernetes. Подготовка машин и настройка кластера Kubernetes.

3.2. Теоретический материал

Kubernetes - открытое программное обеспечение для автоматизации развёртывания, масштабирования контейнеризированных приложений и управления ими. Поддерживает основные технологии контейнеризации, включая Docker, rkt, также возможна поддержка технологий аппаратной виртуализации.

Kubernetes был создан с целью облегчить масштабирование сложно приложения, в том числе на микросервисной архитектуре, с последующей автоматической балансировкой нагрузки.

Как и в Docker, в Kubernetes есть свои термины:

— Node (Узел) — машина в составе кластера Kubernetes.

— Pod — это группа контейнеров с общими разделами, запускаемых как единое целое.

— Controllers — это процесс, который управляет состоянием кластера. Контроллеры управляется Kubernetes Controller Manager. Чаще всего приходится работать с Replication Controller, который отвечает за масштабирование Pod’ов. Так же есть такие контроллеры как DaemonSet Controller, который отвечает за запуск Pod’ов на узле, и Job Controller, задача которого запускать пакетные задания.

— Services — это абстракция, которая определяет логический объединённый набор pod и политику доступа к ним.

— Volumes — это директория, возможно, с данными в ней, которая доступна в контейнере.

— Labels — это пары ключ/значение которые прикрепляются к объектам, например pod'ам. Label'ы могут быть использованы для создания и выбора наборов объектов.

— Kubectl Command Line Interface (kubectl) — интерфейс командной строки для управления Kubernetes.

— Kubelet — программный агент, установленный на узле. Управляет контейнерами и Pod’ами, которые находятся на узле.

— Kube-Proxy — Proxy-сервер, который устанавливается на узле и балансирует нагрузку между контейнерами.

— Etcd — компонент подсистемы управления, отвечающий за согласованное хранение конфигурационных данных кластера, в некотором смысле — распределённый эквивалент каталога /etc Unix-систем. Реализован как легковесная распределённая NoSQL-СУБД класса «ключ — значение»

— Kubernetes API Server — предназначен для того, чтобы быть CRUD сервером со встроенной бизнес-логикой, реализованной в отдельных компонентах или в плагинах. Он, в основном, обрабатывает REST операции, проверяя их и обновляя соответствующие объекты в etcd.

— Scheduler (Планировщик) — компонент подсистемы управления, который выбирает, на каком узле должен выполняться конкретный Pod, опираясь на критерии доступности ресурсов. Планировщик отслеживает использование ресурсов на каждом из узлов, обеспечивая распределение нагрузки так, чтобы она не превышала доступный объём ресурсов.

Архитектурно кластер Kubernetes можно разделить на мастер-узел и на множество управляемых узлов. На мастер-узле находится планировщик, API-сервер и etcd.

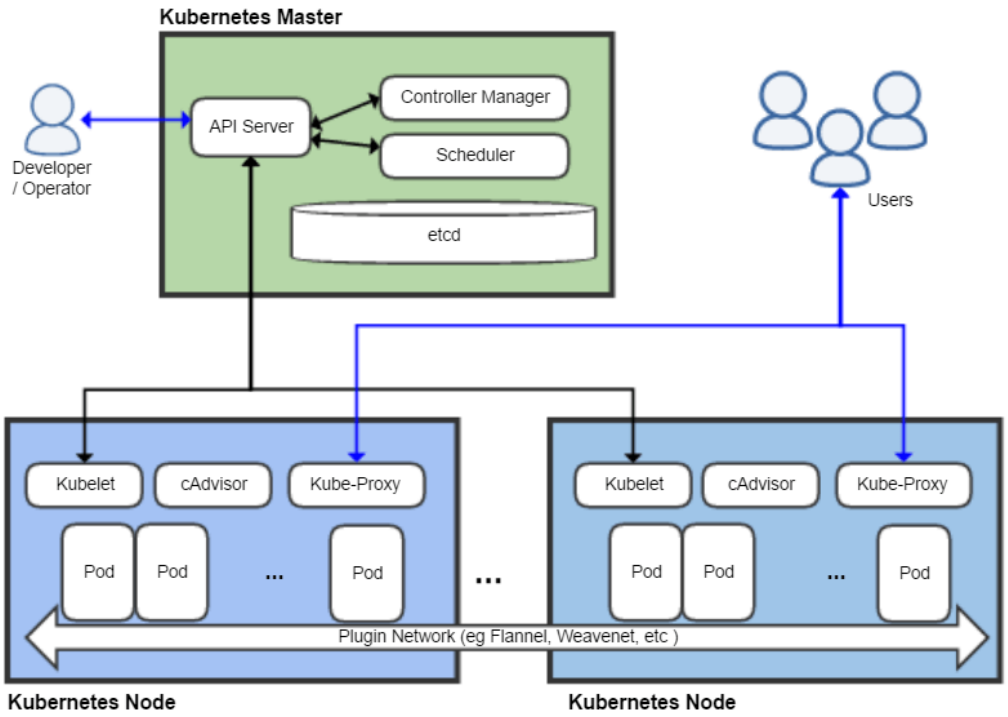


Рисунок 3.1 – Архитектура кластера Kubernetes.

Машины, на которых будет разворачиваться кластер, должны быть на ОС Linux т.к. на Windows могут появиться сложности.

При использовании VirtualBox необходимо сконфигурировать виртуальные машины, объединив их в одну виртуальную локальную сеть. Мастер-узел должен иметь 3-4 ГБ ОЗУ, 2 ядра ЦПУ. Управляемые узлы должны иметь минимум 2 ГБ ОЗУ. При установке ОС Linux необходимо отключить swapping. Если ОС установлено с swapping’ом, то необходимо выполнить команду «swapoff -a» и закомментировать swap-раздел символом ‘#’ в файле /etc/fstab.

3.3. Задание на лабораторную работу

Необходимо создать кластер Kubernetes из 2-х узлов (мастер-узел и управляемый узел), выполнив следующие этапы:

1. Произвести установку и настройку виртуальных машин в среде VirtualBox. Установить на все машины ОС Linux и объединить все машины в одну сеть.

2. Произвести установку Docker и Kubernates на каждом узле. В связи с активным развитием проекта Kubernates, руководство по установки инструментов быстро теряет свою актуальность и может привести к ошибкам. Поэтому желательно смотреть руководство на официальном сайте Kubernates (https://kubernetes.io).

3. На всех узлах должен быть инструмент kubeadm. На мастер-узле нужно запустить команду «kubeadm init --apiserver-advertise-address=[IP-адрес мастер-узла] --pod-network-cidr=192.168.0.0/16», при успешном запуске сервиса проверить командой «kubectl get pods -o wide --all-namespaces», в результате выполнения должны отобразиться ключевые сервисы Kubernates.

4. На управляемом узле выполнить команду «kubeadm join --apiserver-advertise-address==[IP-адрес мастер-узла] --pod-network-cidr=192.168.0.0/16»

3.4. Вопросы для самопроверки

1. Опишите архитектуру кластера Kubernates.
2. Что входит в состав мастер-узла?
3. Что входит в состав управляемого узла?
4. Что такое Kubelet?
5. Как связанны понятие Node и Pod.

3.5. Список использованных источников

1. Сайфан Дж. Осваиваем Kubernetes. Оркестрация контейнерных архитектур //М.: Прогресс книга. – 2019. – Т. 700.

2. Лукша М. Kubernetes в действии. //М.: ДМК Пресс. – 2020. –Т.200.

3. Маркелов А. Введение в технологии контейнеров и Kubernetes //М.: ДМК Пресс. – 2019. –Т.200.

4. Kubernetes Basics [Электронный ресурс]. URL: https://kubernetes.io/docs/tutorials/kubernetes-basics/ (дата обращения: 14.03.2020).

5. Основы Kubernetes [Электронный ресурс]. URL: https://habr.com/ru/post/258443/ (дата обращения: 14.03.2020).

Лабораторная работа №4

«Создание кластера при помощи Rancher v.2»

4.1. Цель работы

Цель работы: Изучение способов развёртывание кластера при помощи Rancher. Конфигурирование кластера при помощи Rancher.

4.2. Теоретический материал

4.2.1. Общие сведения

Rancher – программное обеспечение, признанное облегчить развёртывание и управление кластерами Kubernates при помощи программных компанентов и предоставление пользователю понятного интерфейса. Однако более требователен к ресурсам системы, чем набор инструментов Kubernates. Схема взаимодействия пользователя с кластером при помощи Rancher продемонстрировано на рисунке 3.1.

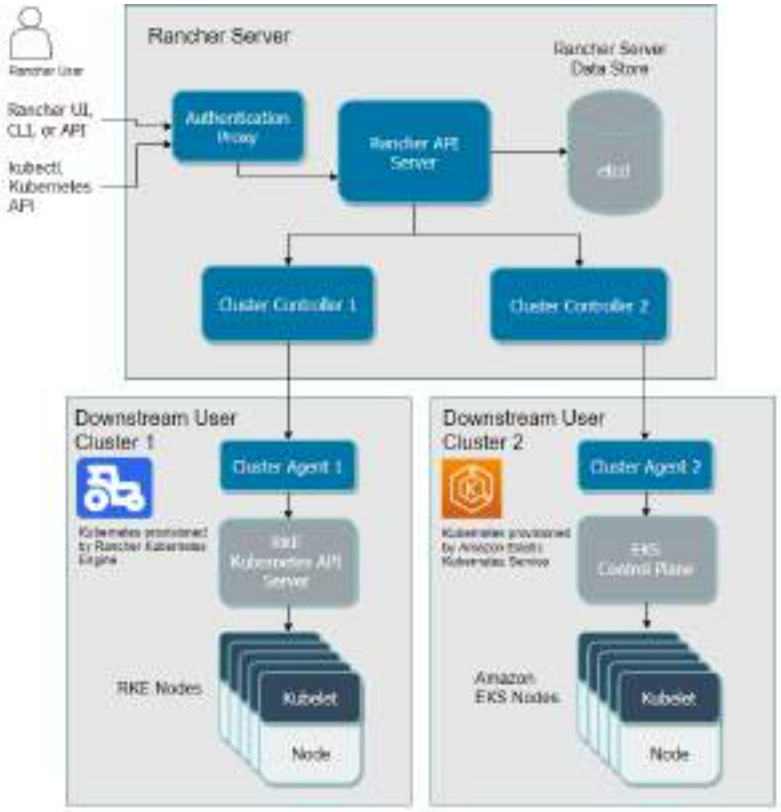


Рисунок 3.1 –Схема взаимодействия пользователя.

В Rancher есть Authentication Proxy, который разграничивает доступ к кластерам Kubernates по привязке учетных записей. По умолчанию Rancher создает файл kubeconfig, содержащий учетные данные для прокси-сервера через центральный сервер Rancher для подключения к API Kubernetes Server в нижестоящем пользовательском кластере. Файл kubeconfig (kube_config_rancher-cluster.yml) содержит полный доступ к кластеру.

Каждый нижестоящий пользовательский кластер имеет агент кластера, который открывает туннель для соответствующего контроллера кластера на сервере Rancher.

Так же существует один контроллер кластера и один агент кластера для каждого нижестоящего кластера. Контроллер кластера нужен, чтобы:

— Отслеживать изменения ресурсов в нижестоящем кластере.

— Переключение состояние контролируемого кластера.

— Настраивать политику контроля доступа к кластерам и проектам.

— Обеспечивать необходимые драйверами машин Docker и Kubernetes- engines, такие как RKE и GKE при работе в облаке.

По умолчанию, чтобы включить Rancher для связи с нижестоящим кластером, контроллер кластера подключается к агенту кластера. Если агент кластера недоступен, контроллер кластера может подключиться к агенту узла.

Агент кластера, также называемый cattle-cluster-agent, является компонентом, который работает в нижестоящем пользовательском кластере. Он выполняет следующие задачи:

— Соединение с Kubernetes API по «Rancher-launched Kubernetes clusters» (Система запуска кластера Kubernetes).

— Управление рабочими нагрузками, созданием и развертыванием модулей в каждом кластере.

— Применение роли и привязки, определенные в глобальных политиках каждого кластера.

— Связь между кластером и сервером Rancher (через туннель до контроллера кластера) о событиях, статистике, информации об узлах и работоспособности.

4.2.2. Пример создание кластера.

Для работы требуется 2 виртуальные машины в среде VirtualBox на базе ОС Linux. На каждой машине должен быть установлен Docker и SSH-сервер.

Далее представлены шаги по развёртыванию кластера и запуску сервера nginx.

1) На машине, выступающей в роли мастер-узла, необходимо скачать запустить образ Rancher командой «docker run -d --restart=unless-stopped -p 80:80 -p 443:443 rancher/server:preview»

2) Открыв браузер, перейти по адресу http://localhost и ввести в окне данные администратора. Окно продемонстрировано на рисунке 3.2

Рисунок 3.2 – Окно приветствия Rancher.

3) В открывшемся окне управления необходимо кликнуть на «Clusters» и добавить новый кластер. Для начала достаточно выбрать имя и нажать Next.

4) На втором окне необходимо в самом низу сконфигурировать параметры для управляемого узла, дав имя узлу и указав внешний и внутренний IP-адрес, так же необходимо поставить галочки возле etcd и Control plane. Управляемый узел, в нашем случае, имеет IP адрес 192.168.56.104, что и указали как внешний и внутренний IP. Ниже сформируется команда для запуска агента, как показано на рисунке 3.3.

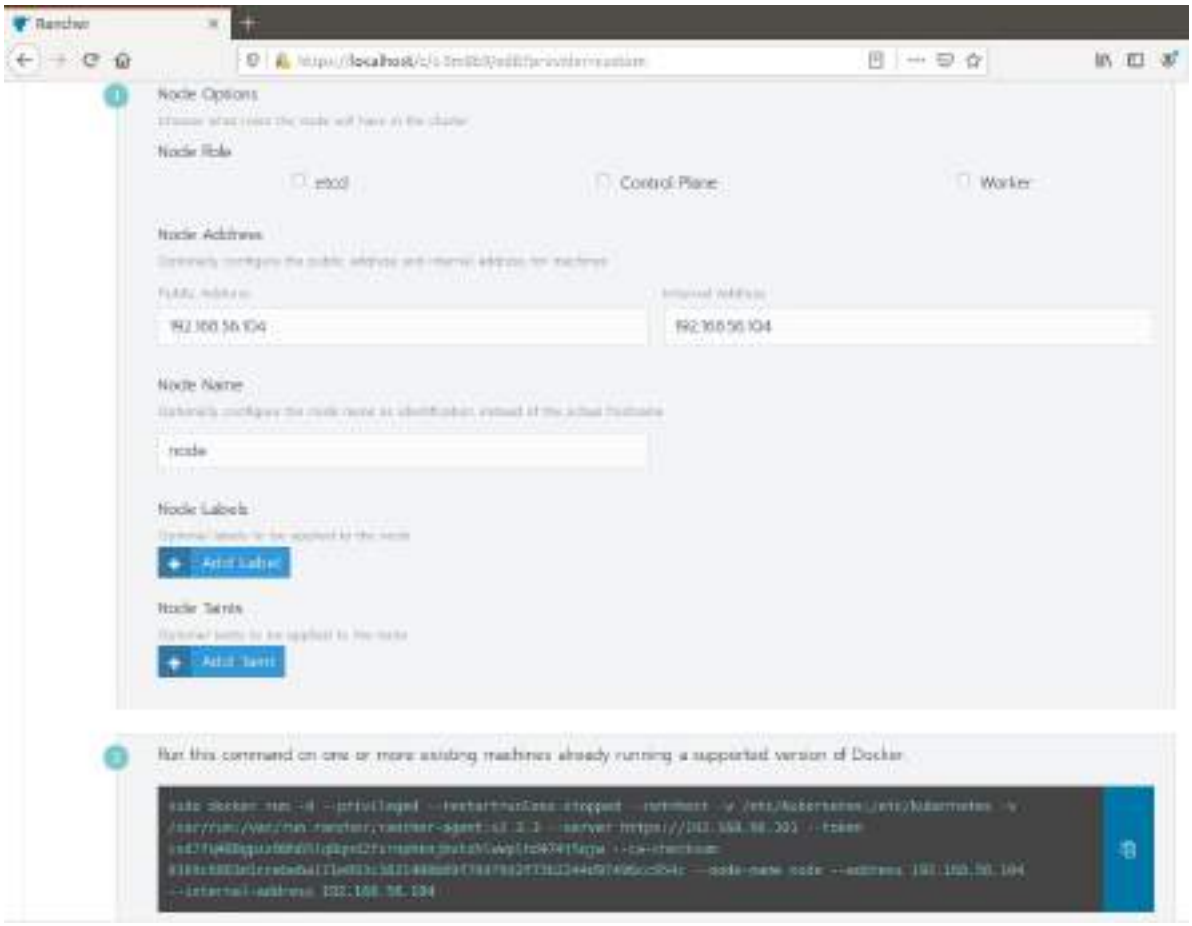


Рисунок 3.3 – Окно конфигурирования управляемого узла.

5) В консоли по SSH клиенту подключиться к управляемому узлу при помощи команды «ssh [ip узла] -l [имя пользователя в группе sudo]». После успешной авторизации, ввести команду запуска агента из шага 4 и дождаться его регистрации в мастере.

6) После удачной регистрации узла в кластере, необходимо перейти в кладку deploy и ввести имя приложения. Запускаться будет сервер nginx, как показано на рисунке 3.4. Через какое-то время приложение будут в состоянии

«Активно», но проброс портов будет, начиная от 30000, что легко исправить через настройку балансировки нагрузки.

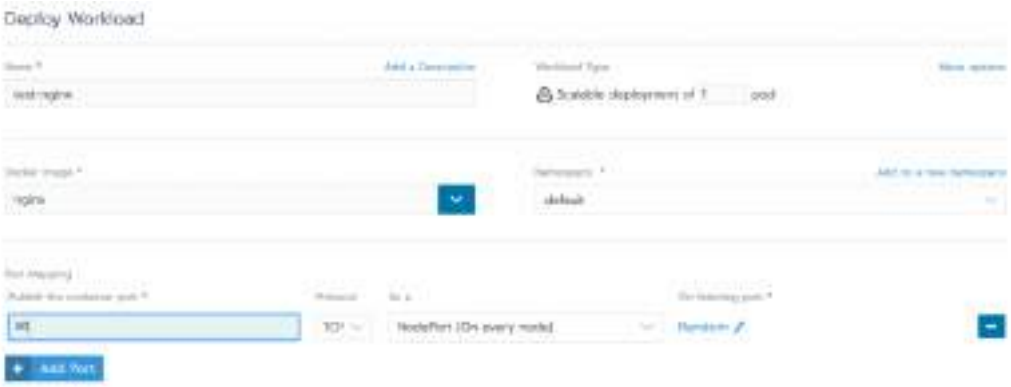


Рисунок 3.4 – Окно конфигурирования управляемого узла.

4.3. Задание на лабораторную работу

Развернуть через Rancher простой кластер, на котором нужно запустить сервер Apache. Дополнительно настроить проксирование запросов через балансировщик нагрузки через интерфейс Rancher.

4.4. Вопросы для самопроверки

1. Опишите архитектуру Rancher.
2. Как взаимодействует Rancher с кластерами Kubernetes.
3. Для чего нужен Authentication Proxy?
4. Какие задачи выполняет агент кластера?
5. Как создать новый кластер в Rancher?

4.5. Список использованных источников

1. Rancher: Kubernetes за 5 минут на голом железе [Электронный ресурс]. URL: https://habr.com/ru/post/418691/ (дата обращения: 14.03.2020).

2. Rancher Deployment Quick Start Guides [Электронный ресурс]. URL: https://rancher.com/docs/rancher/v2.x/en/quick-start-guide/ (дата обращения: 14.03.2020).

3. Установка Kubernetes кластера за 15 минут при помощи Rancher 2.0 [Электронный ресурс]. URL: https://cloud.croc.ru/blog/byt-v-teme/kubernetes-rancher/ (дата обращения: 14.03.2020).

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ ИНТЕЛЛЕКТУАЛЬНЫЕ ИНТЕРНЕТ-ПРИЛОЖЕНИЯ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

Содержание

Лабораторная работа №1 «Подготовка к разработке»

1.1. Цель работы

Цель работы: Подготовить проект и необходимые инструменты для начала разработки приложений на основе библиотеки ReactJS.

1.2. Теоретический материал

1.2.1. Общие сведения о работе

Первый шаг – установка NodeJS, скачивать необходимо последний стабильный релиз под меткой «LTS». Вместе с платформой NodeJS устанавливается пакетный менеджер npm необходимый для удобного скачивания зависимостей. Существует две среды исполнения JavaScript – какой-либо браузер и NodeJS. При разработке front-end приложения, используя ReactJS, итоговое приложение будет исполняться именно в браузере, а NodeJS нужен для таких задач, как:

– сборка проекта – это любые операции с файлами призванные оптимизировать их вес и количество (минификация файлов проекта и создание бандлеров);

– запуск локального сервера для разработки (нужен для динамического применения изменений в файлах).

Второй шаг – установка IDE. Для выполнения лабораторных работ рекомендуется использовать среду программирования «WebStorm» от компании «JetBrains».

Третий шаг – генерация проекта. Первое, что необходимо сделать при создании нового проекта – установить пакет «create-react-app». Пакет create-react-app предоставляет CLI-интерфейс для создания приложений с базовой структурой, устанавливает все нужные зависимости и добавляет в package.json скрипты для запуска, тестов и сборки приложения. Для установки этого пакета необходимо выполнить команду «npm install create-react-app -g» используя командную строку или терминал. Далее необходимо сгенерировать стартовый

шаблон проекта с помощью команды «create-react-app react-course» (команда создаст папку с проектом поэтому следует заранее перейти в место желаемого нахождения проекта в консоли). После окончания выполнения команды следует открыть папку как проект используя WebStorm. Структура проекта:

– node_modules/ – содержит исходные файлы зависимостей проекта (к примеру исходные файлы библиотеки ReactJS);

– public/ – содержит ресурсы проекта;

– src/ – содержит исходные файлы проекта;

– package.json – хранит список пакетов, необходимых для проекта с нужными версиями, и на другой машине можно легко установить все пакеты, которые указаны там с помощью команды «npm install».

– README.md – файл описания проекта, описанный на языке разметки markdown.

Четвертый шаг – сборка проекта и запуск сервера разработки проекта. В поле «scripts» файла package.json описаны основные скрипты проекта:

– start – запускает локальный сервер разработки с автоматическим применением изменений;

– build – выполняет сборку проекта в статику (по умолчанию сборка производится в папку build/ в корне проекта);

– test – выполняет запуск тестов;

– eject – выполняет «выбрасывание» скрытых зависимостей (по умолчанию при генерации проекта с помощью пакета create-react-app для удобства основные зависимости объединены в группы, данная команда вытаскивает эти зависимости из групп).

Для выполнения скрипта необходимо выполнить команду «npm run %script_name%», например, для запуска локального сервера разработки нужно выполнить «npm run start», а для сборки статики «npm run build» и т.д.

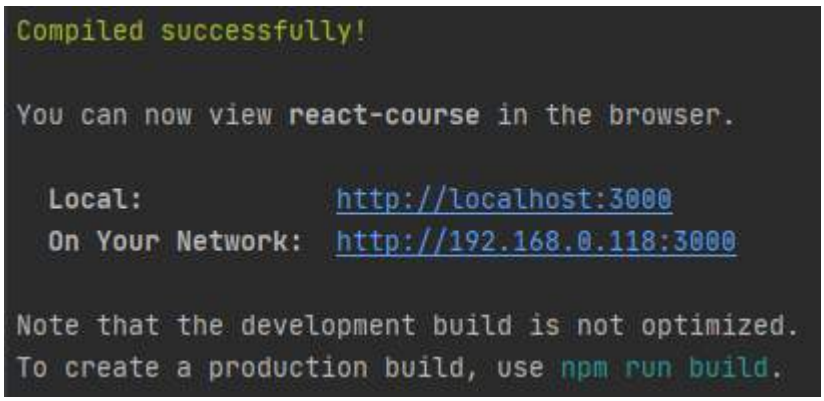


Рис. 1.1. Результат выполнения команды «npm run start» в папке с проектом

1.3. Лабораторное задание

– Установить NodeJS и npm;

– установить среду разработки;

– выполнить установку пакета create-react-app;

– сгенерировать проект под названием react-example;

– выполнить запуск локального сервера разработки;

– убедится, что локальный сервер разработки запущен путем перехода на URL сервера через браузер;

– выполнить сборку проекта;

– убедится, что сборка была выполнена.

1.4. Вопросы для самопроверки

1. Что такое nodeJS и для чего он используется?
2. Что такое npm и для чего он используется?
3. Зачем нужен package.json файл?
4. Что хранится в папке node_modules/?
5. Как установить какой-либо пакет используя пакетный менеджер npm?
6. Как выполнить установку всех пакетов из их списка в файле package.json?
7. Как выполнить запуск скрипта, описанного в файле package.json?

1.5. Список использованных источников

1. NodeJS [Электронный ресурс]. URL: https://nodejs.org/en/ (дата обращения: 12.04.2020).
2. Long Time Support [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/Long-term_support (дата обращения: 12.04.2020).
3. NPM [Электронный ресурс]. URL: https://www.npmjs.com/ (дата обращения: 12.04.2020).
4. Integrated development environment [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/Integrated_development_environment (дата обращения: 12.04.2020).
5. WebStorm [Электронный ресурс]. URL: https://www.jetbrains.com/ru-ru/webstorm/ (дата обращения: 12.04.2020).
6. Документация ReactJS [Электронный ресурс]. URL: https://ru.reactjs.org/docs/getting-started.html (дата обращения: 12.04.2020).
7. Документация npm [Электронный ресурс]. URL: https://docs.npmjs.com/ (дата обращения: 12.04.2020).
8. Документация create-react-app [Электронный ресурс]. URL: https://ru.reactjs.org/docs/create-a-new-react-app.html (дата обращения: 12.04.2020).

Лабораторная работа №2 «Single Page Application»

2.1 Цель работы

Цель работы: изучение способов работы с инструментами VDOM и JSX.

2.2 Теоретический материал

2.2.1 Общие сведения

Существует несколько вариантов жизненных циклов веб-приложений:

– Традиционный, когда серверная сторона (далее «back-end») веб-приложения генерирует html страницы и возвращает их пользователю, который их запросил, далее браузер просто отображает полученный html файл на странице. В таком подходе имеется ряд минусов - например приложение не может обновить информацию на странице без перезагрузки страницы в браузере. Примером такого веб-приложения может служить vk.com образца 2006 года. Пользователь не мог знать о том, что ему пришло новое сообщение не обновляя страницу браузера. Схема работы веб-приложения такого типа представлена на рисунке 2.1.

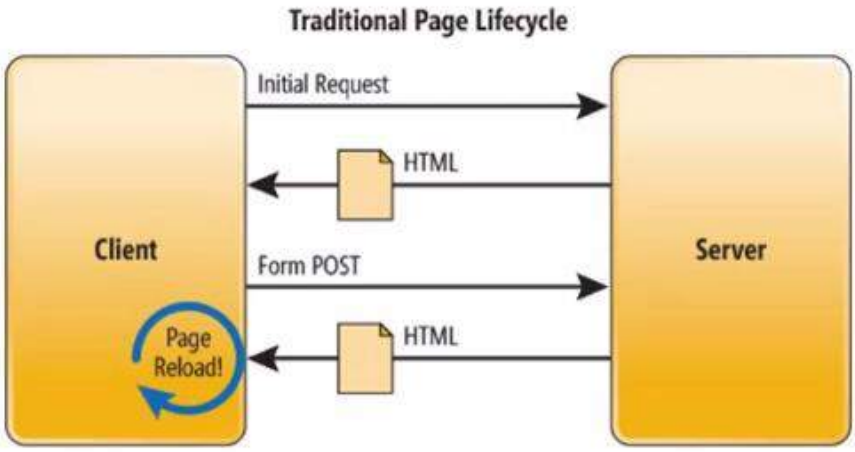


Рис. 2.1. Традиционный жизненный цикл веб-приложения

– Single Page Application (далее «SPA») – когда back-end веб-приложения возвращает по запросу пользователя .js файл(ы), получив который, браузер начинает генерировать html файл(ы) «налету» (т.е. генерация производится на клиентской стороне в отличие от традиционного жизненного цикла) в зависимости от действия пользователя, а back-end веб-приложения возвращает

лишь структурированную информацию, которую нужно отобразить. Таким образом состояние отображения страницы (далее «view») становится возможным изменять без перезагрузки страницы в браузере. Схема работы веб-приложения такого типа представлена на рисунке 2.2.

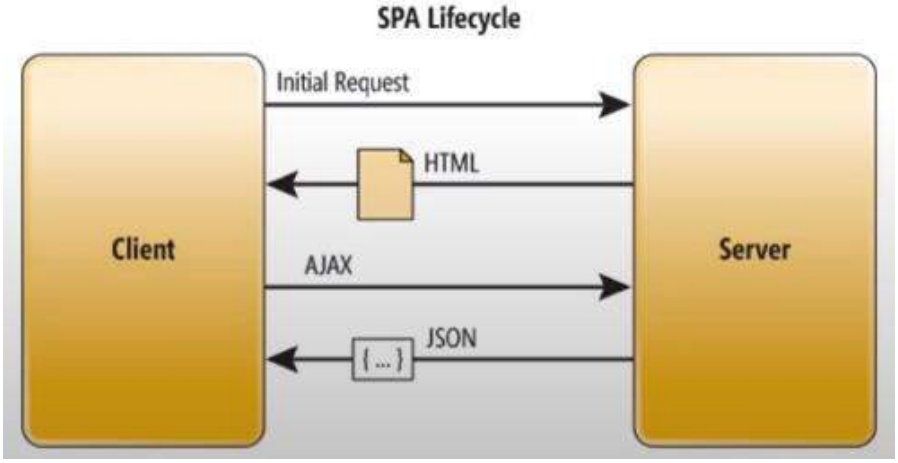


Рис. 2.2. SPA жизненный цикл веб-приложения

С помощью библиотеки ReactJS можно легко создавать SPA приложения, для этого у неё есть очень полезный инструмент под названием Виртуальное Document object module или дом-дерево (далее «DOM») - это концепция программирования, в которой «виртуальное» представление пользовательского интерфейса (User Interface, далее «UI») хранится в памяти и синхронизируется с «настоящим» представлением UI (этот процесс называется согласованием), где функцию представления выполняет DOM. С помощью этого инструмента программисту больше не нужно задумываться о написании кода, выполняющего обновление данных на странице, что существенно сокращает трудозатраты и время на разработку веб-приложения, а также ведет к снижению сложности архитектуры приложения.

Для того, чтобы удобно управлять UI веб-приложения библиотека ReactJS имеет инструмент под названием JavaScript XML (Далее «JSX»). JSX – это препроцессор, который добавляет синтаксис XML к JavaScript. Это никак не влияет на поведение программы, но делает использование функций отрисовки (рендеринга) React компонентов значительно удобнее. Вместо того,

чтобы искусственно разделить технологии, помещая разметку и логику в разные файлы, React использует слабо связанные единицы, называемые «компонентами», которые содержат и разметку, и логику. React можно использовать и без JSX, но его использование повышает наглядность при работе с UI, «живущем» в JavaScript-коде.

2.2.2 Структура проекта

После выполнения первой лабораторной работы был сгенерирован проект используя пакет create-react-app. В папке с проектом есть каталог src/, в котором хранятся основные исходники веб-приложения. Структура каталога представлена на рисунке 2.3.

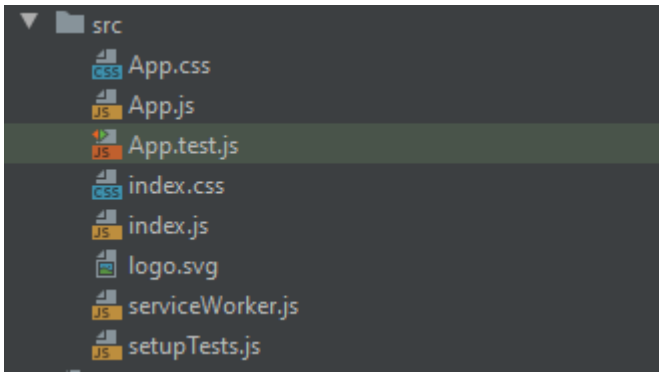


Рис. 2.3. Структура каталога src/

В котором:

– файл App.css описывает стили компонента App;

– файл App.js описывает код компонента App;

– файл index.css описывает стили «коренной» html страницы веб-приложения;

– файл index.js описывает код вставки React компонента(-ов) в коренной дом-элемент страницы index.html.

Рассмотрим файл index.html в папке public/.

```
//.........................
  <div id="root"></div>
//.........................
```

В div-элемент с идентификатором root будет вставлен сгенерированный html библиотекой ReactJS, это можно увидеть исходя из кода в файле index.js каталога src/. Рассмотрим содержимое этого файла.

```
//.........................
ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
//.........................
```

Из листинга выше видно, что выполняется рендеринг корневого React компонента (App) в дом-элемент с идентификатором root. При этом будет производится проверка корневого компонента и всех его потомков на валидность, на это указывает StrictMode.

```
//.........................
  <React.StrictMode>
    <%RootComponentName% />
  </React.StrictMode>
//.........................
```

StrictMode помогает в:

– обнаружении небезопасных методов жизненного цикла;

– предупреждении об использовании устаревшего API библиотеки ReactJS;

– обнаружении неожиданных побочных эффектов;

– обнаружении устаревшего API контекста.

Рассмотрим файл App.js:

```
1: import React from 'react';
2: import logo from './logo.svg';
3: import './App.css';
4: function App() {
5:   return (
6:     <div className="App">
7:       <header className="App-header">
8:         <img src={logo} className="App-logo" alt="logo" />
9:         <p> Edit <code>src/App.js</code> and save to reload. </p>
10:         <a className="App-link" href="https://reactjs.org" target="_blank">
11:           Learn React
12:         </a>
13:       </header>
14:     </div>
15:   );
16: }
17: export default App;
```

Данный файл представляет собой пример React компонента. В первой строчке импортируется React функционал, во второй – логотип библиотеки в формате векторных изображений (svg), а в третьей строчке стили данного компонента.

2.2.3 Создание простейшего функционального компонента

Для того, чтобы описать простейший функциональный React компонент, необходимо:

1) Создать новый файл в формате %имя_компонента%.js в папке src/.
2) Выполнить импорт библиотеки React.
3) Описать именованную функцию.
4) Вернуть из этой функции JSX-код компонента.
5) Выполнить экспорт функции.

Пример простого компонента:

```
1: import React from 'react';
2: const MyComponent = () => {
3:   return (<div>HELLO WORLD</div>);
4: }
5: export default MyComponent;
```

Далее необходимо добавить созданный компонент в нужное место импортируя его:

```
rel="noreferrer"
```

```
//.......................
import MyComponent from './MyComponent';
//.......................
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />
        <p> Edit <code>src/App.js</code> and save to reload. </p>
        <a className="App-link" href="https://reactjs.org" rel="noreferrer" target="_blank">
          Learn React
        </a>
        <MyComponent />
      </header>
//.......................
```

Результат работы представлен на рисунке 2.4

Рис. 2.4. Стартовая страница проекта после добавления компонента MyComponent

2.2.4 JavaScript-выражения внутри JSX

Внутри JSX можно использовать любое js-выражение, заключив его в фигурные скобки. Рассмотрим пример листинга с функцией map():

Листинг файла базового компонента (MyComponent) в котором инициализируется компонент со списком (ListComponent):

```
1: import React from 'react';
2: import ListComponent from './ListComponent'
3: const array = ['first', 'second', 'third'];
4: const MyComponent = () => {
5:   return (<ListComponent array={array} />); // в array ['first', 'second', 'third']
6: }
7: export default MyComponent;
```

Листинг файла компонента списка (ListComponent):

```
1: const ListComponent = (data) => {
2:   const { array } = data; // деструктуризация
3:     return(
4:       <ul>
5:         {
6:     array.map((element, index) => <div key={index}>{element}</div>)
7:         }
8:       </ul>
9:    );
10: };
11: export default ListComponent;
```

В данном случае компонент ListComponent получает на вход объект с полем array в котором хранится передаваемый массив (['first', 'second', 'third']).

Рассмотрим случай с условием внутри JSX:

```
//.........................
const ComponentWithCondition = (data) => {
    const { isNeedDisplaying } = data;
    return(
        <div>
            { isNeedDisplaying
                ? <div>If true, this div well be displayed!</div>
                : ''
            }
        </div>
    );
}
//.........................
```

В данном случае компонент ComponentWithCondition получает на вход объект с полем isNeedDisplaying, содержащее переменную булевого типа, условие выполнено с помощью тернарного оператора. В результате, если на момент рендера isNeedDisplaying равно true, будет отображен div.

2.3 Лабораторное задание

Лабораторная работа выполняется на базе лабораторной работы №1. В исходный проект необходимо добавить компонент, описывающий список (<List value={array}/>), в котором элементы – это тоже компоненты (<Element value={elementValue} />). Компонент <Element /> должен быть отображен с помощью цикла. Разметка самих компонентов должна быть написана используя JSX. Сами компоненты должны быть функциональными.

В качестве входных данных компонентов необходимо использовать произвольный массив типа:

```
//.........................
const array = [‘fistElement’, ‘secondElement’, ‘thirdElement’];
//.........................
```

Код корневого компонента представлен на рисунке 2.5

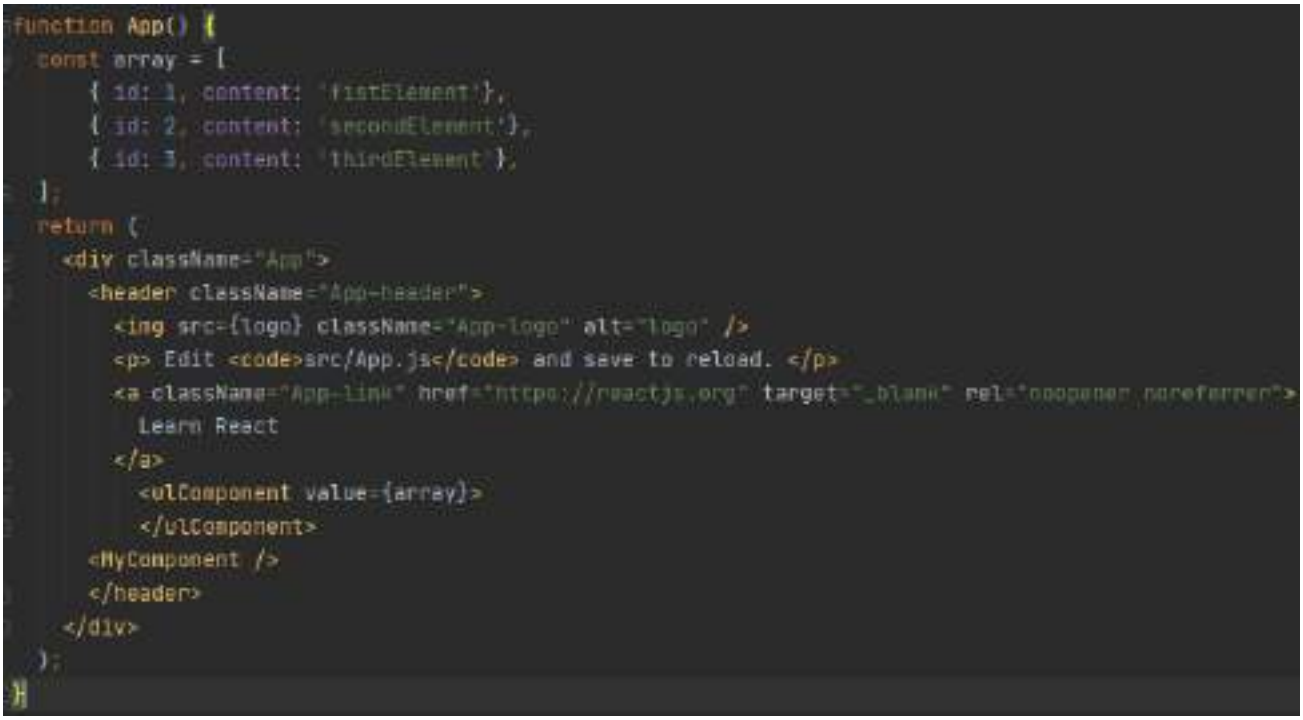


Рис. 2.5. Код корневого компонента

2.4 Вопросы для самопроверки

1. Что такое VDOM и DOM? Для чего они нужны?
2. Что такое JSX и для чего он нужен?
3. Опишите принцип добавления корневого React компонента в корневой html документ.
4. Опишите последовательность действий для создания нового функционального компонента.

2.5 Список использованных источников

1. Document object module [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/Document_Object_Model (дата обращения: 14.04.2020).

2. JavaScript XML [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/React_(web_framework)#JSX (дата обращения: 14.04.2020).

3. Array.prototype.map() [Электронный ресурс]. URL: https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Array/map (дата обращения: 14.04.2020).

4. Тернарный оператор [Электронный ресурс]. URL: https://en.wikipedia.org/wiki/%3F:#JavaScript (дата обращения: 14.04.2020).

5. Знакомство с JSX [Электронный ресурс]. URL: https://ru.reactjs.org/docs/introducing-jsx.html (дата обращения: 14.04.2020).

6. Виртуальный DOM [Электронный ресурс]. URL: https://ru.reactjs.org/docs/faq-internals.html (дата обращения: 14.04.2020)

7. Пробуем React [Электронный ресурс]. URL: https://ru.reactjs.org/docs/getting-started.html#try-react (дата обращения: 14.04.2020).

8. Деструктуризация [Электронный ресурс]. URL: https://learn.javascript.ru/destructuring (дата обращения: 14.04.2020).

Лабораторная работа №3 «React hooks»

3.1 Цель работы

Цель работы: изучение способов работы с хуками библиотеки ReactJS

3.2 Теоретический материал

3.2.1 Общие сведения

Хуки - это функции, которые позволяют вам «зацепить» состояние React и функции жизненного цикла из компонентов-функций. React предоставляет несколько встроенных хуков. Также вы можете создавать и свои собственные для повторного использования поведения, связанного с состоянием, в различных компонентах. Но для начала необходимо ознакомимся со встроенными хуками.

3.2.2 useState hook (хук состояния)

useState - это хук, который позволяет добавлять состояние React к компонентам-функциям. Разберем пример листинга использования этого хука:

```
1: import React, { useState } from 'react';
2: export default function UseStateHookExample() {
3:    // Объявляем новую переменную состояния "count"
4:    const [count, setCount] = useState(0);
5:    return (
6:        <div>
7:            <p>Вы нажали {count} раз</p>
8:            <button onClick={() => setCount(count + 1)}>
9:                Нажми на меня
10:            </button>
11:        </div>
12:    );
13: }
```

Вызов useState объявляет «переменную состояния». Это способ «сохранять» некоторые значения между вызовами функций. Наша переменная называется count, но мы можем назвать ее как угодно, например, banana. Обычно переменные теряются при выходе из функции, но переменные состояния сохраняются React-ом. В качестве аргумента в useState передается Единственный аргумент - это начальное состояние. Мы можем сохранять число или строку, если это все, что нам нужно. В нашем примере мы хотим хранить просто число, показывающее сколько раз пользователь кликал, поэтому передаем 0 в качестве

начального состояния для нашей переменной. Если бы мы хотели сохранить два разных значения в состоянии, мы бы вызвали useState() дважды. useState возвращает два значения: текущее состояние и функцию, которая его обновляет.

Итак, мы объявляем переменную состояния с именем count и устанавливаем ее равной 0. React запоминает ее текущее значение между повторными отрисовками и предоставляет самое последнее значение для нашей функции. Если мы хотим обновить текущее значение счетчика, мы можем вызвать setCount.

Весь процесс, таким образом, выглядит так:

– Строка 1: мы импортируем хук useState из React. Это позволяет нам сохранять локальное состояние в компоненте-функции;

– Строка 4: внутри компонента UseStateHookExample мы объявляем новую переменную состояния, вызывая хук useState. Он возвращает пару значений, которым мы даем имена. Мы называем нашу переменную count, потому что она содержит количество нажатий кнопки. Мы инициализируем её нулем, передавая 0 как единственный аргумент useState. Второй возвращаемый элемент сам по себе является функцией и позволяет нам обновлять счетчик count, поэтому мы назовем его setCount;

– Строка 8: когда пользователь кликает, мы вызываем setCount с новым значением. Затем React повторно выполнит отрисовку компонента UseStateHookExample, передав ему новое значение count.

3.2.3 useEffect hook (хук эффекта)

Этот фрагмент кода основан на примере счетчика из предыдущего раздела. Однако в него была добавлена новая функция: было установлено название документа, содержащее количество нажатий.

Извлечение данных, настройка подписки и ручное изменение DOM в компонентах React - все это примеры побочных эффектов. Возможно, вы выполняли такие действия в своих компонентах, не зная, вероятно, что они так

называются. Компоненты React имеют два основных вида побочных эффектов: требующие очистки, и не требующие. Разберём это различие более подробно.

3.2.3.1 Эффекты, не требующие очистки

Иногда необходимо выполнить дополнительный код после того, как React обновил DOM. Сетевые запросы, ручные изменения DOM и логирование - типичные примеры эффектов, которые не требуют очистки. Разберем пример листинга использования таких эффектов:

```
1: import React, { useState, useEffect } from 'react';
2: export default function UseEffectHookUseExample() {
3:   const [count, setCount] = useState(0);
4:   useEffect(() => {
5:     // Обновляем заголовок документа с помощью API браузера
6:     document.title = `Вы нажали ${count} раз`;
7:   });
8:   return (
9:     <div>
10:       <p>Вы нажали {count} раз</p>
11:       <button onClick={() => setCount(count + 1)}>
12:         Нажми на меня
13:       </button>
14:     </div>
15:   );
16: }
```

При использовании хука useEffect он сообщает React, что компонент должен что-то делать после отрисовки. React запомнит переданную функцию (будем называть ее «эффектом») и вызовет ее после обновления DOM. В текущем случае мы устанавливаем название документа. Кроме это становится возможно извлекать данные или вызывать любой другой императивный API. Вызывая useEffect внутри компонента, мы получаем доступ к переменной count состояния счетчика (или любым другим свойствам) прямо из эффекта. Нам не нужен специальный API для её чтения - она уже находится в области видимости функции. По умолчанию useEffect запускается как после первой отрисовки, так и после каждого последующего обновления. React гарантирует, что DOM будет обновлен к моменту запуска эффектов.

Объединив все выше разобранное можно подытожить, сначала мы объявляем переменную состояния count, а затем говорим React, что нужно

использовать эффект. Мы передаем функцию, которая и является нашим эффектом, в хук useEffect. Внутри эффекта устанавливаем название документа с помощью API браузера document.title. Мы можем прочитать последнее значение счетчика внутри эффекта, потому что он находится в области видимости нашей функции. Когда React отрисовывает компонент, он помнит переданный нами эффект, а затем запускает его после обновления DOM. Это происходит после каждой отрисовки компонента, включая самую первую.

3.2.3.2 Эффекты с очисткой

Ранее было рассмотрен способ создания побочных эффектов, которые не требуют какой-либо очистки. Однако некоторым эффектам она всё же нужна. Допустим, необходимо настроить подписку на некоторый внешний источник данных. В этом случае важно провести очистку, чтобы избежать утечек памяти.

```
//.........................
1: useEffect(() => {
2:   // Код подписки
3:   ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
4:   // Указываем как производить очистку после этого эффекта:
5:   return function cleanup() {
6:     // Код отписки
7      ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
8:   };
9: });
//.........................
```

Почему функция возвращается из эффекта? Это опциональный механизм очистки для эффектов. Каждый эффект может возвращать функцию, которая после него выполнит очистку. Это позволяет нам поддерживать коды добавления и удаления подписок максимально близко друг к другу. Они являются частью одного эффекта.

Когда именно React выполняет очистку в эффекте? React производит очистку, когда компонент демонтируется. Однако, как было сказана раннее, эффекты запускаются для каждой отрисовки, а не единожды. Вот почему React также очищает эффекты предыдущей отрисовки, прежде чем запускать эффекты снова.

3.2.3.3 Резюме

useEffect позволяет определять различные виды побочных эффектов, происходящих после отрисовки компонента. Некоторые эффекты могут требовать очистку, поэтому они должны возвращать функцию:

```
//.........................
1: useEffect(() => {
2:     ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
3:     return () => {
4:       ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
5:     };
6: });
//.........................
```

Эффекты, не имеющие фазы очистки, ничего не возвращают:

```
//.........................
1: useEffect(() => {
2:   document.title = `Вы нажали ${count} раз`;
3: });
//.........................
```

Хук эффекта объединяет оба случая под одним API.

3.3 Лабораторное задание

Лабораторная работа выполняется на базе лабораторной работы №2. При выполнении задания необходимо использовать хук useState и useEffect.

В исходный проект необходимо добавить функцию поиска, т.е. необходимо добавить поле ввода, при изменении данных в котором будет выполняться поиск по элементам массива (принцип его работы основывается на поиске первого вхождения подстроки в строке). Результат поиска должен заменять отображение начального списка на найденный(-ые) элемент(-ы) из этого списка (пока строка пустая, отображается начальный вариант списка). При этом необходимо отображать в title веб-страницы количество найденных элементов (если в данный момент поиск не производится, то выводить «useEffect hook example»).

3.4 Вопросы для самопроверки

1. Что такое React hooks и для чего они нужны?
2. Опишите принцип работы хука useState. Для чего он нужен?

3. Опишите принцип работы хука useEffect. Для чего он нужен?
4. Расскажите в чем различия эффектов с очисткой и без.

3.5 Список использованных источников

1. Введение в хуки [Электронный ресурс]. URL: https://ru.reactjs.org/docs/hooks-intro.html (дата обращения: 18.04.2020).
2. Использование хука состояния [Электронный ресурс]. URL: https://ru.reactjs.org/docs/hooks-state.html (дата обращения: 18.04.2020)
3. Использования хука эффекта [Электронный ресурс]. URL: https://ru.reactjs.org/docs/hooks-effect.html (дата обращения: 18.04.2020).
4. Логирование [Электронный ресурс]. URL: https://ru.wiktionary.org/wiki/логирование (дата обращения 18.04.2020)

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ ИНТЕЛЛЕКТУАЛЬНЫЕ СИСТЕМЫ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

СОДЕРЖАНИЕ

Введение

В пособии рассмотрены понятия искусственного интеллекта и методы представления, использования и приобретения знаний, а также принципы построения и функционирования интеллектуальных систем. Используется представление знаний семантическими сетями, концептуальными графами и логическими выражениями. Описаны также интеллектуальные технологии, позволяющие практически создавать экспертные системы, основанные на правилах и искусственных нейронных сетях.

Представленные практические задания рассматривают создание приложений в различных, в том числе междисциплинарных прикладных областях применения систем и технологий, базирующихся на применении методов искусственного интеллекта.

Модели искусственного интеллекта находят широкое применение в интеллектуальных ***киберфизических*** (реже используется термин ***кибертехнических***) ***системах*** (англ. ***Cyber-Physical Systems***, ***CPS***), которые состоят из различных природных объектов, искусственных подсистем и управляющих контроллеров, позволяющих представить такое образование как единое целое (иногда используется термин "***кибертехнические системы***"). В CPS обеспечивается тесная связь и координация между вычислительными и физическими ресурсами. Компьютеры осуществляют мониторинг и управление физическими процессами с использованием такой петли обратной связи, где происходящее в физических системах процессы оказывает влияние на вычисления и наоборот.

С технической точки зрения CPS имеют много общего со структурами типа грид (grid), реализуемыми посредством ***Интернета вещей*** (***Internet of Things***, ***IoT***), ***Индустрии 4.0***, ***промышленного интернета вещей*** (***Industrial Internet***), ***межмашинного взаимодействия*** (***Machine-to-Machine***, ***M2M***), ***туманного*** и ***облачного компьютинга*** (***Fog*** и ***Cloud Computing***). Для создания этих сложных систем требуются новые кибернетические подходы к

моделям искусственного интеллекта, поскольку именно модели при проектировании являются центральным моментом в науке и инженерии. Приведем далее некоторые понятия и определения в области киберфизических систем.

Киберфизическая система – это система, построенная на основе интеграции вычислительных ресурсов с физическими сущностями любого вида, включая биологические и рукотворные объекты. В киберфизических системах вычислительная компонента, как правило, распределена по всей физической системе, которая является её носителем, и связана с ее составляющими элементами.

Следующие ключевые технологические тенденции лежат в основе киберфизических систем: ***большие данные и аналитика***; ***автономные агенты и роботы***; ***моделирование и симуляторы*** – инженеры уже давно используют 3D-моделирование на стадии проектирования продуктов или процессов; ***облачные вычисления*** позволяют создавать платформы для совместной работы и обмена данными между территориально-распределенными партнерами.

В будущем технологии ***больших данных*** позволят использовать различные симуляторы в режиме реального времени. Например, на стадии производства оператор сможет виртуально смоделировать физический процесс с учетом имеющегося сырья и людей и тем самым снизить время настройки оборудования и повысить качество продукции.

Интернет вещей – показания датчиков и сенсоров обычно попадают в централизованную систему управления производственным процессом, и уже на этом уровне принимаются решения. В дальнейшем возможности, которые предоставляют встраиваемые системы, позволят устройствам «общаться» друг с другом и децентрализовать принятие решений. Например, можно использовать радиочастотные метки для полуфабрикатов — автоматизированная производственная линия, считав метку, сама примет

решение (в реальном времени), какую операцию применить к тому или иному полуфабрикату.

Информационная безопасность – многие компании используют системы управления и производства, основанные на технологиях, не имеющих выход в интернет, но по мере расширения связей с партнерами, использования открытых стандартов и протоколов резко возрастают риски информационной безопасности. Для защиты промышленных систем потребуется не только качественная и безопасная связь, но и системы управления учетными записями и контроля доступа.

Технология дополненной реальности – данная технология находится в начальной стадии своего развития, но в будущем позволит работникам ускорить принятие решений. Например, работник может получить инструкцию, как починить или заменить сломанную деталь в производственной системе, когда он на неё смотрит через очки дополненной реальности (https://ru.wikipedia.org/wiki/Киберфизическая_система).

Частным случаем киберфизических систем следует считать так называемые ***эргатические*** системы, в которых особое внимание уделяется наличию и влиянию «человеческого фактора». Эргатической системе соответствует организация, одной стороной которой является человек или группа людей, а другую сторону представляют технические устройства, посредством которых человек осуществляет свою деятельность. Основными особенностями таких систем являются социально-психологические аспекты. В зависимости от конкретного сценария функционирования эргатической системы присутствие человеческого фактора может рассматриваться как достоинство, так и как недостаток.

На сегодняшний день эргатические системы широко распространены. Примером таких систем являются: система управления блоком станции, система управления самолетом, диспетчерская служба аэропорта, вокзала. Эргатические системы нашли своё применение на объектах, где вмешательство оператора в работу объекта является на сегодняшний день

необходимым условием обеспечения надежной работы данных объектов (https://ru.wikipedia.org/wiki/Эргатическая_система).

В свою очередь, понятия киберфизической и эргатической систем являются частными случаями понятия ***сложной системы***, состоящей из множества взаимодействующих составляющих (подсистем).

Логический подход к интеллектуализации вычислительных и киберфизических систем: общие сведения

Сложность современных вычислительных и киберфизических систем, отсутствие во многих случаях близких по характеристикам и структуре прототипов приводит разработчиков к необходимости использования имитационных моделей различных уровней.

Применение логического подхода к интеллектуализации вычислительных и киберфизических систем позволяет формализовать и строго специфицировать происходящие в них процессы. С логическим подходом тесно связано использование в процессе описания основных моделей искусственного интеллекта – продукционных правил, семантических сетей, сценариев, концептуальных графов, сетей фреймов, а также логических нейронных сетей.

При представлении знаний в математической логике используются логический формализм – исчисление предикатов, которое имеет строгую формальную семантику и механизмы вывода.

В приложениях информатики обычно рассматривают некоторое множество представлений (сигнатуру) Σ с интерпретацией I в множестве S элементов; интерпретация I данному представлению σ ставит в соответствие некоторое абстрактное информационное содержание $I(\sigma)$, т.е. интерпретации соответствует отображение I: $\Sigma \rightarrow S$. Пусть Σ – множество функциональных и предикатных символов различных арностей, S – множество конкретных функций и предикатов.

В многосортных, или многоосновных, системах тип n-арного функционального символа – это кортеж $(i_1, i_2, \ldots, i_n, j)$, а тип n-арного предикатного символа – это кортеж $(i_1, i_2, \ldots, i_n)$, где $i_1, i_2, \ldots, i_n, j$ – названия (сорта) для основ, или носителей, Функции и предикаты из множества S задают структурные связи между понятиями, положенными в основу формального определения интеллектуальной системы. Логические связи между понятиями задаются формулами, допустимыми в исчислении предикатов и включающими функции и предикаты из множества S.

В общем случае построение имитационной модели интеллектуальной системы основано на задании отношения $H \subset \mathbf{P}(S) \times \mathbf{P}(S)$, где S – непустое множество конкретных функций и предикатов (ситуаций в предметной области), а $\mathbf{P}$ – символ булеана. Данное отношение устанавливает зависимость одних множеств ситуаций от других. На более детализированном уровне моделирования между ситуациями задается причинно обусловленное отношение непосредственного следования.

Для представления динамики или эволюции процессов вводятся специальные операции – элементарные обновления функций и предикатов в «модулях-продукциях» и в «модулях-процедурах» некоторого языка (языка сетей абстрактных машин), которые модифицируют, или «обновляют», интерпретацию I, выполняя сгруппированные в блоки так называемые правила обновления вида $I(\sigma_i) \leftarrow s_j$. Моделируемая система эволюционирует, переходя от одной интерпретации $I(t_i)$ к другой $I(t_k)$, где t_i и t_k – последовательные моменты времени.

Модули-продукции» и модули-процедуры фактически представляют собой подпрограммы AI-ориентированного (от ***Artificial Intelligence*** – искусственный интеллект) программирования. Основной частью AI-технологии является переход от графовых моделей к продукционным моделям, которые затем реализуются в виде управляющих программ на каком-либо процедурном или объектно-ориентированном языке программирования.

Лабораторная работа №1

Представление знаний в вычислительных и киберфизических системах семантическими сетями и концептуальными графами

Цель работы: овладение основными навыками построения семантических сетей и концептуальных графов для формирования баз знаний.

В процессе изучения теоретического материала и практической работы необходимо освоить основные аспекты разработки и применения в различных предметных областях семантических сетей, концептуальных графов и сетей фреймов на основе использовании интеллектуального графического редактора, средств добавления связей с внешними файлами базы знаний и с языком логического программирования Пролог.

Основные понятия и определения

Семанти́ческая сеть – это информационная модель предметной области, имеющая вид ориентированного графа. Вершины графа соответствуют объектам предметной области, а дуги задают отношения между ними. Объектами могут быть понятия, события, свойства, процессы. Семантическая сеть – это один из способов представления знаний. В семантической сети роль вершин выполняют понятия базы знаний, а дуги (направленные ребра) задают отношения между ними. Таким образом, семантическая сеть отражает семантику предметной области в виде понятий и отношений.

Семантические сети возникли как попытка визуализации математических формул. Основным представлением для семантической сети является граф. За графическим изображением стоит строгая математическая запись и обе эти формы отображают одно и то же. Чаще всего понятия семантической сети записываются в прямоугольниках (возможны и другие формы) и соединяются стрелками с подписями – дугами. Используя аппарат математической логики, можем сделать вывод о том, что каждая вершина соответствует элементу предметного множества, а дуга – предикату.

В названии соединены термины из двух наук: семантика в языкознании изучает смысл единиц языка, а сеть в математике представляет собой разновидность графа — набора вершин, соединённых дугами (рёбрами), которым присвоено некоторое число. В семантической сети роль вершин выполняют понятия базы знаний, а дуги (причем направленные) задают отношения между ними. Таким образом, семантическая сеть отражает семантику предметной области в виде понятий и отношений.

Концептуальные графы представляют собой графическую интерпретацию формул в исчислении предикатов первого порядка. Многосортное исчисление предикатов первого и высших порядков отличается

ясной формальной семантикой и развитыми механизмами вывода, чем обусловлено его широкое использование для представления знаний. Совокупностью многоместных и многосортных предикатов и функций возможно описать большинство ситуаций в любой предметной области.

Словарь языка исчисления предикатов первого и высших порядков содержит пропозициональные переменные, предметные переменные (обозначают общие имена), индивидные (предметные) константы, предикатные константы и предикатные переменные, функциональные константы и переменные, логические связки, кванторы общности и существования.

Предикатные символы могут обозначать свойства объектов, отношения, свойства отношений. В формулах логики выше первого порядка разрешается ставить кванторы по предикатным переменным. Логике высших порядков в англоязычной литературе соответствует аббревиатура ***HOL*** – ***High Order Logic***. По аналогии, логике первого порядка соответствует аббревиатура ***FOL*** – ***First Order Logic***.

Недостатком логического формализма является его неструктурированность, что затрудняет сбор информации по конкретному объекту. Решение данной проблемы можно облегчить путем использования графических представлений исчисления предикатов, таких как ***концептуальные графы*** и ***семантические сети***, при помощи которых возможно структурировать описание предметной области и собирать знания по конкретному объекту. В общем случае для каждой предметной области возможно построить несколько логических моделей, отличающихся различной степенью детализации.

Ввиду того, что до сих пор при графическом подходе к логике искусственного интеллекта для обозначения одних и тех же понятий, или ***концептов***, предметной области и ***отношений*** между ними используются различные изображения, а сами модели искусственного интеллекта, такие как семантические сети, концептуальные графы и фреймы тесно связаны друг с другом, определимся с терминологией и обозначениями. Как концептуальные графы, так и семантические сети представляют собой графическую версию исчисления предикатов.

В ***концептуальных графах*** прямоугольниками представляются аргументы, или ***концепты***, обозначающие понятия предметной области, а овалами – ***отношения*** между этими понятиями, или концептами. Концептуальный граф, таким образом, может быть описан простой логической формулой, представляющей собой конъюнкцию бинарных предикатов. Семантические сети – это более сложные структуры, состоящие из множества концептуальных графов, и представляющие более сложное соединения формул, отражающих контекст области рассуждений, или ***экспертизы***.

В связи с тем, что используемый нами редактор CharGer называется редактором концептуальных графов, во избежание путаницы, далее не будет делаться различий между семантическими сетями и концептуальными графами, что в общем соответствует классическим работам [4, 20].

Примеры построения концептуальных графов

Основная используемая в настоящей главе терминология иллюстрируется рисунком 1.1.

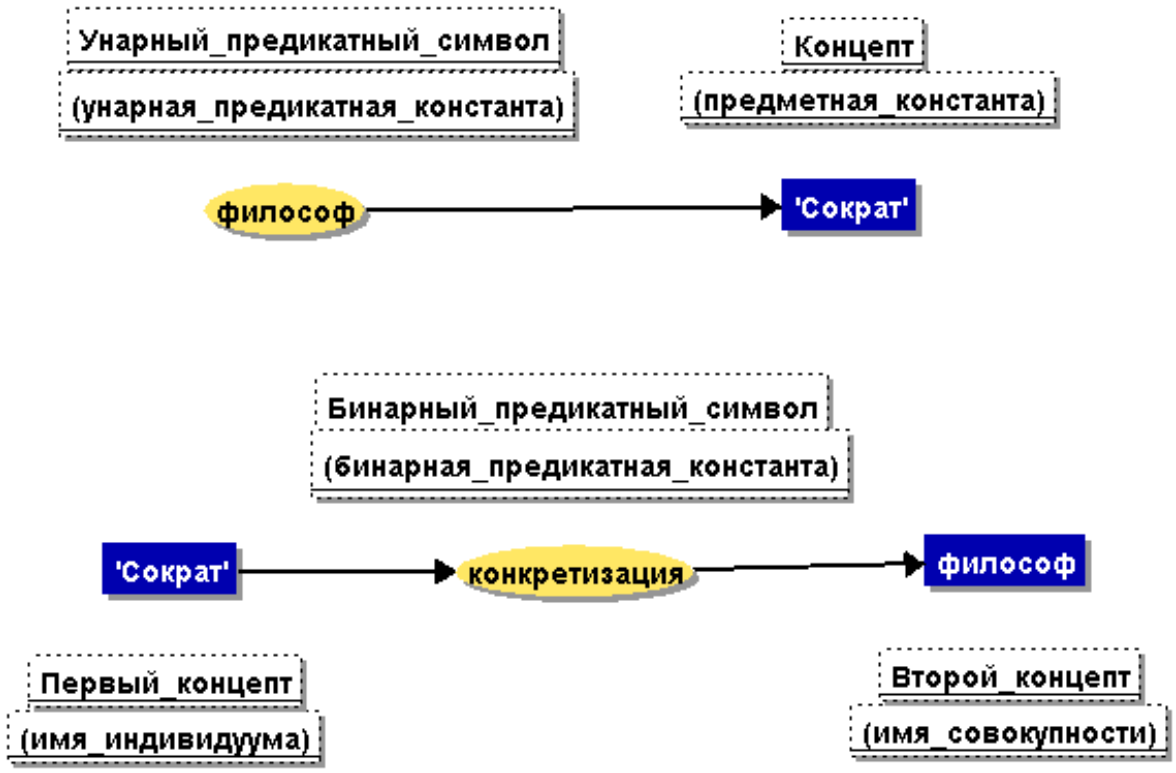


Рисунок 1.1 – Представление фразы “Сократ – философ” концептуальными графами: *а* – унарным предикатом, *б* – бинарным предикатом

Рассмотрим различные способы логического и концептуального описания некоторой фразы, например, утверждения “Сократ – философ”, отдавая дань уважения великому древнегреческому философу. На верхней части рисунка 1.1 дано графическое представление высказывания

философ(‘Сократ’), (1.1)

где *философ* – унарное предикатное имя, или предикатная константа, ‘*Сократ*’ – предметная константа, а на нижней части рисунка – представление той же самой фразы высказыванием

конкретизация(‘Сократ’, философ). (1.2)

Выражения (1.1) и (1.2) и соответствующие им концептуальные графы, естественно, различны, однако с позиций искусственного интеллекта и здравого смысла их можно считать эквивалентными в том плане, что они представляют одну и ту же фразу.

Здесь для бинарного предиката ***конкретизация*** определены: в качестве значения первого аргумента – имя индивидуума, а в качестве значения второго аргумента – имя совокупности. В концептуальном графе для того же самого предиката первый и второй его аргументы представлены ***концептами*** (обозначены прямоугольниками), или ***сущностями*** предметной области. Овалом обозначен сам ***предикат***.

Название ***сущность*** чаще используется при определении диаграмм “***сущность-связь***” при проектировании баз данных. Существуют следующее соответствие понятий баз данных (БД) и логического программирования (ЛП):

отношение (БД) – предикат (ЛП);
атрибут (БД) – аргумент предиката (ЛП);
кортеж (БД) – факт (ЛП);
запрос (БД) – цель (ЛП).

В дальнейшем будет учитываться тот факт, что каждому ***предикат***у соответствует одноименное ***отношение*** – область истинности данного предиката. В концептуальных графах и семантических сетях отношения чаще всего представляются овалами. Концептуальные графы и их композиции – семантические сети, могут использоваться не только для графического представления формул в логике предикатов, но и при проектировании баз данных. По существу, концептуальные графы являются предшественниками используемых при проектировании баз данных диаграмм “***сущность – связь***”.

Пример переинтерпретации концептуальных и логических моделей передачи сообщений на основе *n*-местных и бинарных предикатов

На рисунке 1.2 изображен концептуальный граф следующего высказывания:

event(‘Nadezhda’, ‘Gleb’, ‘11h_40m_01_01_2020’, e_mail, letter_26), (1.3)

формализующего фразу:

“ ‘Nadezhda’ sent to ‘Gleb’ at ‘11h_40m_01_01_2020’ by e_mail the letter_26” (1.4)

Номера в ромбиках (здесь ромбики – не акторы, а простые “бирки” для размещения номеров!) означают порядок следования значений предметных переменных в выражении (1.3).

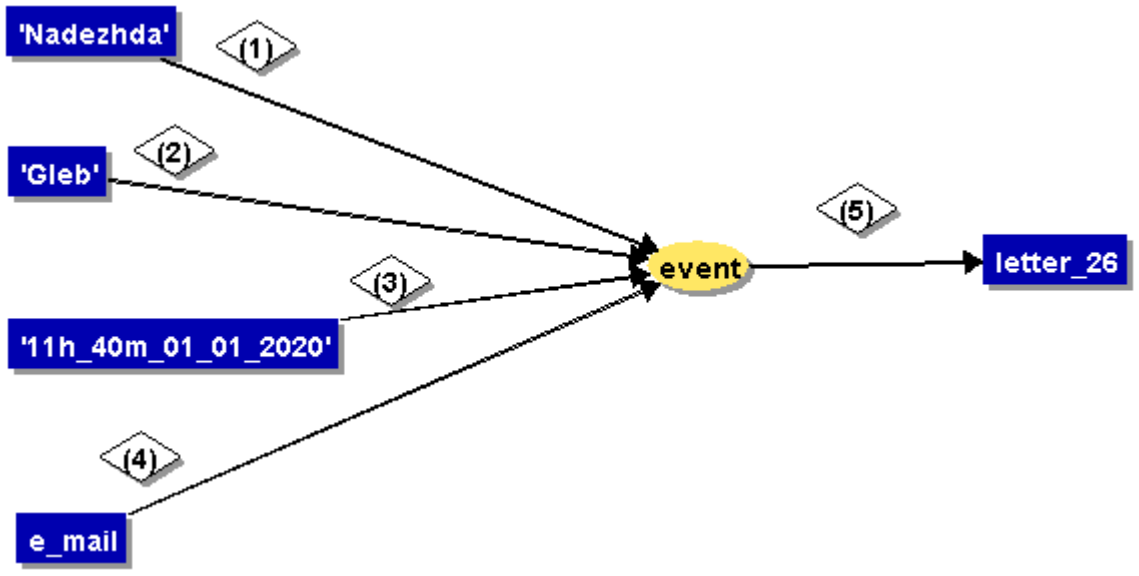


Рисунок 1.2 – Концептуальный граф для высказывания (1.3)

Высказывание (3) получено путем фиксации предметных переменных следующего 5-местного предиката:

event(sender, receiver, time, medium, letter), (1.5)

концептуальный граф которого представлен на рисунке 1.3.

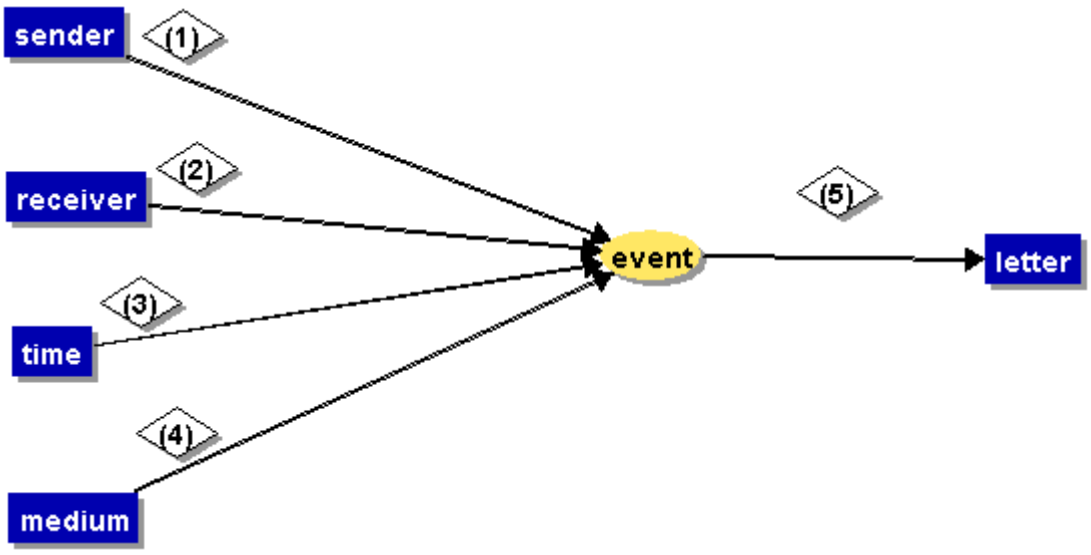


Рисунок 1.3 – Концептуальный граф для 5-местного предиката (1.5)

Совмещая имена предметных переменных и их конкретные значения, получим концептуальный граф, показанный на рисунке 1.4.

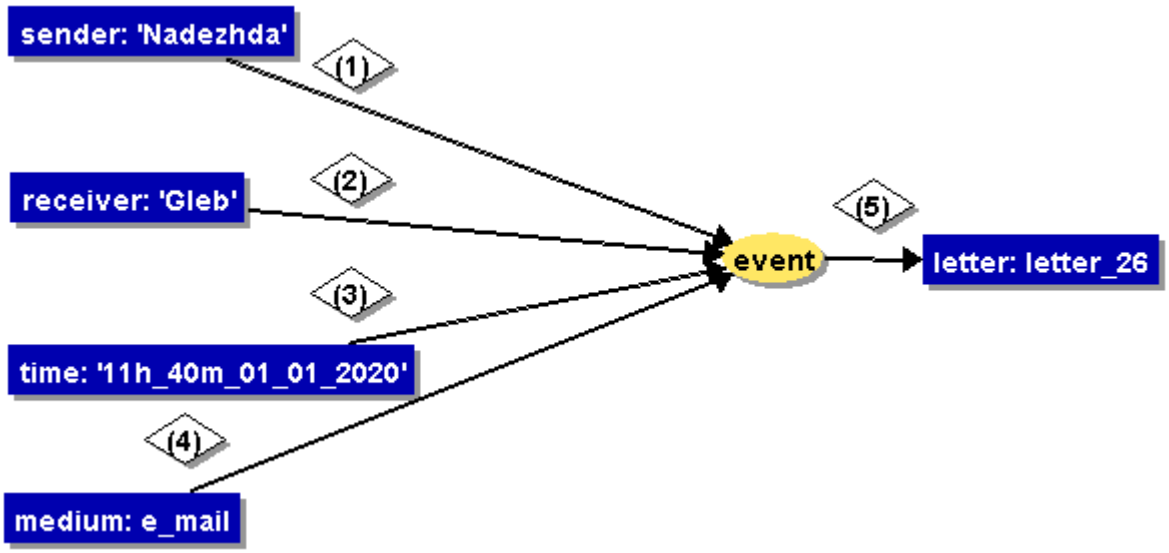


Рисунок 1.4 – Совмещенный концептуальный граф для высказывания (1.3) и 5-местного предиката (1.5)

В задачах искусственного интеллекта чаще предпочитают имеют дело с бинарными отношениями, которые используются как в концептуальных моделях, так и в семантических сетях и фреймах. Поэтому переходим к представлению той же самой фразы конъюнкцией следующих высказываний:

send(message_8, ‘Nadezhda’)&

&accept(message_8, ‘Gleb’)&

&date(message_8, ‘11h_40m_01_01_2020’)&

&means(message_8, e_mail)&

&object(message_8, letter_26), (1.6)

и далее к составному предикату, представляющему собой конъюнкцию бинарных предикатов:

send(event, sender)&

&accept(event, receiver)&

&date(event, time)&

&means(event, medium)&

&object(event, letter). (1.7)

Соответствующие выражениям (1.6) и (1.7) концептуальные графы представлены на рисунках 1.5 и 1.6. Совмещенный граф представлен рисунком 1.7.

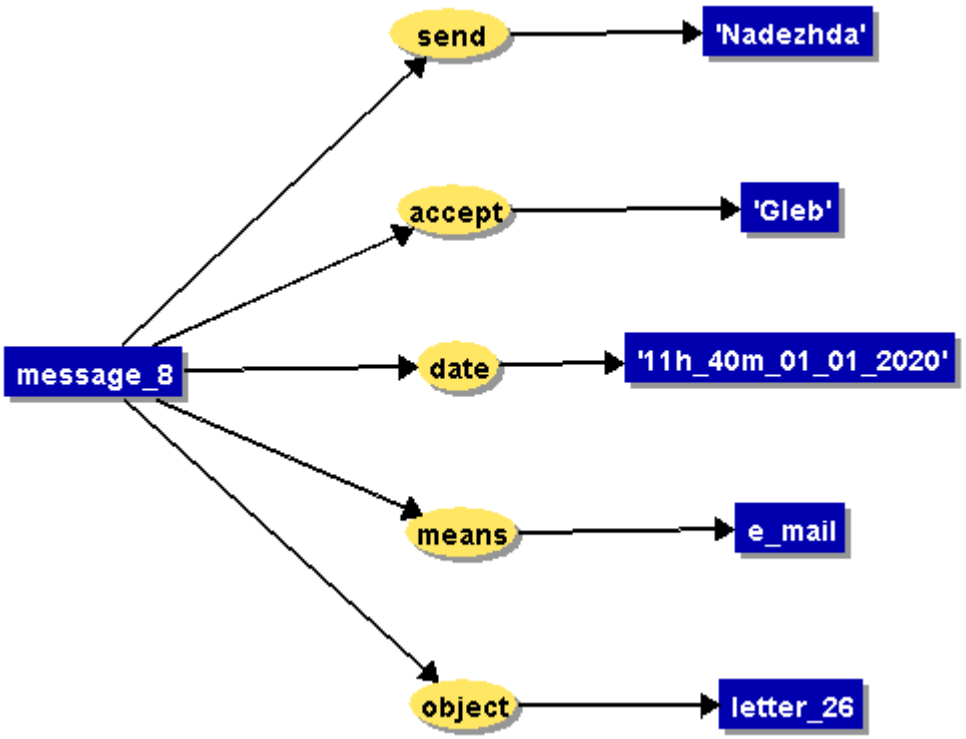


Рисунок 1.5 – Концептуальный граф, соответствующий выражению (1.6)

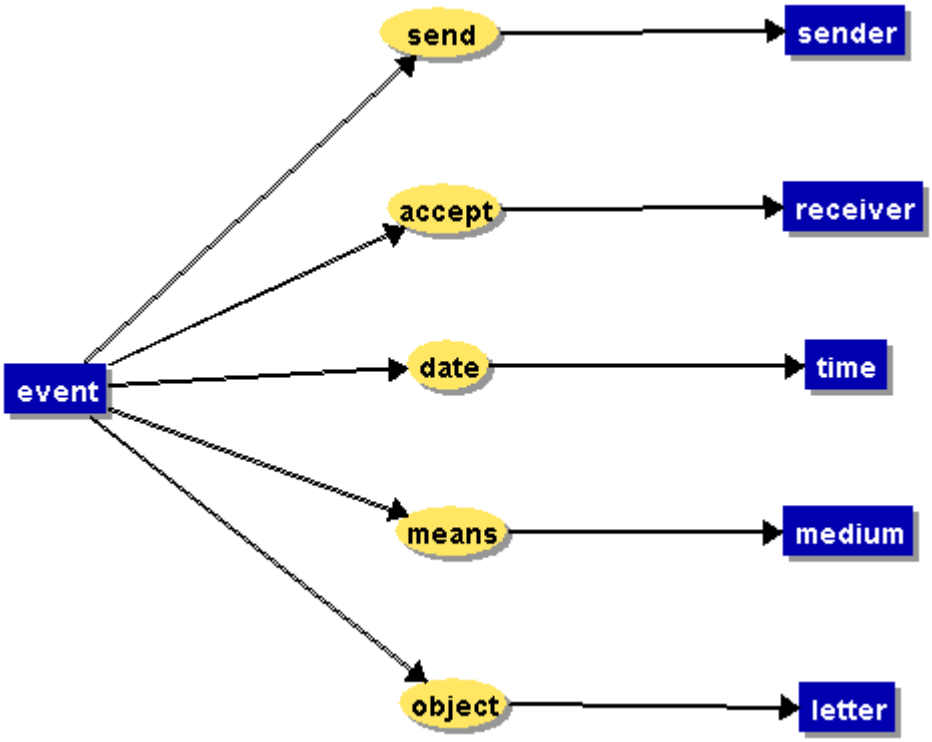


Рисунок 1.6 – Концептуальный граф, соответствующий выражению (1.7)

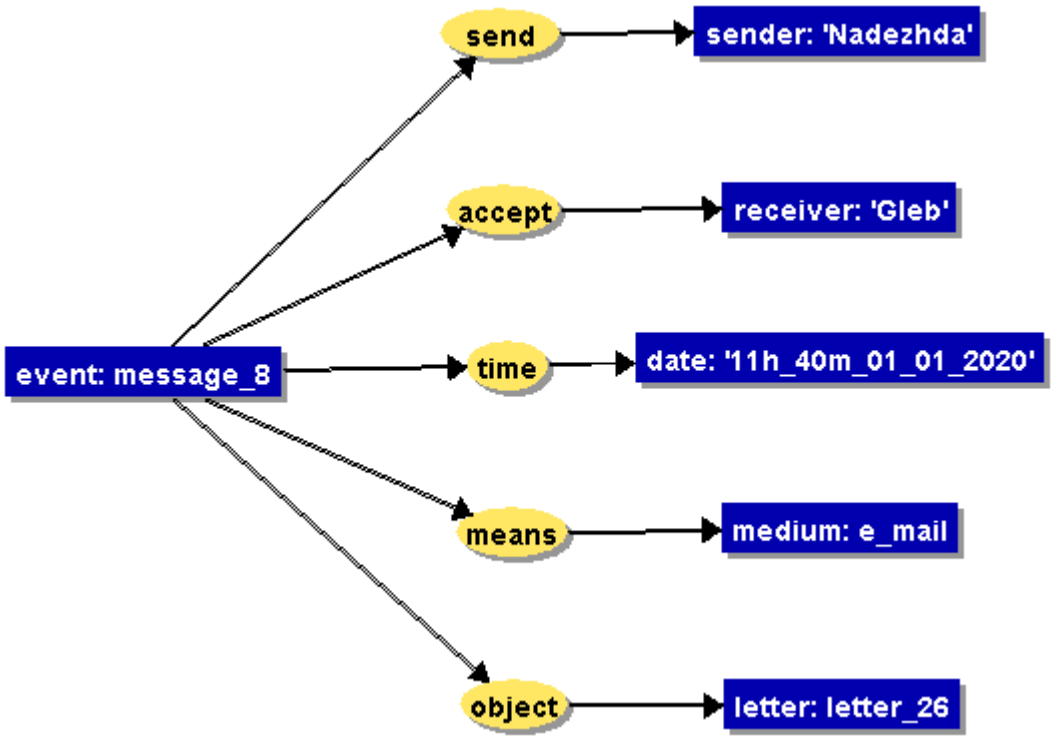


Рисунок 1.7 – Совмещенный концептуальный (канонический) граф, соответствующий выражениям (1.6) и (1.7)

Примечание. Концептуальные графы, подобные представленному на рисунке 1.7, в литературе называют еще ***каноническими графами***. С другой точки зрения, этот граф также представляет ***событийный фрейм***, где имена совокупностей, или предметные переменные, *sender, receiver, time, medium, letter* – могут интерпретироваться как имена слотов, а соответствующие им предметные константы *'Nadezhda', 'Gleb', '11h_40m_01_01_2020', e_mail, letter_26* – это значения слотов. Тогда концептуальный граф на рисунке 1.7 соответствуют ***фрейму-экземпляру***, а на рисунке 1.6 – ***фрейму-образцу***, или ***фрейму-прототипу***.

Многими исследователями в области искусственного интеллекта считается, что сети фреймов – это не что иное, как одна из форм представления семантических сетей, но соответствующие им программные инструментальные средства различны. Фреймы широко используются при проектировании экспертных систем. Кроме того, примечателен тот факт, что иерархии фреймов явились непосредственными предшественниками понятия иерархии классов в объектно-ориентированном программировании.

Для дальнейшей обработки семантической информации, получаемой при помощи концептуальных графов, выбран язык программирования на основе логики предикатов первого порядка Пролог.

На рисунке 1.8 представлен концептуальный граф, ответствующий фразе *"Nadezhda sent a letter to Gleb by e-mail"*, а на рисунке 1.9 представлены

факты и правила, соответствующие этой же фразе, причем факты определяют экстенсиональную базу данных (ЭБД), а правила – построенную на основании ЭБД интенсиональную, или виртуальную базу данных (ИБД); совместно ЭБД и ИБД образуют базу знаний (БЗ).

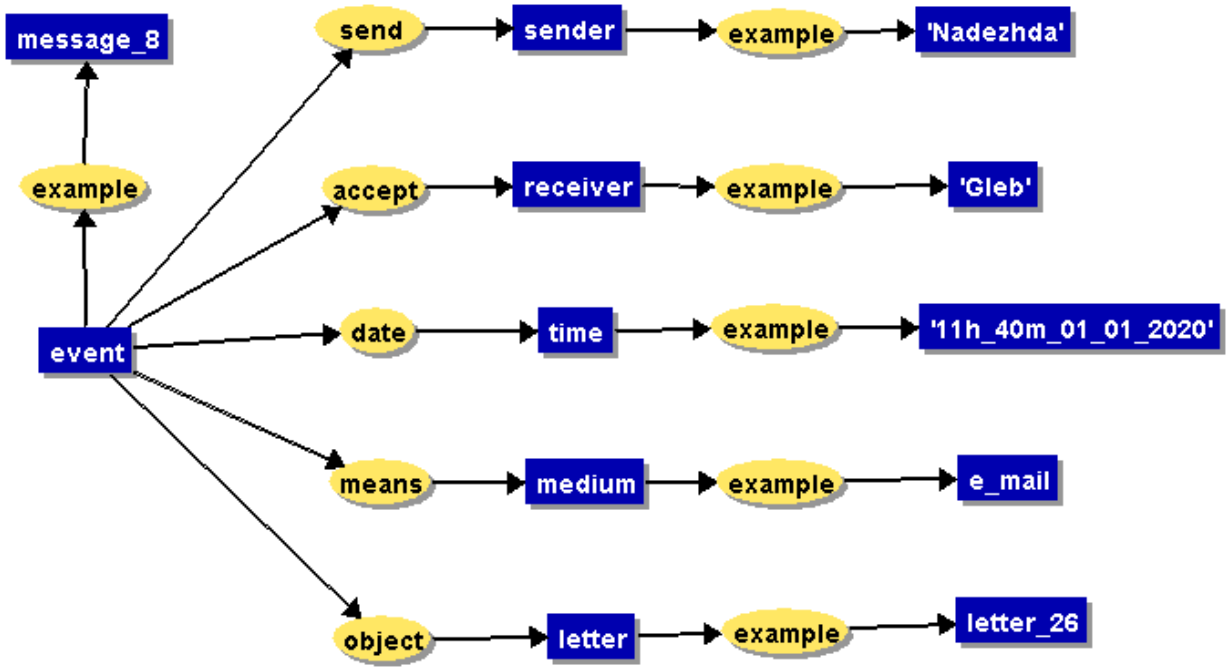


Рисунок 1.8 – Концептуальный граф, соответствующий фразе *"Nadezhda sent a letter to Gleb by e-mail"*

```
example(event,message_8).
example(letter,letter_26).
example(medium,e_mail).
example(receiver,'Gleb').
example(sender,'Nadezhda').
example(time,'11h_40m_01_01_2020').
accept(event,receiver).
accept(X,Y) :- accept(A,B),example(A,X),example(B,Y).
date(event,time).
date(X,Y) :- date(A,B),example(A,X),example(B,Y).
means(event,medium).
means(X,Y) :- means(A,B),example(A,X),example(B,Y).
object(event,letter).
object(X,Y) :- object(A,B),example(A,X),example(B,Y).
send(event,sender).
send(X,Y) :- send(A,B),example(A,X),example(B,Y).
```

Рисунок 1.9 – Экстенсиональная и интенсиональная БД, соответствующие фразе *"Nadezhda sent a letter to Gleb by e-mail"* (реализация в SWI-PROLOG)

Как обычно при построении семантических сетей (семантические сети строятся как композиции концептуальных графов), в графе нередко для уменьшения сложности рисунка определяются не все связи. Например, на рисунке 1.8 не определена непосредственная связь между концептами-индивидуумами *message_8* и *'Nadezhda'*. Однако она может быть получена неявно при помощи следующего правила:

send(X,Y) :- send(A,B),example(A,X),example(B,Y). (1.8)

Следующий пример построения концептуального графа соответствует ответной фразе:

“The pupil Gleb replied to the tutor Nadezhda by e-mail” (1.9)

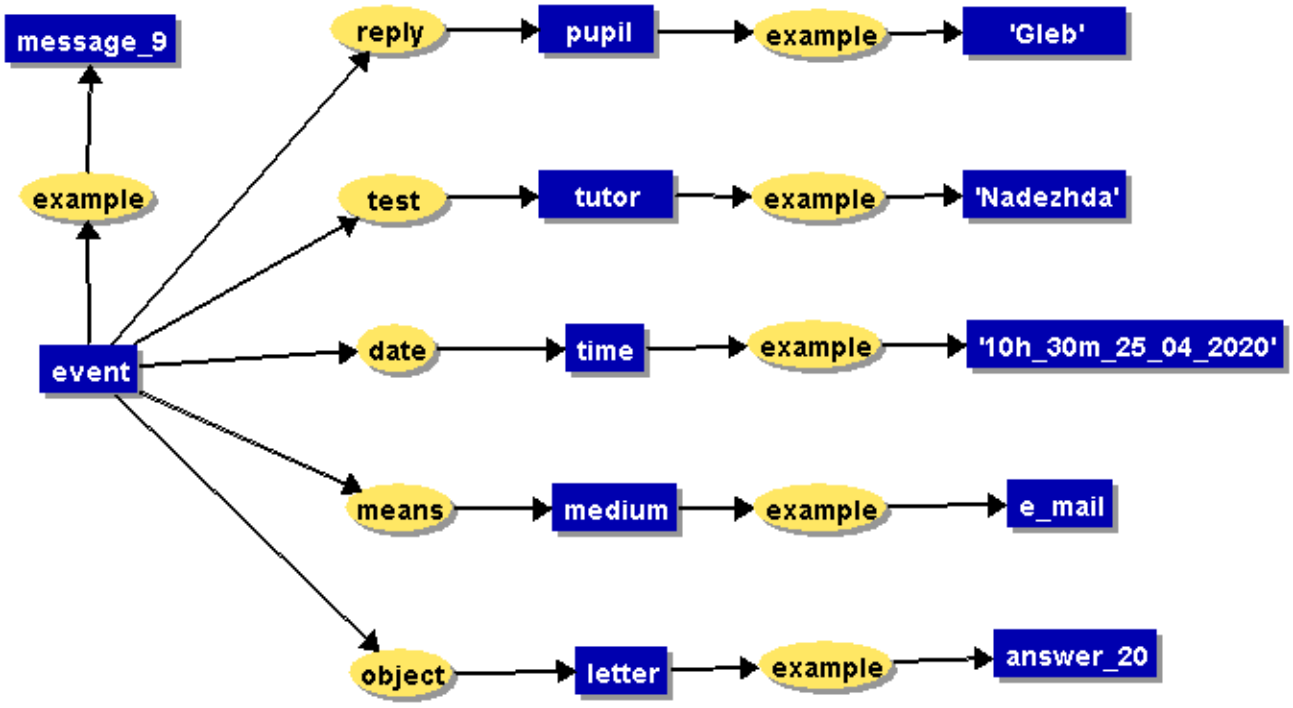


Рисунок 1.10 – Концептуальный граф для представления фразы *“The pupil Gleb replied to the tutor Nadezhda by e-mail”*

```
example(event,message_9).
example(letter,answer_20).
example(medium,e_mail).
example(tutor,'Nadezhda').
example(pupil,'Gleb').
example(time,'10h_30m_25_04_2020').
test(event,tutor).
test(X,Y):- test(A,B),example(A,X),example(B,Y).
date(event,time).
date(X,Y):- date(A,B),example(A,X),example(B,Y).
means(event,medium).
means(X,Y):- means(A,B),example(A,X),example(B,Y).
object(event,letter).
object(X,Y):- object(A,B),example(A,X),example(B,Y).
reply(event,pupil).
reply(X,Y):- reply(A,B),example(A,X),example(B,Y).
```

Рисунок 1.11 – Экстенсиональная и интенсиональная БД, соответствующие фразе *“The pupil Gleb replied to the tutor Nadezhda by e-mail”* (реализация в SWI-PROLOG)

Примечание. Рекомендуемая к использованию версия языка SWI-PROLOG русифицирована, что, естественно, является положительным фактором, поскольку многие задачи из области искусственного интеллекта на чужом языке не имеют смысла.

Порядок выполнения работы (на примере предметной области «Семейные отношения»)

Как считают многие IT-специалисты, во многих понятиях информатики трудно разобраться, не изучая готовых примеров. Поэтому далее целесообразно рассмотреть пример из одной из хорошо известных предметных областей – медицины, биологии, распределенного программирования и др. Одним из показательных примеров являются сложные и в настоящее время ставшие патриархальными семейные отношения: зять, невестка, сноха, тесть, золовка, свояченица, свекор и др.

Особенностью данного примера является отсутствие концептов-событий. Пример прост, но рисунки достаточно громоздки, и по этой причине пришлось разделить концептуальный граф на две части, определенные на одних и тех же концептах, но с отношениями двух типов: отношения “свойства”, возникшие, как в данном примере, после вступления невесты и жениха в брак (рисунок 1.12) и отношения “кровного родства” (далее просто “родства”, рисунок 1.13).

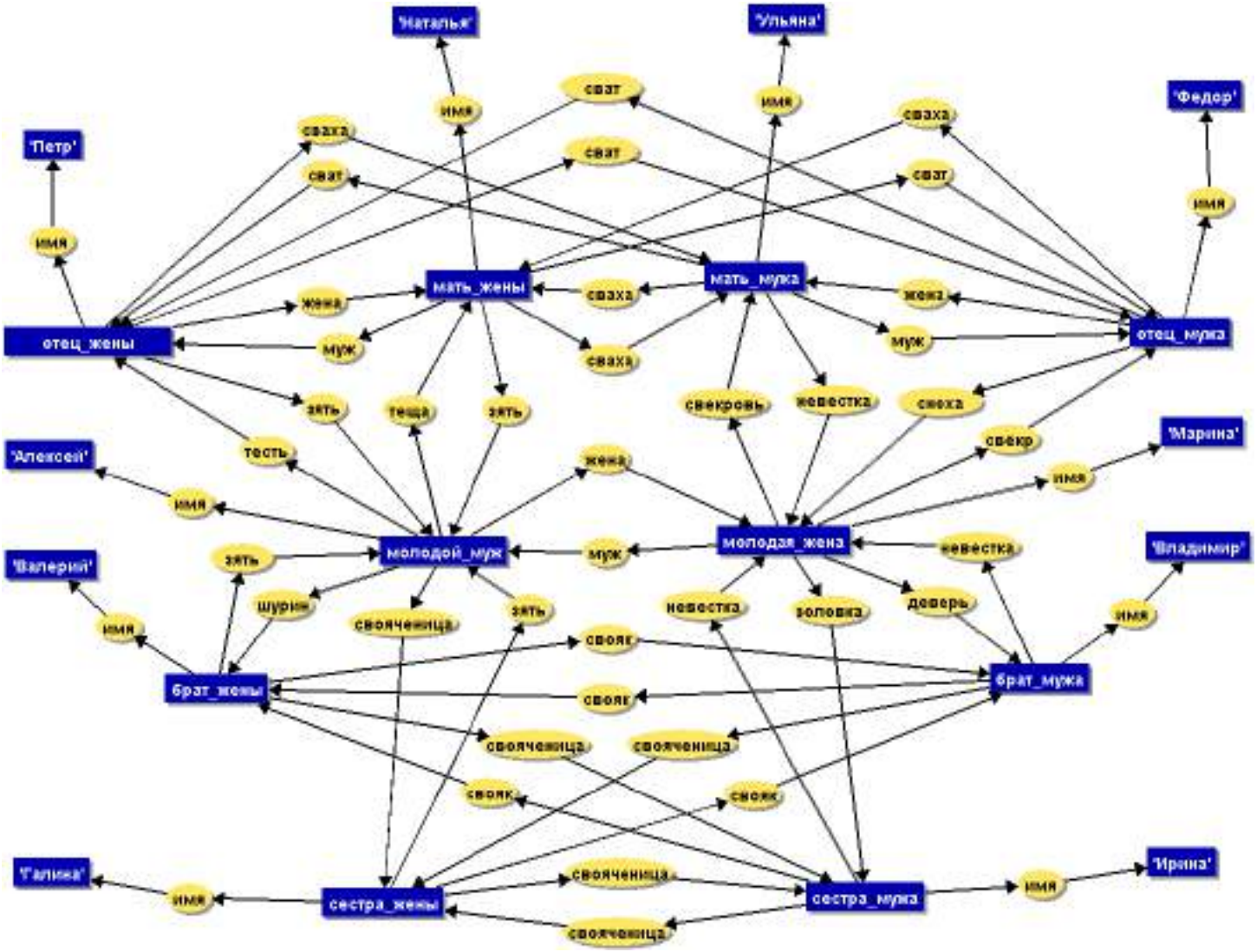


Рисунок 1.12 – Концептуальный граф “Отношения – свойства в браке”

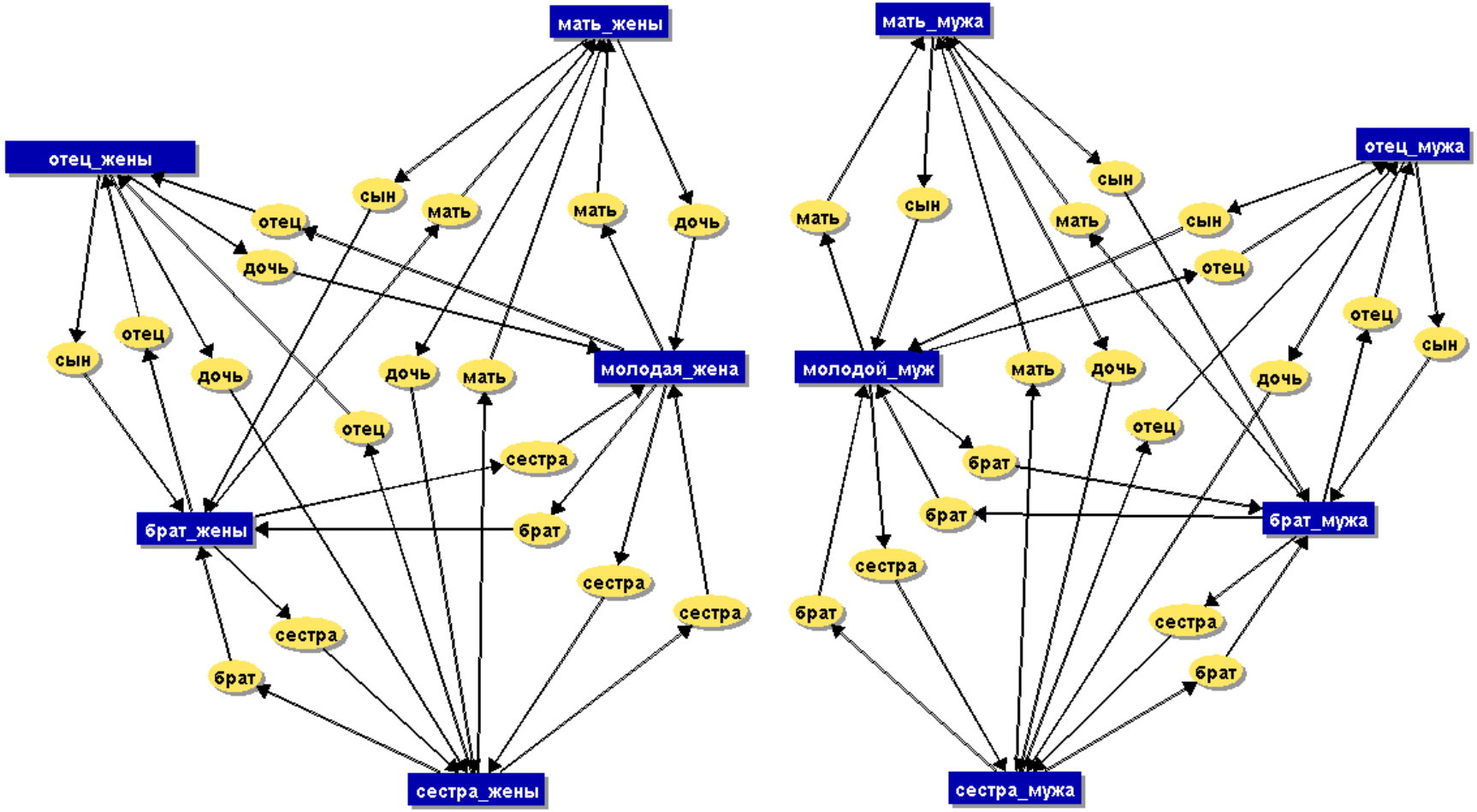


Рисунок 1.13 – Концептуальный граф “Отношения – свойства родства”

README 1. Инструкция по выполнению лабораторной работы № 1. Далее понятия «Семантическая сеть (СеС)» и «концептуальный граф (КГ)» рассматриваются как синонимы. Строго говоря, семантическая сеть представляет собой композицию концептуальных графов [4], однако во многих литературных источниках эти понятия не различают. Преподаватель представляет необходимые файлы и программы. Имеется описание редактора концептуальных графов **CharGer** на английском (амер. вариант) языке, однако редактор настолько прост, что в его использовании нетрудно (и интересно!) разобраться самостоятельно.

Примечание 1. В настоящее время в нашем распоряжении имеются два редактора концептуальных графов, получившие широкое распространение в международной практике – русифицированная версия **CharGer3.6** (далее просто **CharGer**) и более новая, но англоязычная версия **CharGer4**. Функциональные возможности данных версий несколько различны. Существуют и другие графические редакторы концептуальных графов и семантических сетей и им подобных структур [18, 19], однако редактор **CharGer** отличается возможностью вывода информации в текстовых форматах и организации перехода к списку кортежей базы знаний.

Этап 1. Текст программы для Пролога **Семья для презентации.docx** получен при помощи редактора **CharGer.jar**: открыть файл **свойства_в_браках.cgx**: **File|Open**, потом выполнить команду **Operation|Summarize everything**. Полученный в результате текстовый файл наименовать, загрузить в приложение **main4.exe** и конвертировать его в программу на Прологе. Добавить правила для формирования «интенсиональных» отношений – в тексте для программы **Семья для презентации.docx** эти правила выделены красным цветом. Программа для Пролога готова после замены расширений **.txt** или **.docx** на расширение **.pl**. Открыть полученный файл в редакторе Пролога и выполнить запросы.

Этап 2. Затем то же самое проделать с файлом **свойства_родства.cgx** и выполнять запросы независимо от первого этапа.

Примечание 2. Текст программы **семья для презентации.docx** получен путем объединения двух программ, полученных путем последовательной обработки файлов **свойства_в_браках.cgx** и **свойства_родства.cgx**.

Этап 3. Для получения единой программы на Прологе на основе исходных концептуальных графов необходимо объединить текстовые файлы, полученные в редакторе **CharGer.jar**, путем последовательного применения команды **Operation|Summarize everything** к файлам **свойства_в_браках.cgx** и **свойства_родства.cgx**. и объединения сформированных текстов продукционных правил в тестовом редакторе. После этого к объединенному файлу можно применить конвертор **main4.exe**. Факты программы будут автоматически отформатированы по правилам Пролога и отсортированы.

Напомним, что программа для Пролога окончательно готова к исполнению только после замены расширений **.txt** или **.docx** на расширение **.pl**. Промежуточные форматы **.txt** или **.docx** используются здесь на промежуточнных этапах.

Этап 4. Это дополнительный этап, расширяющий возможный состав запросов к семантической сети. Программа на Прологе **Семья для презентации+отношение.pl** получена путем дополнительного формирования фактов **отношение(X,Y,Z)** при помощи приложения **RelationConverter.jar** и добавления нового правила – выделено синим цветом. Тогда можно задавать запросы на определение отношений между концептами. Теперь поподробнее. Поставим перед собой задачу определения какого-либо отношения между концептами. Желательно, чтобы в Прологе были разрешены запросы такого типа:

? – ~~X(молодая_жена, брат мужа).~~

Однако в Прологе такие запросы невозможны, поскольку реализация этого языка основана на логике предикатов первого порядка, то есть недопустимо использование предикатных переменных: в данном примере запроса это переменная *X*. Для того, чтобы было возможно определять отношения между концептами в запросах, применим следующую методику. Пусть, например, в программе на Прологе заданы следующие факты (файл **example.pl**):

брат(сестра_жены,брат_жены).
брат(сестра_мужа,брат_мужа).
брат(брат_мужа,молодой_муж).
брат(молодая_жена,брат_жены).
брат(молодой_муж,брат_мужа).
брат(сестра_мужа,молодой_муж).
деверь(молодая_жена,брат_мужа).
дочь(мать_жены,молодая_жена).
дочь(отец_жены,молодая_жена).
дочь(мать_жены,сестра_жены).
дочь(отец_жены,сестра_жены).
дочь(мать_мужа,сестра_мужа).
дочь(отец_мужа,сестра_мужа).

Преобразуем эти строки следующим образом:

отношение(брат,сестра_жены,брат_жены).
отношение(брат,сестра_мужа,брат_мужа).
отношение (брат,брат_мужа,молодой_муж).
отношение (брат,молодая_жена,брат_жены).

отношение (брат,молодой_муж,брат_мужа).
отношение (брат,сестра_мужа,молодой_муж).
отношение (деверь,молодая_жена,брат_мужа).
отношение (дочь,мать_жены,молодая_жена).
отношение (дочь,отец_жены,молодая_жена).
отношение (дочь,мать_жены,сестра_жены).
отношение (дочь,отец_жены,сестра_жены).
отношение (дочь,мать_мужа,сестра_мужа).
отношение (дочь,отец_мужа,сестра_мужа).

Как видно из этого преобразования (его можно произвести автоматически при помощи приложения **RelationConverter.jar,** предварительно сохранив текст в формате UTF-8), бинарные предикаты заменены трехместными, или тернарными, предикатами. Прежние предикатные имена «брат», «деверь», «дочь» здесь перенесены в списки предметных констант, заключенных в скобки. Прежние имена предикатов заменены одним новым именем «отношение» (имя выбирается произвольно, лишь бы оно не использовалось ранее в Пролог-программе). Преобразование, естественно, не является эквивалентным, но оно поможет нам строить запросы на определение имени отношения между концептами. Строго говоря, описанная процедура является **переинтерпретацией** логики предикатов второго порядка (где возможны предикатные переменные) в первопорядковую логику Пролога (где допускаются только предикатные константы). Например, сформируем следующий запрос:

? – отношение(X,сестра_жены,брат_жены).

Ответом будет, конечно *X = брат*, то есть мы нашли **имя** отношения, связывающего концепты «сестра_жены» и «брат_жены» в соответствии с графом семантической сети.

Далее, для того, чтобы можно было бы определять отношения между именами, то есть между конкретными лицами с именами, к новым фактам следует добавить следующее правило:

отношение(R,X,Y):- отношение(R,A,B),имя(A,X),имя(B,Y).

Тогда будет сформировано виртуальное отношение (интенсионал) между именами людей. Становится возможным задавать следующие запросы:

? – отношение(X, ‘Галина’, ‘Валерий’).

Ответ: *X = брат*, то есть для Галины (первый концепт в концептуальном графе) Валерий (второй концепт в концептуальном графе) является братом.

Полная результирующая программа на Прологе представлена файлом **Семья для презентации+отношение.pl**, полученным после применения (см. **Примечания 1, 2**) редактора **CharGer** и двух конверторов **main4.exe** и **Relation-Converter.jar** к концептуальным графам (или к семантическим сетям), представленных файлами **свойства_в_браках.cgx** и **свойства_родства.cgx**.

Полная результирующая программа **Семья для презентации+отношение.pl** на Прологе имеет следующий вид:

```
брат(сестра_жены,брат_жены).
брат(сестра_мужа,брат_мужа).
брат(брат_мужа,молодой_муж).
брат(молодая_жена,брат_жены).
брат(молодой_муж,брат_мужа).
брат(сестра_мужа,молодой_муж).
брат(X,Y):- брат(A,B),имя(A,X),имя(B,Y).
деверь(молодая_жена,брат_мужа).
деверь(X,Y):- деверь(A,B),имя(A,X),имя(B,Y).
дочь(мать_жены,молодая_жена).
дочь(отец_жены,молодая_жена).
дочь(мать_жены,сестра_жены).
дочь(отец_жены,сестра_жены).
дочь(мать_мужа,сестра_мужа).
дочь(отец_мужа,сестра_мужа).
дочь(X,Y):- дочь(A,B),имя(A,X),имя(B,Y).
жена(молодой_муж,молодая_жена).
жена(отец_жены,мать_жены).
жена(отец_мужа,мать_мужа).
жена(X,Y):- жена(A,B),имя(A,X),имя(B,Y).
золовка(молодая_жена,сестра_мужа).
золовка(X,Y):- золовка(A,B),имя(A,X),имя(B,Y).
зять(брат_жены,молодой_муж).
зять(мать_жены,молодой_муж).
зять(отец_жены,молодой_муж).
зять(сестра_жены,молодой_муж).
зять(X,Y):- зять(A,B),имя(A,X),имя(B,Y).
имя(брат_жены,'Валерий').
имя(брат_мужа,'Владимир').
имя(мать_жены,'Наталья').
имя(мать_мужа,'Ульяна').
имя(молодая_жена,'Марина').
имя(молодой_муж,'Алексей').
имя(отец_жены,'Петр').
```

имя(отец_мужа,'Федор').
имя(сестра_жены,'Галина').
имя(сестра_мужа,'Ирина').
мать(молодая_жена,мать_жены).
мать(молодой_муж,мать_мужа).
мать(брат_жены,мать_жены).
мать(сестра_жены,мать_жены).
мать(сестра_мужа,мать_мужа).
мать(брат_мужа,мать_мужа).
мать(X,Y):- мать(A,B),имя(A,X),имя(B,Y).
муж(мать_жены,отец_жены).
муж(мать_мужа,отец_мужа).
муж(молодая_жена,молодой_муж).
муж(X,Y):- муж(A,B),имя(A,X),имя(B,Y).
невестка(брат_мужа,молодая_жена).
невестка(мать_мужа,молодая_жена).
невестка(сестра_мужа,молодая_жена).
невестка(X,Y):- невестка(A,B),имя(A,X),имя(B,Y).
отец(молодая_жена,отец_жены).
отец(молодой_муж,отец_мужа).
отец(брат_жены,отец_жены).
отец(брат_мужа,отец_мужа).
отец(сестра_жены,отец_жены).
отец(сестра_мужа,отец_мужа).
отец(X,Y):- отец(A,B),имя(A,X),имя(B,Y).
отношение(брат,сестра_жены,брат_жены).
отношение(брат,сестра_мужа,брат_мужа).
отношение(брат,брат_мужа,молодой_муж).
отношение(брат,молодая_жена,брат_жены).
отношение(брат,молодой_муж,брат_мужа).
отношение(брат,сестра_мужа,молодой_муж).
отношение(деверь, молодая_жена,брат_мужа).
отношение(дочь,мать_жены,молодая_жена).
отношение(дочь,отец_жены,молодая_жена).
отношение(дочь,отец_мужа,сестра_мужа).
отношение(дочь,отец_жены,сестра_жены).
отношение(дочь,мать_мужа,сестра_мужа).
отношение(дочь,мать_жены,сестра_жены).
отношение(жена,молодой_муж,молодая_жена).
отношение(жена,отец_жены,мать_жены).
отношение(жена,отец_мужа,мать_мужа).
отношение(золовка,молодая_жена,сестра_мужа).
отношение(зять,брат_жены,молодой_муж).
отношение(зять,мать_жены,молодой_муж).
отношение(зять,отец_жены,молодой_муж).

отношение(зять,сестра_жены,молодой_муж).
отношение(мать,молодая_жена,мать_жены).
отношение(мать,молодой_муж,мать_мужа).
отношение(мать,брат_жены,мать_жены).
отношение(мать,сестра_жены,мать_жены).
отношение(мать,сестра_мужа,мать_мужа).
отношение(мать,брат_мужа,мать_мужа).
отношение(муж,мать_жены,отец_жены).
отношение(муж,мать_мужа,отец_мужа).
отношение(муж,молодая_жена,молодой_муж).
отношение(невестка,брат_мужа,молодая_жена).
отношение(невестка,мать_мужа,молодая_жена).
отношение(невестка,сестра_мужа,молодая_жена).
отношение(отец,молодая_жена,отец_жены).
отношение(отец,молодой_муж,отец_мужа).
отношение(отец,брат_жены,отец_жены).
отношение(отец,сестра_жены,отец_жены).
отношение(отец,сестра_мужа,отец_мужа).
отношение(отец,брат_мужа,отец_мужа).
отношение(сват,мать_жены,отец_мужа).
отношение(сват,мать_мужа,отец_жены).
отношение(сват,отец_жены,отец_мужа).
отношение(сват,отец_мужа,отец_жены).
отношение(сваха,мать_жены,мать_мужа).
отношение(сваха,мать_мужа,мать_жены).
отношение(сваха,отец_жены,мать_мужа).
отношение(сваха,отец_мужа,мать_жены).
отношение(свекровь,молодая_жена,мать_мужа).
отношение(свояк,брат_жены,брат_мужа).
отношение(свояк,брат_мужа,брат_жены).
отношение(свояк,сестра_жены,брат_мужа).
отношение(свояк,сестра_мужа,брат_жены).
отношение(свояченица,брат_жены,сестра_мужа).
отношение(свояченица,брат_мужа,сестра_жены).
отношение(свояченица,молодой_муж,сестра_жены).
отношение(свояченица,сестра_жены,сестра_мужа).
отношение(свояченица,сестра_мужа,сестра_жены).
отношение(свёкр,молодая_жена,отец_мужа).
отношение(сестра,брат_жены,сестра_жены).
отношение(сестра,брат_мужа,сестра_мужа).
отношение(сестра,молодая_жена,сестра_жены).
отношение(сестра,молодой_муж,сестра_мужа).
отношение(сестра,сестра_жены,молодая_жена).
отношение(сестра,брат_жены,молодая_жена).
отношение(сноха,отец_мужа,молодая_жена).

отношение(сын,мать_мужа,молодой_муж).
отношение(сын,отец_мужа,молодой_муж).
отношение(сын,отец_жены,брат_жены).
отношение(сын,мать_мужа,брат_мужа).
отношение(сын,мать_жены,брат_жены).
отношение(сын,отец_мужа,брат_мужа).
отношение(тесть, молодой_муж,отец_жены).
отношение(теща,молодой_муж,мать_жены).
отношение(шурин,молодой_муж,брат_жены).
отношение(R,X,Y):- отношение(R,A,B),имя(A,X),имя(B,Y).
сват(мать_жены,отец_мужа).
сват(мать_мужа,отец_жены).
сват(отец_жены,отец_мужа).
сват(отец_мужа,отец_жены).
сват(X,Y):- сват(A,B),имя(A,X),имя(B,Y).
сваха(мать_жены,мать_мужа).
сваха(мать_мужа,мать_жены).
сваха(отец_жены,мать_мужа).
сваха(отец_мужа,мать_жены).
сваха(X,Y):- сваха(A,B),имя(A,X),имя(B,Y).
свекровь(молодая_жена,мать_мужа).
свекровь(X,Y):- свекровь(A,B),имя(A,X),имя(B,Y).
свояк(брат_жены,брат_мужа).
свояк(брат_мужа,брат_жены).
свояк(сестра_жены,брат_мужа).
свояк(сестра_мужа,брат_жены).
свояк(X,Y):- свояк(A,B),имя(A,X),имя(B,Y).
свояченица(брат_жены,сестра_мужа).
свояченица(брат_мужа,сестра_жены).
свояченица(молодой_муж,сестра_жены).
свояченица(сестра_жены,сестра_мужа).
свояченица(сестра_мужа,сестра_жены).
свояченица(X,Y):- свояченица(A,B),имя(A,X),имя(B,Y).
свёкр(молодая_жена,отец_мужа).
свёкр(X,Y):- свёкр(A,B),имя(A,X),имя(B,Y).
сестра(брат_жены,сестра_жены).
сестра(брат_мужа,сестра_мужа).
сестра(брат_жены,молодая_жена).
сестра(молодая_жена,сестра_жены).
сестра(молодой_муж,сестра_мужа).
сестра(сестра_жены,молодая_жена).
сестра(X,Y):- сестра(A,B),имя(A,X),имя(B,Y).
сноха(отец_мужа,молодая_жена).
сноха(X,Y):- сноха(A,B),имя(A,X),имя(B,Y).
сын(мать_мужа,молодой_муж).

сын(мать_мужа,брат_мужа).
сын(отец_мужа,молодой_муж).
сын(отец_мужа,брат_мужа).
сын(мать_жены,брат_жены).
сын(отец_жены,брат_жены).
сын(X,Y):- сын(A,B),имя(A,X),имя(B,Y).
тесть(молодой_муж,отец_жены).
тесть(X,Y):- тесть(A,B),имя(A,X),имя(B,Y).
теща(молодой_муж,мать_жены).
теща(X,Y):- теща(A,B),имя(A,X),имя(B,Y).
шурин(молодой_муж,брат_жены).
шурин(X,Y):- шурин(A,B),имя(A,X),имя(B,Y).

Примечание 3. Для получения единой программы на Прологе на основе исходных концептуальных графов необходимо предварительно объединить текстовые файлы, полученные в редакторе **CharGer.jar**, путем последовательного применения команды **Operation|Summarize everything** к файлам **свойства_в_браках.cgx** и **свойства_родства.cgx**. После этого можно применять конверторы **main4.exe** (в результате будет получен отсортированный список фактов для программы на Прологе) и **RelationConverter.jar.**

Текст результирующей Пролог-программы представлен файлом **Семья_для_презентации+отношение.docx**, а сама программа, исполняемая в Прологе (версия **SWI-Prolog**) представлена файлом **семья_2_новая+.pl**.

Приложения в электронном виде: все упомянутые программы и файлы.

Лабораторное задание

Изучить и повторить решение задач, приведенных в описании данной лабораторной работы, используя частично готовые файлы. Оформить отчет о проделанной работе.

Содержание отчета

Отчет должен содержать концептуальные графы и соответствующие им программы на языке SWI-Prolog.

Задания на самостоятельную работу

1. Приведите примеры на построение семантических сетей с отношениями следующего вида (результаты выполнения этого задания потребуются для выполнения лабораторного задания к лабораторной работе № 2):

1) таксономические («класс – подкласс – экземпляр», «множество –

подмножество – элемент» и т.п.);
2) структурные («часть – целое»);
3) родовые («предок» - «потомок»);
4) производственные («начальник» - «подчиненный»);
5) функциональные (определяемые обычно глаголами «производит», «влияет» и т.п.);
6) количественные (больше, меньше, равно и т.п.);
7) пространственные (далеко от, близко от, за, под, над и т.п.);
8) временные (раньше, позже, в течение и т.п.);
9) атрибутивные (иметь свойство, иметь значение);
10) логические (И, ИЛИ, НЕ);
11) каузальные (причинно-следственные).

2. Реализовать в виде семантической сети классификацию в одной из предметных областей:

1) распределенных СУБД;
2) интернет-провайдеров;
3) систем контроля знаний;
4) систем искусственного интеллекта;
5) систем поддержки принятия решений;
6) мобильных телефонов;
7) автомобилей;
8) самолётов (вертолётов);
9) садовых растений;
10) лекарственных препаратов;
11) видов спорта;
12) профессий;
13) природных ресурсов;
14) управленческих решений;
15) web-сайтов;
16) фреймворков;
17) облачных систем;
18) грид-систем;
19) распределенных вычислительных систем;
20) локальных вычислительных сетей;
21) глобальных вычислительных сетей;
22) сетевого оборудования;
23) сетевых протоколов;
24) типов компьютеров;
25) фирм-производителей компьютеров;
26) языков программирования.

Лабораторная работа №2
Построение и анализ концептуальных моделей вычислительных и киберфизических систем

Цель работы: овладение основными навыками построения концептуальных моделей представления знаний в вычислительных и киберфизических системах и использования механизмов логического вывода языка SWI-PROLOG.

Основные понятия и определения

Рассматриваются способы представления экстенсиональных и интенсиональных знаний о предметной области средствами языка Пролог (PROLOG, реализация интерпретатора SWI-PROLOG) .

При выполнении ряда лабораторных работ далее потребуются знания некоторых особенностей языка Пролог. В связи с тем, что декларативные языки программирования, к числу которых относится и Пролог, изучаются, как правило, на младших курсах, потребовалось включение в данное учебно-методическое пособие дополнительных сведений, касающихся интеграции логического программирования и баз данных и использования Пролога при разработке концептуальных моделей и экспертных систем. При подготовке материала о реализации и использовании интерпретатора SWI-Prolog использована литература [8, 10, 16, 21].

Интеграция средств логического программирования и баз данных

В настоящее время уделяется большое внимание элементам интеграции логического программирования и баз данных, которая позволяет образовать системы нового типа, расширяющие границы информатики как науки и позволяющие удовлетворять требованиям новых применений. Для таких систем используются следующие названия:

а) дедуктивная база данных (отражает использование стиля логического программирования для реализации дедукции на основе содержимого базы данных);

б) система управления базами знаний (отражает управление сложными знаниями, а не простыми данными);

в) экспертная система баз данных (отражает использование опыта экспертов в определенной прикладной области для решения класса задач при сохранении доступа к большой базе данных).

Слияние логических программ и баз данных происходит в соответствии с общей тенденцией в информатике – совместное исследование различных областей с целью генерации новых идей и получения материального выигрыша за счет использования общих концепций.

Логическое программирование и базы данных развивались одновременно. Пролог – самый популярный язык ПРОграммирования в ЛОГике – появился как результат упрощения более общих методов доказательства теорем для достижения возможности эффективного программирования. Аналогично реляционная модель данных возникла в результате упрощения сложных иерархической и сетевой моделей для обеспечения возможности непроцедурного манипулирования множественными данными. Пролог и реляционные базы данных получили широкое распространение не только в академической или научной среде, но и в деловом мире.

О реализации языка SWI-Prolog

SWI-Prolog – это открытая реализация языка программирования Prolog, часто используемая для преподавания и приложений Semantic Web. Эта реализация предоставляет богатый набор возможностей, библиотеки для программирования на основе логики, многопоточности, тестирования, графический пользовательский интерфейс GUI, интерфейс к языку программирования Java, интерфейс ODBC, содержит реализацию веб-сервера, библиотеки для SGML, RDF, RDFS (SGML – англ. Standard Generalized Markup Language – стандартный обобщённый язык разметки, или метаязык, на котором можно определять язык разметки для документов; RDF – англ. Resource Description Framework – среда описания ресурса; RDFS – англ. RDF Schema – схема

RDF), средства разработчика, включая интегрированную среду разработки IDE (англ. Integrated Development Environment) и обширную документацию.

SWI-Prolog обладает богатым интерфейсом с другими IT-компонентами за счет поддержки многих типов файлов и сетевых протоколов, а также за счет комплексного низкоуровневого интерфейса для языка C, который является основой для высокоуровневых интерфейсов для C++, Java, C#, Python и др. SWI-Prolog работает на платформах Unix, Windows, и Macintosh и постоянно развивается. Полные сведения о реализации языка SWI-Prolog содержатся в документации [16], а также приведены на сайтах производителя (https://www.swi-prolog.org/features.html и http://www.swi-prolog.org).

Общие принципы логического программирования и управления базами данных [21]

Изучение логического программирования и управления базами данных позволяет обнаружить следующие общие их черты:

а) ***Базы данных.*** Системы логического программирования управляют небольшими однопользовательскими базами данных, располагаемыми в оперативной памяти и содержащими правила вывода и фактическую информацию. Системы баз данных, напротив, управляют большими, находящимися в коллективном пользовании совокупностями данных в массово памяти и обеспечивают технологию поддержки эффективного поиска и надежного изменения долговременно хранимых данных.

б) ***Запросы.*** Запрос обозначает процесс, с помощью которого релевантная информация извлекается из базы данных. В логическом программировании запрос (или цель) реализуется построением цепочек логических выводов, комбинирующих правила и фактическую информацию для доказательства истинности или отрицания справедливости первоначального утверждения. В системах баз данных запрос (выражаемый с помощью специализированного языка манипулирования данными) обрабатывается при использовании наиболее

эффективных путей доступа в массовой памяти к большим совокупностям данных для извлечения релевантной информации.

в) ***Ограничения.*** Ограничения задают условия правильности баз данных. Проверка ограничения – процесс, с помощью которого обеспечивается правильность базы данных и предотвращается помещение неправильных данных в базу данных. В логическом программировании ограничения выражаются с помощью универсальных правил, активизируемых при модификации базы данных. В системах баз данных лишь небольшое число типовых ограничений обычно выражается на языке определения данных.

Логическое программирование обладает большей выразительной способностью задания запросов и ограничений в сравнении с языками определения данных и манипулирования данными систем баз данных. Более того, представление запросов и ограничений возможно при этом в однородном формализме, и их проверка требует одних и тех же механизмов вывода, что позволяет вести более сложные рассуждения о содержимом базы данных. Вместе с тем системы логического программирования не обеспечивают технологии управления большими, надежными, долговременно хранимыми совокупностями данных с коллективным доступом.

Естественное расширение логического программирования и управления базами данных заключается в построении новых классов систем, расположенных на пересечении этих двух областей. Такие системы основаны на использовании логического программирования в качестве языка запросов и соединяют формулирование запросов и ограничений в стиле логического программирования с технологией баз данных для эффективного и надежного запоминания данных в массовой памяти.

Правила Пролога можно использовать для построения ***интенсиональной*** базы данных (ИБД) на основе ***экстенсиональной*** базы данных (ЭБД). ЭБД - это просто реляционная база данных.

Примечание. Пролог возвращает результат по одному кортежу, а не как множество результирующих кортежей (для получения всех кортежей в Прологе необходимо нажимать клавишу пробела).

Пролог представляет собой мощный формализм для выражения запросов и алгоритмов. Если задачам присуща рекурсия, то они не могут быть реализованы с помощью стандартных языков запросов. В дополнительной литературе определены архитектура и механизмы, необходимые для связывания механизма Пролога и системы управления базой данных. Для простоты выбрана реляционная модель данных, поскольку многие существующие интерфейсы между Прологом и базами данных в настоящее время используют реляционные базы данных. Реляционная модель данных и соответствующий ей язык являются наиболее удобными средствами для выражения свойств этих интерфейсов. Данные методы связывания легко распространить на нереляционные системы. Некоторые существующие прототипы связанных систем разработаны в основном в промышленных и академических центрах.

Некоторые важные свойства Пролога как языка запросов [21]

Ранее были рассмотрены основные причины для интеграции технологии баз данных и логического программирования, установлено соответствие между основными чертами реляционных баз данных и языков логического программирования. Важные свойства Пролога как языка запросов существенно увеличивают выразительную мощность классических языков запросов, основанных на реляционной алгебре или исчислении:

- ***Интенсиональное определение базы данных.*** Правила в языке Пролог являются мощным средством для определения и использования отношений, полученных путем логического вывода. Множество выведенных отношений, определенных с помощью правил Пролога, образуют *интенсиональную* базу данных (ИБД).
- ***Рекурсия.*** Предикаты и правила в Прологе могут задаваться рекурсивно. Поэтому выразительная мощность Пролога как языка запросов строго

выше выразительных возможностей традиционных языков запросов.

– ***Неполная информация.*** Можно отмечать несущественные значения в правилах и фактах программы с помощью анонимной переменной.
– ***Упорядоченность правил и фактов в базе***. Программист, использующий Пролог, может упорядочить факты и правила наиболее подходящим образом с точки зрения вычислительной эффективности. Механизм Пролога чувствителен к порядку правил и фактов.
– ***Управление выводом***. Пролог позволяет программисту увеличить эффективность программ с помощью некоторых процедурных средств, таких, как предикат отсечения.
– ***Отрицания.*** Пролог позволяет реализовать отрицания. Например, такие литералы, как *not(has(leo, strips))* являются допустимыми в правой части правила.

Некоторые необходимые для выполнения лабораторных работ сведения о языке PROLOG и о работе в среде интерпретатора SWI-PROLOG

Программа на Прологе состоит из фактов и правил, которые образуют базу знаний Пролог-программы, и запроса к этой базе, который задает цель поиска решений.

Предикат описывают отношение между объектами, которые являются аргументами предиката.

Факты констатируют наличие заданного предикатом отношения между указанными объектами.

Программирование на языке Пролог состоит из следующих этапов:

1) объявления некоторых фактов об объектах и отношениях между ними;

2) определения некоторых правил об объектах и отношениях между ними;

3) формулировки вопросов об объектах и отношениях между ними.

В SWI-PROLOG различаются строчные и прописные буквы.

1. Факты

Предположим, что надо сформулировать на Прологе факт, что «Оператор выполняется на узле_сети». Этот факт включает в себя два объекта «оператор» и «узел_сети», связанные отношением «выполняется». В языке Пролог используется стандартная форма записи фактов:

выполняется(оператор, узел сети).

Важно соблюдать при записи фактов следующие правила:

− имена всех отношений и объектов (предметных констант) должны начинаться со строчной буквы. Например, *выполняется*, *оператор*, *узел_сети*;

− сначала записывается имя отношения, затем в круглых скобках через запятую записываются имена объектов;

− каждый факт должен заканчиваться точкой;

− определяя с помощью фактов отношения между объектами, необходимо учитывать порядок перечисления имен объектов внутри круглых скобок; этот порядок может быть произвольным, но, выбрав один раз определенный порядок, необходимо следовать ему и дальше.

Имена объектов (концептов), список которых в каждом факте заключен в круглые скобки, называют ***аргументами***. ***Имя отношения***, которое записывается непосредственно перед скобками, называется также ***именем предиката***. Строго говоря, отношение является областью истинности одноименного предиката.

2. Запросы

Имея некоторую совокупность фактов, мы можем обращаться к Прологу с вопросами о них. В Прологе вопрос записывается почти так же, как и факт, за исключением того, что перед ним ставится специальный символ « ?- ». Например:

?- выполняется(оператор, узел_сети).

Обращение к Прологу с вопросом инициирует процедуру поиска в базе данных, ранее введенной в систему. Пролог ищет факты, сопоставимые с фактом в запросе. Два факта сопоставимы (или соответствуют один другому), если их предикаты одинаковы (побуквенное совпадение) и их соответствующие

аргументы попарно совпадают. Если Пролог находит факт, сопоставимый с вопросом, то он отвечает ***true***. Если в базе данных такого факта не существует – то ***false***.

Все запросы должны заканчиваться точкой. Если вы забудете ее поставить, то Пролог выведет символ '|' и будет ожидать дальнейшего ввода. В этом случае надо ввести точку и нажать клавишу ***Enter***.

3. Переменные

Для того, чтобы обеспечить поиск ответов на более сложные вопросы, например, «Что выполняется на узле_сети?» в Прологе используются ***переменные.*** В Прологе можно не только присваивать имена конкретным данным, но и использовать имена, подобные *X* (неизвестное), для обозначения объектов, которые должны быть определены системой. Имена такого типа называются переменными. В Прологе используется соглашение, согласно которому каждое имя, начинающееся с прописной буквы, рассматривается как переменная. При поиске ответа на вопрос Пролог организует просмотр всех фактов в базе данных, чтобы обнаружить объект, который эта переменная могла бы обозначать. Так при вопросе «выполняется ли на узле_сети *X*?», программа просматривает все известные ему факты для обнаружения тех объектов, которые могут обрабатываться на узле_сети. Такая переменная, как *X*, сама по себе не является именем какого-то конкретного объекта, но она может быть использована для обозначения объектов, для которых пока не не определено имя.

Переменная в SWI-PROLOG обозначается как последовательность латинских букв, кириллицы и цифр, начинающаяся с заглавной буквы.

Если значение или имя предметной константы – аргумента предиката начинается с заглавной буквы, то оно должно заключаться в апострофы:

звание('Александр', майор). % Александр имеет звание майора.

Комментарий в строке программы начинается с символа % и заканчивается концом строки. Блок комментариев выделяется специальными скобками: /* *(начало)* и **/ (конец)*.

Имена предикатов и объектов (концептов) должно начинаться со строчной буквы и может содержать латинские буквы, кириллицу, цифры и символ подчеркивания (_). Кириллица используется наравне с латинскими буквами. Обычно предикатам дают такие имена, чтобы они отражали смысл отношения. Два предиката могут иметь одинаковые имена, тогда система распознает их как разные предикаты, если они имеют различное число аргументов (арность). Например, *выполняется/2*, *выполняется/3*.

Имя предиката может совпадать с именем какого-либо встроенного предиката SWI-PROLOG. Однако, если совпали имена пользовательского и встроенного предиката, то при обращении к нему (либо из интерпретатора, либо из программы), будет вызван пользовательский предикат, то есть пользовательское определение «перекроет» предопределенное в SWI-PROLOG.

4. Правила

Правила описывают связи между предикатами. Правило *B:-A* соответствует импликации $A \rightarrow B$ («ЕСЛИ *A* , ТО *B*»).

В общем виде правило – это конструкция вида:

P0:-P1,P2,…,Pn.

которая читается «*P0 истинно, если P1 и P2 и ... Pn истинны*".

Предикат *P0* называется заголовком правила, выражение *P1,P2,…,Pn* – телом правила, а предикаты *Pi* – подцелями правила. Запятая означает логическое "И". Точка с запятой означает логическое "ИЛИ".

Факты и ***правила*** называются также ***утверждениями*** или ***клозами***. Факт можно рассматривать как правило, имеющее заголовок и пустое тело.

5. Процедура – это совокупность утверждений, заголовки которых имеют одинаковый функтор и одну и ту же арность. Процедура задает определение предиката. Конец предложения всегда отмечается точкой, поэтому все факты, правила и запросы должны заканчиваться точкой. Заметим также, что между именем предиката и скобкой не должно быть пробелов.

6. Унификация – это процесс получения свободной переменной значения в результате сопоставления при логическом выводе в SWI-PROLOG.

Набор текста программы и ее выполнение

Для набора текста программы используется встроенный текстовый редактор. Для создания нового файла используется команда **File/New**, в диалоговом окне следует указать имя нового файла. Для редактирования уже созданного файла с использованием встроенного редактора можно воспользоваться командой меню **File/Edit**. После набора программы следует ее сохранить, выполнив команду **File/Save buffer**.

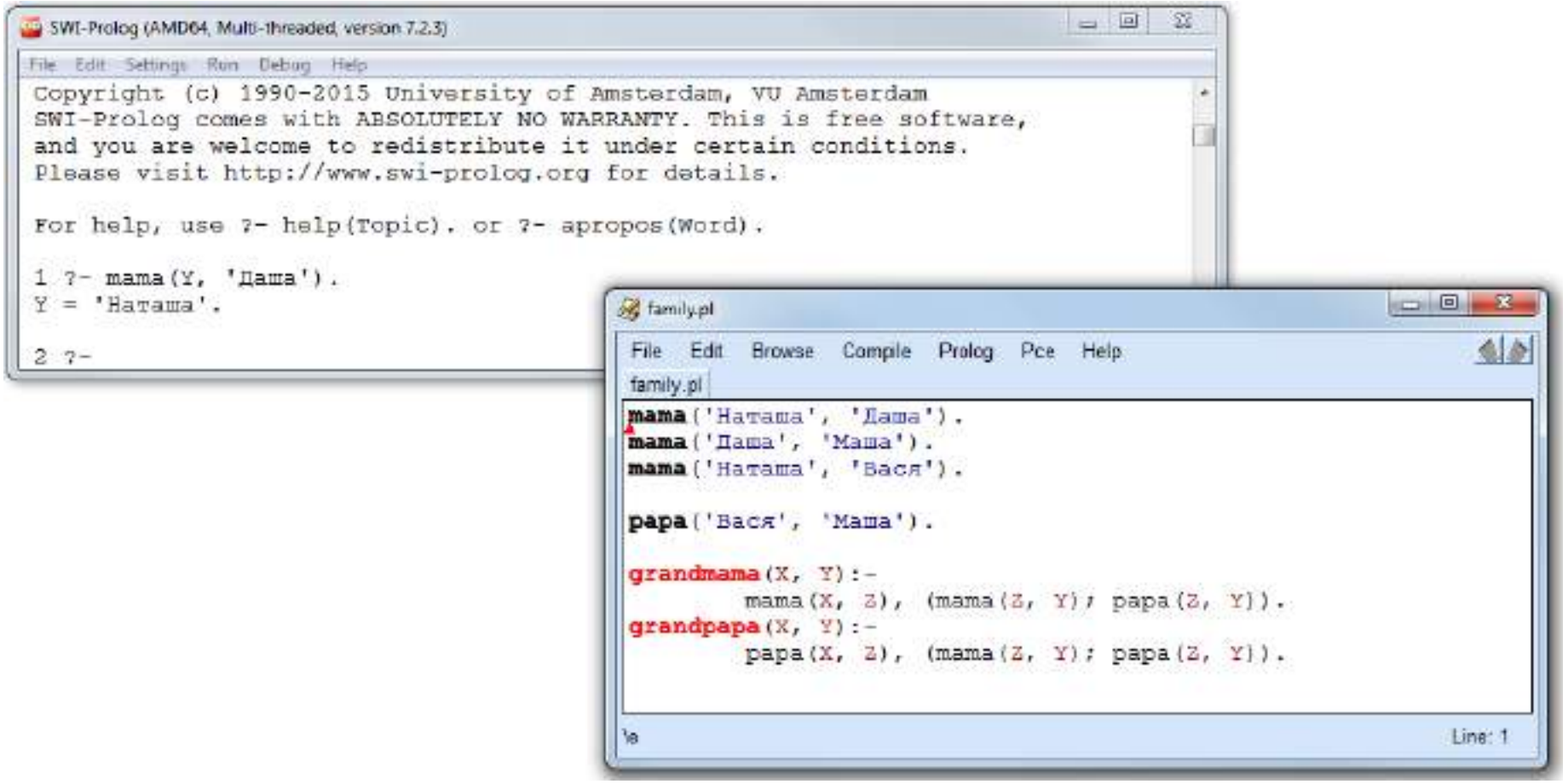
Красным цветом подсвечиваются предикаты в заголовках предложений, которые с точки зрения синтаксиса SWI-PROLOG корректны. Указатель "*курсор*" можно использовать для выверки (например, корректности) расстановки скобок. Зелёным цветом выделяются комментарии, темно-красным цветом – переменные. Подчеркиванием выделяются предикаты в теле правила, которые совпадают с предикатом заголовка: таким образом акцентируется внимание на возможном зацикливании программы.

Чтобы запустить программу, сначала необходимо ее загрузить в SWI-PROLOG для выполнения. Это делается выбором опции **Compile/Compile buffer** из окна редактора. Результат отображается в окне интерпретатора SWI-PROLOG. Там же указываются ошибки, возникшие на данном этапе; чаще всего они отображаются и во всплывающем окне ошибок. Обычно перед компиляцией предлагается сохранить файл.

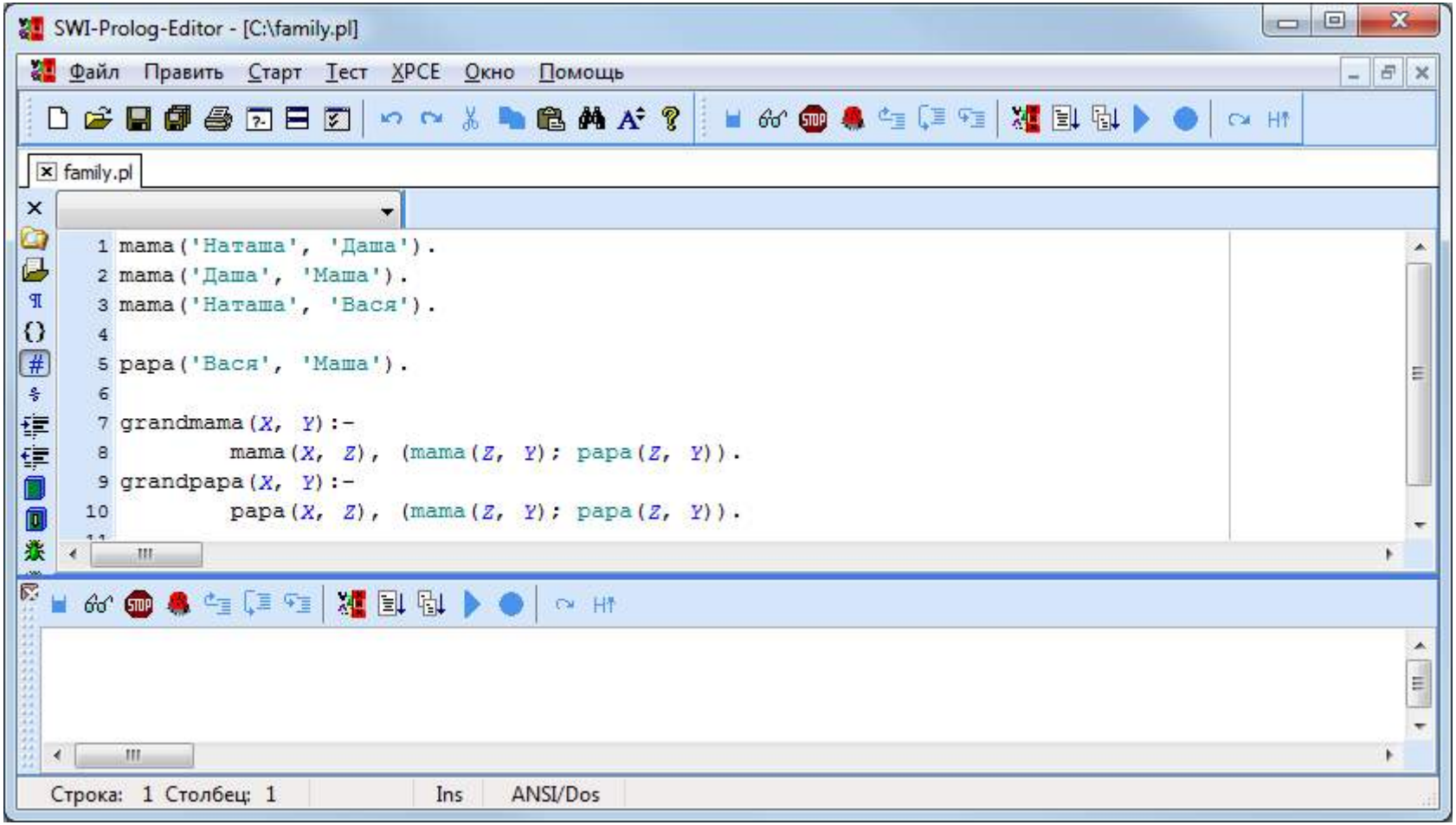
Если необходимо продолжить поиск в базе по этому же запросу и получить альтернативные решения, то вводится точка с запятой « ; ».

Набор и исполнение программы в среде SWI-PROLOG возможны в разных вариантах [9]:

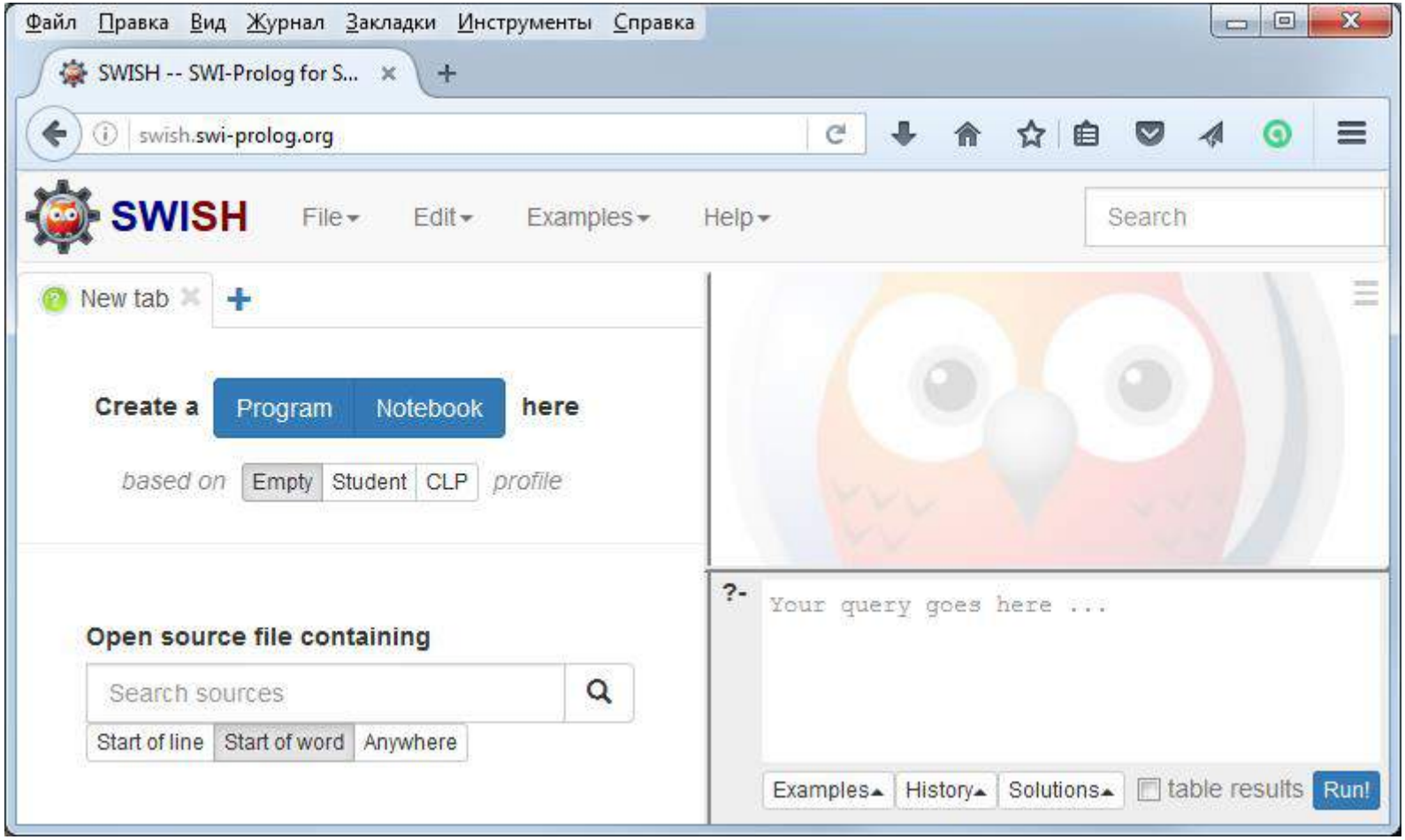
− при помощи стандартной offline-среды программирования (http://www.swi-prolog.org):



– при помощи offline-среды программирования SWI-Prolog-Editor (http://lakk.bildung.hessen.de/netzwerk/faecher/informatik/swiprolog/indexe.html):



− при помощи стандартной online-среды программирования SWI-PROLOG (http://swish.swi-prolog.org):



Примеры решения задач

Задача 1. Составить программу на языке Пролог для нахождения неизвестных отношений в концептуальном графе, представленном на рисунке 2.1. Здесь считается, что первоначально даны идентификаторы конкретных детей – мальчика_2 и девочки_3 в группе детей, которым присвоены имена Саша и Валя. Далее для данного примера могли бы быть, например, определены имена классов “*мальчик*” и “*девочка*”.

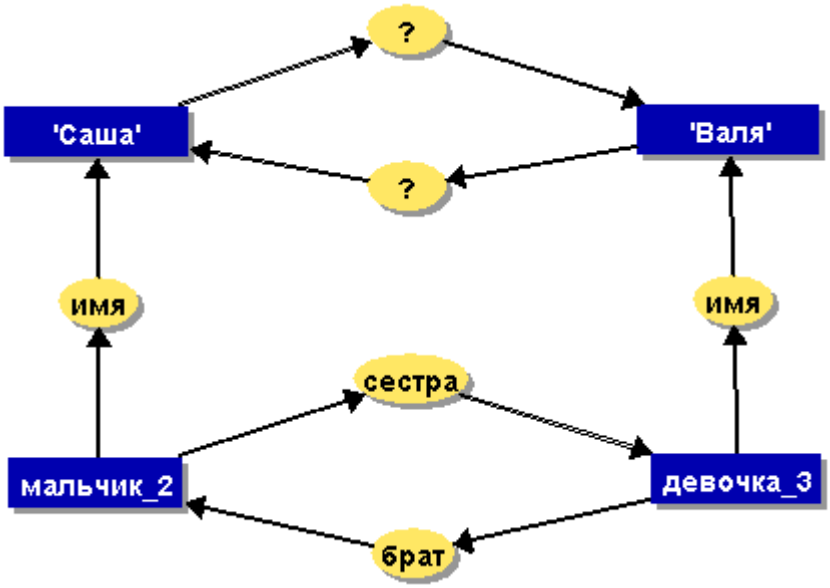


Рисунок 2.1 – Концептуальный граф с неизвестными отношениями

Решение. В представленной ниже программе использован тернарный, или трехместный, предикат “*родство*”, в котором заданные ранее факты с именами бинарных, или двуместных, предикатов “*брат*” и “*сестра*” используются в качестве значений первого аргумента:

имя(мальчик_2,'Саша').

имя(девочка_3,'Валя').

сестра(мальчик_2, девочка_3).

брат(девочка_3, мальчик_2).

родство(сестра, мальчик_2, девочка_3).

родство(брат, девочка_3, мальчик_2).

родство(C, A, B) :- родство(C, X, Y), имя(X, A), имя(Y, B).

родство(D, B, A) :- родство(D, Y, X), имя(X, A), имя(Y, B).

В данной программе используются рекурсивные правила. В результате выполнения запроса “какими отношениями связаны определенные в графе концепты?” получены следующие результаты:

? - родство(X, Y, Z).

X = сестра,

Y = мальчик_2,

Z = девочка_3 ;

X = брат,

Y = девочка_3,

Z = мальчик_2 ;

X = сестра,

Y = 'Саша',

Z = 'Валя' ;

X = брат,

Y = 'Валя',

Z = 'Саша' ;

Значения переменной *X* “*брат*” и “*сестра*” соответствуют именам искомых отношений.

Следующий запрос – “*кем является Валя для Саши?*”:

?- родство(X, 'Саша', 'Валя').

X = сестра,

то есть для Саши Валя является сестрой.

Задача_2. Следующий пример является модификацией предыдущего. Здесь требуется дополнить экстенсиональные отношения “*брат*” и “*сестра*” в базе знаний интенсиональными отношениями, получаемыми в результате логического вывода. Концептуальный граф представлен на рисунке 2.2.

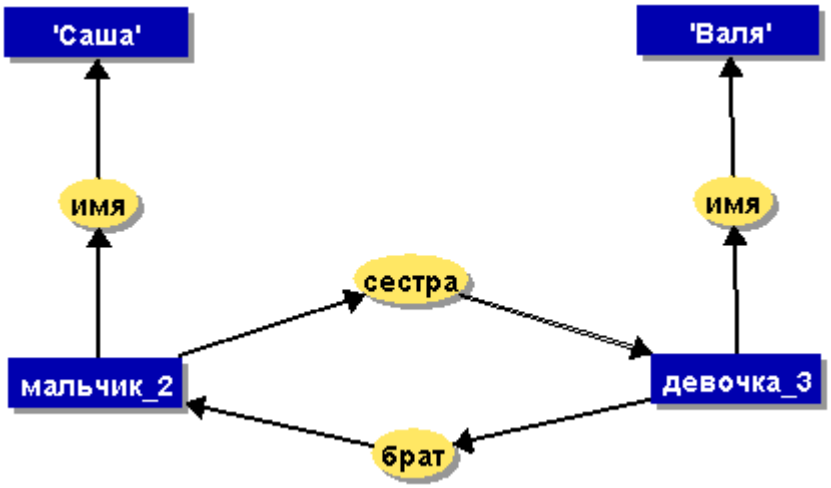


Рисунок 2.2 – Концептуальный граф с неполными отношениями “*брат*” и “*сестра*”

Решение. Программа имеет следующий вид:

имя(мальчик_2,'Саша').

имя(девочка_3,'Валя').

сестра(мальчик_2, девочка_3).

брат(девочка_3, мальчик_2).

сестра(A, B) :- сестра(X, Y), имя(X, A), имя(Y, B).

брат(A, B) :- брат(X, Y), имя(X, A), имя(Y, B).

Как и в предыдущем примере, в данной программе используются рекурсивные правила. Например, ответ на запрос “*выдать все пары брат-сестра*” имеет следующий вид:

?- сестра(X,Y).

X = мальчик_2,

Y = девочка_3 ;

X = 'Саша',

Y = 'Валя' ;

Ответ на запрос “*правда ли, что Саша – брат Вали?*” имеет следующий вид:

?- брат('Валя', 'Саша').

true .

Задача 3. На рисунке 2.3 задан концептуальный граф. Используя рекурсивное правило, составить программу на Прологе для определения отношения, отмеченного вопросительным знаком (выполнить самостоятельно!).

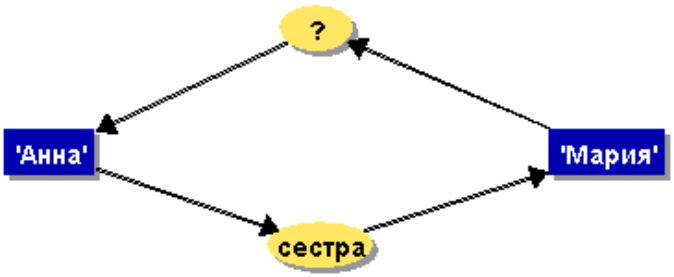


Рисунок 2.3 – Концептуальный граф с двумя вершинами-концептами

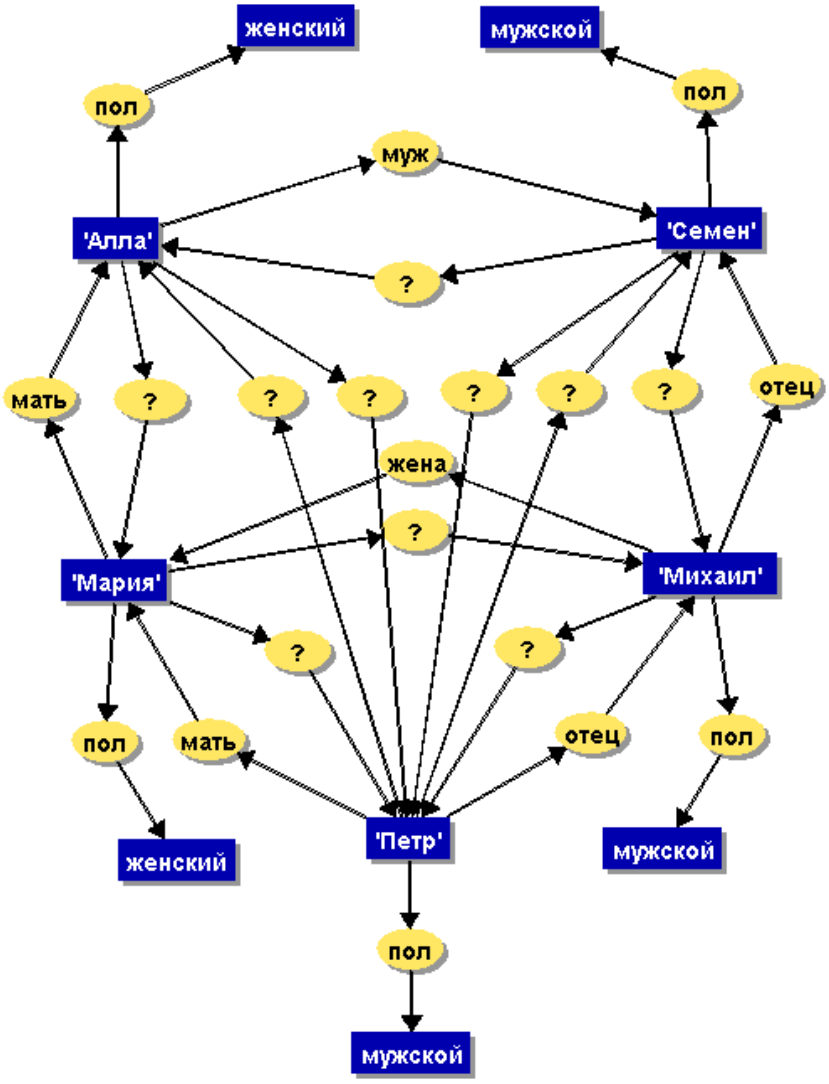


Рисунок 2.4 – Концептуальный граф “Родство”

Задача 4. Доопределить по смыслу предметной области неизвестные отношения в концептуальном графе “Родство”, представленном на рисунке 2.4, разработать программу на языке Пролог с правилами для интенсиональной базы знаний с отношениями, первоначально обмеченными символом вопроса “?” на рисунке (выполнить самостоятельно!).

Задача 5. Используя унарные предикаты и правила, разработать программу на языке Пролог для простой экспертной системы. Например, задавая различные признаки, требуется угадать животное – лев, тигр или зебра. Разработать программу на каком-либо недекларативном языке типа C++, Java, Python и др. (по указанию преподавателя).

Решение. Концептуальный граф представлен на рисунке 2.5. Соответствующая этому графу программа представлена ниже:

```
признак(имеет_клыки).
признак(млекопитающее).
признак(плотоядный).
признак(семейство_кошачьих).
признак(хищник).
признак(охота_круглосуточная).
признак(обитает_в_Африке_и_Индии).
признак(живет_в_прайдах).
признак(самец_имеет_гриву).
признак(умеет_плавать).
признак(поддается_дрессировке).
признак(имеет_хвост).
признак(обитает_в_Африке).
признак(имеет_гриву).
признак(не_приручается).
признак(имеет_копыта).
признак(непарнокопытное).
признак(травоядное).
```

признак(защищается_лягание).

признак(имеет_полосы).

признак(обитает_в_Азии).

признак(охота_в_одиночку).

признак(охота_ночью_из_засады).

это(лев) :- признак(имеет_клыки),признак(млекопитающее),

признак(плотоядный), признак(семейство_кошачьих),

признак(хищник), признак(охота_круглосуточная),

признак(обитает_в_Африке_и_Индии),

признак(самец_имеет_гриву), признак(умеет_плавать),

признак(поддается_дрессировке), признак(имеет_хвост).

это(тигр) :- признак(млекопитающее), признак(семейство_кошачьих),

признак(хищник), признак(плотоядный), признак(обитает_в_Азии),

признак(поддается_дрессировке), признак(умеет_плавать),

признак(охота_в_одиночку), признак(имеет_хвост),

признак(охота_ночью_из_засады), признак(имеет_клыки),

признак(имеет_полосы).

это(зебра) :- признак(обитает_в_Африке), признак(имеет_хвост),

признак(имеет_гриву), признак(имеет_клыки),

признак(не_приручается), признак(имеет_копыта),

признак(непарнокопытное), признак(травоядное),

признак(защищается_лягание), признак(имеет_полосы).

Построение концептуального графа, изображенного на рисунке 2.5 основан на графическом представлении унарных предикатов типа “*признак(X)*” и “*это(Y)*”, где переменная *X* принимает значения в множестве признаков, характерных для различных животных, а *Y* – в множестве имен животных. Напомним, что запятая “,” между литералами в теле правила означает конъюнкцию. Тело правила может быть представлено и в дизъюнктивной форме, при этом между литералами ставится точка с запятой “;”.

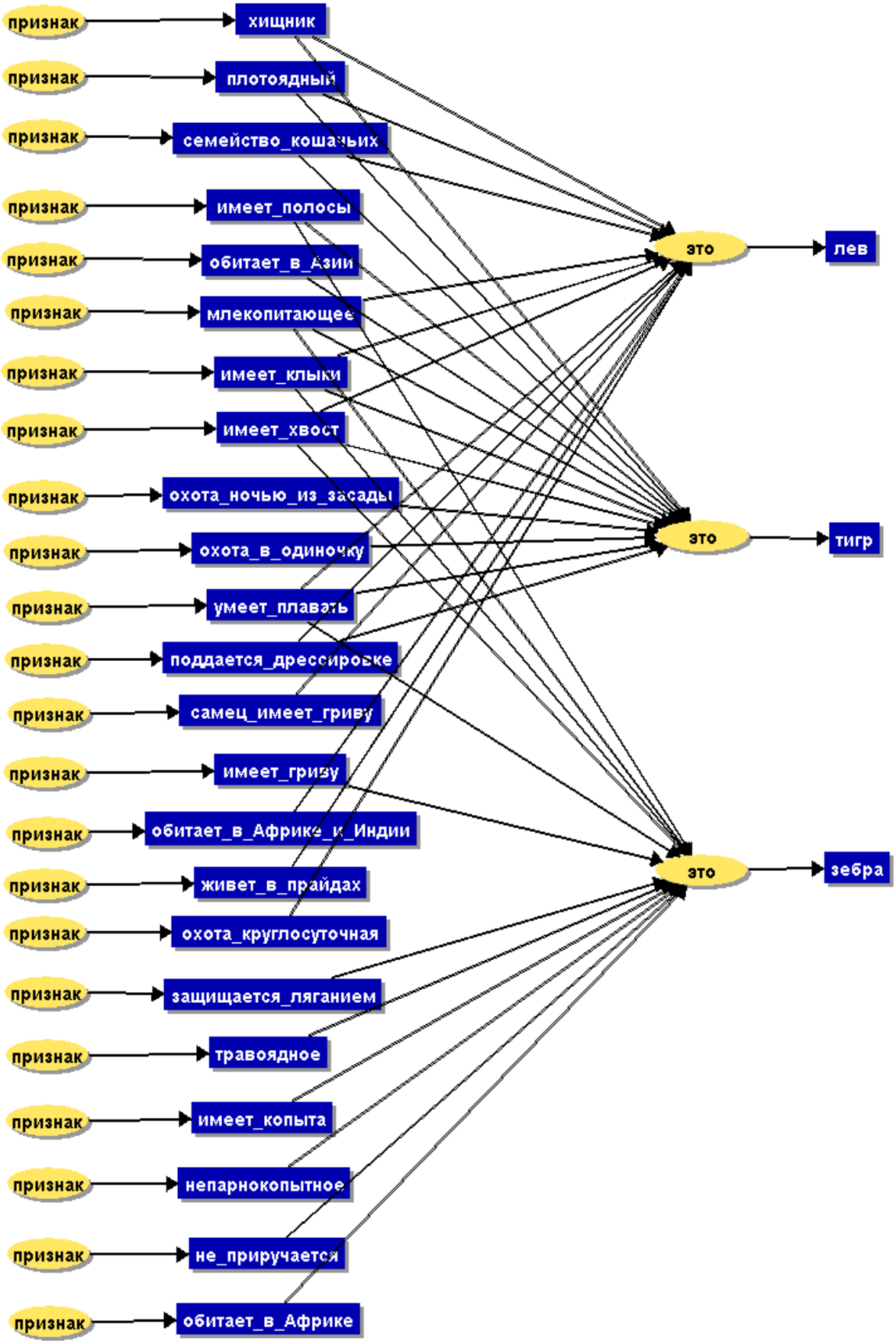


Рисунок 2.5 – Концептуальный граф для прототипа экспертной системы

Отрицание предиката задается при помощи унарного предиката *not*. Например, тот факт, что животное не имеет полос, записывается в Прологе как *not*(*признак(имеет_полосы)*). Строго говоря, в теле правила возможно использовать отрицание, при этом литерал начинается с функции *not*. Например, такой литерал, как *not(признак(самец_имеет_гриву))*, является допустимым в правой части правила.

При использовании недекларативных языков для решения данной задачи можно использовать вместо унарных предикатов булевы переменные. Более сложный вариант решения предполагает использования элементов нечеткой логики, позволяющей сделать выводы при совпадении не всех признаков с эталоном.

Задача 6. Работа с *n*-арными ($n > 2$) отношениями. При построении концептуальных графов используются в основном унарные и бинарные предикаты и отношения (отношение – это область истинности одноименного в общем случае предиката). Предикаты с арностью (местностью) более 2-х редактором концептуальных графов ***CharGer*** не поддерживаются, что не позволяет обозначать все связи в графическом представлении *n*-арного предиката ($n > 2$). В качестве примера рассмотрен граф 4-арного (четырехместного) предиката на рисунке 2.6.

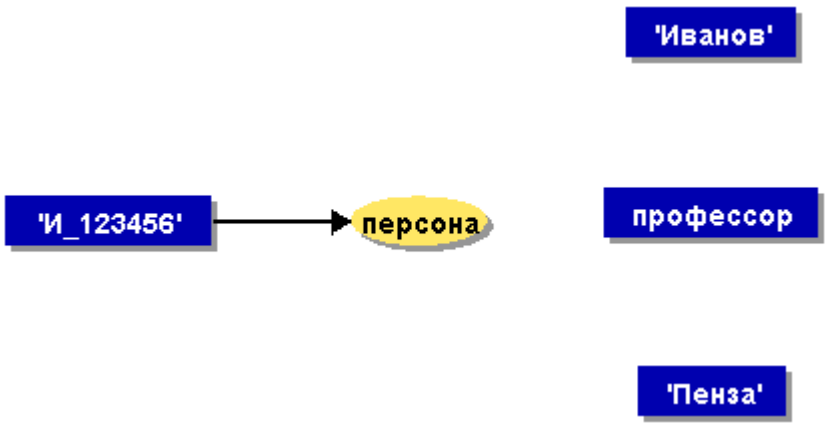


Рисунок 2.6 – Концептуальный граф с недостроенными связями для 4-арного предиката “*персона*”

Концептуальный граф 4-арного предиката “*персона*” с недостроенными, но подразумеваемыми связями, формализует фразу “*персона с табельным номером И_123456 является профессором Ивановым, проживающим в Пензе*”. Соответствующее данному графу высказывание имеет следующий вид:

персона(‘И_123456’, ‘Иванов’, профессор, ‘Пенза’),

где “*персона*” – имя предиката, а в скобках указаны предметные константы – значения аргументов предиката. Подобное высказывание можно использовать в качестве факта экстенсиональной базы знаний. Этого высказывания недостаточно для раскрытия смысла (семантики) фразы “*персона с табельным номером И_123456 является профессором Ивановым, проживающим в Пензе*”. Как известно, понятиям логического программирования соответствуют понятия реляционных баз данных: предикат – отношение, аргумент предиката (предметная переменная) – атрибут, факт – кортеж, цель – запрос. Отношению реляционной алгебры для рассматриваемой задачи соответствует следующая таблица:

персона			
Таб_номер	Фамилия	Должность	Место_проживания
…	…	…	…
‘И_123456’	‘Иванов’	профессор	‘Пенза’
…	…	…	…

Описание предметной области, или область экспертизы, здесь расширена за счет введения атрибутов “Таб_номер”, “Фамилия”, “Должность”, “Место_проживания”. Эти новые, но подразумеваемые ранее знания можно представить в концептуальном графе (рисунок 2.7).

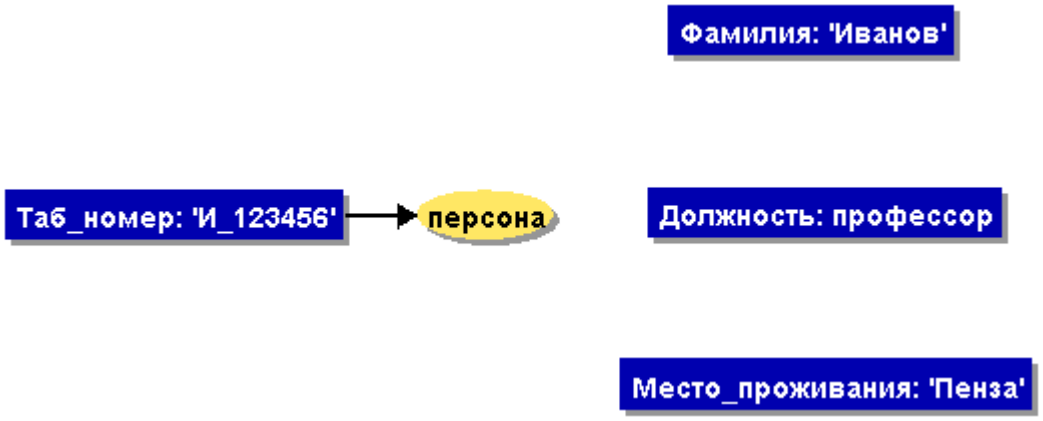


Рисунок 2.7 – Концептуальный граф с недостроенными связями 4-арного предиката “*персона*” с указанием имен атрибутов

Этот граф более информативен, но для достройки необходимых связей потребовался бы другой редактор концептуальных графов, отличный от редактора ***CharGer***.

На рисунке 2.8 приведен концептуальный граф другого логического представления той же фразы бинарными предикатами. Введенным ранее именам атрибутов здесь соответствуют одноименные предикаты.

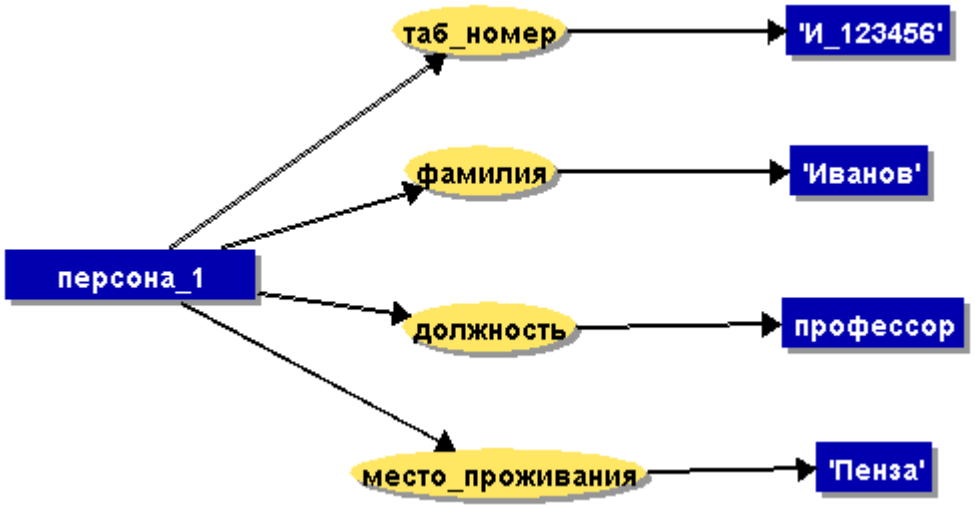


Рисунок 2.8 – Концептуальный граф логического представления фразы бинарными предикатами.

Подобное описание более информативно и расширяет семантику предметной области по сравнению с графом на рисунке 2.6. Теперь факты экстенсиональной базы знаний в Прологе следует представить таким образом:

таб_номер(персона_1, ‘И_123456’).
фамилия(персона_1, ‘Иванов’).
должность(персона_1, профессор).
место_проживания(персона_1, ‘Пенза’).

Предметная константа “*персона_1*” указывает на конкретную личность. Имя “*персона*” может быть использовано в качестве имени класса персон, как представлено на рисунке 2.9. Здесь отношение “*например*” означает, что объект “*персона_1*” является экземпляром класса “*персона*”.

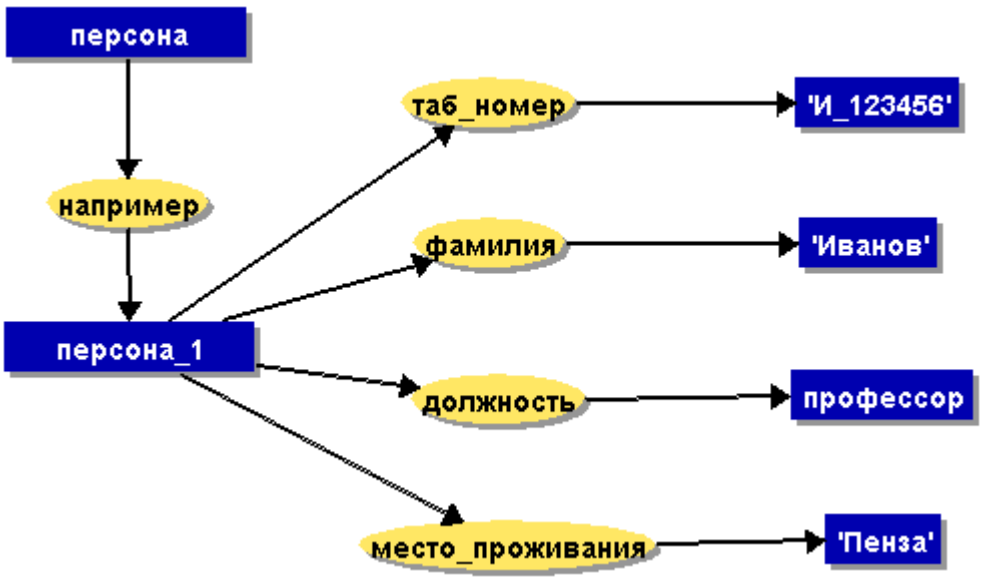


Рисунок 2.9 – Концептуальный граф с отношением “*класс – элемент класса*”

Лабораторное задание

Построить концептуальные графы для различных предметных областей по заданию преподавателя и подготовить на их основе факты и правила языка Пролог. Нарисовать и исследовать концептуальный граф для своего варианта в редакторе **CharGer.jar**: **File|New.** Повторить все этапы из лабораторной работы №1 для заданного преподавателем концептуального графа.

Отчет по лабораторной работе должен включать:

1. Цель работы.
2. Условие задачи.
3. Сформулированные требования к семантической сети пользователя интеллектуальной системы на основе функциональной модели предметной области.
4. Разработку семантической сети (в любом редакторе), концептуального графа на ее основе (в редакторе **CharGer**), экстенсиональной и интенсиональной баз фактов и знаний о выбранной предметной области.
5. Выводы о проделанной работе.

Вопросы и задания для самопроверки

1. Дайте характеристику основным элементам семантической сети.
2. Назовите отличие однородных семантических сетей от неоднородных.
3. Перечислите достоинства и недостатки семантических сетей.
4. Используя соответствующие отношения, постройте семантическую сеть для перечисленных предметных областей:

4.1. География какого-либо региона. Отношения: государство, страна, континент, широта.

4.2. Диагностика заболеваний внутренних органов. Отношения: категории болезней, патофизиологическое состояние, жалобы пациента, наблюдения, симптомы.

4.3. Распознавание химических структур. Отношения: формула вещества, свойства вещества, область применения, меры предосторожности.

4.4. Процедура поиска полезных ископаемых. Отношения: наименование ископаемого, расположение месторождения, глубина залегания, методы добычи.

4.5. Судебная процедура. Отношения: юридическое лицо, событие, меры воздействия, способы расследования.

4.6. Распределение продуктов по магазинам. Отношения: источник снабжения, наименование продукта, способ транспортировки, конечный пункт транспортировки.

4.7. Определение принадлежности животного к определенному виду, типу, семейству. Отношения: место обитания, строение, особенности поведения, вид питания.

4.8. Классификация пищевых продуктов. Отношения: наименование продукта, составляющие части, способ приготовления, срок хранения.

4.9. Распознавание типа компьютера. Отношения: страна изготовитель, стандартная конфигурация, область применения, используемое программное обеспечение.

Лабораторная работа №3

Анализ и описание предметной области экспертной системы

Цель работы: Изучение и проектирование функциональной модели экспертной системы для заданной предметной области.

Основные понятия и определения

Экспертная система (ЭС) представляет собой человеко-машинную систему, обычно содержащую пять основных компонентов:

1) интерфейс с пользователем;
2) подсистему логического вывода, обеспечивающую манипулирование знаниями при решении прикладных проблем;
3) базу знаний, представляющую собой совокупность знаний описанных с использованием выбранной формы их представления;
4) подсистему приобретения знаний;
5) подсистему отображения и объяснения решений.

ЭС обычно имеют прикладной характер и предназначены для узкой области применения. С функциональной точки зрения прикладная ЭС является системой, которая использует знания специалистов (экспертов) о некоторой конкретно узкоспециализированной предметной области, и в пределах этой области способна принимать решения на уровне эксперта – профессионала.

Взаимодействие пользователя с ЭС осуществляется через интерфейс с пользователем на проблемно-ориентированном языке непроцедурного типа, часто на некотором редуцированном варианте естественного языка. Интерфейс с пользователем служит средством трансляции предложения некоторого входного языка на внутренний язык представления знаний данной ЭС. После прохождения опроса ЭС выдаёт окончательные решения. Функция подсистемы приобретения знаний состоит в поддержке процесса

извлечения знаний о соответствующей предметной области от определенного источника знаний о соответствующей предметной области от определенного источника знаний (эксперта). В качестве потенциальных источников знаний могут также выступать личный опыт пользователя и специальная литература по данной проблеме. Таким образом, ЭС в отличие от компьютерных программ, решающих задачи в соответствии с определенными алгоритмами, не исключает пользователя из решения, а, наоборот, сохраняет за ним инициативу. В то же время ЭС не является просто пассивным источником полезной информации подобно книжному справочнику или базе данных. В нужные моменты ЭС подсказывает необходимое направление решения задачи, развивает цепочки умозаключений, объясняет свои действия.

Рисунок 3.1 – Архитектура экспертной системы

Лабораторное задание

Составить подробное описание предметной области по заданному варианту (общие понятия, свойства, объекты, симптомы и т.д.).

Примерные варианты лабораторных заданий:

1. Подбор автомобиля.
2. Постановка медицинского диагноза.
3. Локализация неисправности персонального компьютера по внешним симптомам.
4. Определение конфигурации персонального компьютера в зависимости от потребностей пользователя.
5. Выбор комплекта программного обеспечения для компьютера в соответствии с задачами, которые необходимо решать пользователю.
6. Выбор оптимального языка программирования в зависимости от поставленной задачи.
7. Определение «пробелов» в знаниях учащегося по конкретному курсу.
8. Определение места отдыха для предстоящего отпуска (подбор туристической путевки).
9. Выбор высшего учебного заведения абитуриентом.
10. Профориентация школьника.
11. Диагностика неисправности телевизора (телефона, пылесоса, …)
12. Выбор книги в соответствии с пожеланиями читателя.
13. Выбор оптимального способа путешествия (самолет, поезд, пароход, автостопом, на машине, автобусом).
14. Определение животного.
15. Определение породы дерева.
16. Выбор телевизора.

Пример 1:

Необходимо разработать экспертную систему для выбора фотоаппарата, в соответствии с запросами пользователя.

ЭС «помогает» подобрать фотоаппарат, основываясь на пристрастии пользователя к определенному виду/стилю/месту съёмок. Поэтому, выбор

фотоаппарата будет зависеть от такой характеристики, как **«качество»**. **«Качество»** складывается из трёх основных составляющих – **«статическое качество»**, **«динамическое качество»** и **«творческое качество»**. Рассмотрим каждое из них поподробнее.

Статическое качество.

Под этим подразумевается техническое качество картинки (отпечатка), не зависящее ни от динамичности сюжета (скорость реакции и т.п.), ни от эстетической ценности полученного результата. Важно для тех, кто любит делать портретные и массовые, семейные снимки.

Таблица 3.1 – Перечисление основных свойств фотоаппарата, влияющих на статическое качество

Параметр	Пояснение	От чего зависит
Разрешение	Способность передать мелкие детали.	Обратно пропорционально от зума и прямо пропорционально от светосилы.
Аберрации	Искажения объектива, «не вошедшие» в разрешение.	От величины зума и оптики (чем меньше зум, тем лучше).
Натуральность цветов и динамический диапазон	Плавность и натуральность оттенков, способность передавать контрастные сюжеты без «завала» деталей в тенях или светах.	От геометрического размера матрицы и формата съемки
Резкость	«Попадание» фокусировки .	От фокусов (желательно, фиксированный)
Экспозиция	Отработка экспо автоматики.	От оптики
Баланс белого	Точность автомата баланса белого.	От режимов настройки (лучше ручной ББ), формата съемки (необходим RAW).

Динамическое качество.

Способность оперативно «поймать момент», и именно тот, который хочет фотограф. Важно для тех, кто любит снимать спортивные соревнования, делать естественные, а не постановочные фотографии.

Таблица 3.2 – Перечисление основных свойств фотоаппарата,

влияющих на динамическое качество

Параметр	Пояснение	От чего зависит
Лаг	«Тормознутость» — время от нажатия на кнопку до самого кадра.	От оптики (лучше зеркальная).
Скорость и точность автофокуса.	Подмножество «тормознутости», но иногда имеет самостоятельное значение...	От оптики (лучше зеркальная).
Ручной фокус.	Скорость и удобство ручной установки фокуса.	От оптики (лучше зеркальная или псевдозеркальная).
Точки фокусировки	Наличие нескольких точек и возможность быстрого переключения между ними часто удобнее чем пользование «фокус-лок»-ом.	От количества точек фокусировки
Скорострельность	Способность быстро снять длинную серию кадров. Любителям нужна не часто.	От размера буфера и скоростью записи на носитель.

«Творческое» качество».

Собственно «творческое качество снимка» определяется в основном самим фотографом и именно про это качество прежде всего известная поговорка «снимает фотограф а не камера». Важно как, для любителей постановочных фотографий, так и для тех, кто хочет поймать момент.

Таблица 3.3 – Перечисление основных свойств фотоаппарата, влияющих на динамическое качество

Параметр	Пояснение	От чего зависит и с какими камерами
ГРИП	Недостаточно МАЛАЯ для портрета или недостаточно БОЛЬШАЯ для макро.	От размера матрицы.
ЭФР	Недостаточно широкий угол или недостаточно длинный фокус при невозможности «скадрировать ногами» (подойти или отойти).	От кратности зума и значения ЭФР на «коротком конце»
Минимальный макрообъект.	Обычно указывается либо масштаб, либо минимальная дистанция фокусировки. Выше объяснялось, почему это не всегда удобно...	От размера матрицы и конструкции объектива.
Чувствительность	Способность снять при низкой освещённости (без вспышки и «шевелёнки»). Для цифры важное замечание — «при нормированных шумах».	От светосилы объектива и площади матрицы.
Динамика	Смысл сводится именно к «точно поймать момент».	От оптики и скорострельности

В этих таблицах собрались конфликтующие свойства, например, чувствительность и разрешение, выигрывая в одном, мы проиграем в другом. Поэтому ЭС должна определять, какие свойства фотоаппарата важны покупателю, и именно по этим свойствам выбирать модель.

Отчет по лабораторной работе должен включать:

1. Цель работы.

2. Условие задачи.

3. Сформулировать отличительные признаки объектов для экспертной системы на основе ее функциональной модели.

4. Разработку общих требований к средствам реализации экспертной системы.

5. Выводы о проделанной работе.

Лабораторная работа №4
Проектирование базы знаний для экспертной системы

Цель работы: В соответствии с требованиями, выявленными при анализе предметной области, спроектировать базу знаний разрабатываемой экспертной системы.

База знаний (txt, xml-файл) должна обязательно включать несколько уровней рассуждений (то есть окончательные выводы не должны напрямую следовать из комбинаций входных данных, обязательно присутствие промежуточных выводов) и демонстрировать некоторую интеллектуальность в принятии решений.

Основные понятия и определения в области проектирования базы знаний для экспертной системы на основе примеров

Пример 1:

ЭС должна осуществлять подбор модели фотоаппарата, следовательно, в БЗ необходимо хранить информацию об имеющихся моделях. Для реализации была выбрана модель базы знаний на правилах. В системе, базирующейся на правилах, утвердительный результат является действием одного из продукционных правил. Эти продукционные правила определяются входными данными.

Итак, для идентификации модели используется список атрибутов:

— разрешение;
— аберрация;
— натуральная передача цвета;
— резкость;
— экспозиция;
— баланс белого;
— лаг;

— скорость и точность автофокуса;

— ручной фокус;

— точки фокусировки;

— скорострельность;

— ГРИП;

— ЭФР;

— чувствительность;

— динамика.

Каждая характеристика для определенной модели верна или не верна.

Характеристики для каждой модели фотоаппарата приведены в таблице 4.4.

Таблица 4.4 – Модели и их характеристики

Canon Mark II	1, 3, 5, 6, 9, 11, 15
Sony CyberShot DSC-T50	1, 5, 6, 11, 13, 14, 15
Panasonic Lumix	1, 4, 7, 8, 9, 13, 14
Olympus NB	2, 5, 6, 10, 7, 11, 12

Важность или неважность для покупателя любой из характеристик определяется при помощи ответов на вопросы:

— Вы часто печатаете фотографии больших форматов?

— Вы считаете, что чем больше кратность зума, больше вы теряете?

— Для вас важна натуральность красок фотографии?

— Вас сильно расстраивают «смазанные» фотографии?

— Вы расстраиваетесь, видя засвеченные снимки?

— Вы часто снимаете на вечеринках, дискотеках (в помещениях с несколькими источниками света разного происхождения)?

— Вы предпочитаете, чтобы фотоаппарат снимал сразу после нажатия кнопки?

— Вы любите подкорректировать вручную резкость фотографии?

— Любите снимать как портреты, так и массовые фотографии?

— Любите заниматься фотоохотой?

— Вы любите заниматься портретной съемкой?

— Любите снимать памятники архитектуры на маленьких улочках Европы?

— Вы хотели бы сфотографировать своих друзей на концерте, где ягоде негде упасть?

— Вы хотели бы иметь несколько фотографий с прогулки по ночному городу?

— Вы предпочитаете поймать момент, а не делать постановочные фотографии?

Порядок вопросов совпадает с порядком характеристик, приведенных выше.

Запишем данную таблицу в виде правил следующего вида:

Модель Canon Mark II :-

Вы часто печатаете фотографии больших форматов,

Для вас важна натуральность красок фотографии,

Вас сильно расстраивает смазанные фотографии,

Вы расстраиваетесь, видя засвеченные снимки,

Любите снимать как портреты, так и массовые фотографии,

Вы любите заниматься портретной съемкой,

Вы предпочитаете поймать момент, а не делать постановочные фотографии.

Модель Sony CyberShot DSC-T50 :-

Вы часто печатаете фотографии больших форматов,

Вас сильно расстраивают смазанные фотографии,

Вы расстраиваетесь, видя засвеченные снимки,

Вы любите заниматься портретной съемкой,

Вы хотели бы сфотографировать своих друзей на концерте, где ягоде негде упасть,

Вы хотели бы иметь несколько фотографий с прогулки по ночному городу,

Вы предпочитаете поймать момент, а не делать постановочные фотографии.

Модель Panasonic Lumix :-

Вы часто печатаете фотографии больших форматов,

Вас сильно расстраивают смазанные фотографии,

Вы предпочитаете, чтобы фотоаппарат снимал сразу после нажатия кнопки,

Вы любите подкорректировать вручную резкость фотографии,

Любите снимать как портреты, так и массовые фотографии,

Вы хотели бы сфотографировать своих друзей на концерте, где ягоде негде упасть,

Вы хотели бы иметь несколько фотографий с прогулки по ночному городу,

Любите заниматься фотоохотой,

Вы любите заниматься портретной съемкой,

Любите снимать памятники архитектуры на маленьких улочках Европы.

Модель Olympus NB :-

Вы считаете, что чем больше кратность зума, больше вы теряете,

Вы расстраиваетесь, видя засвеченные снимки,

Вы часто снимаете на вечеринках, дискотеках (в помещениях с несколькими источниками света разного происхождения),

Вы предпочитаете, чтобы фотоаппарат снимал сразу после нажатия кнопки.

Таким образом, каждую модель однозначно характеризует список значений характеристик.

Пример 2.

Необходимо сформулировать вопросы к пользователю экспертной системы, помогающей ему выбрать автомобиль. Дополнительное задание на самостоятельную работу: для каждого вопроса составить расширенный набор возможных признаков автомобиля.

Вопросы:

Важен Вам цвет машины? (1)

Важен Вам цвет салона? (2)

Важен Вам размер автомобиля? (3)

Вы выбираете машину на лето? (4)

Вы выбираете машину на любой сезон? (5)

Вы выбираете машину на весну? (6)

Вы выбираете машину на зиму? (7)

Вы выбираете машину на осень? (8)

Вам надо с ручной коробкой? (9)

Вам надо с автоматической коробкой? (10)

Вам нужна машина для мужчин? (11)

Вам нужна машина для женщин? (12)

Вам важна форма кузова? (13)

Вам важна масса автомобиля? (14)

Вы собираетесь использовать машину в городе? (15)

Вы собираетесь использовать машину в деревне? (16)

В таблице 4.5 определены признаки для 17 видов автомобилей.

Таблица 4.5 – Машина и ее характеристики

1	2, 3, 10, 13, 15
2	1, 4, 9, 11, 14, 16
3	1, 2, 5, 10, 12, 13, 15
4	3, 4, 7, 9, 11, 13, 14
5	1, 3, 6, 9, 11, 13, 14, 16
6	2, 3, 5, 9, 11, 13, 14, 15
7	3, 10, 12, 13, 14, 15
8	1, 3, 5, 9, 13
9	1, 2, 3, 10, 13, 15
10	1, 2, 10, 11, 13, 15
11	7, 9, 12, 13, 15
12	2, 4, 6, 10, 12, 13, 14, 15
13	1, 3, 5, 10, 11, 15
14	1, 2, 3, 9, 10, 11, 13, 14, 15
15	4, 5, 9, 10, 11, 13, 14, 15
16	3, 4, 6, 7, 9, 11, 13, 15
17	1, 2, 4, 9, 10, 11, 13, 15

Лабораторное задание

По заданию преподавателя выберите вариант лабораторного задания из описания лабораторной работы № 3 и сформулируйте продукционные правила для выбора или идентификации объектов, учитывая особенности предметной области.

В соответствии с требованиями, выявленными при анализе предметной области, спроектировать базу знаний разрабатываемой экспертной системы.

Отчет по лабораторной работе должен включать:

1. Цель работы.

2. Условие задачи.

3. Сформулировать вопросы к пользователю экспертной системы на основе ее функциональной модели.

4. Разработку требований к средствам реализации экспертной системы.

5. Выводы о проделанной работе.

Лабораторная работа № 5

Проектирование экспертной системы для заданной предметной области.

Цель работы: Проектирование экспертной системы для заданной предметной области. Экспертная система должна вести диалог с пользователем: задавать вопросы и запоминать ответы пользователя. Диалог системы должен быть построен на основе утверждений, с которыми можно соглашаться или нет.

Задание:

Для предметной области, заданной преподавателем, разработать экспертную систему. Языки программирования: C++, C#, Java. База знаний (txt, xml-файл) должна обязательно включать несколько уровней рассуждений (то есть окончательные выводы не должны напрямую следовать из комбинаций входных данных, обязательно присутствие промежуточных выводов) и демонстрировать некоторую интеллектуальность в принятии решений.

Экспертная система должна вести диалог с пользователем: задавать вопросы и запоминать ответы пользователя. Диалог системы должен быть построен на основе утверждений, с которыми можно соглашаться или нет. При анализе будет последовательно проверяться на совпадение значение характеристик у модели БЗ и положительные ответы пользователя.

Поддержка диалога означает, что консультация должна завершаться ясным утверждением, выдаваемым системой, и последовательностью, объясняющей его.

Результатом поиска станет модель, имеющая 100% совпадение, а если такой нет, то модель, имеющая максимальное число совпадений. Можно использовать вероятностный подход, чтобы ЭС могла находить решения не только полностью соответствующие требованиям пользователя, но и варианты решений со степенью соответствия менее 100%.

Например, в базе данных может не содержаться ответа, полностью соответствующего всем требованиям пользователя, но после анкетирования на экран выводится предлагаемый вариант и его свойства, по которым был сделан выбор, “Вероятнее всего это - …”, “Скорее всего это - …”

Рассмотрим принципы функционирования простой ЭС диагностического типа. При начальном запуске ЭС на экран выдается окно системы, включающее в себя заголовок окна программы, управляющие кнопки и основное окно, в котором есть три клавиши меню (рисунок 5.1). Выбор конкретного пункта меню осуществляется однократным нажатием левой кнопкой мыши по нужной клавише меню. Загрузка базы знаний (левая клавиша меню) выполняется в случае, если база знаний была создана ранее и находится на носителе данных. При выборе этого пункта система выдаёт на экран оконную форму для выбора папки с имена всех файлов, содержащихся на носителе данных. Загрузка файла базы знаний инициируется клавишей “открыть” либо двойным нажатием на файл, в котором сохранена база знаний. Если база знаний в текущий момент отсутствует, пользователь должен выбрать вторую (среднюю) клавишу меню – «Изменение данных».

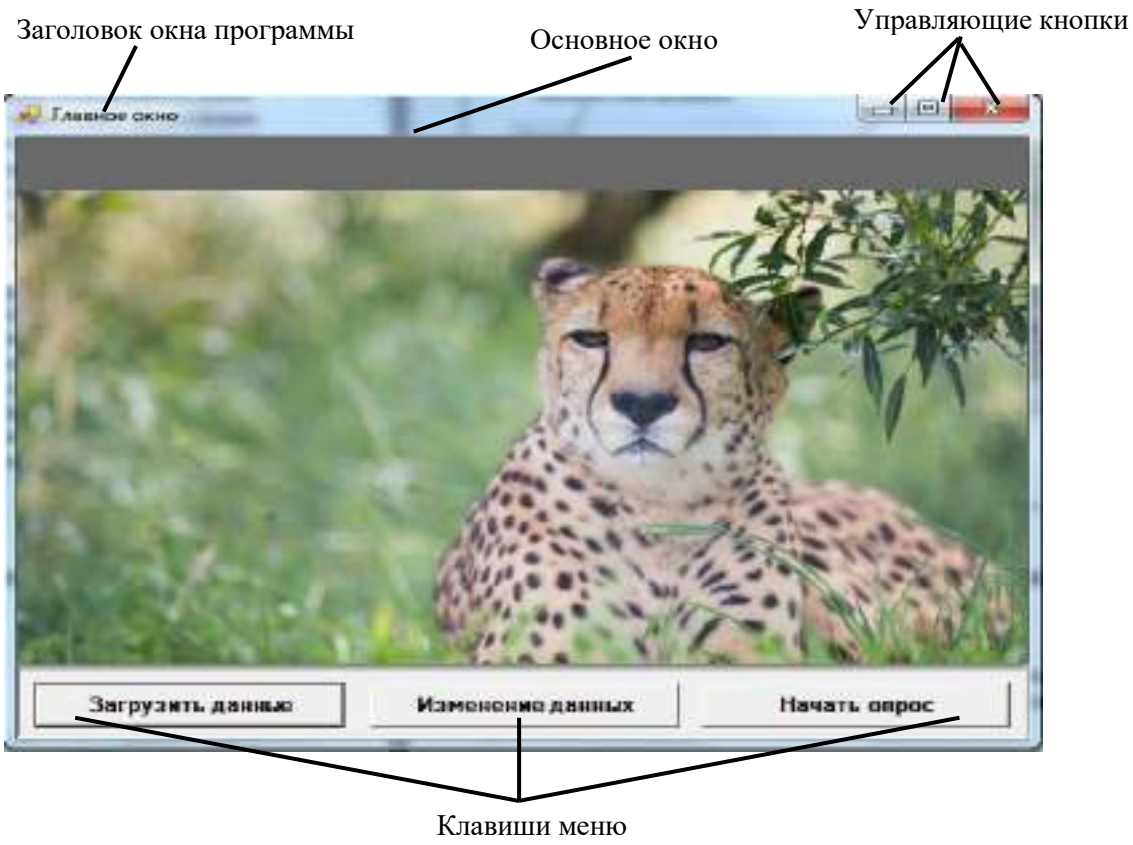


Рисунок 5.1

Общение с пользователем в интегрированной среде ЭС ведется на русском языке. Знания в ЭС заносятся на основе упрощенной продукционной модели представления знаний, причем обеспечивается возможность формулирования условий правил и заключений в терминах предметной области. Структура правила базы знаний ЭС включает следующие элементы:

1) ключевое слово «rule»;

2) список параметров, заключенный в круглые скобки.

Параметры правила в списке отделяются друг от друга запятой и содержат следующую информацию:

параметр 1 – номер правила;

параметр 2 – название общего понятия предметной области (класса);

параметр 3 – название частного понятия предметной

области (представителя), которое заключается в общее понятие (параметр 2);

параметр 4 – список условий, принадлежащих частному понятию и являющиеся критерием поиска.

Список условий продукционного правила заключается в квадратные скобки, а сами условия обозначаются числами и отделяются друг от друга запятой.

Структура условия отдельного правила, подобно самому правилу, включает ключевое слово «cond» и список параметров, заключенные в круглые скобки. Параметры условия в списке отделяются друг от друга запятой и содержат следующую информацию:

параметр 1 – номер условия;

параметр 2 – формулировка условия (текст) в терминах предметной области.

Условия в базе знаний располагаются после правил. Последний элемент базы знаний ЭС содержит имя файла на носителе данных, в котором хранится база знаний, и формулируется автоматически при создании базы.

В процессе изменения знаний пользователь осуществляет либо создание базы знаний ЭС (рисунок 5.2),

Рисунок 5.2

либо добавление необходимых правил и условий (рисунок 5.3).

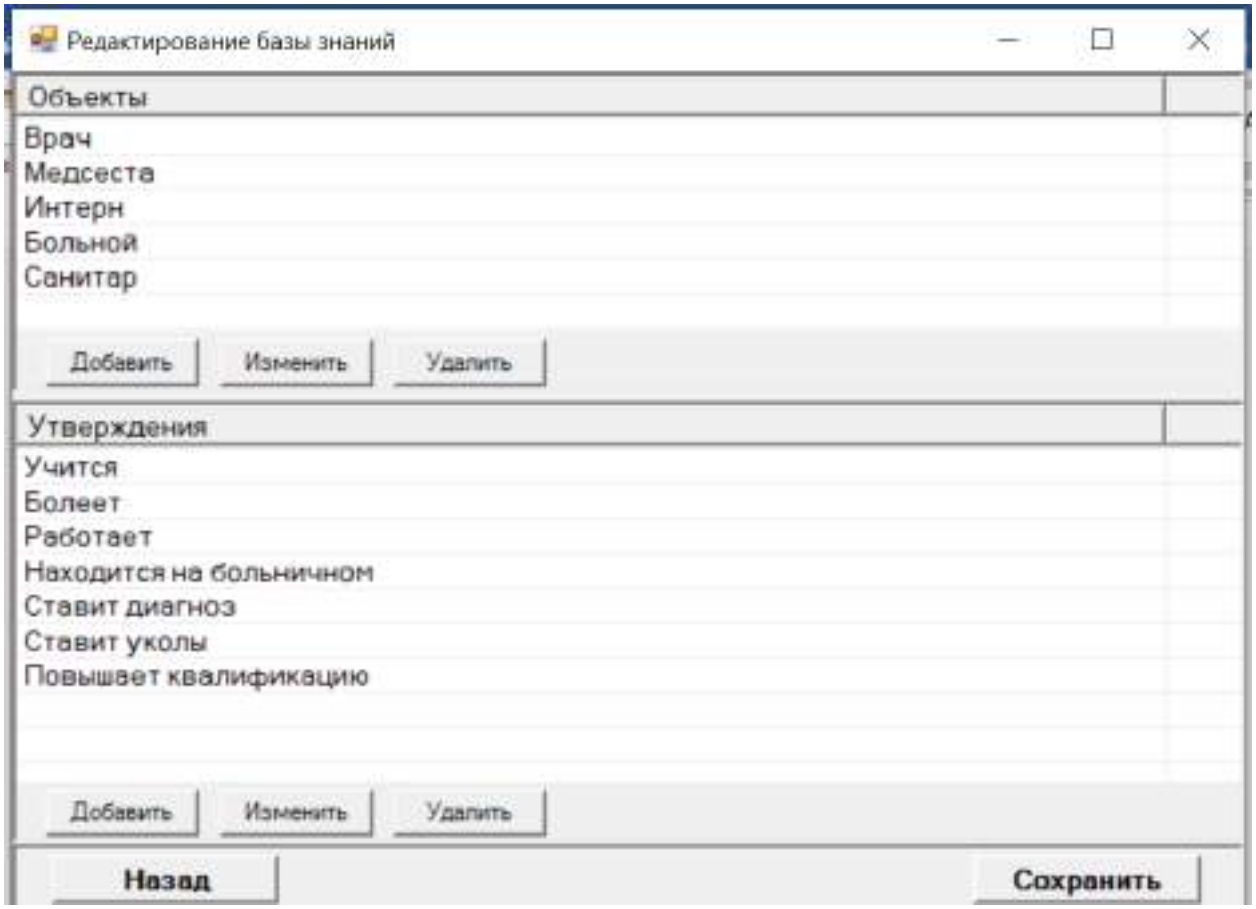


Рисунок 5.3

Для добавления новых «Объектов» в базу знаний пользователю необходимо нажать кнопку «Добавить» в верхней части окна на рисунке 5.3. При нажатии кнопки «Добавить», система выдаёт окно «Настройка объекта» (рисунок 5.4), в поле «Название» пользователю необходимо внести название субъекта и нажать кнопку «Ок».

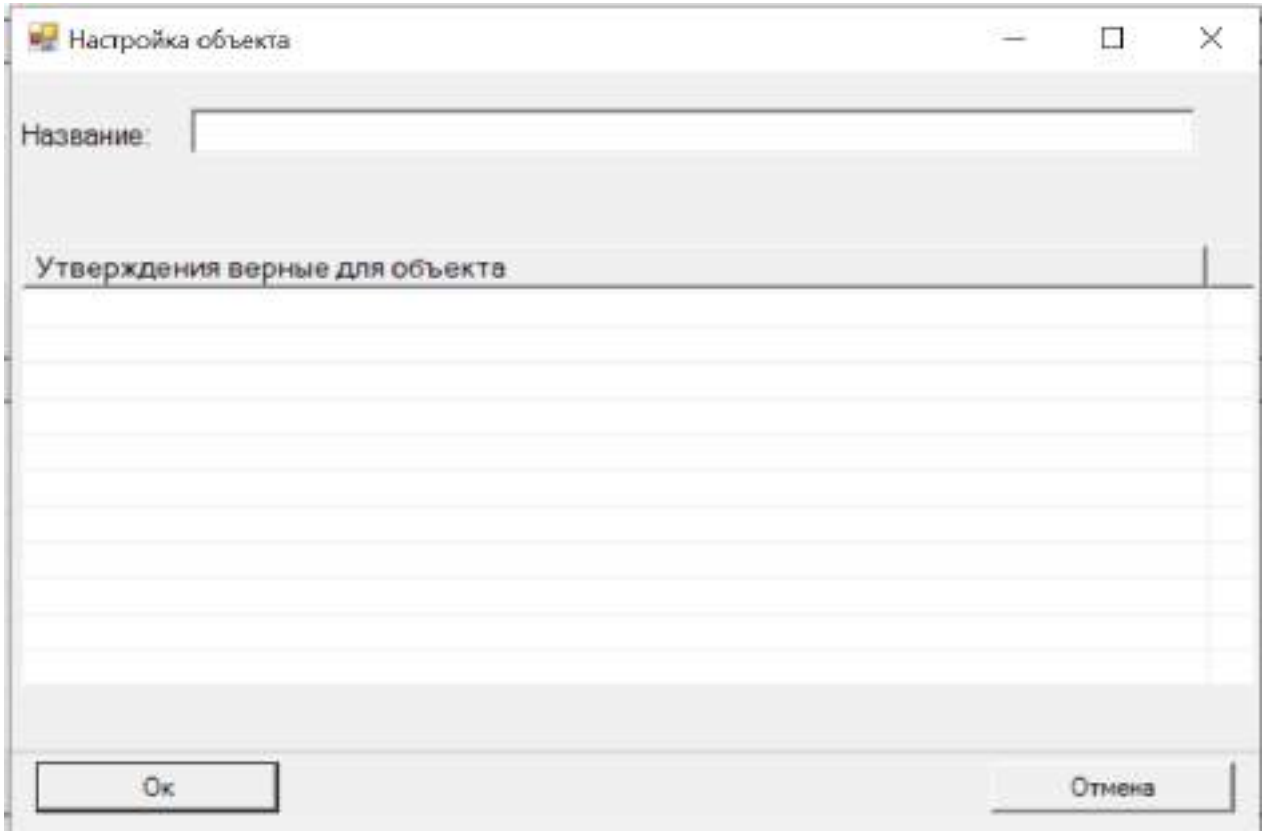


Рисунок 5.4

Переход к формированию следующего элемента базы знаний осуществляется путем повторения пользователем описанных выше действий для всех необходимых субъектов экспертной системы (рисунок 5.5).

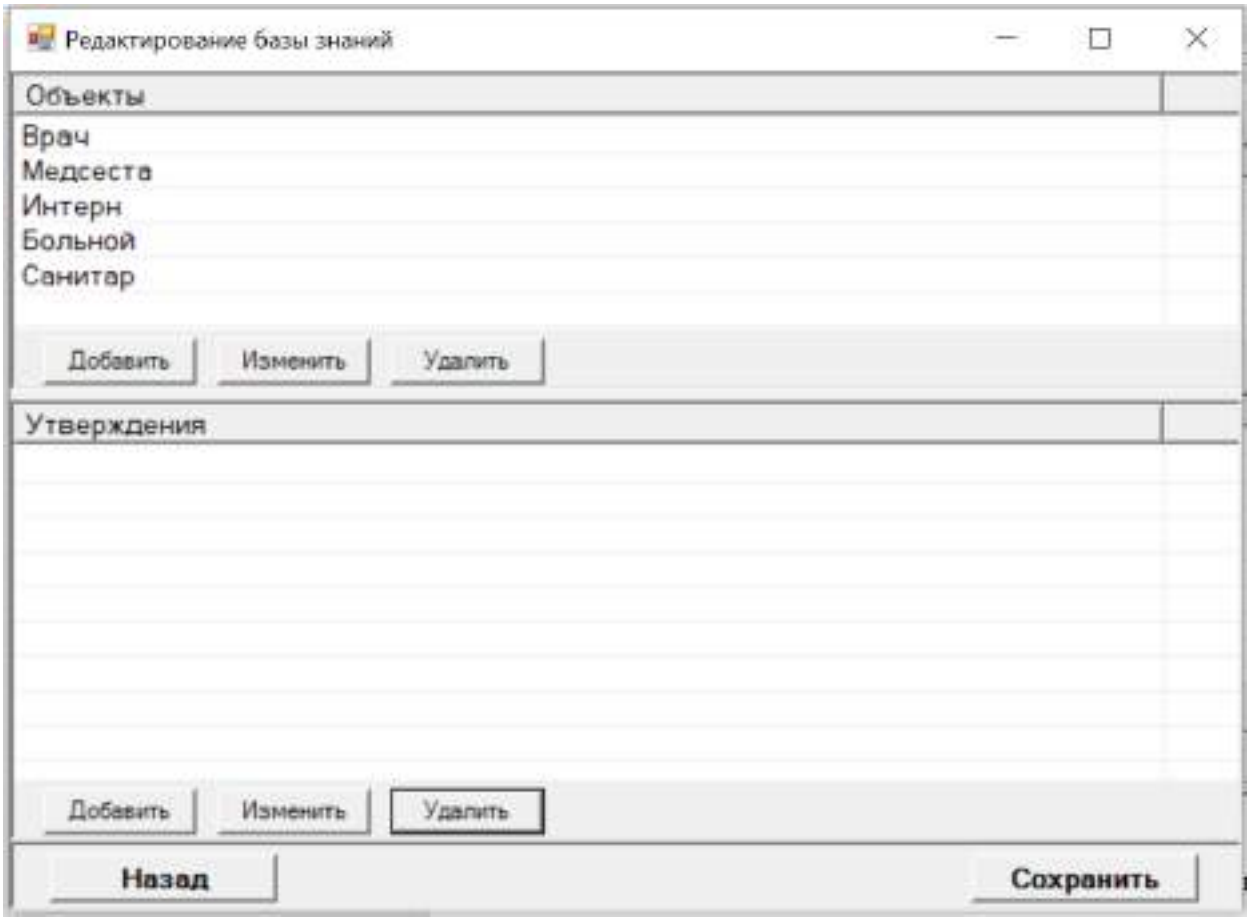


Рисунок 5.5

Чтобы не потерять уже внесённые данные, пользователь долженвоспользоваться кнопкой «Сохранить» (рисунок 5.5).

Для добавления новых «Утверждений» в базу знаний пользователю необходимо нажать кнопку «Добавить» в нижней части окна (рисунок 5.5). При нажатии кнопки «Добавить», система выдаёт окно «Утверждение» (рисунок 5.6), в поле «Описание» необходимо внести описание субъекта и нажать кнопку «Ок».

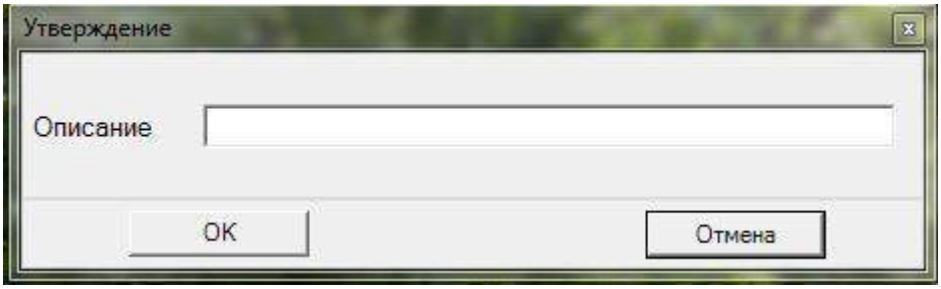


Рисунок 5.6

Переход к формированию следующих утверждений субъектов экспертной системы осуществляется путем повторения пользователем описанных выше действий для всех необходимых субъектов экспертной системы (рисунок 5.7).

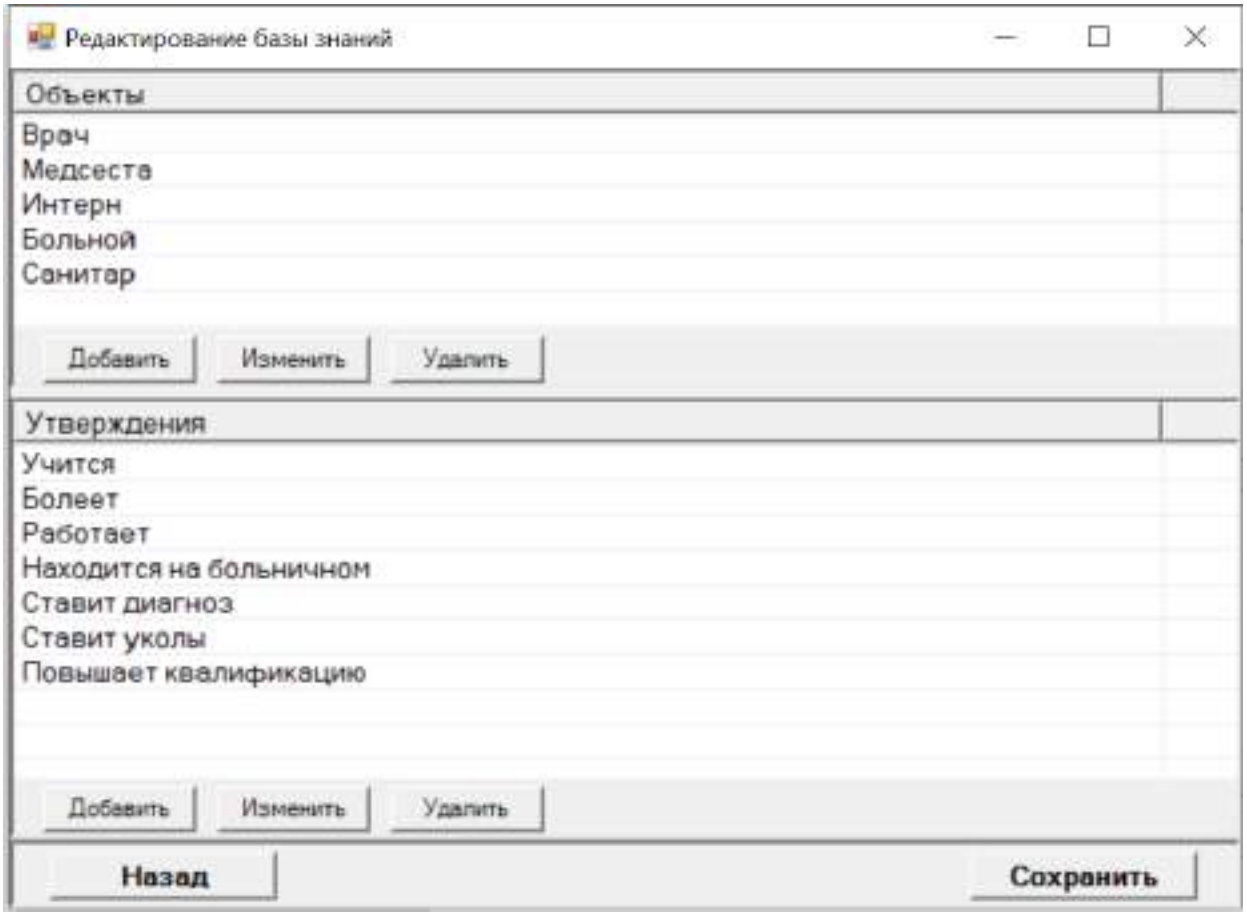


Рисунок 5.7

Получили окно системы «Редактирование базы знаний» с полностью заполненной базой знаний по выбранной предметной области.

Теперь пользователю необходимо привязать описание субъектов к самим субъектам.

Для этого из верхней части окна, изображенного на рисунке 5.7, пользователь должен выбирать один из субъектов и нажать на кнопку «Изменить» или двойным нажатием левой кнопкой мыши открыть. Появляется окно, представленное на рисунке 5.8.

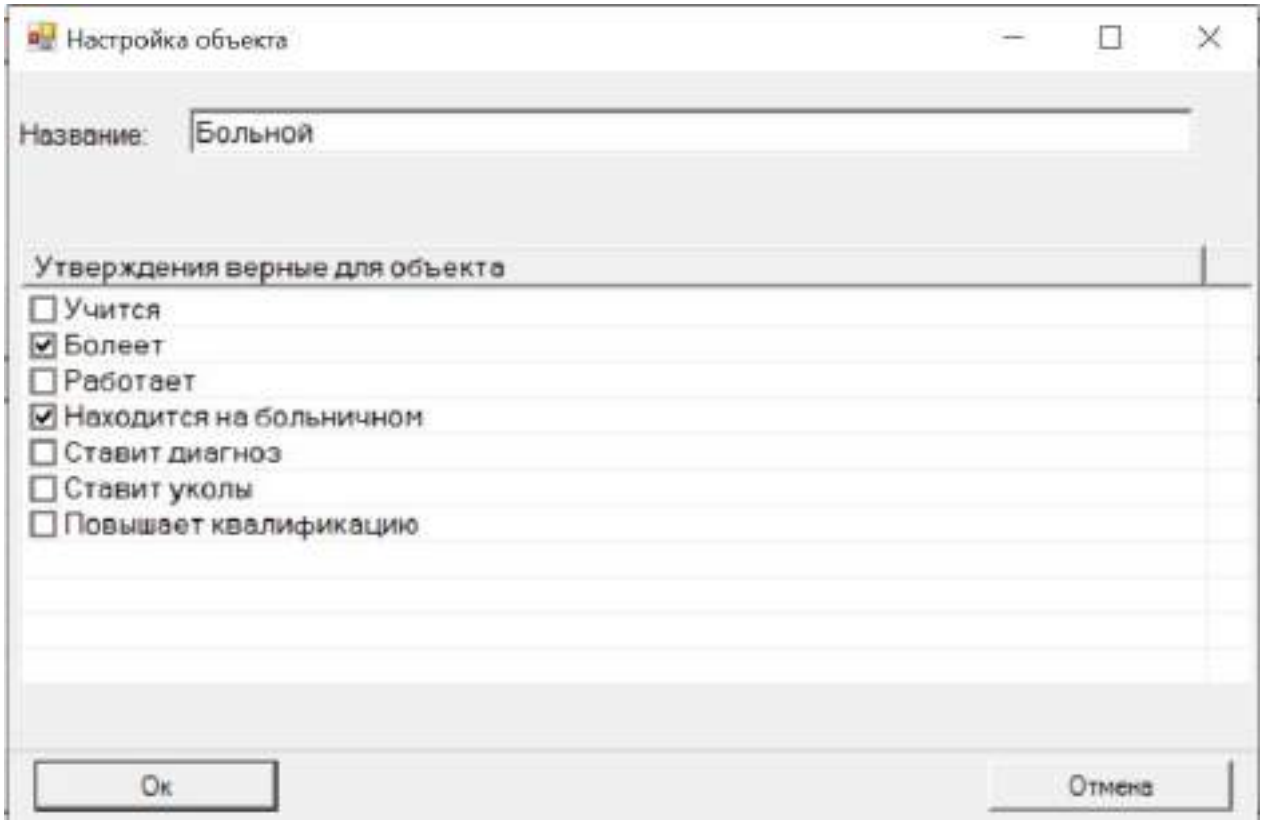


Рисунок 5.8

На рисунке 5.8 можно увидеть, что ранее незаполненные поля «Утверждения верны для объекта» теперь заполнены всеми описаниями всех субъектов экспертной системы.

Для выбранного субъекта пользователю требуется пометить галочками (рисунок 5.9) свойственные ему описания и нажать клавишу «Ок».

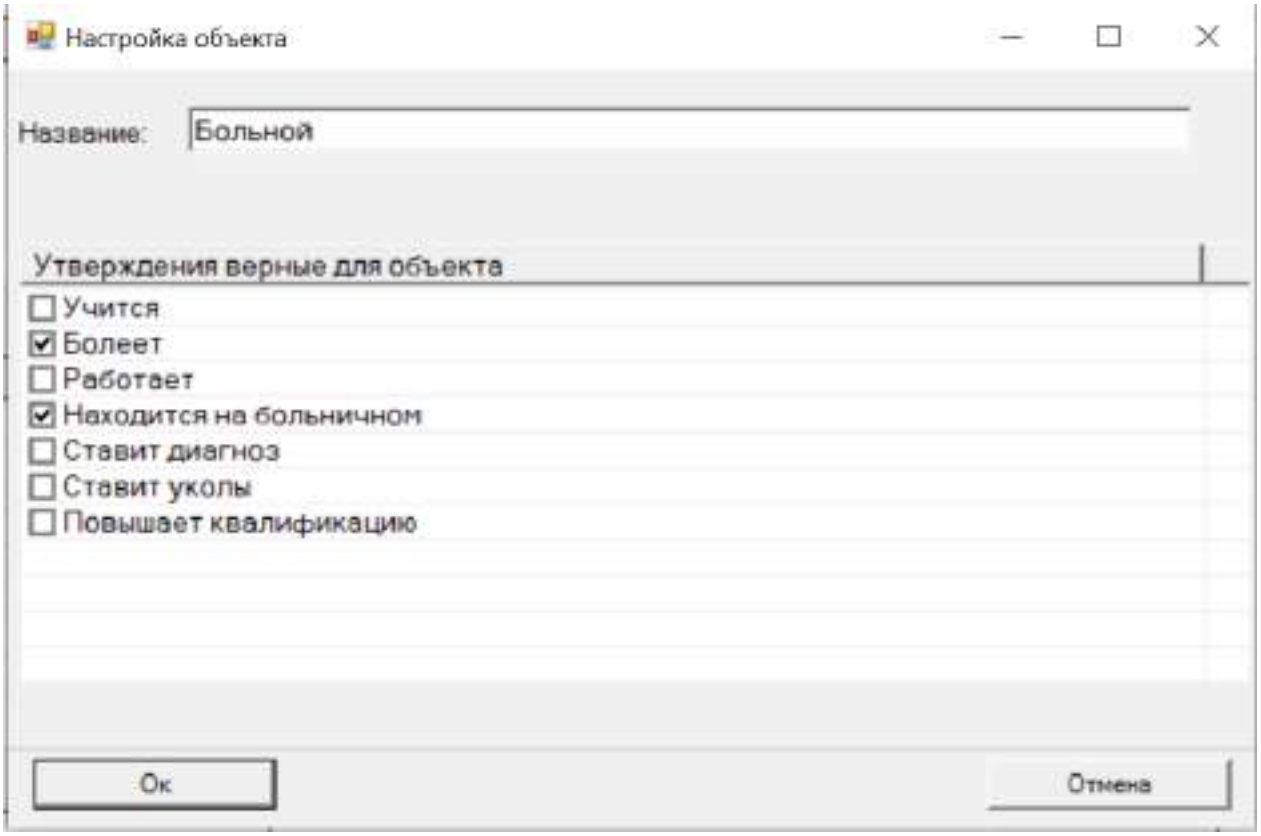


Рисунок 5.9

Для привязки всех описаний ко всем субъектам пользователь должен повторить вышеописанные действия.

В базе знаний некоторые описания могут быть свойственны нескольким объектам, но при этом не должно быть абсолютно одинаковых субъектов, так как ЭС не сможет дать правильный ответ, такие субъекты должны отличаться хотя бы одним описанием.

Для сохранения созданной базы знаний необходимо нажать на кнопку «Сохранить».

После нажатия система выдаст окно, в котором пользователю необходимо с помощью правой кнопки мыши создать текстовый документ. Далее надо выбрать созданный файл и нажать кнопку «Открыть», таким образом, база знаний будет сохранена.

После редактирования базы знаний ЭС так же необходимо выполнить аналогичные действия, реализуемым после создания знаний.

После сохранения базы знаний пользователь может начать опрос. Для этого необходимо выйти из окна «Редактирование базы знаний» (Рис. 10) при помощи кнопки «Назад». Таким образом, пользователь перешел в главное окно системы. Теперь необходимо начать опрос. После нажатия пользователю будет выведено окно, представленное на рисунке 5.10.

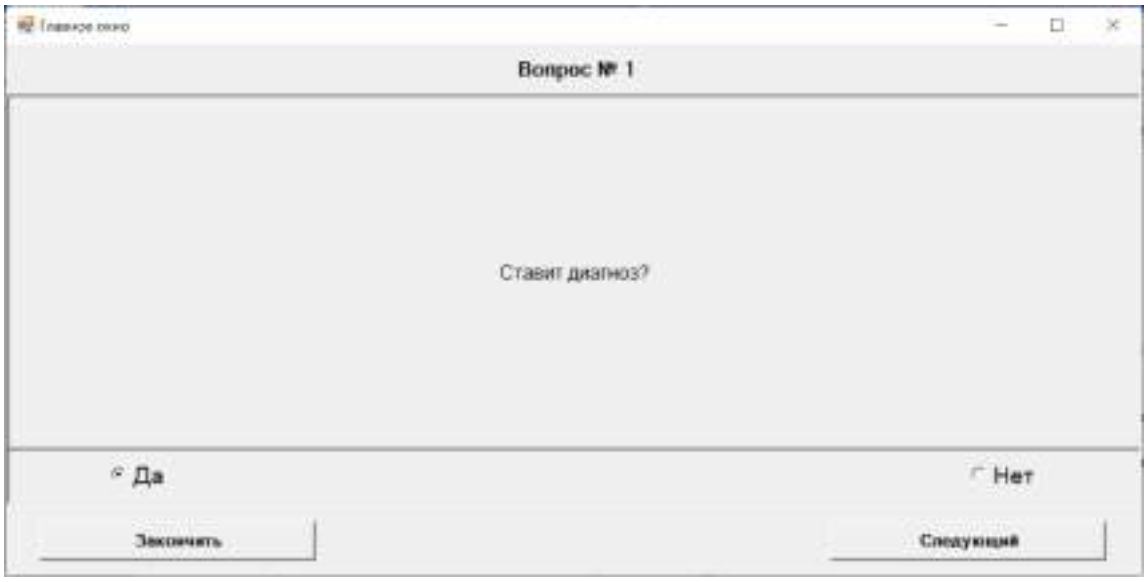


Рисунок 5.10

В представленном выше окне пользователю нужно отвечать на поставленные системой вопросы только «да» или «нет». Можно ответить на все вопросы, которые представляет система, либо закончить опрос раньше.

Чем больше будет ответов до момента завершения опроса, тем более точный результат получит пользователь от экспертной системы. Результат опроса представлен на рисунке 5.11.

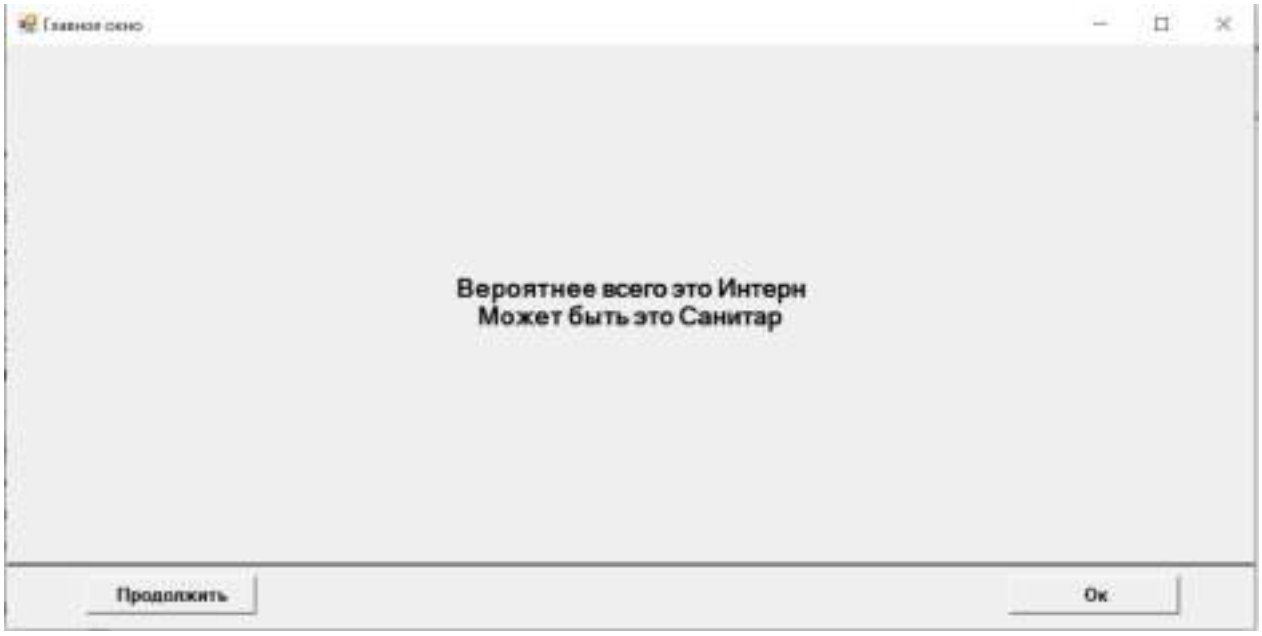


Рисунок 5.11

Отчет по лабораторной работе должен включать:

1. Титульный лист.
2. Краткое описание предметной области и решаемой задачи (что является входными данными, каков результат работы системы).
3. Разработанный набор правил экспертной системы.
4. Код программы, тестирование работы экспертной системы.
5. Выводы.

Контрольные вопросы:

1. Перечислите основные компоненты экспертной системы и дайте им краткую характеристику.
2. Укажите отличия экспертных систем от любых других интеллектуальных систем.
3. Дайте определение прикладной экспертной системы с функциональной точки зрения.
4. Укажите основную функцию компоненты экспертной системы “Интерфейс с пользователем”.
5. Укажите источники знаний для прикладных экспертных систем.
6. Сформулируйте определение инженерии знаний.
7. Приведите требования к инженеру по знаниям.
8. Объясните, почему необходимо привлечение инженера по знаниям для формирования качественной базы знаний при построении ИИС.
9. Опишите понятие поля знаний и процесс формирования поля знаний.
10. Скажите, что такое извлечение знаний и что такое приобретение знаний, вчем разница между двумя понятиями.
11. Приведите классификацию методов работы с экспертами.
12. Опишите пассивные методы получения знаний.
13. Опишите активные методы получения знаний.
14. Приведите классификацию вопросов при подготовке интервью с экспертом.

Лабораторная работа № 6

Работа с нейронной сетью

Цель работы: изучение программы по распознаванию рукописных цифр при помощи трехслойной нейронной сети.

Методические указания

Для выполнения данной лабораторной работы вам необходимо установить Python, среду Spyder(Anaconda), а также скачать готовые наборы (тренировочные и тестовые данные) рукописных цифр, представленных в удобном CSV формате.

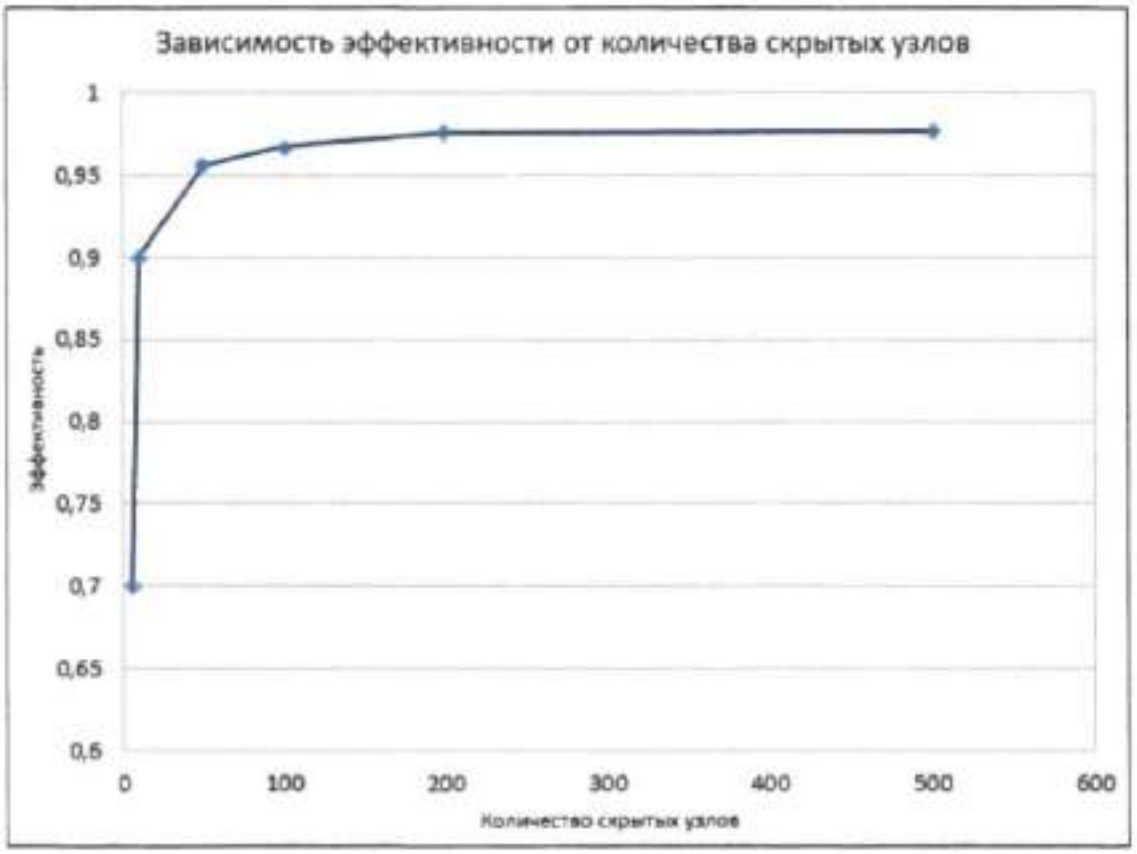
Задание: изучить теоретическую информацию и программу по распознаванию рукописных цифр на языке Python. Запустить программу в среде Spyder(Anaconda), проанализировать полученные результаты. Затем внести правки в код в соответствии с вариантом и сравнить результаты с полученными ранее.

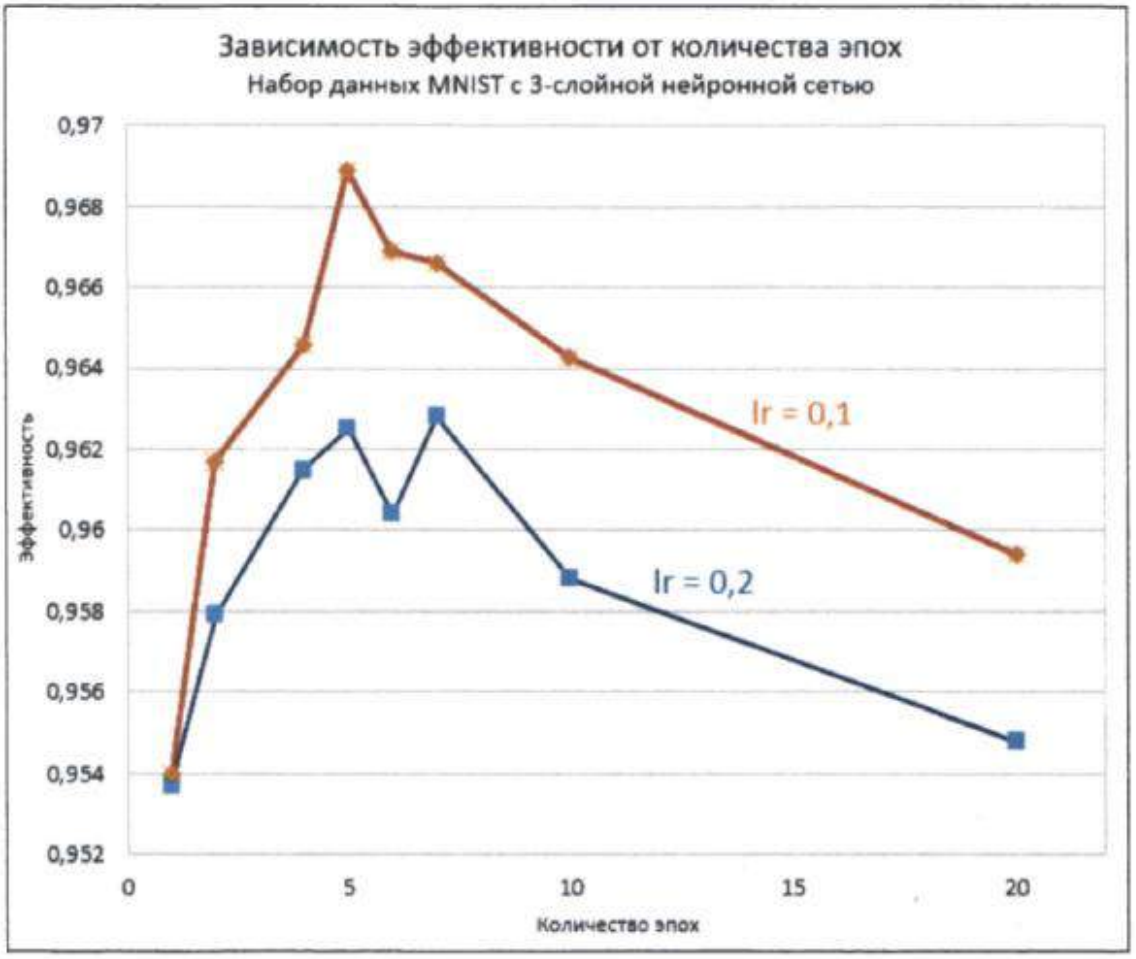
Опросить сеть с использованием тестовой базы данных (собственноручно написанных цифр).

Варианты лабораторных заданий:

В соответствии с таблицей задаются коэффициент обучения, количество эпох, количество скрытых узлов:

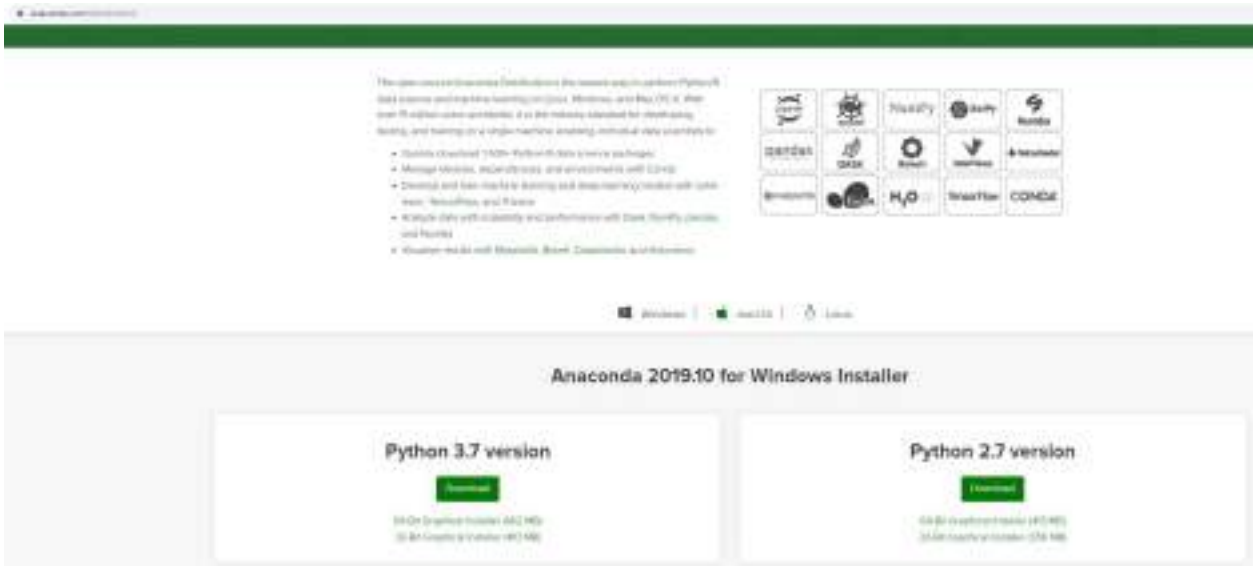
	1	2	3	4	5	6	7	8	9	10
Коэф. обучения	0.1	0.2	0.3	0.1	0.2	0.3	0.1	0.2	0.3	0.1
Кол. эпох	4	5	6	7	4	5	6	7	5	7
Кол. скр. узлов	80	100	120	140	120	140	160	180	200	180


Зависимость эффективности от количества скрытых узлов
1
0,95
0,9
0,85
0,8
0,75
0,7
0,65
0,6
Эффективность
0
100
200
300
400
500
600
Количество скрытых узлов

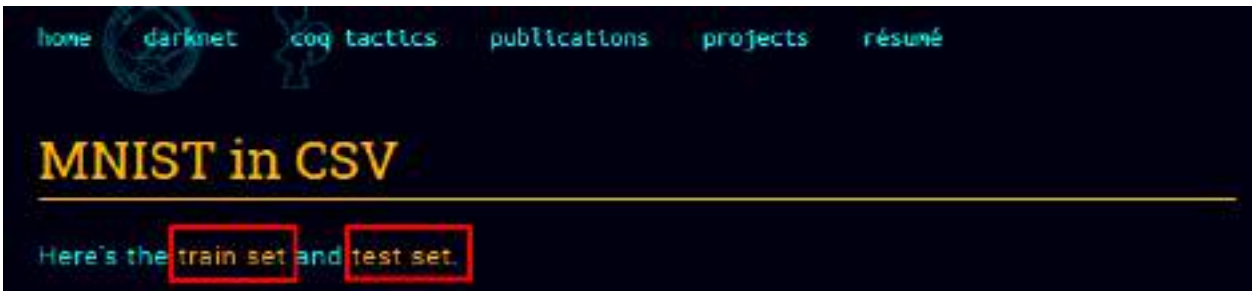

Зависимость эффективности от количества эпох
Набор данных MNIST с 3-слойной нейронной сетью
0,97
0,968
0,966
0,964
0,962
0,96
0,958
0,956
0,954
0,952
Эффективность
lr = 0,1
lr = 0,2
0
5
10
15
20
Количество эпох

Порядок выполнения работы

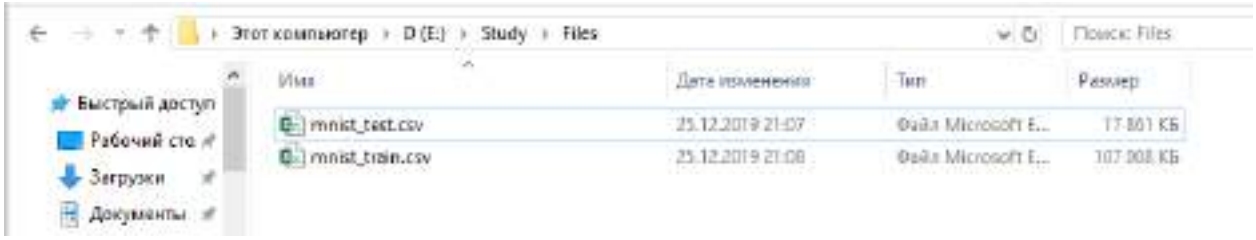
Первым делом необходимо установить всё необходимое для начала работы. Установить Python можно с официально сайта https://www.python.org/getit/, можно взять версию Python 3.7. Затем с официального сайта установим среду Spyder (Anaconda) https://www.anaconda.com/distribution/, выбираем версию для Python 3.7 и соответствующую разрядность.



И наконец, скачаем файлы тестового и тренировочного набора с сайта https://pjreddie.com/projects/mnist-in-csv/. Ссылки выделены красным на рисунке ниже.

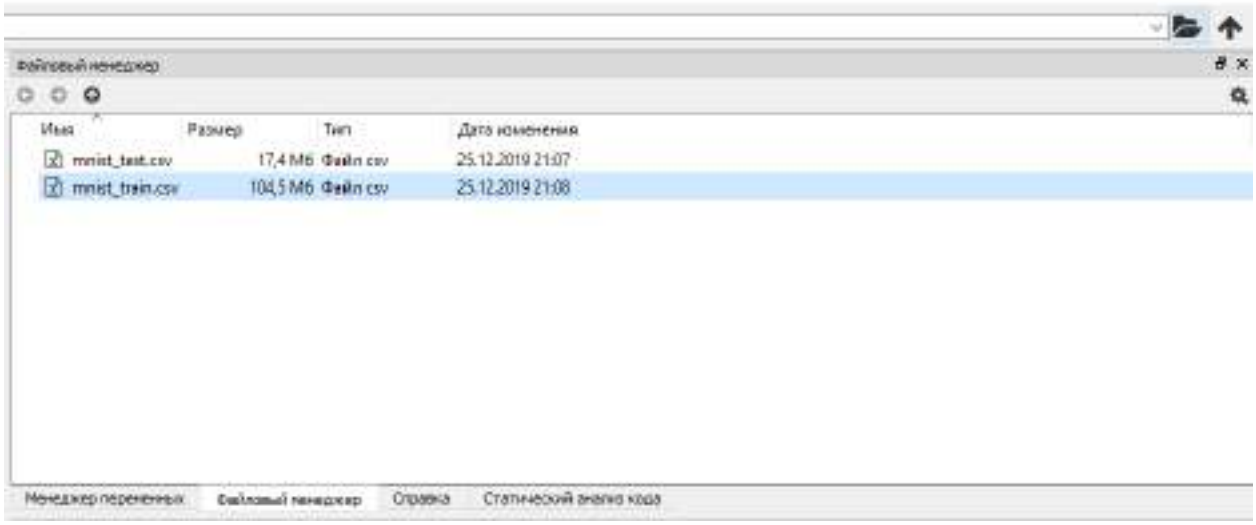


Скачанные файлы расположим в удобной для нас папке.



После запуска среды Spyder в правой части видим консоль, где нажав на шестеренку выполним перезапуск ядра.

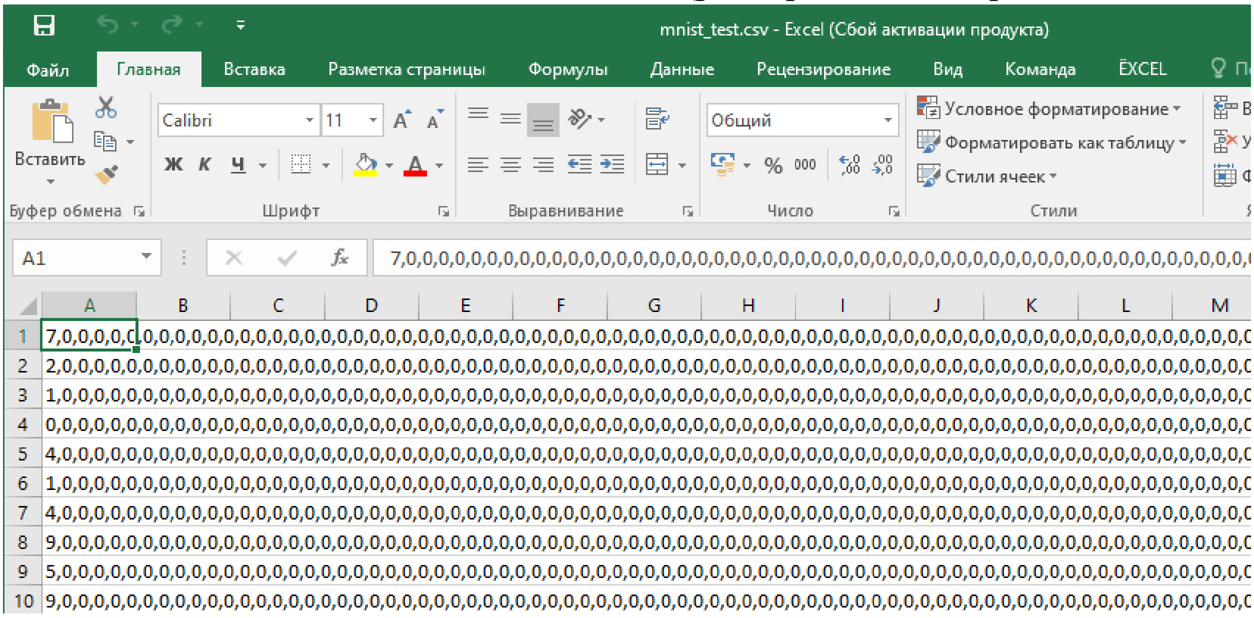
Также в правой части выше видим значок папки для выбора пути «Файлового менеджера». Необходимо выбрать ту папку, куда мы поместили файлы тестового и тренировочного набора.



Тренировочный набор содержит 60 000 промаркированных экземпляров, используемых для тренировки нейронной сети. Слово “промаркированные” означает, что для каждого экземпляра указан соответствующий правильный ответ.

Меньший тестовый набор, включающий 10 000 экземпляров, используется для проверки правильности работы идей или алгоритмов. Он также содержит корректные маркеры, позволяющие увидеть, способна ли наша нейронная сеть дать правильный ответ.

Присмотримся к файлу. Он содержит маркер цифры в начале строки и затем значения 784 пикселей (размерность картинки 28x28).



После этого представленный ниже код вставляем в рабочую область.

```
# Блокнот Python для книги "Создаем нейронную сеть".
# Код для создания 3-слойной нейронной сети вместе с
# кодом для ее обучения с помощью набора данных MNIST.
#  (c)  Tariq Rashid,  2016
# лицензия GPLv2
import numpy
# библиотека scipy.special содержит сигмоиду expit ()
import scipy.special
# библиотека для графического отображения массивов
#import matplotlib.pyplot
# гарантировать размещение графики в данном блокноте,
# а не в отдельном окне
# определение класса нейронной сети
class neuralNetwork:
    # инициализировать нейронную сеть
    def __init__(self,  inputnodes,  hiddennodes,
outputnodes, learningrate):
    # задать количество узлов во входном,  скрытом и
#выходном слое
        self.inodes = inputnodes
        self.hnodes = hiddennodes
        self.onodes = outputnodes
        # Матрицы весовых коэффициентов связей wih
#(между входным и скрытым слоями)и who  (между скрытым и
```

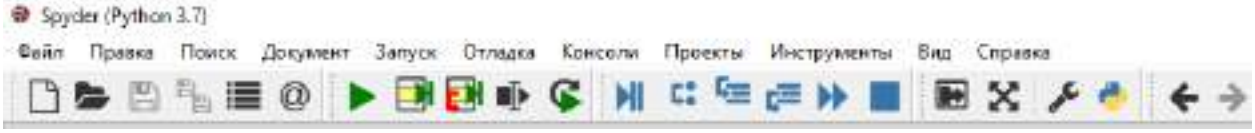
```
#выходным слоями).
        # Весовые коэффициенты связей между узлом i и
#узлом j следующего слоя обозначены как w__i_j :
        # wll w21
        # wl2 w22 и т.д.
        self.wih = numpy.random.normal(0.0,
pow(self.hnodes,  -0.5),(self.hnodes,  self.inodes))
        self.who = numpy.random.normal(0.0,
pow(self.onodes,  -0.5),(self.onodes,  self.hnodes))
        # коэффициент обучения
        self.lr = learningrate
        # использование сигмоиды в качестве функции
#активации
        self.activation_function = lambda x:
scipy.special.expit(x)
        pass
    # тренировка нейронной сети
    def train(self,  inputs_list,  targets_list):
        # преобразование списка входных значений
        # в двухмерный массив
        inputs = numpy.array(inputs_list,  ndmin=2).T
        targets = numpy.array(targets_list,  ndmin=2).T
        # рассчитать входящие сигналы для скрытого слоя
        hidden_inputs = numpy.dot(self.wih,  inputs)
        # рассчитать исходящие сигналы для скрытого слоя
        hidden_outputs = self.activation_function
(hidden_inputs)
        # рассчитать входящие сигналы для выходного слоя
        final_inputs = numpy.dot(self.who,
hidden_outputs)
        # рассчитать исходящие сигналы для выходного
слоя
        final_outputs = self.activation_function
(final_inputs)
        # ошибки выходного слоя =
        #  (целевое значение - фактическое значение)
        output_errors = targets - final_outputs
        # ошибки скрытого слоя - это ошибки
#output_errors,
        # распределенные пропорционально весовым
#коэффициентам связей
        # и рекомбинированные на скрытых узлах
        hidden_errors = numpy.dot(self.who.T,
output_errors)
```

```
        # обновить веса для связей между скрытым
#и выходным слоями
        self.who += self.lr * numpy.dot((output_errors *
final_outputs *
        (1.0 - final_outputs)), numpy.transpose
        (2.0 (hidden_outputs))
        # обновить весовые коэффициенты для связей между
        # входным и скрытым слоями
        self.wih += self.lr * numpy.dot((hidden_errors *
hidden_outputs *
        (1.0 - hidden_outputs)), numpy.transpose
        (2.0 (inputs))
        pass
    # опрос нейронной сети
    def query(self,  inputs_list):
        # преобразовать список входных значений
        # в двухмерный массив
        inputs = numpy.array(inputs_list,  ndmin=2).T
        # рассчитать входящие сигналы для скрытого слоя
        hidden_inputs = numpy.dot(self.wih,  inputs)
        # рассчитать исходящие сигналы для скрытого слоя
        hidden_outputs = self.activation_function
hidden_inputs)
        # рассчитать входящие сигналы для выходного слоя
        final_inputs = numpy.dot(self.who,
hidden_outputs)
        # рассчитать исходящие сигналы для выходного
слоя
        final_outputs = self.activation_function
final_inputs)
        return final_outputs
# количество входных,  скрытых и выходных узлов
input_nodes = 784
hidden_nodes =10
output_nodes =10
# коэффициент обучения
learning_rate =0.1
# создать экземпляр нейронной сети
n = neuralNetwork(input_nodes,hidden_nodes,
output_nodes,learning_rate)
# загрузить в список тренировочный набор данных
# CSV-файла набора MNIST
training_data_file = open("mnist_train.csv", 'r')
training_data_list = training_data_file.readlines()
```

```
training_data_file.close()
# тренировка нейронной сети
# переменная epochs указывает,  сколько раз
#тренировочный набор данных используется для тренировки
#сети
epochs = 2
for e in range(epochs):
    # перебрать все записи в тренировочном наборе данных
    for record in training_data_list:
    # получить список значений из записи,  используя
символы
    # запятой  (',')  в качестве разделителей
        all_values = record.split(',')
        # масштабировать и сместить входные значения
        inputs =  (numpy.asfarray(all_values[1:])  /
255.0  *  0.99)  + 0.01
        # создать целевые выходные значения  (все равны
#0,01,  за исключением желаемого маркерного значения,
#равного 0,99)
        targets = numpy.zeros(output_nodes)  +0.01
        # all_values[0]  - целевое маркерное значение
        # для данной записи
        targets[int(all_values[0])]  =0.99
        n.train(inputs, targets)
    pass
pass
# загрузить в список тестовый набор данных
# CSV-файла набора MNIST
test_data_file = open("mnist_test.csv", 'r')
test_data_list = test_data_file.readlines()
test_data_file.close()
# тестирование нейронной сети
# журнал оценок работы сети,  первоначально пустой
scorecard =  []
# перебрать все записи в тестовом наборе данных
#for record in test_data_list:
    # получить список значений из записи,  используя
#символы запятой  (',*)  в качестве разделителей
    all_values = record.split(',')
    # правильный ответ - первое значение
    correct_label = int(all_values[0])
    # масштабировать и сместить входные значения
    inputs = (numpy.asfarray(all_values[1:]) / 255.0 *
0.99) + 0.01
```

```
    # опрос сети
    outputs = n.query(inputs)
    # индекс наибольшего значения является маркерным
#значением
    label = numpy.argmax(outputs)
    # присоединить оценку ответа сети к концу списка
    if  (label == correct_label):
        # в случае правильного ответа сети присоединить
        # к списку значение 1
        scorecard.append(1)
    else:
        # в случае неправильного ответа сети
# присоединить к списку значение 0
        scorecard.append(0)
    pass
pass
# рассчитать показатель эффективности в виде
# доли правильных ответов
scorecard_array = numpy.asarray(scorecard)
print  ("Эффективность = ",  scorecard_array.sum()  /
scorecard_array.size)
```

Выполнение программы можем осуществить с помощью выполнения всего файла или выполнять отдельными блоками. Нужные кнопки найдем на панели инструментов.



Результат выполнения будем наблюдать в консоли.

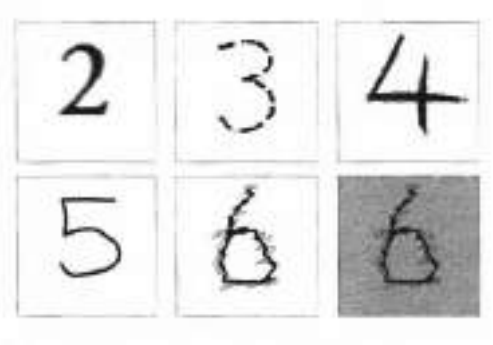
На последнем этапе можно изменять различные параметры и проанализировать полученные результаты.

Использование собственного рукописного текста.

Создадим тестовую базу данных, используя собственноручно написанные цифры. Можно сымитировать различные виды почерка, а также зашумленные и

прерывистые изображения, чтобы посмотреть, как с ними справится нейронная сеть.

Для создания изображений можно использовать любой графический редактор типа Photoshop, бесплатную программу с открытым исходным кодом GIMP (www.gimp.org). Можно даже написать цифры карандашом на листе бумаги, затем отсканировать или сфотографировать их с помощью смартфона или фотокамеры. Единственным требованием является то, что изображение цифры должно быть квадратным (длина равна ширине) и представленным в формате png. Этот формат можно выбрать в списке допустимых форматов большинства графических редакторов при выполнении команды меню Файл→Сохранить как или Файл→Экспорт.



Создадим уменьшенные версии PNG-изображений, масштабировав их до размера 28×28 пикселей, чтобы привести в соответствие с использовавшимися ранее данными MNIST. Для этого можно использовать свой графический редактор.

Для чтения и декодирования данных из распространенных форматов изображений, включая PNG, используем библиотеки Python.

```
import scipy.misc
img_array = scipy.misc.imread(image_file_name,
flatten=True)
img_data  = 255.0 - img_array.reshape(784)
img_data = (img_data / 255.0 * 0.99) + 0.01
```

Функция scipy.misc.imread() поможет в получении данных из файлов изображений, таких как PNG- или JPG-файлы. Чтобы использовать библиотеку scipy.misc, ее необходимо импортировать. Параметр flatten=True превращает изображение в простой массив чисел с плавающей запятой и, если изображение цветное, переводит цветовые коды в шкалу оттенков серого, что нам и надо.

Следующая строка преобразует квадратный массив размерностью 28×28 в длинный список значений, который нужен для передачи данных нейронной

сети. Вычитание значений массива из 255,0 необходимо сделать, т.к. обычно коду 0 соответствует черный цвет, а коду 255-белый, но в наборе данных MNIST используется обратная схема, в связи с чем мы должны инвертировать цвета для приведения их в соответствие с соглашениями, принятыми в MNIST.

Последняя строка выполняет масштабирование данных, переводя их в диапазон значений от 0,01 до 1,0.

Образец кода, демонстрирующий чтение PNG-файлов:

https://github.com/makeyourownneuralnetwork/makeyourownneuralnetwork/blob/master/part3_load_own_images.ipynb

Версия программы, использовавшейся ранее для создания базовой нейронной сети и ее обучения с помощью набора данных MNIST, но теперь будем тестировать программу с использованием набора данных, созданного на основе наших изображений:

https://github.com/makeyourownneuralnetwork/makeyourownneuralnetwork/blob/master/part3_neural_network_mnist_and_own_data.ipynb

В отчете должны быть результаты опроса сети с использованием ваших изображений.

Сделать выводы о том, что удалось или не удалось распознать нейронной сети.

Отчет по лабораторной работе должен включать:

1. Титульный лист.
2. Краткое описание предметной области и решаемой задачи (что является входными данными, каков результат работы системы).
3. Модифицированую программную реализацию нейронной сети.
4. Код программы, тестирование работы нейронной сети.
5. Выводы.

Контрольные вопросы

1. Что такое нейронная сеть и каковы области её применения?
2. Какие основные этапы работы нейронной сети?

3. Какие параметры можно изменять для улучшения результата?

4. Какие рукописные цифры удалось/не удалось распознать нейронной сети и почему.

Для понимания темы будет полезно изучить материалы книг и статей; гиперссылки на ресурсы приведены ниже.

1. Создаем нейронную сеть: https://ru.pdfdrive.com/%D0%A1%D0%BE%D0%B7%D0%B4%D0%B0%D0%B5%D0%BC-%D0%BD%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D1%83%D1%8E-%D1%81%D0%B5%D1%82%D1%8C-e189509202.html

2. Наглядное введение в нейросети на примере распознавания цифр: https://proglib.io/p/neural-network-course/

3. Нейросети и глубокое обучение, глава 1: использование нейросетей для распознавания рукописных цифр: https://habr.com/ru/post/456738/

Лабораторная работа № 7

Логические модели представления знаний на примере проблемно-ориентированной облачной системы

Цель работы: овладение основными навыками анализа предметной области при построения логических моделей искусственного интеллекта. В процессе изучения теоретического материала и практической работы необходимо освоить основные аспекты построения логических моделей для заданной предметной области (например, для интеллектуальной киберфизической системы, облачной среды, регулирующей доступ пользователей к вычислительным ресурсам – кластерам компьютеров или к суперкомпьютерам и т.п.)

Описание данной лабораторной работы составлено при участии аспиранта М. Е. Федосина и является внедрением его диссертационной работы.

Описание предметной области

Облачные вычисления – это модель предоставления сервиса (услуги), при которой пользователь имеет возможность получить повсеместный, удобный, доступ по требованию к пулу (англ. *pool*) разделяемых, конфигурируемых ресурсов (например, сетей, серверов, памяти, приложений), которые могут быть быстро предоставлены пользователю и с минимальными для пользователя усилиями по взаимодействию с сервис-провайдерами в процессе получения доступа к ресурсам.

Национальным институтом стандартов и технологий США зафиксированы следующие обязательные характеристики облачных вычислений:

– самообслуживание по требованию (англ. *self service on demand*), при котором потребитель самостоятельно определяет и изменяет вычислительные потребности, такие как серверное время, скорости доступа и обработки данных, объем хранимых данных без взаимодействия с представителем поставщика услуг;

– универсальный доступ по сети, услуги доступны потребителям по сети передачи данных вне зависимости от используемого терминального устройства;

– объединение ресурсов (англ. *resource pooling*), когда поставщик услуг объединяет ресурсы для обслуживания большого числа потребителей в единый пул для динамического перераспределения мощностей между потребителями в условиях постоянного изменения спроса на мощности; при этом потребители контролируют только основные параметры услуги (например, объем данных, скорость доступа), но фактическое распределение

ресурсов, предоставляемых потребителю, осуществляет поставщик (в некоторых случаях потребители все-таки могут управлять некоторыми физическими параметрами перераспределения, например, указывать желаемый центр обработки данных из соображений географической близости);

– «эластичность», когда услуги могут быть предоставлены, расширены, сужены в любой момент времени, без дополнительных издержек на взаимодействие с поставщиком, как правило, в автоматическом режиме;

– учет потребления, когда поставщик услуг автоматически исчисляет потребленные ресурсы на определенном уровне абстракции (например, объем хранимых данных, пропускная способность, количество пользователей, количество транзакций) и на основе этих данных оценивает объем предоставленных потребителям услуг.

С точки зрения поставщика, благодаря объединению ресурсов и непостоянному характеру потребления со стороны потребителей, облачные вычисления позволяют экономить на масштабах, используя меньшие аппаратные ресурсы, чем требовались бы при выделенных аппаратных мощностях для каждого потребителя, а за счет автоматизации процедур модификации выделения ресурсов существенно снижаются затраты на абонентское обслуживание.

С точки зрения потребителя эти характеристики позволяют получить услуги с высоким уровнем доступности (англ. *high availability*) и низкими рисками неработоспособности, обеспечить быстрое масштабирование вычислительной системы благодаря эластичности без необходимости создания, обслуживания и модернизации собственной аппаратной инфраструктуры.

Удобство и универсальность доступа обеспечивается широкой доступностью услуг и поддержкой различного класса терминальных устройств (персональных компьютеров, мобильных телефонов, планшетных ПК).

Такой подход к организации услуг дает возможность получить следующие преимущества:

– удаленный доступ к вычислительным ресурсам с любого рабочего места вне зависимости от географического положения пользователей;

– масштабируемость предоставляемых услуг;

– возможность полного самообслуживания пользователей в отношении подготовки к работе, управлению и завершению предоставления услуг;

– высокая степень доступности услуг за счет возможности быстрого восстановления после вероятных катастроф;

– высокий уровень эффективности использования аппаратных ресурсов (обслуживание большего числа пользователей за счет обеспечения возможности общего использования вычислительных ресурсов системы для решения задач с разными, часто взаимоисключающими требованиями к вычислительной инфраструктуре); за счет этого возможно обеспечить снижение стоимости единицы ресурса;

– экономия за счет унифицированных механизмов обслуживания и администрирования.

Существуют различные сервисные модели облачных вычислений, но наибольшее распространение получили:

– программное обеспечение как услуга (SaaS, англ. Software as a Service): модель, в которой потребителю предоставляется возможность использования прикладного программного обеспечения провайдера, работающего в облачной инфраструктуре и доступного из различных клиентских устройств, например, из веб-браузера. Контроль и управление основной физической и виртуальной инфраструктурой облака, в том числе сети, серверов, операционных систем, хранения, или даже индивидуальных возможностей приложения (за исключением ограниченного набора пользовательских настроек конфигурации приложения) осуществляется облачным провайдером;

– платформа как услуга (PaaS, англ. Platform as a Service): модель, в которой потребителю предоставляется возможность использования облачной инфраструктуры для размещения базового программного обеспечения для последующего размещения на нем новых или существующих приложений (собственных, разработанных на заказ или приобретенных тиражируемых приложений). В состав таких платформ входят инструментальные средства создания, тестирования и выполнения прикладного программного обеспечения – системы управления базами данных, связующее программное обеспечение, среды исполнения языков программирования, предоставляемые облачным провайдером.

В модели SaaS распространяются многие типы прикладного программного обеспечения: офисные пакеты (Google Docs, Office 360, ZohoDocs), графические редакторы (Pixlr, picnik), клиенты электронной почты (Gmail, iNotes), ERP-системы (Microsoft Dynamics AX, SAP Business ByDesign), CRM (Salesforce.com, ZohoCRM) и др. Существует также множество облачных платформ для разработки таких приложений (Google Apps Engine, Microsoft Azure, Amazon AWS).

Однако для класса высокопроизводительного программного обеспечения с графическим интерфейсом пользователя отсутствует возможность его распространения в модели SaaS. Одновременно с этим не существует облачной платформы по предоставлению средств разработки и тестирования таких приложений в модели PaaS.

Веб-лаборатории. В связи с тем, что в мире наблюдается потребность в объединении ученых, исследователей и учащихся в социальные научные сети, где они могут найти необходимые им публикации, выявить актуальные тенденции в данной тематике, узнать о проведении тематических конференций, установить контакт друг с другом и, наконец, проводить совместные исследования, была разработана концепция виртуальных информационно-

вычислительных веб-лабораторий или «хабов» (англ. *hubs*). Затем на основе этой концепции была разработана референсная архитектура системы под названием HUBzero. Основной целью, для которой используется эта система, является организация предметно-ориентированных научных сообществ с возможностью совместной работы и доступа к прикладным моделям и данным.

С точки зрения пользователя веб-лаборатория представляет собой сайт, доступный через Интернет, который позволяет:

- получать безопасный доступ к размещенным ресурсам;
- публиковать информацию о себе и своих исследовательских работах;
- осуществлять поиск коллег в своей области, а также смежных научных областях;
- поддерживать связь со своими коллегами;
- организовывать коллективы по интересам для дальнейшего совместного участия;
- получать актуальную информацию о направлениях деятельности различных научных коллективов, проводимых конференциях и тому подобное.
- использовать материалы веб-лаборатории: обучающие курсы, статьи, инструкции по использованию приложений и др.;
- запускать приложения в вычислительной инфраструктуре веб-лаборатории;
- получать помощь исследователей в разработке, установке и дальнейшей поддержке приложений.

Важным аспектом организации таких веб-лабораторий является сокрытие от пользователей сложностей, связанных с авторизацией, обменом данными, запуском вычислительных приложений, получением результатов и их визуализацией.

Возможности веб-лаборатории. Взаимодействие пользователей с веб-лабораторией происходит через веб-браузер. Веб-лаборатория предоставляет пользователям следующие возможности:

- получение интерактивного доступа к прикладному программному обеспечению в концепции облачных вычислений;
- предоставление площадки для разработки и тестирования собственных приложений, которые в дальнейшем могут быть размещены в веб-лаборатории;
- средства для взаимодействия пользователей.

Для получения доступа к программному обеспечению веб-лаборатории пользователю необходимо авторизоваться на сайте, выбрать нужное ему приложение из предложенного списка и запустить его.

Запуск приложения приводит к созданию сессии, которая выполняется на вычислительных узлах веб-лаборатории и отображается в браузере пользователя. При этом работа с приложением происходит в интерактивном графическом режиме и не отличается от работы с локальным настольным

приложением. Тем не менее, иногда работа с приложением в окне браузера бывает недостаточно удобной и для таких случаев присутствует возможность использования автономного окна для приложения.

Важной особенностью является то, что выполнение сессии происходит удаленно, пользователь может завершить работу своего компьютера и позднее вернуться к запущенным сессиям с помощью любого компьютера с доступом в Интернет.

Пользователь может получить результаты выполнения с помощью протокола WebDAV. Через него можно работать с удаленной файловой системой так, как если бы она была локальной.

Для организации совместной работы сервисы веб-лаборатории позволяют разделять рабочую сессию запускаемого приложения между несколькими пользователями, список которых должен определить ее владелец путем обновления параметров запущенной сессии, то есть добавления регистрационных имен учетных записей выбранных пользователей. Помимо этого, владелец может ограничить доступ на изменение сессии и она будет доступна только для просмотра. Эти пользователи могут также получить доступ к результатам завершенной сессии приложения.

Для предоставления доступа к описанным выше сервисам веб-лаборатория должна в частности содержать следующие аппаратные компоненты (рисунок 7.1):

– группу серверов, обеспечивающих функционирование веб-ядра (1);
– хранилище виртуальных контейнеров (2);
– систему хранения для пользовательских данных и приложений (3);
– высокопроизводительные вычислительные ресурсы (4).

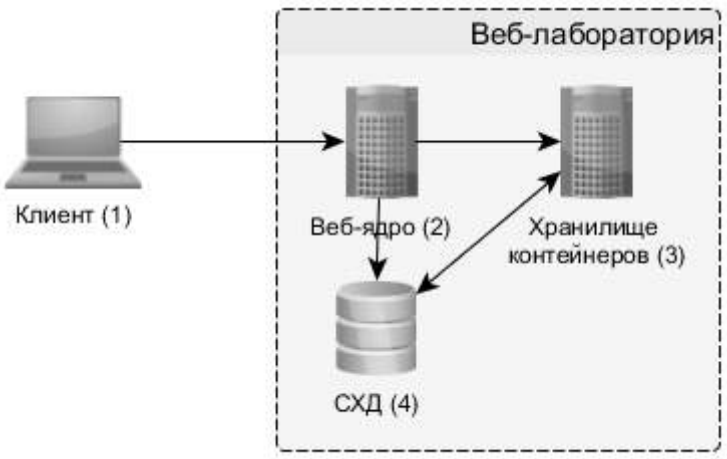


Рисунок 7.1 – Аппаратная архитектура веб-лаборатории

Работа с веб-лабораторией происходит с помощью клиента (1) через веб-ядро (2), которое предоставляет все возможности по взаимодействию пользователей в рамках социальной научной сети и отвечает за доступ к размещенным ресурсам.

Хранилище виртуальных контейнеров (3) содержит множество работающих виртуальных машин, где выполняются запущенные пользователями приложения, доступ к которым предоставляется с помощью специального клиента в веб-браузере.

Система хранения данных (СХД) (4) содержит домашние каталоги пользователей и бинарные файлы приложений веб-лаборатории, которые подключаются к контейнеру сессии при запуске. Основные этапы проектирования архитектуры ПООС представлены на рисунке 7.2.

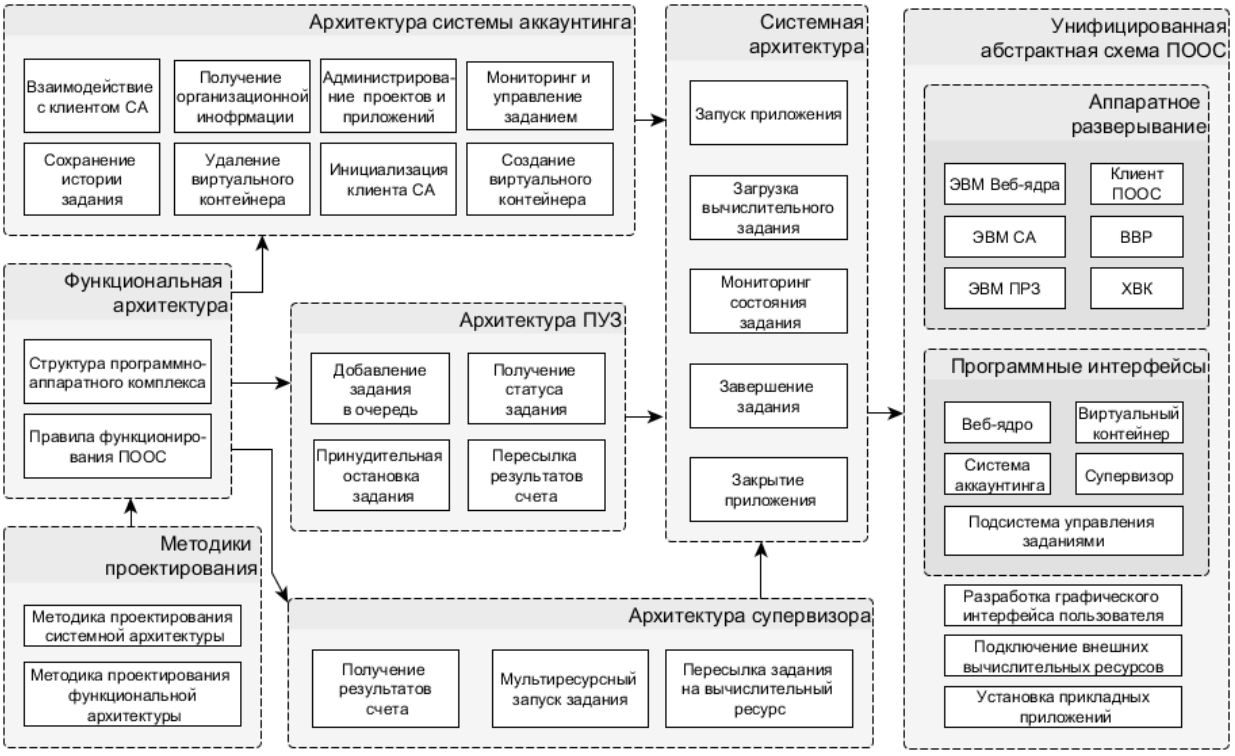


Рисунок 7.2 – Основные этапы проектирования архитектуры ПООС

Архитектура предметно-ориентированной облачной среды

Архитектура ПООС, представленная рисунком 7.3, определяется множеством объектов различных типов, структурных и логических связей между ними. Функциональная архитектура ПООС, определяется фиксацией состава ее программных компонентов с определенными функциями, структурными и логическими межкомпонентными связями, а также сценариями ее поведения. Дополнительным уточнением функциональной архитектуры ПООС являются ***логические правила функционирования***, задающие инварианты поведения при выполнении операций.

Аппаратная часть ПООС, спроектированной на основе разработанной методики, содержит:

– веб-ядро, которое предоставляет интерфейс для работы с ПООС, отвечает за авторизацию и хранение данных пользователей и выполняет рассылку информационных сообщений;

– хранилище контейнеров, внутри которых выполняются запущенные приложения, удаленно управляемые пользователями с помощью клиентов ПООС;

– систему хранения данных (СХД) для бинарных файлов приложений и домашних каталогов пользователей, которые подключаются к контейнеру виртуальной машины при запуске;

– ЭВМ подсистемы распределения заданий, которая занимается распределением заданий пользователей по необходимым вычислительным ресурсам, а также обработкой результатов выполнения заданий;

– ЭВМ системы аккаунтинга, в задачи которой входит обработка запросов на возможность проведения счета, а также хранение данных о доступных пользователям ресурсах и истории запусков;

– внешние подключаемые высокопроизводительные вычислительные ресурсы – кластеры, на которых происходит основное выполнение приложений.

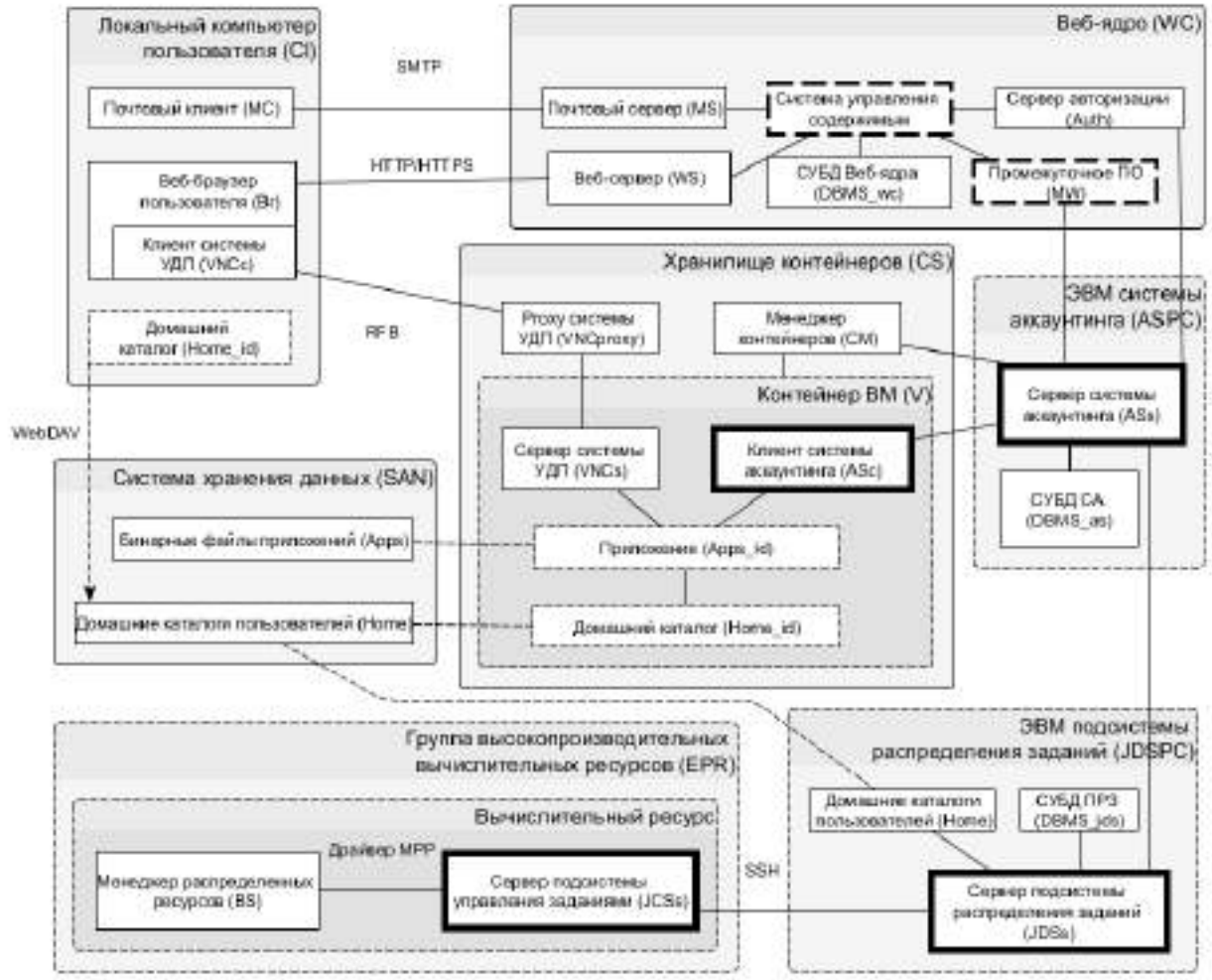


Рисунок 7.3 – Архитектура программно-аппаратного комплекса ПООС

Взаимодействие пользователя с системой происходит через веб-браузер по протоколу HTTP/HTTPS. Для этого он заходит на специальный портал ПООС, где ему после авторизации предоставляется список сервисов.

Отличительной особенностью разработанной концепции является наличие системы аккаунтинга ресурсов (СА). Система состоит из четырех частей: клиентской, работающей в контейнере виртуальной машины, сервера подсистемы распределения заданий (ПРЗ) и основного сервера СА, запущенных на выделенных ЭВМ, а также множества серверов подсистемы управления заданиями (ПУЗ), которые выполняются на каждом подключенном к ПООС вычислительном ресурсе. Соответствующие новые компоненты выделены на рисунке рамками, а ранее существовавшие модифицированные компоненты обозначены пунктиром.

В задачи системы аккаунтинга ресурсов входят:

1. Контроль над использованием ресурсов ПООС.
2. Распределение заданий пользователей по необходимым вычислительным ресурсам.
3. Осуществление проверки на возможность получения пользователем доступа к ресурсам ПООС.
4. Ограничение доступа в случае, если проверка не пройдена.
5. Обработка и хранение результатов запусков.
6. Поддержка формирования отчетов по осуществленным запускам.

Клиентская часть выполняется внутри виртуального контейнера приложения. Она занимается перехватом и обработкой запросов от приложения к менеджеру распределенных ресурсов (МРР) и последующей передачей запросов серверу системы аккаутинга.

Сервер подсистемы распределения задний, запущенный на выделенной ЭВМ ПРЗ, выполняет ряд ключевых функций: оперативный выбор вычислительного ресурса для проведения расчетов, пересылку вычислительных заданий и запросов СА необходимому серверу ПУЗ, обработку полученных результатов счета и перемещение их в домашний каталог пользователя.

Сервер системы аккаунтинга запущен на выделенной ЭВМ СА и является центральной частью ПООС. Главной задачей сервера СА является прием вычислительных заданий от различных клиентов СА и проверка возможности их запуска на основе количества доступных средств на счету соответствующих проектов. Сервер также обрабатывает информационные запросы о статусе выполняемых заданий, осуществляет управление ходом выполнения расчетов, хранит результаты запусков и формирует отчеты о проведенных исследованиях.

Сервер подсистемы управления заданиями выполняется на каждом вычислительном ресурсе ПООС. Сервер ПУЗ запускается от имени специально созданного служебного пользователя ПООС и осуществляет взаимодействие с локальным менеджером распределенных ресурсов. Принимая запросы от сервера ПРЗ, сервер ПУЗ реализует добавление задания в очередь выполнения,

мониторинг текущего состояния, отмену выполнения задания по требованию пользователя и пересылку результатов выполнения серверу ПРЗ.

Логическое описание функциональной архитектуры ПООС

Для построения функциональной архитектуры ПООС было предложено математическое описание, соответствующее структуре на рисунке, и базирующееся на математическом аппарате многоосновного исчисления предикатов первого порядка,

Для ПООС в настоящей работе определены основные множества объектов. Элементы каждого из основных множеств обладают общими свойствами, характеризуемыми «сортами». В настоящей работе, например, используются сорта «программа», «ЭВМ», «виртуальный объект» и др.

Каждому функциональному k-арному символу $f \in F$ сопоставлена схема функции $s_1 \times s_2 \times \ldots \times s_k \rightarrow s_{k+1}$; $s_i \in Sort$ ($i = 1, 2, \ldots, k+1$), а каждому k-арному предикатному символу $p \in P$ сопоставлена схема предиката (высказывательной функции) $s_1 \times s_2 \times \ldots \times s_k \rightarrow T$; $T, s_i \in Sort$ ($i = 1, 2, \ldots, k$). Нульарные функции выделяют элементы в некотором множестве.

Далее будут использоваться предикаты вида

$$p: A_1 \times A_2 \ldots \times A_k \rightarrow \{true, false\}$$

и функции вида

$$f: A_1 \times A_2 \ldots \times A_k \rightarrow A_{k+1}$$

с определенными выше схемами.

Специальный тип T интерпретируется как двухэлементное множество истинностных значений (другими словами, $T = \{true, false\}$). В логико-алгебраических моделях сортам соответствуют одноименные множества.

В разработанных логических моделях классам сущностей системы предписаны следующие ***сорта***, или ***типы*****:** *app* – прикладное приложение, установленное в ПООС; *job* – выполняющееся на ресурсе ПООС счетное задание; *client* – подключенный к контейнеру удаленный клиент, с помощью которого происходит управление приложением; *user* – зарегистрированный пользователь ПООС; *time* – время; *cputime* – процессорное время; *account* – кредитный счет проекта; *resource* – подключенный к ПООС вычислительный ресурс, на котором выполняются вычислительные задания; *project* – проект, в рамках которого запущено приложение и будут выполняться задания; *container* – виртуальный контейнер, содержащий работающее приложение. В логико-алгебраических моделях сортам соответствуют одноименные множества.

В логико-алгебраической модели функциональной архитектуры ПООС определены следующие функции. Основные функции для работы с заданиями, возвращающие название приложения, из которого запускается задание (*jobapp*),

имя ресурса, на котором происходит счет (*jobres*), идентификатор контейнера задания (*jobcont*), общее количество требуемых заданию ресурсов (*jobtime*) и затраченное заданием процессорное время (*jobwork*):

jobapp: *job* → *app*; *jobres*: *job* → *resource*; *jobcont*: *job* → *container*; *jobtime*: *job* → *cputime*; *jobwork*: *job*×*time*×*time* → *cputime*.

Временны́е функции задания, задающие время отправки задания (*send*), начала (*start*) и конца (*end*) выполнения задания:

send: *job* → *time*; *start*: *job* → *time*; *end*: *job* → *time*.

Функции клиента ПООС, определяющие имя пользователя, запустившего клиент (*clientuser*), и идентификатор контейнера, к которому клиент подключен (*clientcont*):

clientuser: *client* → *user*; *clientcont*: *client* → *container*.

Функции владельца, которые возвращают имя пользователя, создавшего контейнер (*owner*), и название проекта, от имени которого из этого контейнера запускаются задания (*ownerproj*):

owner: *container* → *user*; *ownerproj*: *container* → *project*.

Счетное множество констант, обозначающих время:

0: → *time*; 1: → *time*; …; *t*: → *time*.

Функции работы с денежными величинами для получения текущего состояния счета проекта в определенный момент времени (*acc*) и определения количества средств, необходимых для выполнения задания (*cost*):

acc: *project*×*time* → *account*; *cost*: *app*×*cputime* → *account*.

Служебные предикаты, определяющие наличие у вычислительного ресурса возможности запуска задания на требуемый интервал времени (*free*); наличие у вычислительного ресурса поддержки определенного типа приложений (*appsupport*); авторизован ли пользователь в системе (*auth*); состоит ли пользователь в определенном проекте (*userproj*); доступен ли контейнер, созданный первым пользователем, для второго пользователя (*shared*):

free: *resource*×*cputime* → *T*; *appsupport*: *resource*×*app* → *T*; *auth*: *user* → *T*; *userproj*: *user*×*project* → *T*; *shared*: *user*×*user*×*container* → *T*.

Дополнительные двуместные функции и предикаты, определенные в инфиксной форме $e_1 f e_2$, где e_1 и e_2 – типы, а f – имя функции или предиката, предназначенные соответственно для сложения натуральных чисел, представляющих денежные величины ($+), а также предикаты сравнения временных (≤) и денежных ($≤) величин на «меньше или равно»:

$+: $account \times account \rightarrow account$; ≤: $time \times time \rightarrow T$; $≤: $account \times account \rightarrow T$.

На основе описанных выше функций и предикатов представлены знания о работе ПООС и, в частности, системы аккаунтинга:

1. Владелец контейнера должен состоять в проекте, от имени которого создается контейнер:

$(\forall c \in container)\ (\exists\ u \in user, p \in project)\ [((owner(c) = u)\ \&\ (ownerproj(c) = p)) \rightarrow (auth(u)\ \&\ userproj(u, p))]$.

2. Доступ к контейнеру может получить либо участник соответствующего проекта, выбранный владельцем, либо сам владелец:

$(\forall cl \in client)(\exists c \in container, u \in user)\ [(clientuser(cl) = u)\ \&\ (clientcont(cl) = c)\ \&\ auth(u)\ \&\ userproj(u, ownerproj(c))\ \&\ ((owner(c) = u) \vee shared(owner(c), u, c))]$.

3. Запуск задания разрешен только владельцу или пользователю, которому предоставлен доступ:

$(\forall j \in job)(\exists c \in container, u \in user)\ [(jobcont(j) = c)\ \&\ auth(u)\ \&\ ((owner(c) = u) \vee shared(owner(c), u, c))]$.

4. Выполнение расчетов возможно только при наличии необходимых средств на счету проекта:

$(\forall j \in job)(\exists r \in resource, c \in container, p \in project, a \in app)\ [((jobres(j) = r)\ \&\ (jobapp(j) = a)\ \&\ (jobcont(j) = c)\ \&\ (ownerproj(c) = p)\ \&\ free(r, jobtime(j))\ \&\ appsupport(r, a)\ \&\ (1 \le send(j))) \rightarrow (cost(a, jobtime(j))\ \$\le acc(p))]$.

5. Средства проекта резервируются со счета на время выполнения задания:

$(\forall j \in job)(\exists r \in resource, c \in container, p \in project, a \in app, t_1, t_2 \in time, ac_1, ac_2, acjr \in account)\ [((jobres(j) = r)\ \&\ (jobapp(j) = a)\ \&\ (jobcont(j) = c)\ \&$

(*ownerproj*(*c*) = *p*) & (*send*(*j*) = t_1) & ($t_2 \le$ *end*(*j*)) & (*acc*(*p*, t_1) = ac_1) & (*acc*(*p*, t_2) = ac_2) & (*cost*(*a*, *jobtime*(*j*)) = *acjr*)) & *appsupport*(*r*, *a*)) → (ac_2 $+ *acjr* = ac_1)].

6. Неизрасходованные средства возвращаются на счет проекта после выполнения задания:

(∀ *j* ∈ *job*)(∃ *r* ∈ *resource*, *c* ∈ *container*, *p* ∈ *project*, *a* ∈ *app*, t_1, t_2 ∈ *time*, ac_1, ac_2, *acjs* ∈ *account*) [((*jobres*(*j*) = *r*) & (*jobapp*(*j*) = *a*) & (*jobcont*(*j*) = *c*) & (*ownerproj*(*c*) = *p*) & (*start*(*j*) = t_1) & (*end*(*j*) = t_2) & (*acc*(*p*, t_1) = ac_1) & (*acc*(*p*, t_2) = ac_2) & (*cost*(*a*, *jobwork*(*j*, t_1, t_2)) = *acjs*)) & *appsupport*(*r*, *a*)) → (ac_2 $+ *acjs* = ac_1)].

Задание: на основе описания правил использования виртуальной лаборатории разработать проект собственного приложения

В разработанной концепции вводится новое понятие проекта [41].

Проект – это особый тип группы участников, отличающийся тем, что он дополнительно снабжен собственным счетом в системе аккаунтинга. Проект в рамках ПООС предназначен для координации его участников, предоставления им средств для эффективной работы над исследованием (форум, wiki-раздел, блог проекта, To-do раздел и др.) и контроля за программными и аппаратными ресурсами, израсходованными за время выполнения проекта.

Проекты разделены на два типа по их целям:

1. Исследовательский проект, целью которого является проведение научного исследования.
2. Проект по разработке собственного приложения.

При заведении нового проекта в ПООС создавший его пользователь становится ***Руководителем*** проекта и ему доступны все средства управления, в том числе управление участниками и приложениями.

При создании проекта руководителю необходимо указать:

1. Название и его краткое описание, где приводятся основные цели и задачи проекта.

2. Уровень доступа к проекту:

– публичный: проект будет виден всем пользователям ПООС, в том числе и незарегистрированным;

– ограниченный: проект будет виден только зарегистрированным пользователям ПООС;

– закрытый: проект доступен только для ее участников.

3. Принцип добавления новых участников:

– открытый: все запросы на вступление в проект автоматически удовлетворяются;

– подтверждаемый: заявка на добавление отправляется на рассмотрение руководителю проекта;

– по приглашению: пользователи не могут самостоятельно оставлять заявку на вступление, вместо этого они получают приглашение на вступление от руководителя;

– закрытый: никто не имеет права вступать в проект. Список участников определяется руководителем при создании.

4. Уровень видимости проекта:

– видимый: проект отображается в списке проектов и его материалы доступны при поисковых запросах;

– невидимый: проект не виден в списке, он не отображается в поиске, вступление в него возможно только по приглашению.

После заведения проекта в ПООС руководителю предоставляется доступ к панели администрирования, где определяются дальнейшие необходимые для работы системы аккаунтинга характеристики. Изначально руководителю необходимо определить состав участников проекта и их статус в зависимости от выбранных настроек при создании. Участниками обязаны быть только зарегистрированные пользователи ПООС. На основе введенных руководителем данных система аккаунтинга ресурсов сохраняет профиль проекта в собственной базе данных.

В зависимости от типа проекта выделяются следующие роли:

– руководитель (для всех типов). В его задачи входит обработка заявок от пользователей на вступление в проект, назначение ролей и удаление участников из проекта, а также назначение доступных ресурсов в проекте;

– модератор (для всех типов). Отвечает за соблюдение участниками правил проекта. В его задачи входят удаление неправомерных сообщений и ограничение прав участников, написавших их, на редактирование и просмотр тем форума;

– пользователь (для всех типов). Может общаться на форуме и иметь доступ к интеллектуальным ресурсам проекта и к коммуникационным разделам;

– исследователь (для всех типов). Может запускать приложения и проводить расчеты;

– разработчик (для проектов по разработке). Имеет доступ к репозиторию приложения и может вносить изменения в исходный код;

– тестировщик (для проектов по разработке). Может запускать разрабатываемое приложение и просматривать его исходный код, но не может вносить изменения.

Исследовательский проект **–** совокупность работ, проводимых пользователями ПООС в рамках одного исследования с помощью конкретного набора приложений.

Проект по разработке приложений **–** совокупность работ пользователей, проводимых ими в рамках разработки и тестирования высокопроизводительного программного обеспечения средствами ПООС. Такому проекту дополнительно предоставляется ряд инструментов:

1. Все разработчики приложения получают доступ к рабочей среде (IDE) через веб-браузер, в которой они могут редактировать файлы, компилировать исходный код и тестировать свое приложение.

2. В качестве инструмента управления проектами и отслеживания ошибок используется система Trac. Она позволяет эффективно организовывать работу, помогает избежать многих трудностей, связанных с управлением и сопровождением проекта. Trac использует веб-интерфейс, основанный на технологии wiki, что облегчает ведение документации по проекту, его версиям и этапам разработки.

3. Для хранения исходных кодов разрабатываемого приложения используются системы управления версиями GIT и Subversion. Системы позволяет хранить несколько версий одного и того же файла, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, что упрощает коллективную разработку.

4. Для упрощения использования не имеющих графического пользовательского интерфейса приложений, ПООС предоставляет разработчикам набор инструментов Rappture, который автоматически генерирует интерфейс после считывания и обработки XML-описания входных и выходных данных приложения.

После финального тестирования приложение может быть опубликовано в ПООС, что означает, что доступ к нему могут получить не только разработчики, но также и остальные пользователи. В момент публикации приложения разработчики должны указать, под какой лицензией оно распространяется, и в зависимости от типа лицензии код приложения будет доступен, либо скрыт от остальных пользователей ПООС.

Лабораторная работа № 8

Концептуальные и логические модели представления знаний о структуре и функционировании распределенной вычислительной системы

Цель работы: овладение основными навыками анализа предметной области при построения концептуальных и логических моделей искусственного интеллекта для заданной предметной области. В процессе изучения теоретического материала и практической работы необходимо освоить основные аспекты построения концептуальных и логических моделей для распределенных вычислительных систем облачно-сетевого типа [36 – 40].

Описание данной лабораторной работы составлено при участии аспирантов М.С. Джафара и А.С. Амирханян и является внедрением их выпускных работ.

Описание предметной области

Современный подход к организации распределенных вычислений предполагает выбор сетевой архитектуры ***Grid Computing and Networking*** – что дословно переводится как сетевой компьютерный грид (grid – решетка). Для использования практических преимуществ Grid Computing and Networking организациям необходимо обосновать переход на эту относительно новую вычислительную парадигму, современная реализация которой предполагает также взаимодействие грид-системы с клиентом при посредстве облачной среды, которой соответствует термин ***Cloud Computing*** – облачные вычисления. Несмотря на наличие в названии термина “вычисления”, собственно вычисления, как правило, не являются глобальными. При наличии спроса на вычисления, в том числе высокопроизводительные вычисления, через “облако” чаще всего организуется связь с отдельной грид-системой, кластером компьютеров или с суперкомпьютером.

Концепции, так и приложения грид-вычислений не всегда являются ясными и понятными для нетехнических специалистов. При выборе распределенных грид-вычислений необходимо объяснить заказчику

лежащий в основе грид сетевой механизм и ответить на важные для бизнеса вопросы.

В настоящей работе распределенная обработка структурированных в виде таблиц данных связана с выполнением заданий многочисленных пользователей, организацией запросов к распределенным по вычислительной сети базам данных, вычислениями значений атрибутов. На рисунке 8.1

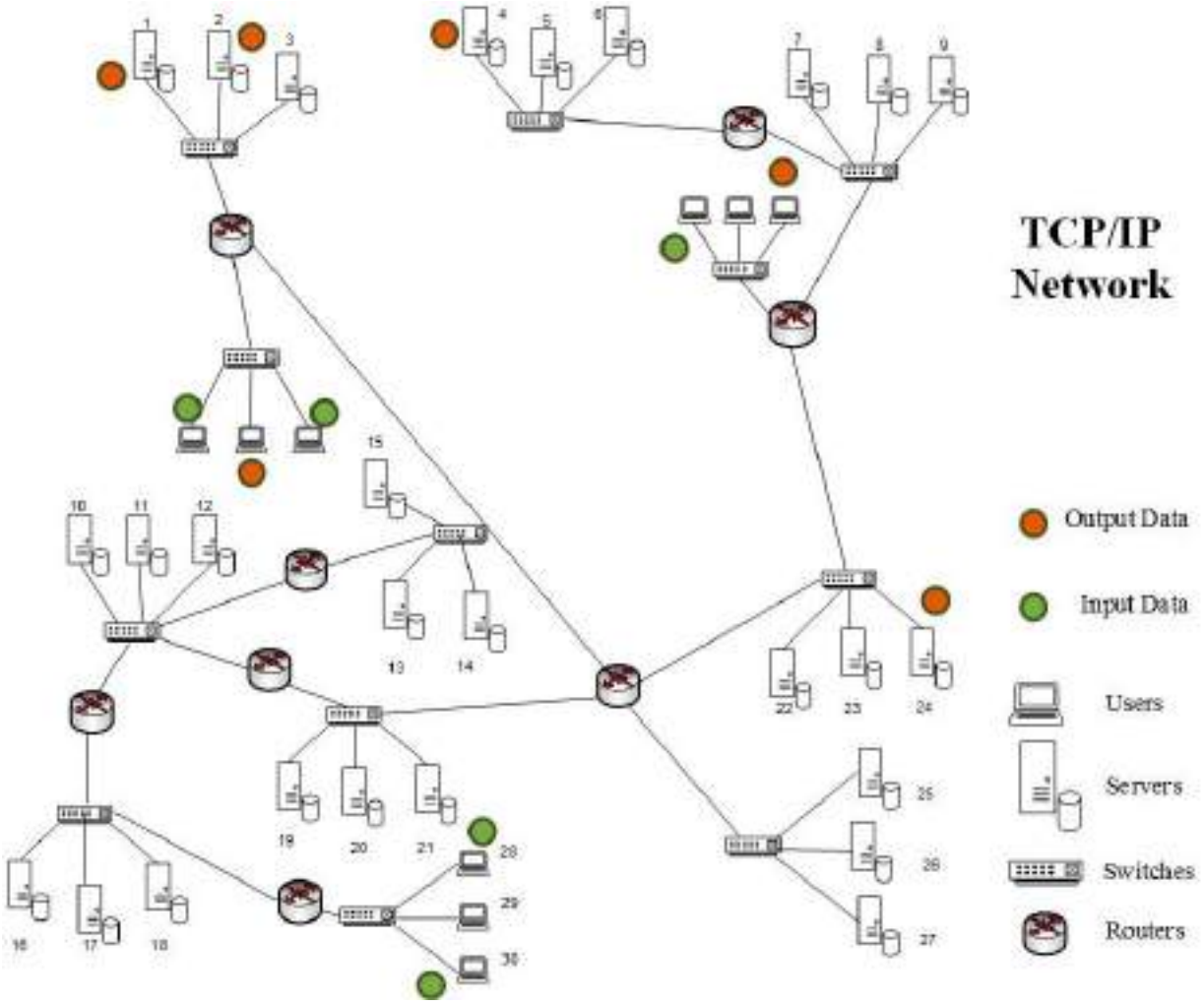


Рисунок 8.1 – Распределенная обработка данных в компьютерной сети

представлена структура вычислительной сети, в которой могут быть реализованы распределенные вычисления. Например, это может быть

компьютерная TCP/IP сеть. Эта сеть содержит клиентские компьютеры, серверы, коммутаторы, маршрутизаторы и каналы связи.

Зелеными кружками на рисунке 8.1 обозначены данные или запросы к распределенной базе данных, вводимые с пользовательских компьютеров. Коричневыми кружками обозначены конечные результаты, сохраняемые в локальных базах данных или принимаемые клиентами.

Грид-вычисления все еще являются развивающейся областью и связаны с несколькими другими инновационными вычислительными системами, некоторые из которых являются частными случаями грид-вычислений.

Совместно используемые вычисления обычно относятся к совокупности компьютеров, которые совместно используют вычислительную мощность для выполнения определенной задачи. В этой связи в облачных средах реализуется сервис “программное обеспечение как услуга” (SaaS), известная как служебные вычисления, в которой компания предлагает определенные услуги (например, хранение данных или увеличение мощности процессора) за определенную плату. Облачные вычисления – это система, в которой приложения и хранилище являются ресурсами сети, а не компьютера пользователя.

На рисунках 8.2 и 8.3 представлены примеры возможных сценариев распределенной обработки данных в среде вычислительной сети. Эти сценарии соответствуют обработке данных в распределенной среде типа «грид» с возможной организацией в виде частного облачного сервиса. Сценарии имеют абстрактный характер и не привязаны к какой-либо конкретной обработке данных.

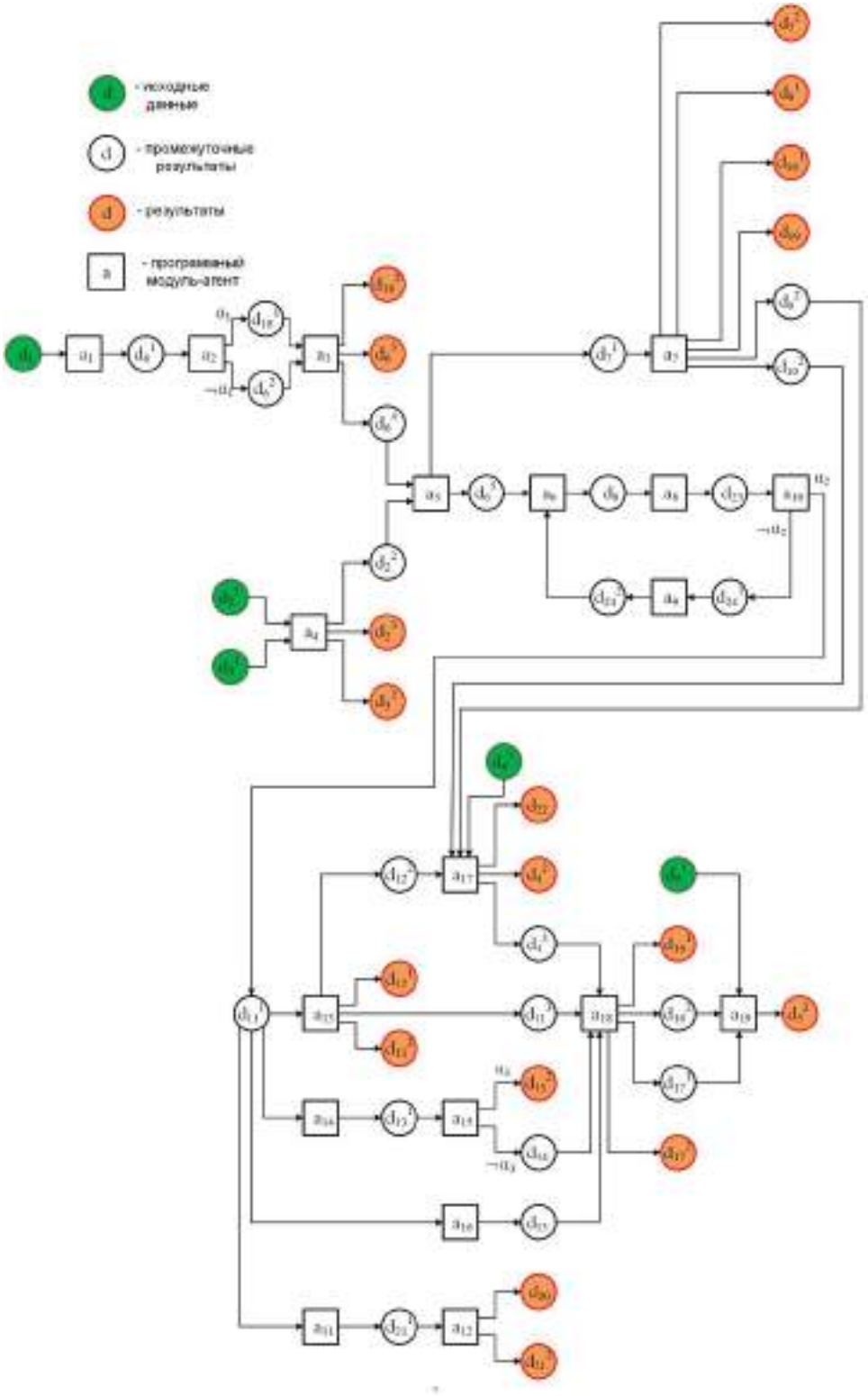


Рисунок 8.2 – Сценарий (пример 1) распределенной обработки данных в среде вычислительной сети

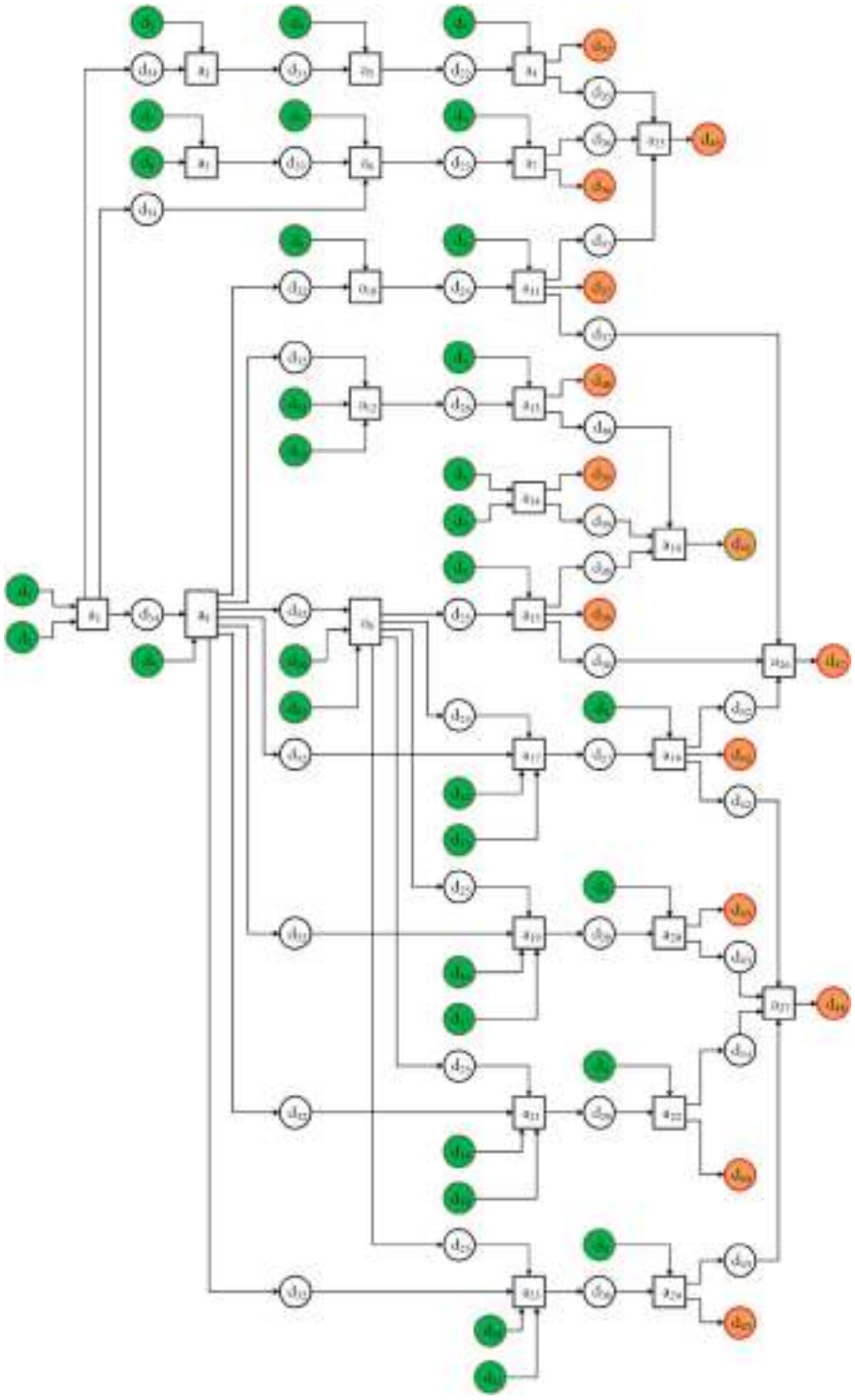


Рисунок 8.3 – Сценарий (пример 2) распределенной обработки данных в среде вычислительной сети

Похожие сценарии выбирались в качестве примеров в работах [30, 31], но, в отличие от этих работ, здесь предлагается в технологии создания распределенных приложений использовать некоторые методы искусственного интеллекта на основе представления фреймов и семантических сетей в виде концептуальных графов. Обработка графических представлений автоматизирована путем приложения ***main4*** (см. описание лабораторной работы № 1), позволяющего по концептуальному графу получить его продукционное представление и в дальнейшем при проверке правильности структурного проектирования приложения использовать язык Пролог.

На рисунках 8.2 и 8.3 кружками обозначены входные данные, промежуточные и окончательные результаты. Квадратами обозначены модули-агенты – программы, которые взаимодействуют при обработке распределенной информации. На входе каждого модуля-агента реализуется ввод всех данных, обозначенных входными стрелками. После обработки этих данных результаты передаются на дальнейшую или окончательную обработку.

Необходимо разработать концептуальный и логический подход к структурному синтезу распределенных приложений. Дополнительно к предыдущим понятиям возможно использовать понятия "***фрейм***" и "***функциональный фрейм***", а также логические формализации концептуальных графов. Введем понятие ***модулей-агентов*** – программ, которые взаимодействуют при обработке распределенной информации. На входе каждого модуля-агента реализуется ввод всех данных, обозначенных входными стрелками. После обработки этих данных результаты передаются на дальнейшую или окончательную обработку.

На рисунке 8.4 представлен концептуальный граф типа "звезда" ***функционального фрейма-экземпляра*** для одного из типовых модулей-агентов обработки данных.

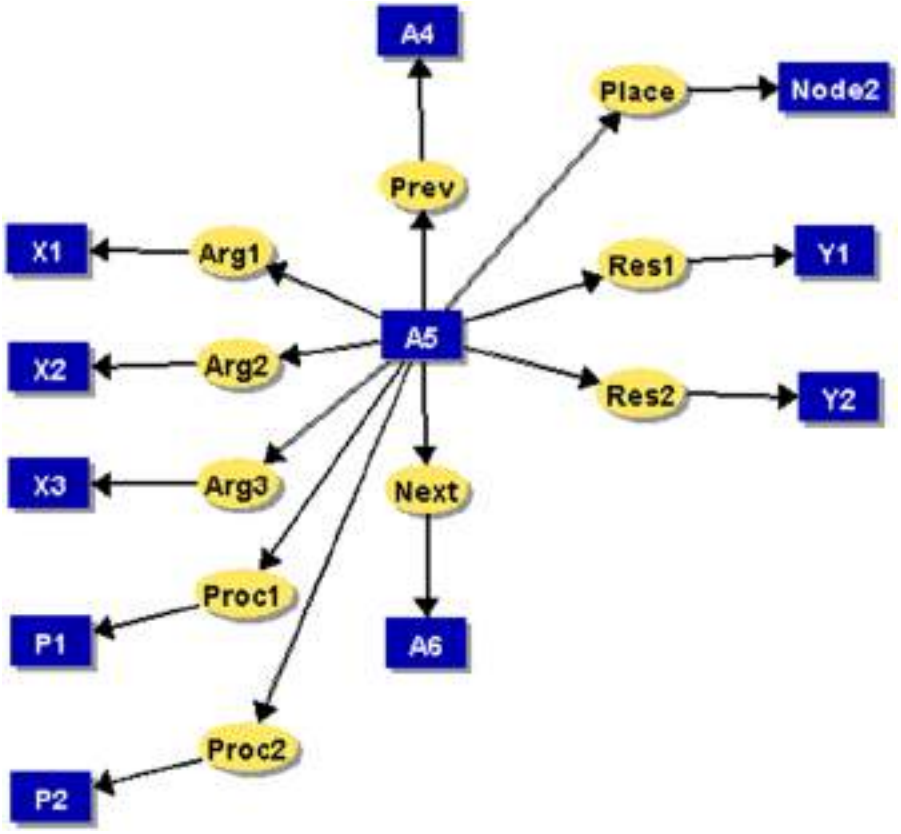


Рисунок 8.4 – Концептуальный граф функционального фрейма-экземпляра для примера модуля-агента *A5* обработки данных.

Логическое представление функционального фрейма-экземпляра (факта) «Агент *A5*»:

Basic_properties(A5)=
Prev(A5, A4)&
Next(A5, A6)&
Place(A5, Node2)&
Proc1(A5, P1)&
Proc2(A5, P2)&
Arg1(A5, X1)&
Arg2(A5, X2)&
Arg3(A5, X3)&
Res1(A5, Y1)&
Res2(A5, Y2).

Здесь *Prev* (предыдущий агент), *Next* (следующий агент), *Place* (местонахождение агента), *Proc1* (первая процедура обработки), *Proc2* (вторая процедура обработки), *Arg1* (первый входной аргумент), *Arg2* (второй входной аргумент), *Arg3* (третий входной аргумент), *Res1* (первый результат), *Res2* (второй результат) – имена отношений, а *A4* (агент 4), *A6* (агент 6), *Node2* (узел сети 2), *P1* (процедура 1), *P2* (процедура 2), *X1* (аргумент 1), *X2* (аргумент 2), *X3* (аргумент 3), *Y1* (результат 1), *Y2* (результат 2) – имена объектов-концептов предметной области. Формально модуль-агент готов к исполнению в распределенной среде, если составное высказывание *Basic_properties(A5)* истинно, то есть для выполнения модуль-агент должен обладать всеми описанными высказываниями свойствами.

Табличное представление функционального фрейма-экземпляра (факта) «Агент *A5* и его свойства» имеет следующий вид:

A5		
	Prev:	*A4*
	Next:	*A6*
	Place:	*Node2*
	Proc1:	*P1*
	Proc2:	*P2*
	Arg1:	*X1*
	Arg2:	*X2*
	Arg3:	*X3*
	Res1:	*Y1*
	Res2:	*Y2*

Правило вывода свойств модуля-агента *A5* имеет следующий вид:

$$
\begin{aligned}
Is\_a(A5, A) \supset (&Prev(A5, A4)\&\\
&Next(A5, A6)\&\\
&Place(A5, Node2)\&\\
&Proc1(A5, P1)\&\\
&Proc2(A5, P2)\&\\
&Arg1(A5, X1)\&\\
&Arg2(A5, X2)\&\\
&Arg3(A5, X3)\&\\
&Res1(A5, Y1)\&\\
&Res2(A5, Y2)).
\end{aligned}
$$

Представление данного правила в виде концептуального графа приведено на рисунке 8.5, из которого видно, что фрейм-экземпляр *A5* наследует все свойства фрейма-прототипа *A*. Перефразируя сказанное в терминах объектно-ориентированного программирования, истинность высказывания *Is_a(A5, A)* означает, что объект модуль-агент *A5* принадлежит к классу *A*.

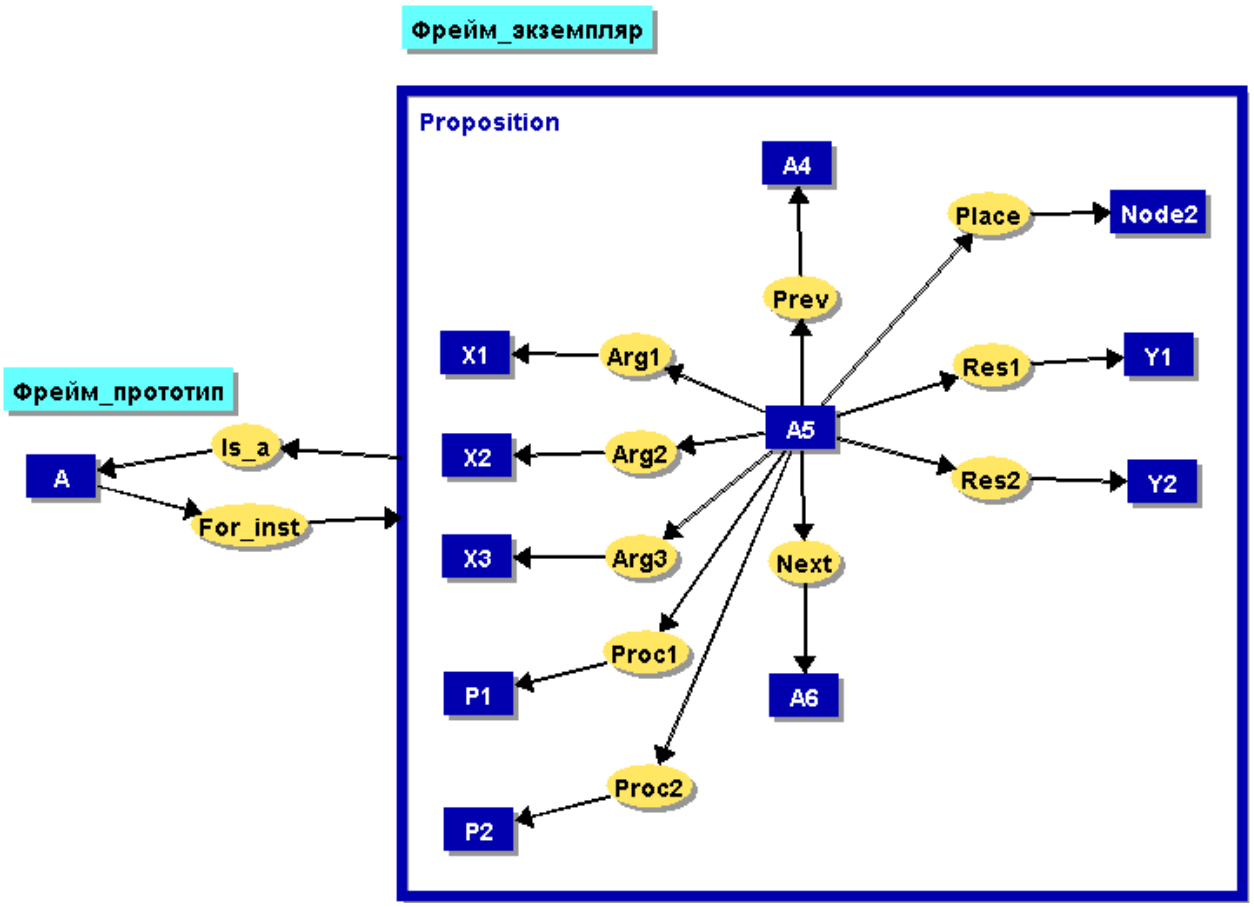


Рисунок 8.5 – Отношения между фреймом-экземпляром *A5* и фреймом-прототипом *A*

Расширяя состав свойств модуля-агента *A5*, построим концептуальный граф для семантической сети, в основе которой лежит графическое представление фрейма (рисунок 8.6). Дополнительные отношения: *Send* (отправление сообщения с данными о передаче управления), *Link* (связность узлов компьютерной сети). Добавлены концепты *Node1* и *Node3* – обозначающие узлы компьютерной сети для смежных модулей-агентов.

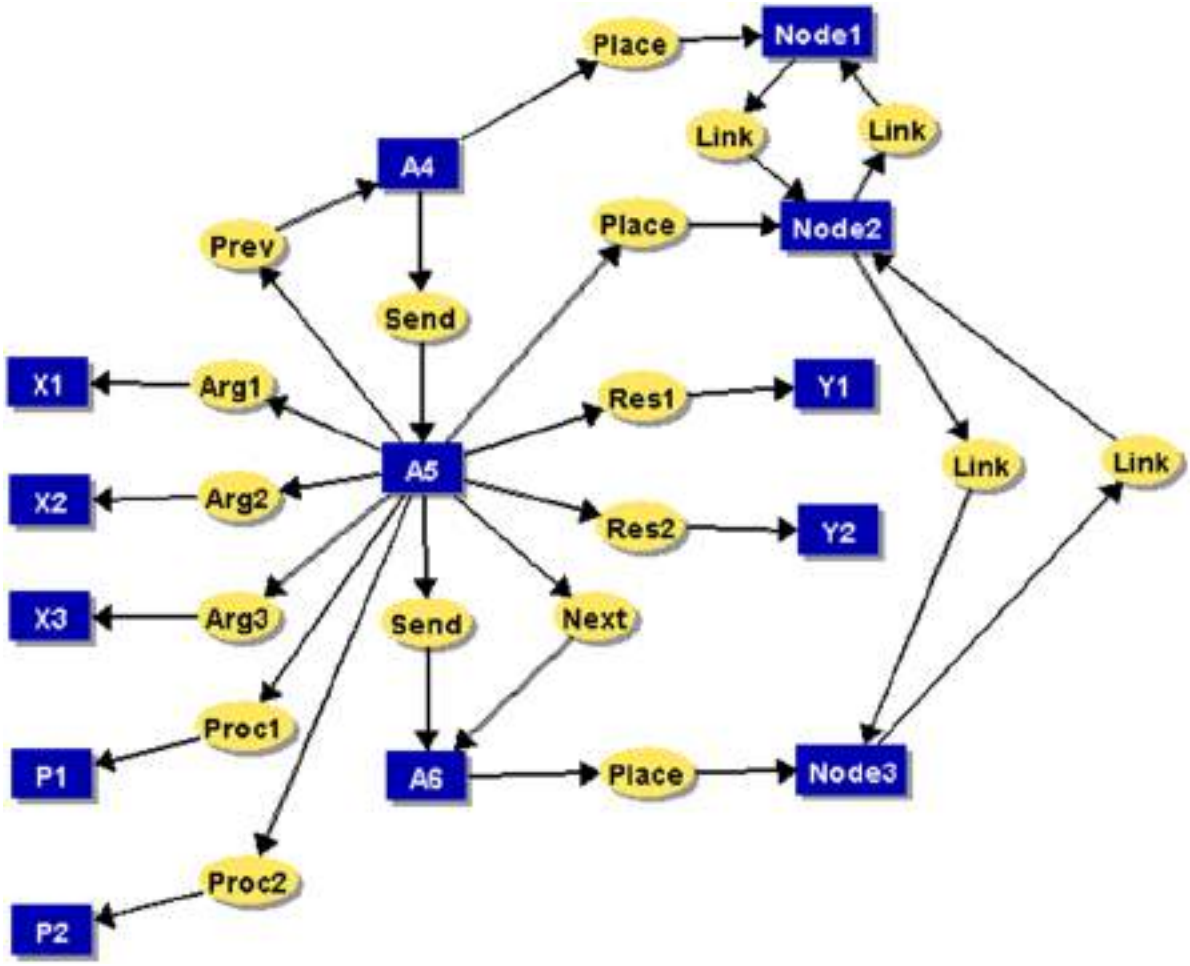


Рисунок 8.6 – Концептуальный граф для семантической сети с дополнительными отношениями для окружения модуля-агента *A5*

Логическое представление семантической сети, построенной на основе функционального фрейма *A5*, изображенной на рисунке 8.6 и описывающей расширенный состав свойств агента *A5*, имеет следующий вид:

Extendent_properties(A5)=
Prev(A5, A4)&
Next(A5, A6)&
Place(A5, Node2)&
Proc1(A5, P1)&
Proc2(A5, P2)&

Arg1(A5, X1)&
Arg2(A5, X2)&
Arg3(A5, X3)&
Res1(A5, Y1)&
Res2(A5, Y2)&
Place(A4, Node1)&
Place(A6, Node3)&
Link(Node1, Node2)&
Link(Node2, Node1)&
Link(Node2, Node3)&
Link(Node3, Node1)&
Send(A4, A5)&
Send(A5, A6).

Применяя приложения ***CharGer3*** и ***main4***, разработанные на каф. ВТ ПГУ, получим набор фактов для программы на Прологе, соответствующих данной семантической сети:

arg1(a5,x1).
arg2(a5,x2).
arg3(a5,x3).
link(node1,node2).
link(node2,node1).
link(node2,node3).
link(node3,node2).
next(a5,a6).
place(a4,node1).
place(a5,node2).
place(a6,node3).
prev(a5,a4).

proc1(a5,p1).

proc2(a5,p2).

res1(a5,y1).

res2(a5,y2).

send(a4,a5).

send(a5,a6).

Например, на запрос

? - *next(X, Y)*

будет получен ответ *X = a5, Y = a6.*

Все отношения в перечисленном выше наборе фактов – бинарные, что не дает возможность нахождения отношений, связывающих концепты (у стандартных версий языка Пролог такая возможность отсутствует). Поэтому данный набор дополним фактами, у которых использованные выше имена отношений станут концептами. Для формирования подобных фактов дополнительно используется приложение ***RelationConverter***, разработанное на каф. ВТ ПГУ авторами:

relation(arg1,a5,x1).

relation(arg2,a5,x2).

relation(arg3,a5,x3).

relation(link,node1,node2).

relation(link,node2,node1).

relation(link,node2,node3).

relation(link,node3,node2).

relation(next,a5,a6).

relation(place,a4,node1).

relation(place,a5,node2).

relation(place,a6,node3).

relation(prev,a5,a4).

relation(proc1,a5,p1).

relation(proc2,a5,p2).

relation(res1,a5,y1).

relation(res2,a5,y2).

relation(send,a4,a5).

relation(send,a5,a6).

Здесь *relation* – имя тернарного (трехместного) отношения. Теперь можно расширить состав запросов, например, запрос

? – *relation(X, a5,a6)*

позволяет получить ответ на вопрос «какие отношения связывает агенты *A5* и *A6* ?». Ответ: это отношения *X = send* и *X = next*.

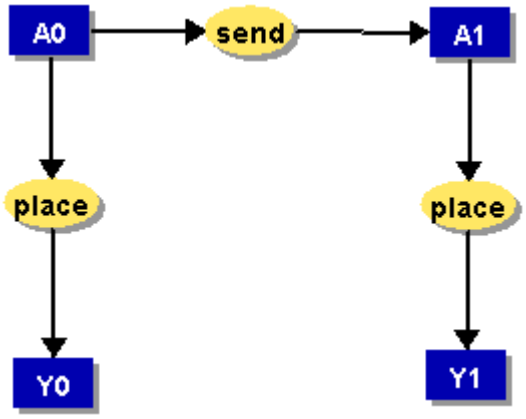
Таки образом, использован логический подход к построению распределенных приложений. Элементы технологии реализации распределенных приложений в компьютерных сетях на базе логического подхода к формализации декларативных и процедурных знаний о процессе обработки структурированной информации в распределенной среде вычислительной сети будут представлены далее.

В этом разделе представлены принципы реализации графического сервис-ориентированного метаязыка MPL – MetaProgramming Language, предназначенный для разработки грид-приложений. При его использовании считается, что, что все ресурсы распределенной вычислительной грид-системы составляют единое логическое пространство, где каждый хост содержит ресурс в форме службы, реализованной программными модулями-агентами, а приложение решает проблему, работая с этими службами. Метаязык MPL позволяет программистам сконцентрироваться на логике своих приложений, такой как логика реализации сервиса, логика вызова сервиса, логика связей и передач управления. Детали нижнего уровня, такие как распределение ресурсов, привязка ресурсов и развертывание служб, поддерживаются программным обеспечением промежуточного уровня (*middleware*), в качестве которого

выбрана платформа агентно-ориентированного программирования на основе Java. Используя такие методы виртуализации, метаязык MPL помогает повысить продуктивность программистов при разработке распределенных приложений.

Конструкции метаязыка MPL соответствуют дейкстровским принципам структурного программирования:

1. *Последовательность* – данная конструкция представлена в двух вариантах. На приведенном ниже рисунке дано графическое представление фразы “оператор A_0, размещенный на узле Y_0, передает сообщение оператору A_1, размещенному на узле Y_1”:



При помощи сообщений в распределенном алгоритме передается управление от одного оператора другому; сообщения могут содержать результаты вычислений, запросы к базам данных и результаты выполнения данных запросов. Напомним, что концептуальные графы являются графической формой представления формул в исчислении предикатов первого порядка. Графическому представлению на указанном рисунке соответствует следующая формула :

$$place(A_0, Y_0) \ \& \ place(A_1, Y_1) \ \& \ send(A_0, A_1).$$

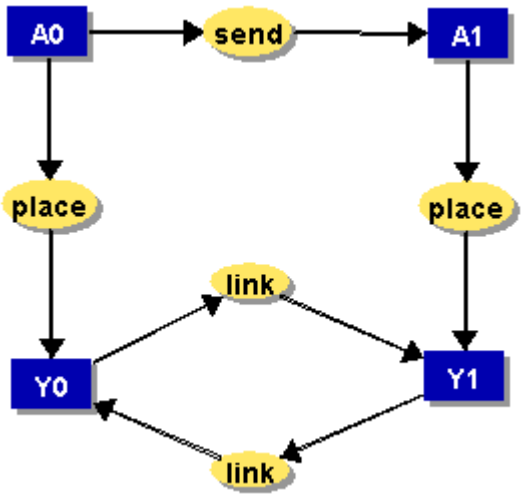
Графу соответствуют следующие факты, составляющие экстенсиональную базу данных в программе на языке Пролог:

place(a0, y0).

place(a1, y1).

send(a0, a1).

Второй вариант реализации может быть получен путем логического вывода из первого. То есть для того, чтобы передача управления от оператора A_0 оператору A_1 состоялась, необходимо, чтобы узлы Y_0 и Y_1 были связаны двунаправленными каналами:



Для данного концептуального графа новые связи *link* находятся при помощи следующих правил:

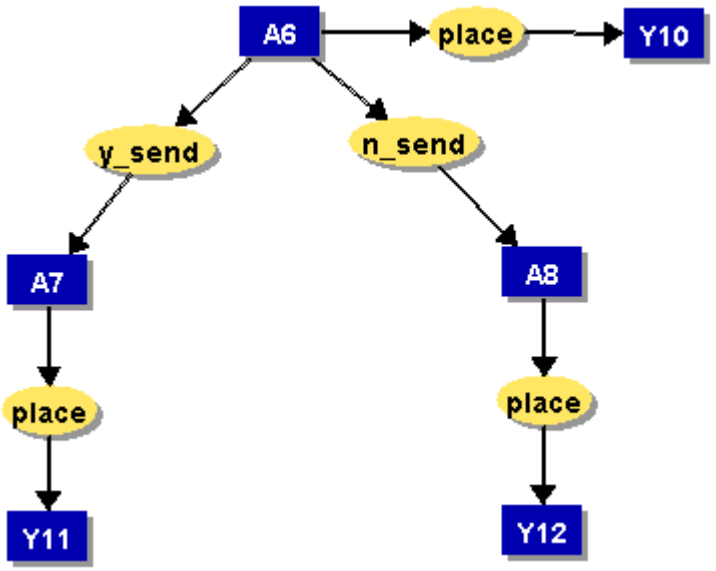
link(X, Y) :- send(A, B), place(A, X), place(B, Y).

link(Y, X) :- send(A, B), place(A, X), place(B, Y).

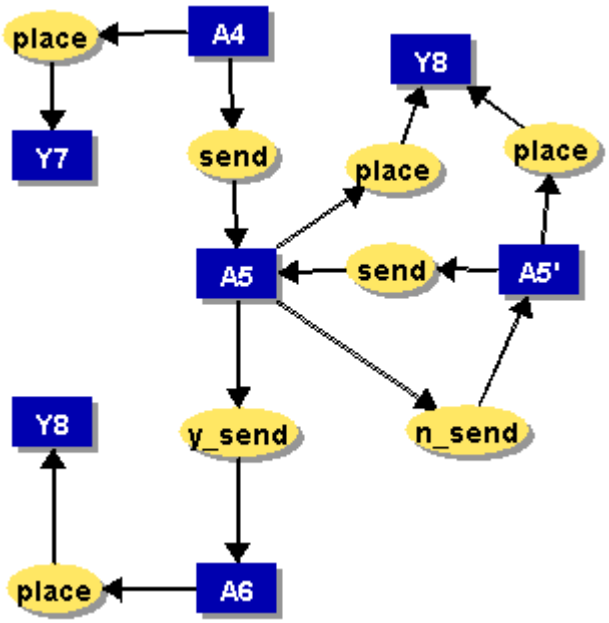
Здесь переменные *A* и *B* принимают значения в множестве имен операторов, а переменные *X* и *Y* – в множестве имен узлов распределенной вычислительной системы.

Рекурсивное правило *link(Y, X) :- link(X, Y)* не использовано по причине появления в ответах копий кортежей в связи с особенностью выполнения рекурсивных правил в различных версиях Пролога.

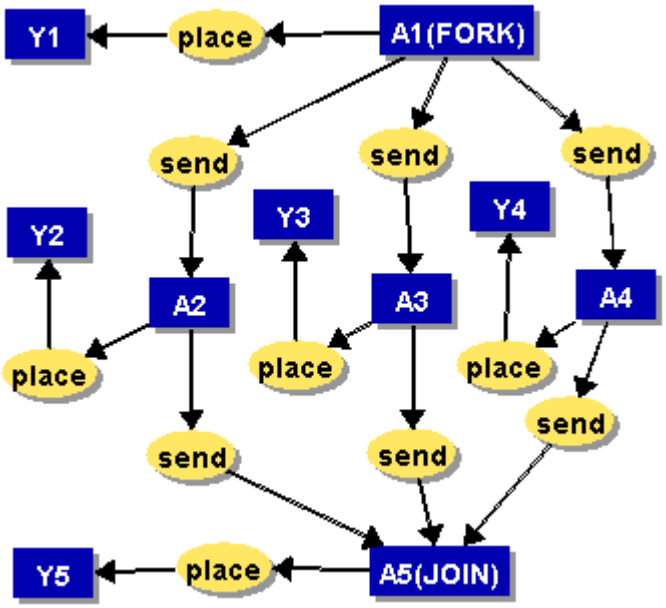
2. *Ветвление*, или ***альтернатива***. Особенность конструкции состоит в следующем. По умолчанию полагается, что в результате выполнения оператора A_6 определяется и проверяется значение логического условия Z_6. Если $Z_6 = true$, то сообщение передается оператору A_7, а при $Z_6 = false$ – оператору A_8:



3. *Цикл.* Во фрагменте **“*цикл*”** оператор A_5 выполняется циклически следующим образом: после каждого выполнения данного оператора A_5 по умолчанию вычисляется значение логической переменной Z_5. Если Z_5 = *true*, то осуществляется выход из цикла путем передачи сообщения оператору A_6. При Z_5 = *false* управление передается вспомогательному оператору A'_5, размещенному на том же узле Y_8, что и оператор A_5, и далее оператор A'_5 передает управление оператору A_5:



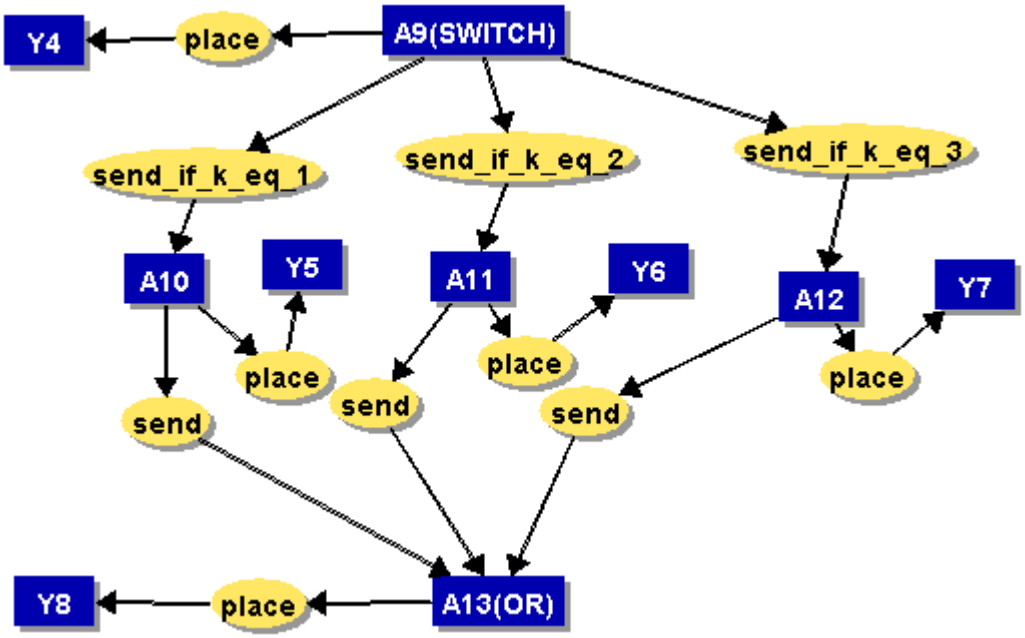
4. *Fork-Join*. Фрагмент **“*fork-join*”** описывает передачу сообщений по параллельным ветвям распределенного алгоритма:



Здесь по завершении выполнения всех параллельных ветвей сообщения объединяются в одно результирующее сообщение. В распределенном варианте выполнения ветви алгоритма не обязательно выполняются параллельно. Эти ветви выполняются “возможно одновременно”, независимо друг от друга: во-первых, они запускаются на выполнение поочередно выдаваемые агентом A_1 сообщениями, во-вторых, их выполнение происходит в различных фрагментах вычислительной сети. В зависимости от ситуации, сложившейся в вычислительной сети, операторы, размещенные на различных узлах, могут выполняться с пересечением временных интервалов или вовсе в непараллельном режиме.

Возможны варианты реализации. Например, агентом A_5 вместо оператора соединения ***join*** может использоваться оператор ***gather***, осуществляющий барьерную синхронизацию всех сообщений, выходящих их параллельных ветвей и их последующую передачу следующему оператору. При реализации другого варианта сообщения, выходящие их параллельных ветвей, могут передаваться по независимым путям различным агентам.

5. Switch-Or. Фрагмент **“switch-or”** реализует передачу сообщения по одной из ветвей распределенного алгоритма по значению переменной *k*. Возможен вариант реализации, при котором каждое результирующее сообщение передается различным операторам:



Пример концептуального графа распределенного алгоритма, описывающего декларативные и процедурные знания о распределенном приложении, выполняемом в сетевой грид-среде, представлен на рисунке 8.7.

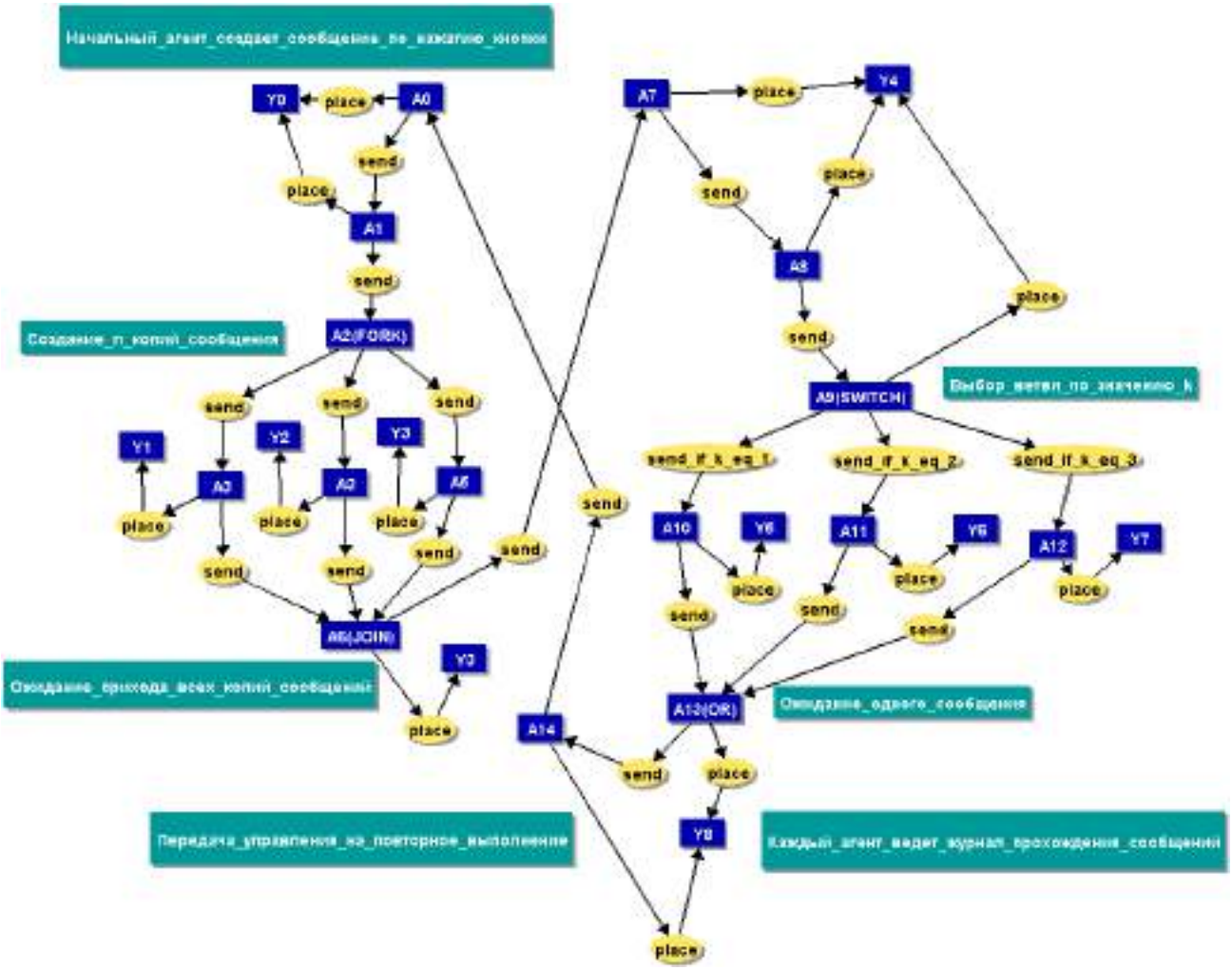


Рисунок 8.7 – Концептуальный граф распределенного алгоритма, описывающий декларативные и процедурные знания о распределенном приложении, выполняемом в сетевой грид-среде

Для приведенного на рисунке 8.7 концептуального графа, описывающего декларативные и процедурные знания о распределенном приложении, выполняемом в сетевой грид-среде, нетрудно разработать ее логическое представление и программу на языке Пролог для размещения распределенного алгоритма в физической сети, представленной на рисунке 8.1.

Основным отличием метаязыка программирования распределенных систем MPL является его платформенная независимость, ориентация на парадигмы передачи сообщений и, в случае мобильных агентов, перемещения

агентов. Кроме того, функционирование распределенного приложения может быть организовано по принципу перемещения транзактов в системе имитационного моделирования GPSS, то есть активности-сообщения или активности-агенты могут генерироваться агентами-источниками, не дожидаясь окончания обработки предыдущих сообщений или агентов.

Пример перехода от логической функциональной архитектуры облачно-сетевой РВС к ее логической системной архитектуре на основе логического вывода в логике предикатов

В качестве примера рассматривается распределенный алгоритм, формульная запись которого на метаязыке MetaProgramming Language (MPL) имеет следующий вид:

$$R_A = A_0, A_1, A_2, A_3, [A_3|C_3](\{A_4, A_5, AND\_SIM(A_6, A_7, A_8, A_9), A_{10}\} \vee \{A_{11}, A_{12}, [A_{12}|C_{12}](A_{13} \vee \{\downarrow^1 A_{14}, [A_{14}|C_{14}](E \vee \{A_{15}, Ret\uparrow^1\}))), A_k.$$

Данный алгоритм содержит разветвления (α-дизъюнкции), параллельные (*AND_SIM*) и циклический участки. Для организации цикла используется оператор передачи управления $Ret\uparrow^1$ оператору A_{14}.

Формульная запись данного распределенного алгоритма с указанием местонахождения операторов в глобальной сети представлена в следующем виде:

$$R_{AY} = A_0/Y_0, A_1/Y_1, A_2/Y_2, A_3/Y_3, [(A_3/Y_3)|C_3](\{A_4/Y_4, A_5/Y_5, AND\_SIM(A_6/Y_5, A_7/Y_6, A_8/Y_7, A_9/Y_8), A_{10}/Y_5\} \vee \{A_{11}/Y_4, A_{12}/Y_5, [(A_{12}/Y_5)|C_{12}](A_{13}/Y_6 \vee \{\downarrow^{Y7,1} A_{14}/Y_7, [(A_{14}/Y_7)|C_{14}](E \vee \{A_{15}/Y_7, Ret\uparrow^{Y7,1}\}))), A_k/Y_0.$$

Здесь оператор передачи управления $Ret\uparrow^{Y7,1}$ содержит наименование логического узла вычислительной сети, где находится оператор A_{14}, которому передается управление. На рисунке 8.8 представлен построенный на основании формулы R_{AY} концептуальный граф распределенного алгоритма (КГ РА), определяющий «логическую» функциональную архитектуру РВС. Здесь

предикаты *Send*, *Y_Send*, *N_Send* и *Place* и концепты определяют функциональную архитектуру распределенного приложения РВС.

Концептами здесь являются операторы из множества $A = \{A_0, A_1, \ldots, A_{15}, A_k\}$ и логические узлы из множества $Y = \{Y_0, Y_1, \ldots, Y_8\}$. Как было определено в первой главе, передача управления в КГ РА задается путем передачи сообщений, возможно, содержащих данные. Эти передачи определяются предикатами *Send*, *Y_Send* и *N_Send*. Размещение операторов по логическим узлам РВС задает предикат *Place*. Тогда при заданных предикатами *Send*, *Y_Send*, *N_Send* и *Place* фактах предикат *Link*, необходимый для задания логических связей между виртуальными узлами облачно-сетевой РВС, определяется как заключение в следующих правилах вывода:

$(\forall a \in A, b \in A, y \in Y, z \in Y)\ [Send(a, b) \& Place(a, y) \& Place(b, z) \rightarrow Link(y, z)];$

$(\forall a \in A, b \in A, y \in Y, z \in Y)\ [Y\_Send(a, b) \& Place(a, y) \& Place(b, z) \rightarrow Link(y, z)];$

$(\forall a \in A, b \in A, y \in Y, z \in Y)\ [N\_Send(a, b) \& Place(a, y) \& Place(b, z) \rightarrow Link(y, z)];$

$(\forall a \in A, b \in A, y \in Y, z \in Y)\ [Send(a, b) \& Place(a, y) \& Place(b, z) \rightarrow Link(z, y)];$

$(\forall a \in A, b \in A, y \in Y, z \in Y)\ [Y\_Send(a, b) \& Place(a, y) \& Place(b, z) \rightarrow Link(z, y)];$

$(\forall a \in A, b \in A, y \in Y, z \in Y)\ [N\_Send(a, b) \& Place(a, y) \& Place(b, z) \rightarrow Link(z, y)],$

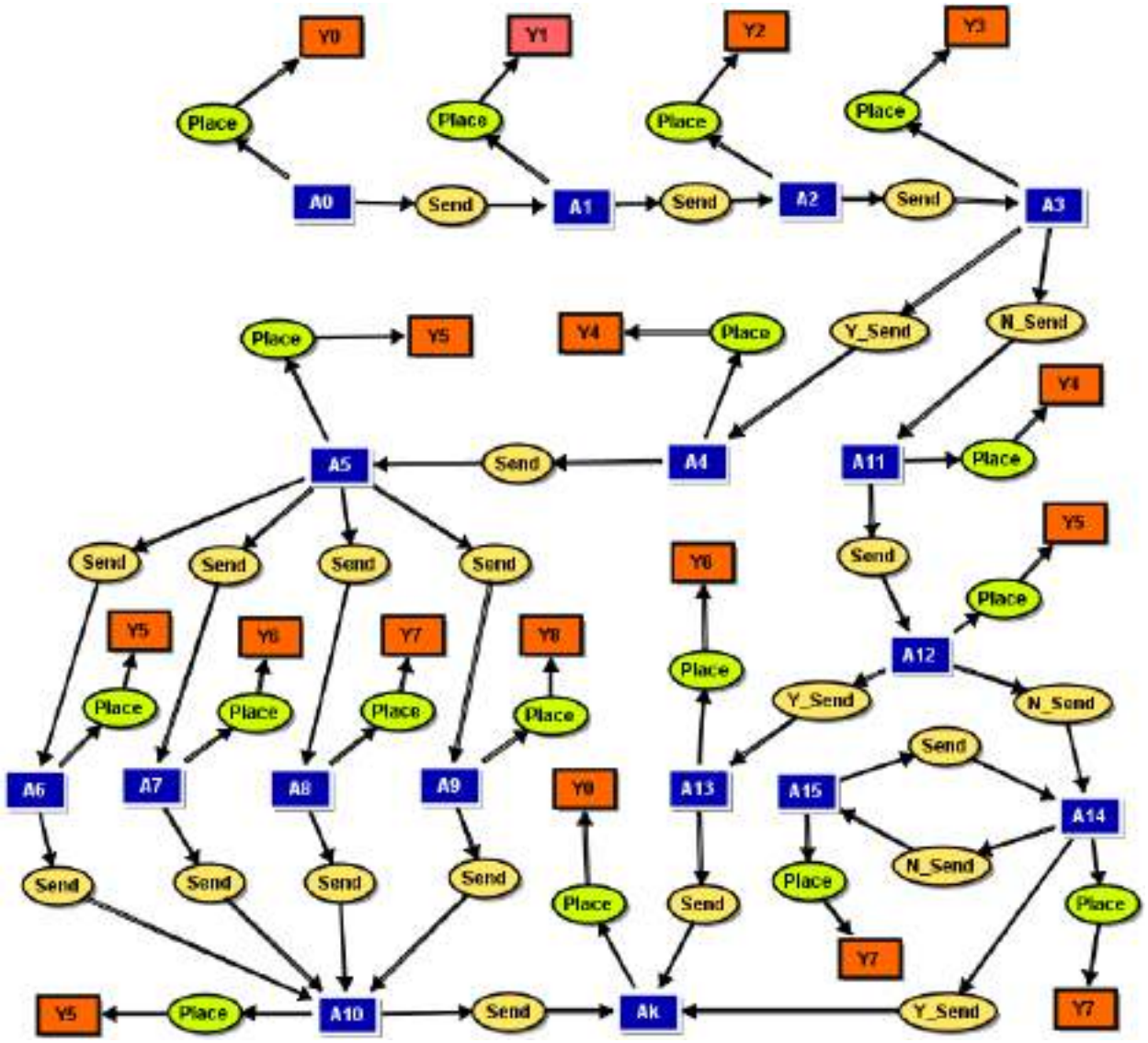


Рисунок 8.8 – Концептуальный граф распределенного алгоритма (КГ РА), определяющий «логическую» функциональную архитектуру РВС

где первые три правила определяют связи типа *Link*(*y*, *z*) для реализации управляющих связей между операторами, а при помощи остальных правил формируются обратные связи типа *Link*(*z*, *y*), необходимые для отправки подтверждений (квитанций) от операторов-приемников сообщений к операторам-отправителям, то есть все *Link*-связи – двунаправленные, то есть истинно выражение

$$(\forall y \in Y, z \in Y)\ [Link(y, z) \rightarrow Link(z, y)]).$$

По умолчания полагается, что отношение *Link* рефлексивно, то есть выражение $(\forall\, y \in Y)\ Link(y, y)$ истинно – каждый узел связан сам с собой.

Программа на языке SWI-PROLOG для перехода от «логической» функциональной архитектуры облачно-сетевой РВС к «логической» системной архитектуре представлена на рисунке 8.9. Запрос на формирование виртуальной системной архитектуры облачно-сетевой РВС имеет следующий вид:

? – link(Y,Z).

Концептуальный граф «логической» системной архитектуры (КГ СА), полученный в результате логического вывода, представлен рисунком 8.10. Дополнительно на графе приведено исходное отношение *Place*, задающее размещение операторов на логических узлах. Вложение логической структуры в физическую осуществляется следующим образом. Провайдером облачного сервиса, или автоматически, формируется информационный объект (таблица) на основе унарной функции

$$Map: Y \rightarrow IPaddr, \qquad (8.1)$$

где *IPaddr* – множество IP-адресов физических узлов TCP/IP сети, выделенных для реализации РВС.

Для перехода от логической структуры к физической, “вложенной” в вычислительную сеть, введено следующее правило вывода:

$$(\forall\, x \in Y)(\forall\, y \in Y)[Link(x, y) \rightarrow PhLink\,(Map(x), Map(y))], \qquad (8.2)$$

где $PhLink: IPaddr \times IPaddr \rightarrow \{\mathbf{true}, \mathbf{false}\}$ – бинарный предикат, область истинности которого задает отношение, связывающее узлы в физической вычислительной сети.

```
send(a0,a1).
send(a1,a2).
send(a2,a3).
send(a4,a5).
send(a5,a6).
send(a5,a7).
send(a5,a8).
send(a5,a9).
send(a6,a10).
send(a7,a10).
send(a8,a10).
send(a9,a10).
send(a10,ak).
send(a11,a12).
send(a13,ak).
send(a15,a14).
ysend(a3,a4).
ysend(a12,a13).
ysend(a14,ak).
nsend(a12,a14).
nsend(a14,a15).
nsend(a3,a11).
place(a0,y0).
place(a1,y1).
place(a2,y2).
place(a3,y3).
place(a4,y4).
place(a5,y5).
place(a6,y5).
place(a7,y6).
place(a8,y7).
place(a9,y8).
place(a10,y5).
place(a11,y4).
place(a12,y5).
place(a13,y6).
place(a14,y7).
place(a15,y7).
place(ak,y0).
link(Y,Z):-send(A,B),place(A,Y),place(B,Z).
link(Y,Z):-ysend(A,B),place(A,Y),place(B,Z).
link(Y,Z):-nsend(A,B),place(A,Y),place(B,Z).
link(Z,Y):-send(A,B),place(A,Y),place(B,Z).
link(Z,Y):-ysend(A,B),place(A,Y),place(B,Z).
link(Z,Y):-nsend(A,B),place(A,Y),place(B,Z).
```

Рисунок 8.9 – Программа на языке SWI-PROLOG для перехода от «логической» функциональной архитектуры облачно-сетевой РВС к «логической» системной архитектуре

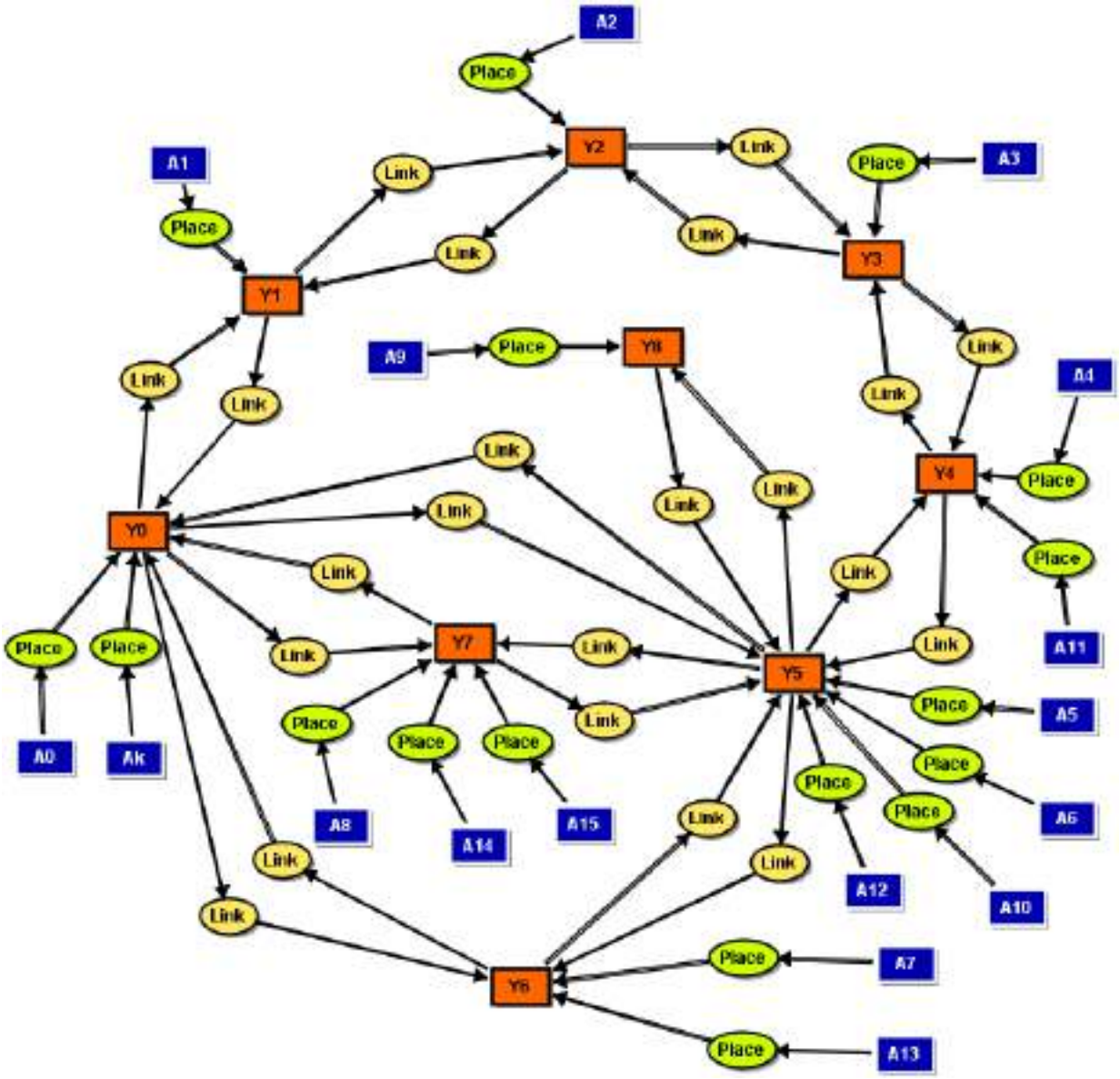


Рисунок 8.10 – Концептуальный граф «логической» системной архитектуры (КГ СА), полученный в результате логического вывода

Как обычно в TCP/IP сетях, конкретные физические пути следования сообщений определяются маршрутизаторами. Предполагается, что граф связности узлов фрагмента физической сети, на котором реализуется облачно-сетевая РВС, является сильносвязным.

Таким образом, проиллюстрирована примерами методика синтеза виртуальных и физических системных и функциональных архитектур РВС.

Лабораторное задание

Для заданной преподавателем граф-схемы распределенного алгоритма разработать ее логическое представление и программу на языке Пролог для размещения распределенного алгоритма в физической сети. Далее осуществить синтез виртуальной и физической системной и функциональной архитектуры РВС.

Отчет по лабораторной работе должен включать:

1. Цель работы.
2. Условие задачи.
3. Сформулированные требования к семантической сети пользователя интеллектуальной системы на основе функциональной модели предметной области.
4. Разработку семантической сети (в любом редакторе), концептуального графа на ее основе (в редакторе **CharGer**), экстенсиональной и интенсиональной баз фактов и знаний о выбранной предметной области.
5. Программу на языке Пролог для размещения распределенного алгоритма в физической сети.
6. Результаты синтеза виртуальной и физической системной и функциональной архитектуры РВС.
5. Выводы о проделанной работе.

Пример выполнения самостоятельной работы на тему: «Концептуальные, сценарные и имитационные модели роботизированного производства, управляемого при помощи беспроводной сети»

Введение.

Объектом исследования являются информационные процессы в асинхронных дискретно-событийных системах. Примерами таких систем могут быть организационно-экономические, производственные и вычислительные системы. Предметом исследования является методология формализации и анализа асинхронных дискретно-событийных систем на основе интеграции методов искусственного интеллекта и поведенческого дискретно-событийного моделирования. Методология исследования базируется на сценарных моделях искусственного интеллекта, логике предикатов первого и второго порядков, концептуальных моделях искусственного интеллекта и на впервые предлагаемых ***концептуальных сетях Петри***. Новизна исследования состоит в получении в конечном итоге на основе концептуальной и поведенческой модели асинхронной дискретно-событийной системы формализованных непосредственно исполнимых спецификаций, например, пригодных для последующей реализации сетевого программного обеспечения беспроводной вычислительной сети, управляющей гибким автоматизированным производством изделий.

Показано, что семантические сети с событиями, ролевыми и каузальными связями между объектами (сценарные сети) могут в достаточно полной степени представлять логическую, структурную и процедурную парадигмы интеллектуальных систем, в том числе систем ситуационного управления. Формализован процесс построения сценарных моделей на основе концептуальных графов, что позволяет использовать указанные модели в составе экспертно-имитационных моделей систем ситуационного управления. Показано, что концептуальное представление сценариев позволяет естественным образом автоматизировать переход к построению базы знаний на основе реляционного подхода, что обеспечивает повышение оперативности внесения изменений в концептуальную модель и базу знаний (например, при изменениях в предметной области). Запросы к базе знаний, построенной на основе концептуального графа, могут быть формализованы на языке сетей абстрактных машин, что упрощает дальнейшее проектирование человеко-машинного интерфейса интеллектуальной системы на базе известных языков типа C++, C#, SQL, Prolog, Python и др.

Описание данного примера составлено при участии аспиранта М. С. Джафара и магистранта У. Н. Пучковой по совместным публикациям в журнальных статьях [37, 38].

Примечание. В описании примера используется самостоятельная нумерация подразделов, рисунков и таблиц.

1. Концептуальные и сценарные поведенческие модели

Концептуальные и поведенческие модели нашли широкое применения в экспертно-имитационном моделировании дискретно-событийных систем различной природы. Согласно одному из распространенных определений, ***концептуальная модель*** – это «абстрактная модель предметной области, состоящая из множества понятий и связей между ними, классифицированных по типам и ситуациям свойств и характеристик, являющихся смысловой структурой (описанием) рассматриваемой предметной области, содержит необходимые для достижения цели моделирования причинно-следственные связи, присущие системе». Данное определение предполагает использование как неформализованного, так и формализованного описания предметной области. Формализация концептуальных знаний о проблемной области является сложной проблемой и может быть основана на представлении знаний в системах искусственного интеллекта.

Для представления основных парадигм искусственного интеллекта, таких как логической, структурной и процедурной используются ***логические, сетевые*** **и** ***продукционные модели***. К сетевым моделям относятся ***семантические сети*** и ***сети фреймов***.

Сценарием обычно называют формализованное представление совокупности взаимосвязанных фактов, определяющих смену ситуаций какой-либо предметной области. Знания о ситуациях реального мира, представленные в экспертных системах и системах ситуационного управления, помогают восстанавливать информацию, неточно представленную или утраченную при описании ситуаций, устанавливать закономерности в смысле ситуаций и зависимости одних ситуаций от других, предсказывать появление новых ситуаций. При помощи сценарных моделей возможно представлять проблемно-зависимые каузальные знания о событиях, действиях и процедурах, происходящих в предметной области.

Сценарный подход к решению сложных плохо формализованных задач приобретает все большую популярность. Он активно используется в системах динамического и интеллектуального моделирования, а также как средство представления и структурирования знаний.

Под сценарием будем понимать последовательность взаимосвязанных событий, которая может иметь место при определенных условиях. Между событиями существуют причинно-следственные связи, которые можно представить правилами, записанными на языке логики. Сценарии можно рассматривать как разновидности семантических сетей и сетей фреймов с ролевыми отношениями, дополненными концептами-событиями и причинно-следственными отношениями между ними. Концептуальные графы распределенных алгоритмов, рассмотренные в описании лабораторной работы № 8, можно также рассматривать как частные случаи сценариев.

Целью настоящей работы является формализация процесса построения и анализа сценарных моделей при учете логической, сетевой и продукционной компонент модели представления знаний о предметной области. Для достижения этой цели необходимо осуществить переход от концептуальной модели сценария к его представлению в реляционной базе данных и возможному дальнейшему использованию в составе экспертно-имитационной модели системы ситуационного управления. При представлении сценариев будут использоваться традиционные модели представления знаний – семантические сети с событиями и причинно-следственными, или каузальными, связями между ними, концептуальные графы.

2. Описание предметной области и постановка задачи на содержательном уровне

В качестве примера подходящей предметной области рассматривается участок гибкого автоматизированного производства, сокращенно ГАП (FMS – Flexible Manufacturing Systems). Применяемые на различных участках ГАП промышленные роботы (роботы-манипуляторы, робокары, погрузчики-робокары, роботы-погрузчики, складские роботы-штабелеры и др.) различаются по специализации, грузоподъемности, возможностям передвижения, способам установки на рабочем месте, адаптивности к окружающей среде, видам управления, способам программирования. Управляющие программы для роботов с программным управлением реализуют различные алгоритмы для исполнительных устройств, связанные с выполнением функций робота и позволяющие ему преодолевать различные препятствия в работе.

Промышленная мобильная связь, сокращенно ПМС (IMC – Industrial Mobile Communication), обеспечивающая возможность обмена данными через беспроводные каналы связи сетей, позволяет использовать для управления промышленными роботами различные радиосети, например, WLAN, GSM, GPRS, UMTS; она охватывает программные и аппаратные компоненты.

Рассмотрим представление знаний на содержательном уровне в виде следующего предложения, несколько модифицируя и комбинируя соответствующие описания из работ [47, 48]: «Автомат подает партию заготовок на станок (точнее, в подающий накопитель – часть станка), затем партия деталей обрабатывается по одной детали и оказывается в выходном накопителе станка, откуда их забирает первый робот-манипулятор, который упаковывает партию деталей в кассету. Затем второй робот-погрузчик грузит кассеты с деталями на робокар (число кассет, размещаемых на робокаре, ограничено), который перевозит их на склад. На складе, в зависимости от некоторого условия α робот-штабелер помещает кассеты в ячейки (при α = **true**) или же, при наличии транспорта (при α = **false**), третий робот упаковывает имеющиеся кассеты в контейнер (объем контейнера ограничен),

который по заполнении грузится роботом-автопогрузчиком на грузовой автомобиль по его прибытии».

3. Концептуальный граф для модели роботизированного производства и его контекстное представление

Сценарий, или сценарную сеть, работы участка ГАП возможно представить в виде семантической сети с событиями, ролевыми и причинно-следственными (каузальными) связями между объектами. Для этой сети определены следующие множества концептов и отношений:

– множество концептов-событий (фактов):

$F = \{F_0, F_1, F_3, F_4, F_5, F_6, F_7, F_8, F_9\}$,

имеющих, соответственно, следующие названия (имена):

X_N = {*Подача*, *Обработка*, *Упаковка_1*, *Погрузка_1*, *Перевозка*, *Размещение*, *Упаковка_2*, *Погрузка_2*, *Местонахождение_1*, *Местонахождение_2*};

– множество R всех отношений:

R = {*Имя*, *Агент*, *Объект*, *Место*, *След*, *Возм_след*};

– множество $R_1 = R \setminus \{След, Возм\_след\}$ ролевых отношений, то есть отношений без отношений следования (*След*) и возможного следования (*Возм_след*):

R_1 = {*Имя*, *Агент*, *Объект*, *Место*};

– множество X всех концептов, включая все концепты-события из множества F и имена (названия) этих событий из множества X_N;

– множество $X_1 = X \setminus F$ концептов модели без концептов-событий;

– множество $X_2 = X_1 \setminus X_N$ концептов, обозначающих физические объекты.

Сценарий роботизированного производства, или семантическая сеть с событиями, представлена на рисунке 1.

Концептуальное представление C сценария работы участка ГАП приведено на рисунке 2.

Здесь прямоугольниками представлены концепты – концепты-объекты и концепты-события, а овалами – отношения между концептами. Причинно-следственные (каузальные) связи между событиями представлены бинарным отношением *След*. Так, истинность высказывания *След*(F_i, F_j), где *След* – одноименный отношению бинарный предикатный символ, означает, что вслед за событием F_i выполняется событие F_j. В овале, связывающем в концептуальном графе концепты-события F_4 и F_5, записана формула

α&*След*(F_4, F_5), а в овале, связывающем концепты-события F_4 и F_6, записана формула ¬α&*След*(F_4, F_6), то есть первая связь реализуется при α = **true**, а вторая – при α = **false**. Однако в целях упрощения последующего анализа концептуального графа и представления его в реляционной базе данных в обоих случаях будем использовать отношение “возможного следования” *Возм_след*.

При построении концептуального графа *C* использовался распространенный в мировой практике редактор ***CharGer*** [12, 13, 14], позволяющий представить граф в виде XML-файла. Такое представление позволяет осуществить в дальнейшем автоматический переход от концептуального графа к отношениям реляционной базы данных. Полученные отношения, представленные таблицами 1 – 7, используются далее при анализе сценария, представленного концептуальным графом *C* на рис. 2.

5. Логика запросов и сети абстрактных машин

Построение сетей из абстрактных модулей, составляющих основу концептуальных моделей, строится на логике предикатов первого и второго порядков и элементов нотации алгебры алгоритмов В. М. Глушкова. Например, выражение [α]($m_1 \vee m_2$), где α – некоторое условие, а m_1 и m_2 – выражения для модулей абстрактных машин, соответствует тернарной операции α-дизъюнкции, когда при α = **true** выполняется модуль m_1, а при α = **false** выполняется модуль m_2. Одним из модулей в приведенном выражении может быть пустой (“тождественный”) оператор *E*, не выполняющий никаких действий. Выражения m_1 и m_2 могут, в свою очередь, содержать другие α-дизъюнкции. Запятыми в выражениях для абстрактных модулей разделяются последовательно выполняемые действия.

В логике предикатов первого порядка используется четыре типа выражений: константы, служащие именами индивидуумов (объектов, людей, событий); переменные, обозначающие имена совокупностей; предикатные имена, задающие правила соединения констант и переменных; функциональные имена, представляющие такие же правила, как и предикаты, для отношений, имеющих функциональный характер.

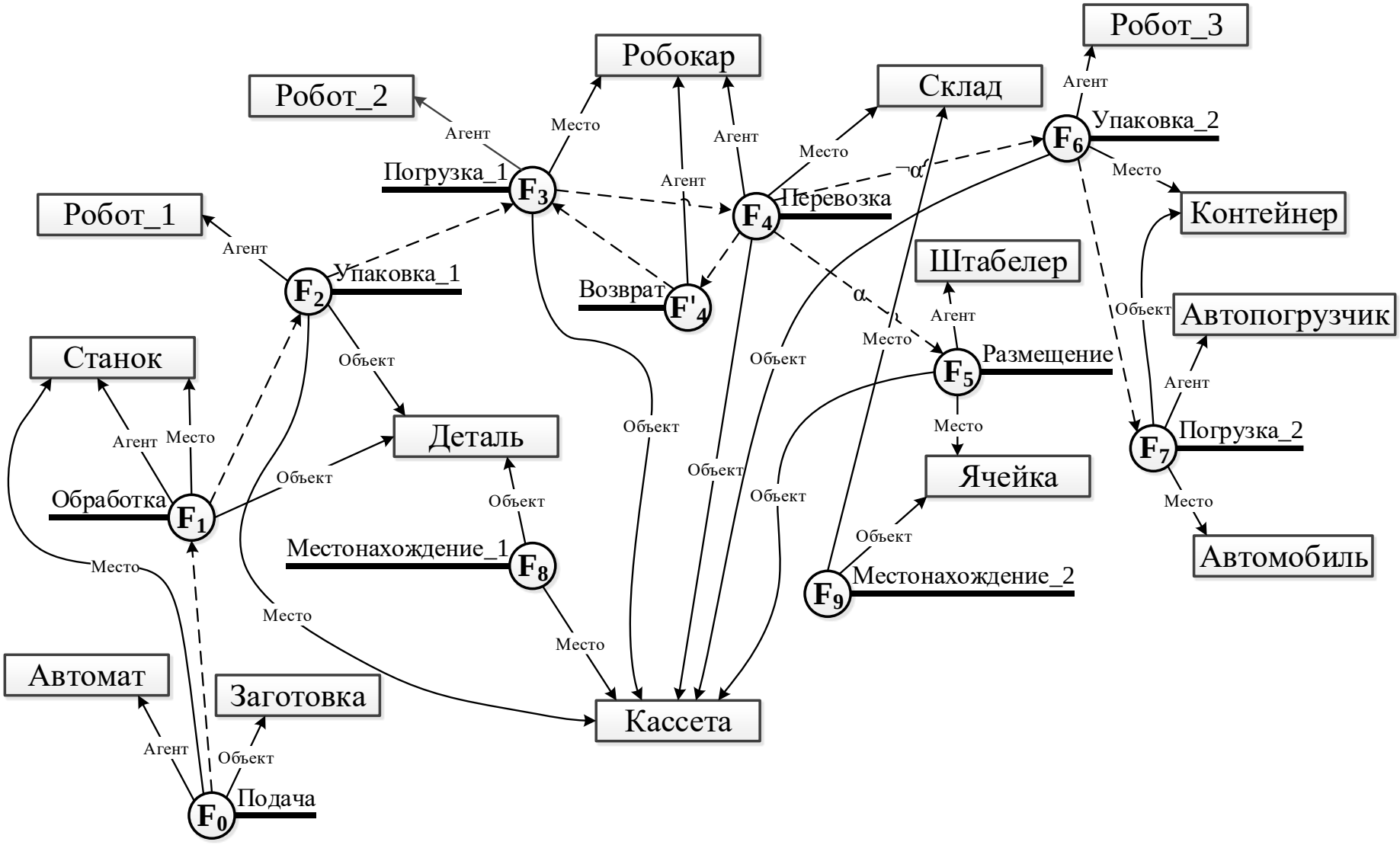


Рисунок 1 – Семантическая сеть с событиями (сценарий) для участка роботизированного производства

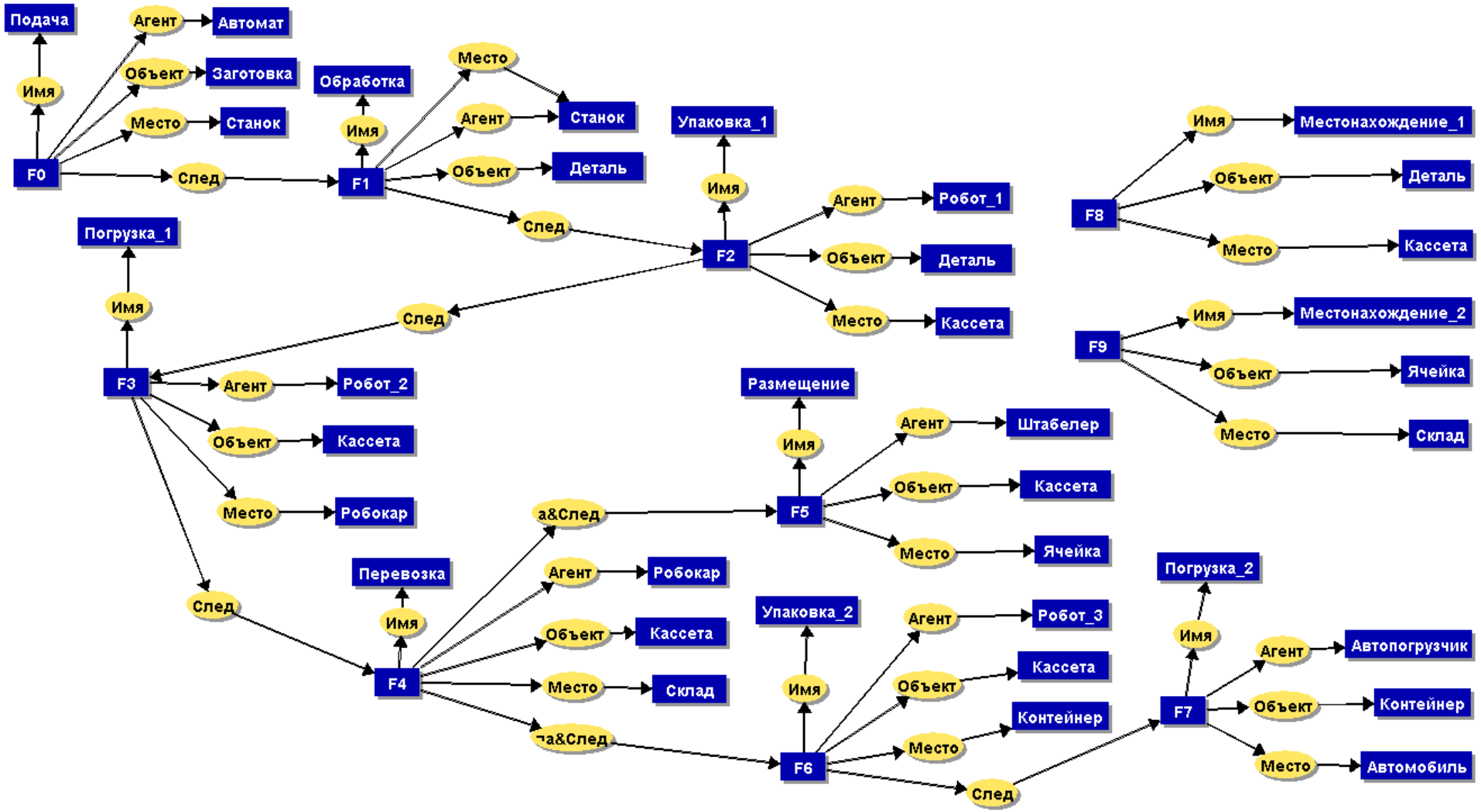


Рисунок 2 – Концептуальный граф *C* сценария роботизированного производства

Таблица 1 – Имя

Факт	Имя
F_0	Подача
F_1	Обработка
F_2	Упаковка_1
F_3	Погрузка_1
F_4	Перевозка
F_5	Размещение
F_6	Упаковка_2
F_7	Погрузка_2
F_8	Местонахождение_1
F_9	Местонахождение_2

Таблица 2 - Агент

Факт	Агент
F_0	Автомат
F_1	Станок
F_2	Робот_1
F_3	Робот_2
F_4	Робокар
F_5	Штабелер
F_6	Робот_3
F_7	Автопогрузчик
F_8	–
F_9	–

Таблица 3 - Объект

Факт	Объект
F_0	Заготовка
F_1	Деталь
F_2	Деталь
F_3	Кассета
F_4	Кассета
F_5	Кассета
F_6	Кассета
F_7	Контейнер
F_8	Деталь
F_9	Ячейка

Таблица 4 - Место

Факт	Место
F_0	Станок
F_1	Станок
F_2	Кассета
F_3	Робокар
F_4	Склад
F_5	Ячейка
F_6	Контейнер
F_7	Автомобиль
F_8	Кассета
F_9	Склад

Таблица 5 - След

Факт_1	Факт_2
F_0	F_1
F_1	F_2
F_2	F_3
F_3	F_4
F_6	F_7

Таблица 6 – Возм_след

Факт_1	Факт_2
F_4	F_5
F_4	F_6

Таблица 7 - Одновр

Факт_1	Факт_2
F_8	F_2
F_8	F_3
F_8	F_4
F_8	F_5
F_8	F_6
F_8	F_7
F_8	F_9
F_2	F_8
F_3	F_8
F_4	F_8
F_5	F_8
F_6	F_8
F_7	F_8
F_9	F_8
F_9	F_0
F_9	F_1
F_9	F_2
F_9	F_3
F_9	F_4
F_9	F_5
F_9	F_6
F_9	F_7
F_0	F_9
F_1	F_9
F_2	F_9
F_3	F_9
F_4	F_9
F_5	F_9
F_6	F_9
F_7	F_9

В логике второго порядка дополнительно используются предикатные и функциональные переменные. Несмотря на неполноту, характерную также и для теории множеств, выразительные возможности логики второго порядка богаче выразительных возможностей логики первого порядка. В настоящей работе рассмотрены такие логические модели, которые допускают простую *переинтерпретацию* логики второго порядка в многосортную первопорядковую логику, обладающую полнотой. Один из способов подобной переинтерпретации изложен в описании к лабораторной работе № 1 настоящего пособия.

В абстрактных модулях используются специальные операции – элементарные обновления функций и предикатов. Элементарные правила обновления функций или предикатов записывается в виде правил вывода следующего вида:

$$\frac{t_1, t_2, ..., t_k, t_{k+1}}{s(t_1, t_2, ..., t_k) \leftarrow t_{k+1}},$$

где $t_1, t_2, ..., t_k$ – термы различных сортов, а s – функциональный или предикатный символ. В случае, если s – функциональный символ, t_{k+1} – суть терм любого сорта, а если s – предикатный символ, то t_{k+1} – булево выражение. Отметим, что подобные правила активно используются в теории машин абстрактных состояний Ю. Гуревича [49]. В модулях абстрактных машин второго порядка s может быть предикатной или функциональной переменной. Дополнительно отметим, что символ двойной стрелки "⇐", используемый вместо символа обычной стрелки "←", означает множественное обновление предиката или функции.

При построении выражений для абстрактных модулей используются "квантифицированные" операторы выборки $\exists_{Sel\_any}$, $\exists_{Sel\_one}$, $\forall_{Sel}$ и $\forall_{Sel\_all}$ кортежей из отношений при условии истинности выражений справа от данных операторов. При выполнении оператора $\exists_{Sel\_any}$ в области истинности предиката, описываемого выражением справа, отмечается произвольный кортеж. При выполнении оператора $\exists_{Sel\_one}$ отмечается единственный кортеж, находящийся в области истинности записанного справа предиката. При выполнении оператора $\forall_{Sel}$ отмечаются все кортежи, составляющие область истинности соответствующего предиката. Оператор $\forall_{Sel\_all}$ позволяет отметить все кортежи из области истинности предиката в случае, если его область определения совпадает с его же областью истинности. Во всех случаях подразумевается, что предикат описывается выражением, стоящим справа от символа квантифицированного оператора. Описанные операторы не модифицируют области истинности предикатов, а только выделяют некоторые кортежи в этих областях. Дальнейшие действия, связанные с модификацией предикатов и функций реализуются при помощи описанных выше помощи правил обновления, заключенных в круглые скобки и, возможно, сгруппированных в блоки, ограниченные фигурными скобками.

Каждому из описанных квантифицированных операторов, в случае его использования в условной части выражения (в квадратных скобках) для модуля, ставится в соответствие элементарное логическое условие, истинное в случае успешного выполнения оператора и ложное в противном случае.

Отличительной особенностью используемого формализма являются его расширенные логические возможности за счет использования логики второго порядка, что дает возможность использования не только предметных, но и предикатных и функциональных переменных и упрощение описания и выполнения модулей запросов.

5. Примеры запросов к базе знаний в виде выражений для модулей сети абстрактных машин

Ниже представлены примеры выражений для абстрактных модулей, выполняющих запросы к базе знаний, сформированной на основании концептуального представления модели роботизированного производства. Выполнение подобных запросов позволяет получить недостающую информацию, то есть пополнить экспертные знания о предметной области в процессе проектирования. На рис. 3 представлена основная часть концептуального графа C с некоторыми неизвестными клиенту экспертной системы концептами и отношениями. Выполнении всех запросов производится с использованием таблиц 1 – 7.

5.1. Какие ролевые отношения связывают событие F_7 с концептами *Погрузка_2*, *Автопогрузчик*, *Контейнер* и *Автомобиль*?

При построении запроса на языке сетей абстрактных машин, основанного на логике предикатов первого и второго порядков, определены предикатная переменная $r \in R$ и унарный предикат *Роль_ответ*(r) для отображения результата поиска по отношениям и концептам графа. Ответ на указанный запрос может быть получен при реализации модулями запросов следующих выражений:

$[(\forall_{Sel}\, r \in R)\, r(F_7, \textit{Погрузка\_2})](\, \textit{Роль\_ответ}(r) \Leftarrow \mathbf{true} \vee E)$;

$[(\forall_{Sel}\, r \in R)\, r(F_7, \textit{Автопогрузчик})](\, \textit{Роль\_ответ}(r) \Leftarrow \mathbf{true} \vee E)$;

$[(\forall_{Sel}\, r \in R)\, r(F_7, \textit{Контейнер})](\, \textit{Роль\_ответ}(r) \Leftarrow \mathbf{true} \vee E)$;

$[(\forall_{Sel}\, r \in R)\, r(F_7, \textit{Автомобиль})](\, \textit{Роль\_ответ}(r) \Leftarrow \mathbf{true} \vee E)$.

В результате выполнения данных запросов переменная r будет принимать значения в множестве отношений {*Имя*, *Агент*, *Объект*, *Место*}.

Использование предикатной переменной r означает применения логики предикатов второго порядка.

В приведенных выражениях использован квантифицированный оператор выбора $\forall_{Sel}$ (а не оператор $\exists_{Sel\_one}$ или оператор $\exists_{Sel\_any}$), поскольку некоторые концепты-события в концептуальной модели могли быть связаны с концептами-объектами несколькими отношениями; например, концепт-событие F_1 *Обработка* связано с концептом-объектом *Станок* отношениями *Место* и *Агент*, то есть станок является и агентом, поскольку производит обработку деталей, и местом, на котором размещаются детали при обработке. Так как в общем случае для всего графа при этом оператор $\forall_{Sel}$ находит, возможно, не единственное значение предикатной переменной r, то используемый символ "$\Leftarrow$" означает возможное множественное обновление, или модификацию, предиката *Роль_ответ*(r). Например, в результате выполнения запроса

$$[(\forall_{Sel}\, r \in R)\, r(F_1, \textit{Станок})](\textit{Роль\_ответ}(r) \Leftarrow \mathbf{true} \vee E)$$

будут истинны высказывания *Роль_ответ*(*Агент*) и *Роль_ответ*(*Место*).

5.2. Какие несобытийные концепты связаны с реализацией события F_0?

В случае, когда клиенту экспертной системы известно, что все отношения, представленные концептуальным графом, имеют функциональный характер, выражения для модулей запросов имеют следующий вид:

$$[(\exists_{Sel\_one}\, x \in X_1)\, \textit{Имя}(F_0, x)](\textit{Концепт\_ответ}(x) \leftarrow \mathbf{true} \vee E);$$

$$[(\exists_{Sel\_one}\, x \in X_1)\, \textit{Агент}(F_0, x)](\textit{Концепт\_ответ}(x) \leftarrow \mathbf{true} \vee E);$$

$$[(\exists_{Sel\_one}\, x \in X_1)\, \textit{Объект}(F_0, x)](\textit{Концепт\_ответ}(x) \leftarrow \mathbf{true} \vee E);$$

$$[(\exists_{Sel\_one}\, x \in X_1)\, \textit{Место}(F_0, x)](\textit{Концепт\_ответ}(x) \leftarrow \mathbf{true} \vee E).$$

В результате выполнения этих запросов будет определен унарный предикат *Концепт_ответ*(x), область истинности которого составят искомые концепты {*Подача*, *Автомат*, *Заготовка*, *Станок*}.

Другой вариант решения возможен на основе использования характеристической функции (унарного предиката) $Q_{R1}(r)$, определенной для множества отношений R_1. Более компактное представление запроса, заменяющее приведенные выше четыре выражения для запросов к базе знаний тогда можно представить в следующем виде:

$[(\forall_{\textbf{All}}\, r \in R_1)\, Q_{R1}(r)]([(\exists_{\textbf{Sel\_one}}\, x \in X_1)\, r(F_0, x)]$
$(Концепт\_ответ(x) \Leftarrow \textbf{true} \vee E) \vee E)$.

Построение данного выражения основано на логике предикатов второго порядка. Здесь при выполнении квантифицированного оператора $(\forall_{\textbf{All}}\, r \in R_1)$ $Q_{R1}(r)$ выбираются все значения предикатной переменной r, поскольку область определения предиката $Q_{R1}(r)$ совпадает с областью его истинности. Эти значения затем учитываются при дальнейшем выполнении квантифицированного оператора $(\exists_{\textbf{Sel\_one}}\, x \in X_1)\, r(F_0, x)$. Таким образом, выполнение множественного обновления предиката

$$Концепт\_ответ(x) \Leftarrow \textbf{true}$$

будет эквивалентно выполнению множества одиночных обновлений того же самого предиката:

$\{Концепт\_ответ(Подача) \leftarrow \textbf{true}$,
$Концепт\_ответ(Автомат) \leftarrow \textbf{true}$,
$Концепт\_ответ(Заготовка) \leftarrow \textbf{true}$,
$Концепт\_ответ(Станок) \leftarrow \textbf{true}\}$.

В случае, когда пользователь экспертной системы не уверен в функциональности ролевых отношений, в выражениях для запросов оператор $\exists_{\textbf{Sel\_one}}$ следует заменить на $\forall_{\textbf{Sel}}$ и использовать далее операцию "$\Leftarrow$" множественного обновления предиката $Концепт\_ответ(x)$:

$[(\forall_{\textbf{All}}\, r \in R_1)\, Q_{R1}(r)]([(\forall_{\textbf{Sel}}\, x \in X_1)\, r(F_0, x)]$
$(Концепт\_ответ(x) \Leftarrow \textbf{true} \vee E) \vee E)$.

5.3. Определить концепты-события, необходимо следующие за событием F_2.

Выполнение следующего выражения для запроса позволит получить ответ на заданный вопрос:

$$[(\forall_{\textbf{Sel}}\, x \in F)\, След(F_2, x)](След\_ответ(x) \Leftarrow \textbf{true} \vee E).$$

При составлении выражения предполагалось, что пользователю неизвестно, возможно ли распараллеливание процесса смены событий при выполнении сценария действий. Например, возможен был бы случай, когда за событием F_2 необходимо следует несколько событий, то есть в таком случае отношение *След* не имеет функционального характера. В случае, когда известно, что для события F_2 (как и в рассматриваемом примере сценария) имеется лишь одно последующее событие, выражение для запроса можно было бы заменить следующим выражением, применение которого даст тот же ответ – единственное событие F_3:

$[(\exists_{\textbf{\textit{Sel\_one}}}\, x \in F)\ След(F_2, x)](След\_ответ(x) \leftarrow \textbf{true} \vee E).$

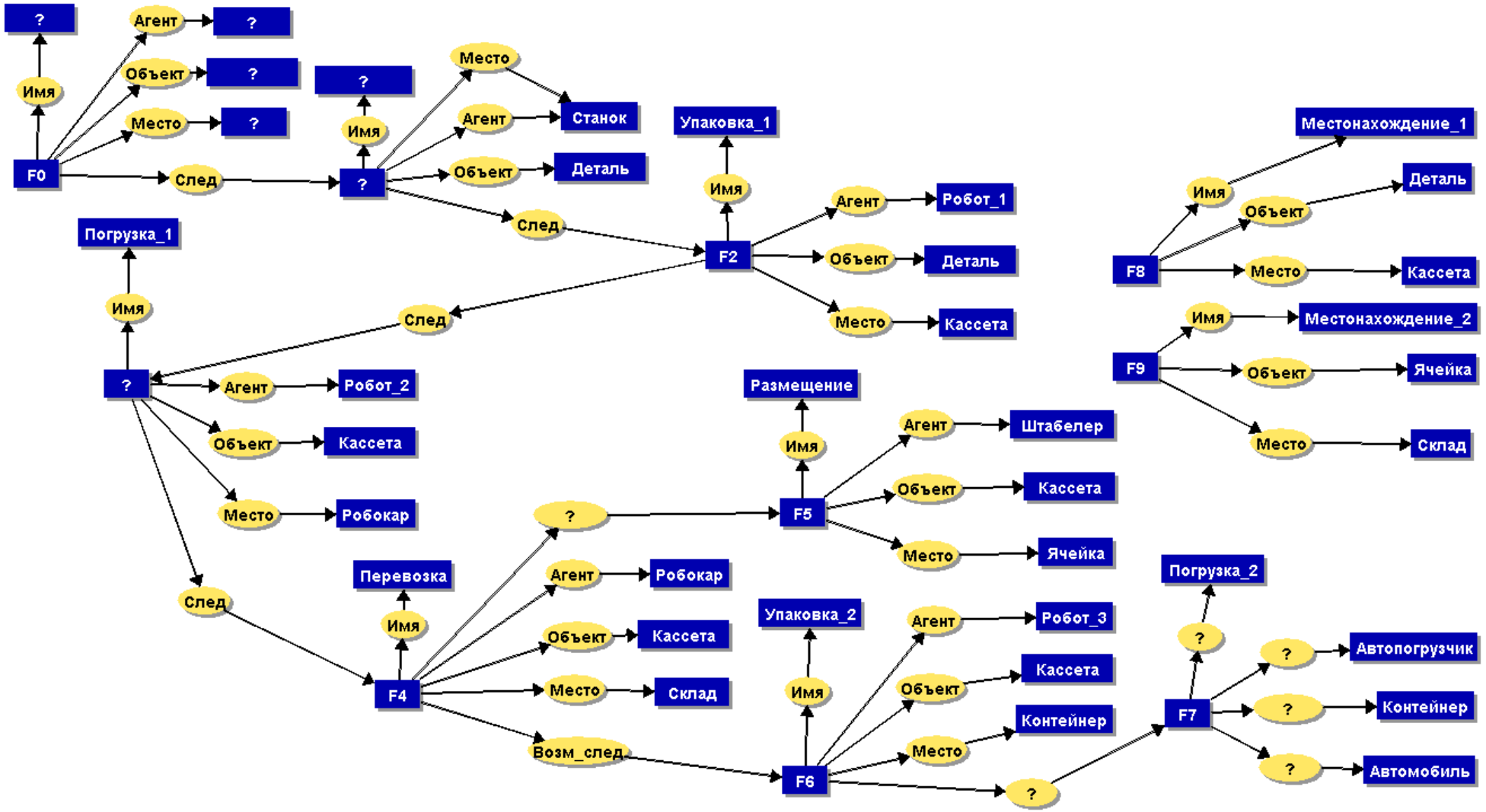


Рис. 3. Вид концептуального графа *C* с некоторыми неизвестными концептами и отношениями, которые необходимо найти при помощи запросоов

5.4. Определить концепты-события, «возможно» следующие за событием F_4.

Исследователь сценария может предполагать, что непосредственно после или при выполнении события F_4 формируется одно или несколько условий, определяющих возможные дальнейшие события. Запрос в данном случае может быть описан следующим выражением:

$$[(\forall_{\textbf{Sel}}\, x \in F)\ \textit{Возм\_след}(F_4, x)](\textit{Возм\_след\_ответ}(x) \Leftarrow \textbf{true} \vee E).$$

Для рассматриваемого сценария после выполнения данного запроса область истинности предиката *Возм_след_ответ*(x) будет содержать два события – F_5 и F_6.

5.5. Какое отношение связывает концепты-события F_4 и F_5? Другой вариант вопроса: «Возможно ли следование события F_5 непосредственно после события F_4?».

$$[(\exists_{\textbf{Sel\_one}}\, r \in \{\textit{След}, \textit{Возм\_след}\})\ r(F_4, F_5)]$$
$$(\textit{Отношение\_ответ}(r) \leftarrow \textbf{true} \vee E).$$

В результате применения подобного запроса к базе знаний рассматриваемого примера будет истинно высказывание *Отношение_ответ*(*Возм_след*), означающее, что события F_4 и F_5 связывает отношение непосредственного следования, или что возможно следование события F_5 непосредственно после события F_4.

5.6. В каких событиях и в каких ролях участвовал робокар?

$$[(\forall_{\textbf{All}}\, r \in R_1)\ Q_{R1}(r)]([(\forall_{\textbf{Sel}}\, x \in F)\ r(x, \textit{Робокар})]$$
$$(\textit{Результат\_1}(x, r) \Leftarrow \textbf{true} \vee E) \vee E).$$

В результате выполнения данного запроса область истинности бинарного предиката *Результат* должны содержать кортежи <F_3, *Место*> и <F_4, *Агент*>, то есть должны оказаться истинными высказывания *Результат*(F_3, *Место*) и *Результат*(F_4, *Агент*). Это означает, что робокар при выполнении события F_3 был местом для размещения кассет, а при выполнении события F_4 стал агентом для этих же кассет.

Для того, чтобы узнать названия событий F_3 и F_4, необходимо выполнить два следующих запроса:

$$[(\exists_{\textbf{Sel\_one}}\, y \in X_N)\ \textit{Имя}(F_3, y)](\textit{Имя\_события}(F_3, y) \leftarrow \textbf{true} \vee E);$$
$$[(\exists_{\textbf{Sel\_one}}\, y \in X_N)\ \textit{Имя}(F_4, y)](\textit{Имя\_события}(F_4, y) \leftarrow \textbf{true} \vee E),$$

в результате чего будут истинны высказывания *Имя_события*(F_3, *Погрузка_1*) и *Имя_события*(F_4, *Перевозка*) , утверждающие, что событие F_3 – это *Погрузка_1*, а событие F_3 – *Перевозка*. Оба отношения имеют функциональный характер, так как символам событий имена присвоены однозначно.

5.7. Связаны ли события F_6 и F_7 отношением непосредственного следования?

При поиске ответа на данный вопрос просто проверяется истинность или ложность высказывания *След*(F_6, F_7). Для рассматриваемого примера это высказывание истинно.

Приведенные выражения для запросов к базе знаний представляют точные формализованные спецификации для сценариев получения новых знаний о предметной области или уточнения прежних забытых или утраченных знаний. При проектировании человеко-машинного интерфейса интеллектуальной системы возможно использовать известные языки, например, C++, C#, Python, SQL, Prolog. Последние два языка не используют логику второго порядка, поэтому при реализации запросов необходимо просто перечислять отношения, из которых выбираются кортежи. Другой способ решения данной проблемы заключается в переинтерпретации запросов, записанных с использованием логики второго порядка, в первопорядковую логику, как это предлагалось в лабораторной работе № 1.

6. Переход от концептуальной сценарной модели роботизированного производства к имитационной модели

Логические формализмы позволяют описывать структурные и логические связи между сущностями предметной области в виде составных высказываний, истинность которых соответствует ситуациям, возникающим в заданной предметной области. В качестве логического языка нами было использовано многосортное исчисление предикатов (с равенством) первого и, частично, второго порядков.

При построении поведенческих – исполнимых, или операционных, моделей необходимы не только операционная поддержка, обеспечиваемая меха-низмами вывода используемого логического формализма, но и правила, изменяющие интерпретацию сигнатуры при моделировании поведения дискретно-событийной системы. Эти правила и механизмы их исполнения основаны на правилах модификации предикатов и функций.

На рисунке 4 представлен модифицированный концептуальный граф для сценария работы участка роботизированного производства и его контекстное представление. Модификация заключается во введении события

F'4 (возвращение робокара к станку после доставки очередной партии деталей, упакованных в кассеты, на склад). После доставки деталей на склад робокар возвращается к участку ГАП, где его ожидает новая партия кассет, а на складе принимается решение, размещать ли кассеты кассеты в ячейках склада (при истинном условии α), либо упаковывать их в контейнер (при ложном α), после чего автопогрузчик погрузит контейнер на автомобиль. Так как заранее неизвестно, где будут находиться кассеты, будем считать, что между парами событий *F4, F5* и *F4, F6* установлены отношения возможного следования.

Модели, описанные графами на рисунках 1, 2, 3 и 4, имеют декларативный характер, и мы их относим к классу концептуальных. Эти модели далее положены в основу построения поведенческих моделей.

Непосредственно при построении поведенческой модели семантические сети и концептуальные графы могут найти лишь ограниченное применение. Например, рассмотрим смену состояний сети, введя предикат *Execute*(*F*), определенный на множестве значений предметной переменной $F \in \{F_1, F_2, F_3, F_4, F_5, F_6, F_7, F_8, F_9\}$. Изменение интерпретации предикатного символа *Execute* иллюстрирует рисунок 5.

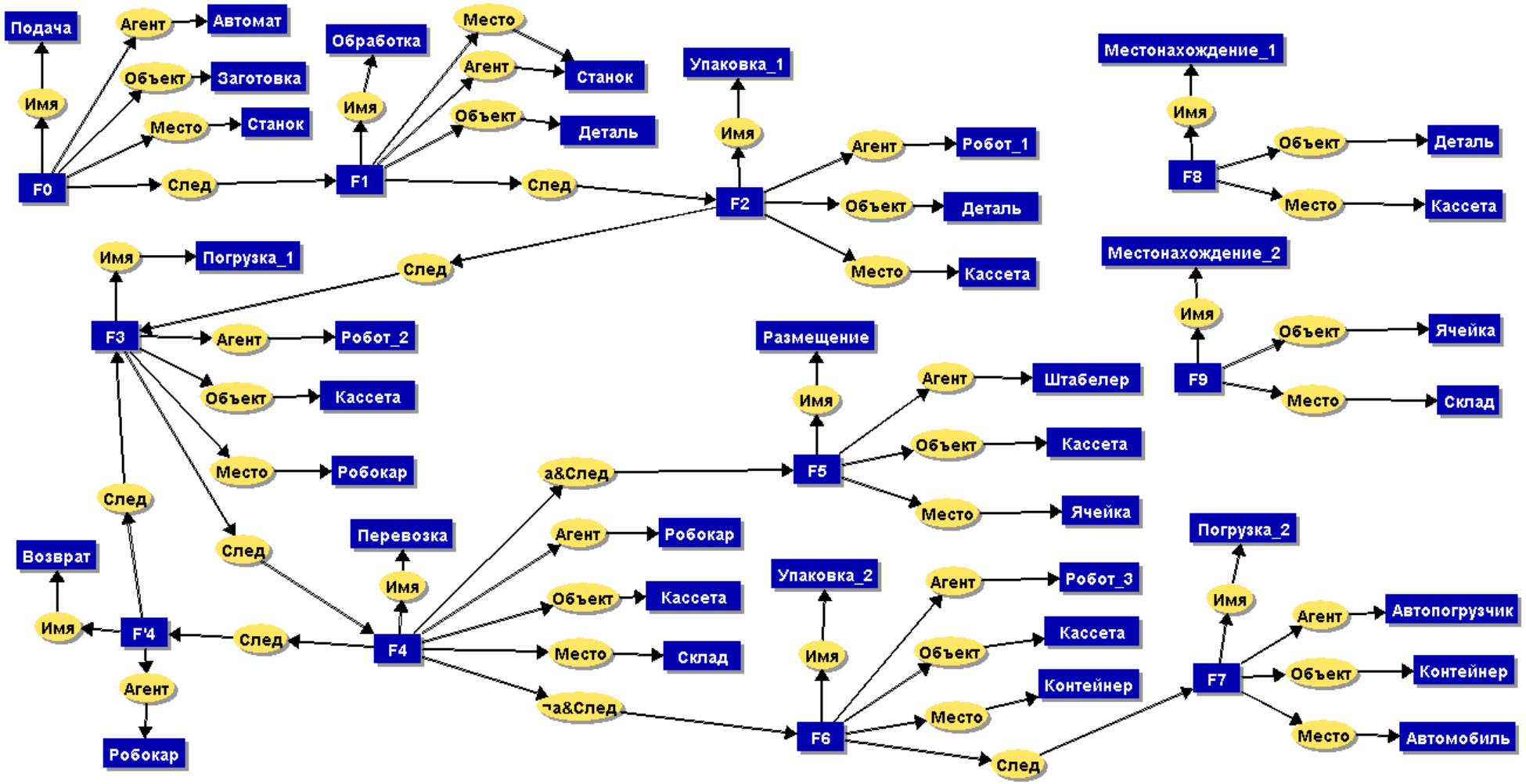


Рисунок 4 – Модифицированный концептуальный граф для сценария работы участка роботизированного производства и его контекстное представление

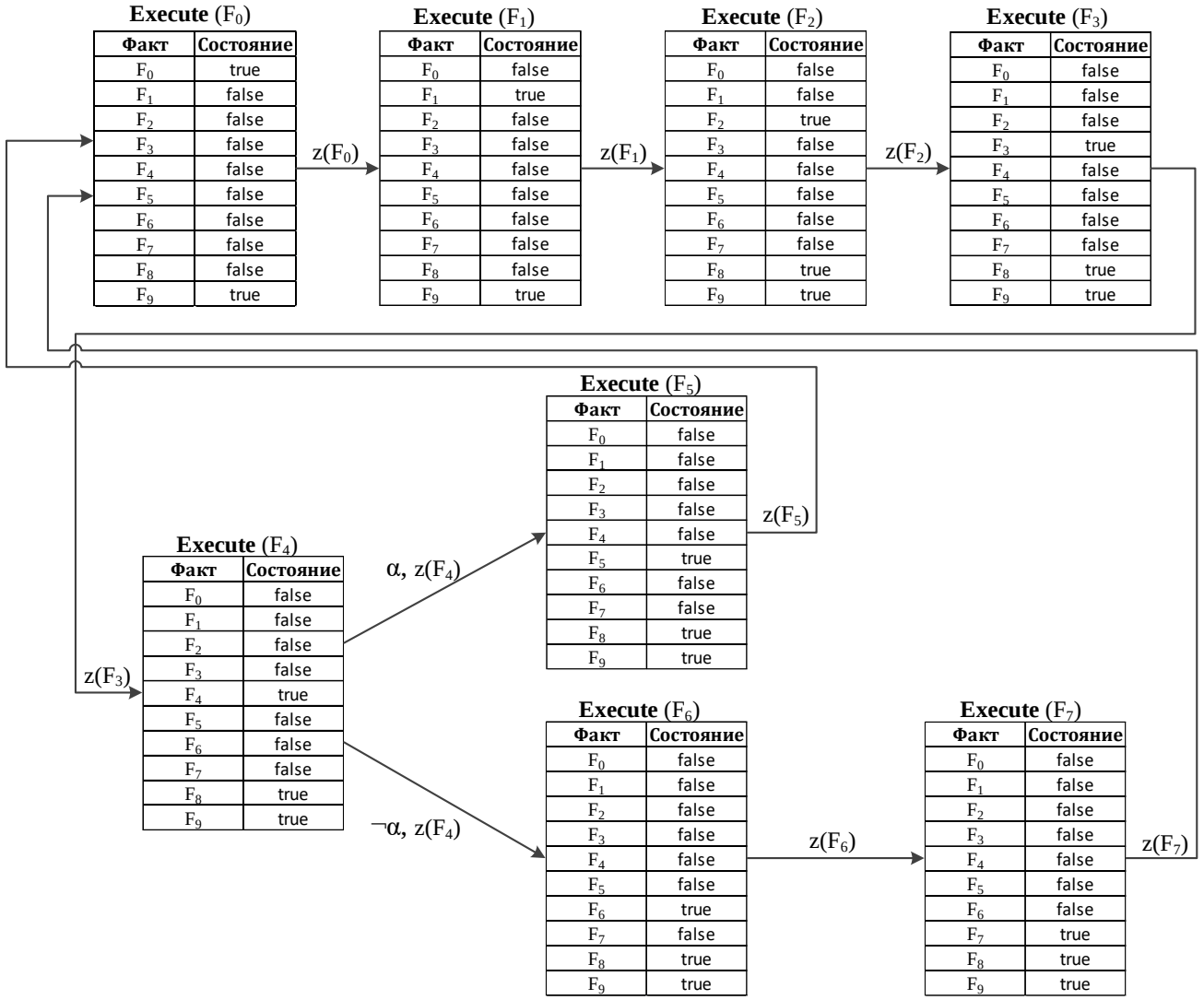


Рисунок 5 – Модификации предиката **Execute**(F) в процессе выполнения сценария

Здесь $z(F_i)$ – признак завершения события (свершения одноименного факта) F_i, i = 1, 2, …, 9. Смена состояний таблиц соответствует простейшему случаю работы моделируемой системы при отсутствии возможности для параллельного выполнения событий.

7. Сеть Петри, описывающая работу участка роботизированного производства ГАП

Граф, представленный на рисунке 5, описывает смену состояний для частного случая прохождения лишь одной кассеты с деталями. Вместе с тем очевидно, что ряд событий может выполняться параллельно. Поэтому из ряда исполнимых, или операционных, моделей дискретно-событийных систем выберем сети Петри.

Рассмотрим в качестве примера сеть Петри, описывающую работу участка роботизированного производства ГАП. На рисунке 6 позициями R_0, R_1, …, R_8 в модели представлены ресурсы-агенты участка ГАП – автомат (R_0), подающий партии заготовок деталей к станку; станок (R_1), обрабатывающий детали; робот 1 (R_2), упаковывающий готовые изделия в кассеты; робот 2 (R_3), осуществляющий погрузку кассет с деталями на робокар – мобильный объект (R_4); штабелер – робот-погрузчик (R_5), размещающий кассеты с деталями в ячейки на складе; робот 3 (R_6), упаковывающий кассеты в контейнер; автопогрузчик (R_7), грузящий контейнеры с кассетами на автомобиль (R_8). Наличие меток в позициях соответствует незанятости данных ресурсов, или готовности выполнения своих функций. При имитации занятия ресурса метка удаляется из соответствующей позиции.

Позиции W_0, W_1, имитируют текущее размещение заготовок для будущих деталей до упаковки в кассеты, а позиции W_2, W_3, W_4, W_5, W_6, W_7 используются для имитации текущего размещения кассет уже с деталями. Предполагается, что число заготовок в каждой партии, поступающей на обработку, равно числу деталей, упакованных в кассету. Позиция W'_4 – вспомогательная.

Переходы, соответствующие событиям F_1, F_2, F_3, F_4, F'_4, F_5, F_6, F_7, в модели на рисунке 6, реализуют события, представленные ранее в семантической сети (сценарии) на рисунке 1 и в концептуальном графе на рисунке 4. Введен дополнительный переход, реализующий в сети событие F_k, имитирующий учет изделий в кассетах и завершающий работу сети Петри. Вместимости робокара (не более C кассет) и контейнера (не более M кассет) ограничены, что учтено при определении правил срабатывания переходов F_3, F_6 и F_7.

Как обычно в сетях Петри, асинхронные срабатывания переходов приводят к изменению текущей разметки сети. В отличие от семантической сети с событиями (рисунок 1) и концептуального графа (рисунок 2) сетью Петри (рисунок 6) эффективно представляется динамика модели, ее параллельное выполнение.

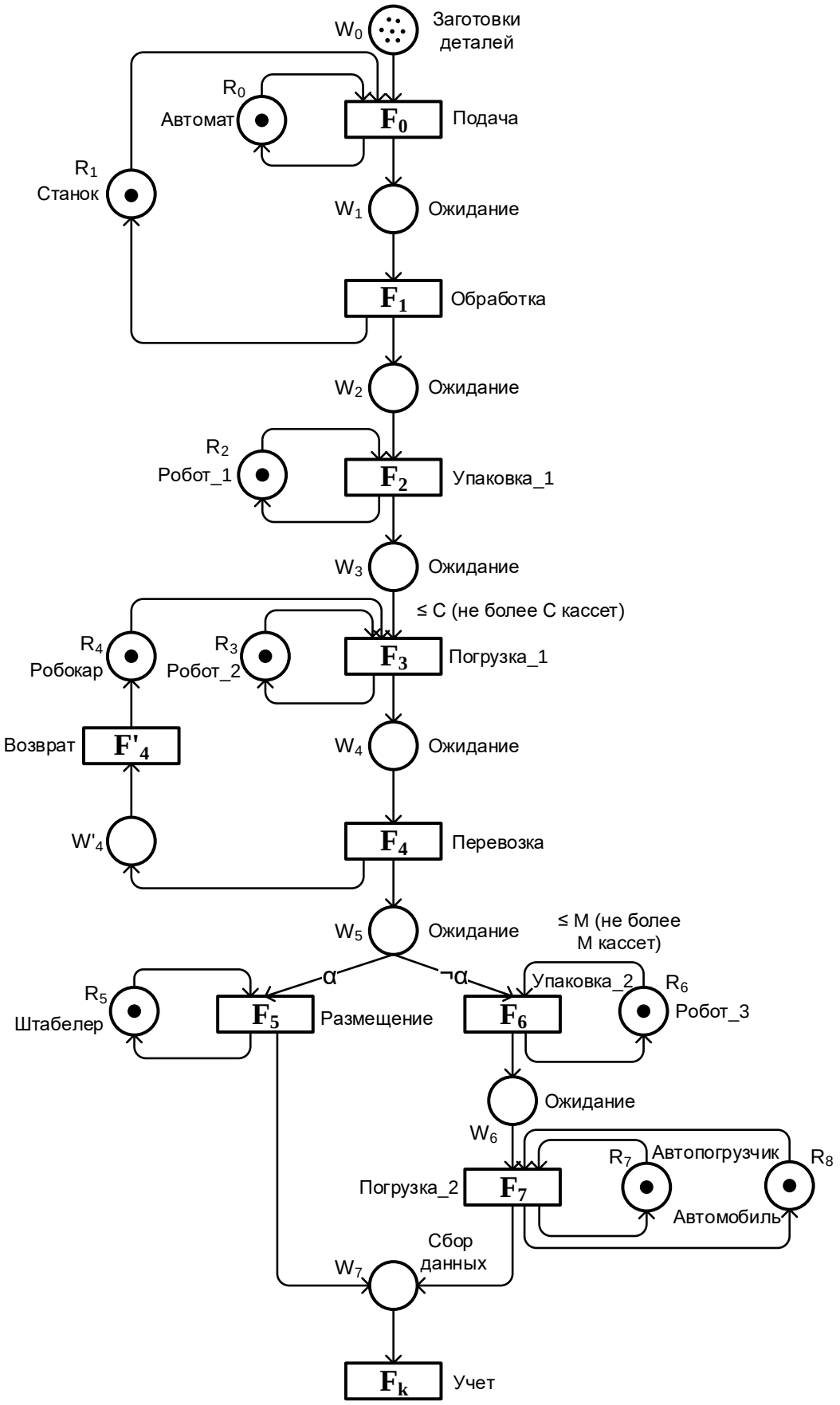


Рисунок 6 – Сеть Петри для сценария работы участка роботизированного производства

Например, обработка заготовок может выполняться одновременно с перемещением робокара, то есть в отмеченном и в очевидных других случаях в модели реализуется «конвейерный» параллелизм. Поэтому модель, построенная в терминах сетей Петри и доопределенная правилами и процедурами срабатывания переходов, обладает всеми признаками поведенческой. Сведения об основных инструментальных средствах, специально предназначенных для исследования сетей Петри, приведены на Web-сайте [45]. Правильность построения и работы сетей Петри, представленных в настоящей работе, проверялась при помощи программы PIPE [46], получившей широкое распространение в международной практике.

8. Интеграция концептуальных и поведенческих моделей дискретно-событийных систем: концептуальные сети Петри

Достоинством концептуальных моделей дискретно-событийных систем, представленных семантическими сетями с событиями и концептуальными графами является их декларативный характер, позволяющий строить экспертные системы и системы ситуационного управления. В дискретно-событийных моделях систем на основе разновидностей сетей Петри проще учитывается динамика процессов, занятие и освобождение ресурсов, управление очередями. Предложим вариант интеграции моделей обоих классов в виде концептуальных сетей Петри (КСП). Пример КСП-сети для сценария работы участка ГАП роботизированного производства представлен на рисунке 7.

КСП представляет собой концептуальный граф, в котором события, или первые концепты, представленные переходами обычной сети Петри на рисунке 6, связаны ролевыми отношениями с вторыми концептами – именами событий, агентами, объектами, местами. Представленные графически ролевые отношения соответствуют семантической сети (рисунок 1) и концептуальному графу (рисунок 4).

Подобное представление позволяет строить удобные в использовании экспертно-имитационные модели дискретно-событийных систем, сочетающих признаки инфологических моделей с операционными (исполнимыми) моделями типа сетей Петри.

Принцип детализации иллюстрирует рисунок 8. На рисунке 8, *а* представлен пример «концептуального» перехода КСП для события F_1. На рисунке 8, *б* графически представлена конкретизация ролевых отношений. Рисунок 8, *в* показывает, что в графе можно было бы представить и дополнительные отношения, например, *Тип_ст* – тип станка (токарный); *Число_партий* – число партий заготовок, принимаемое станком (в примере – одна партия); *Тип_детали* – тип обрабатываемой детали (ступенчатый вал); *Тип_обраб* – тип обработки детали (токарная обработка).

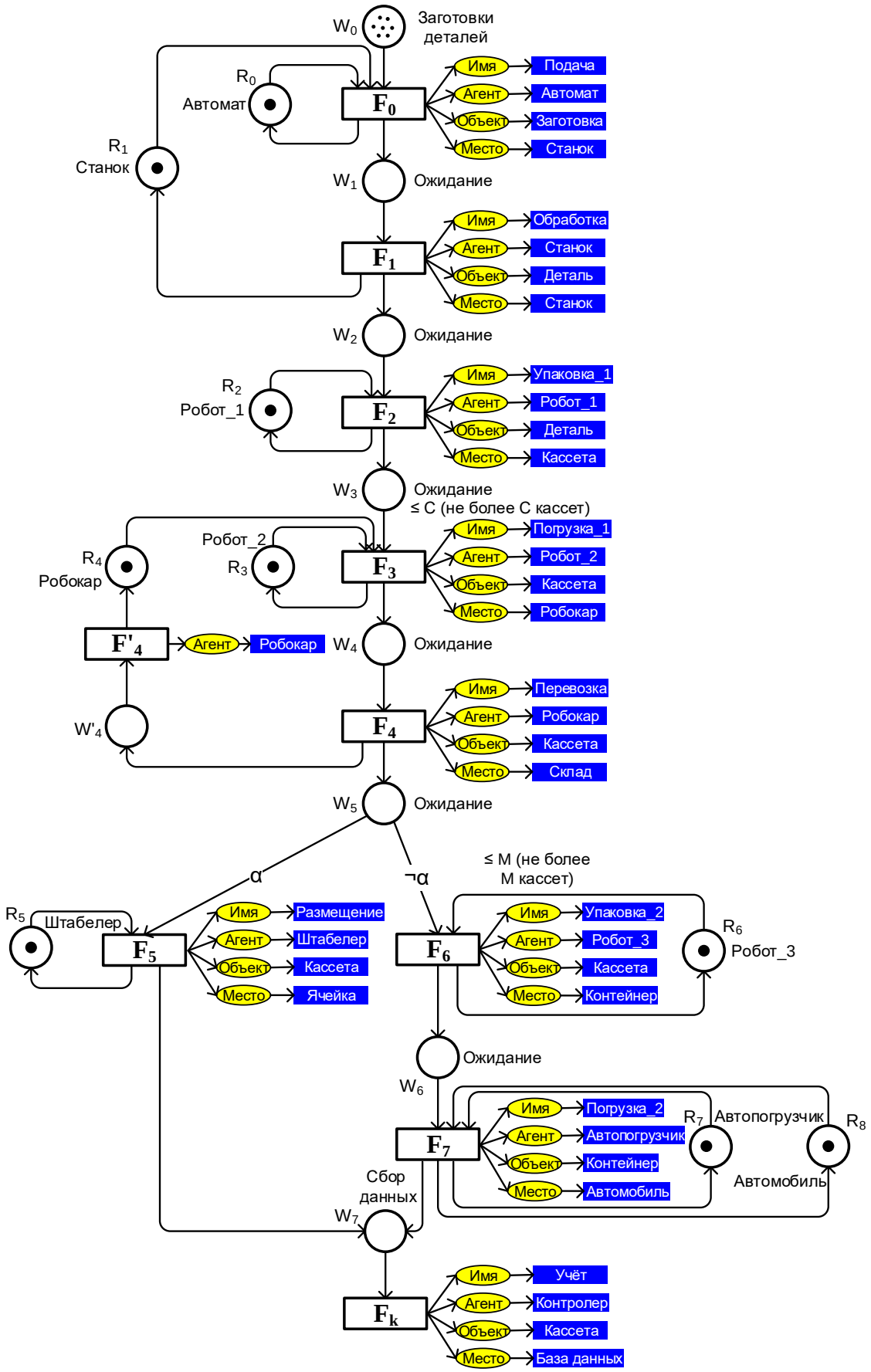


Рисунок 7 – Концептуальная сеть Петри для сценария работы участка

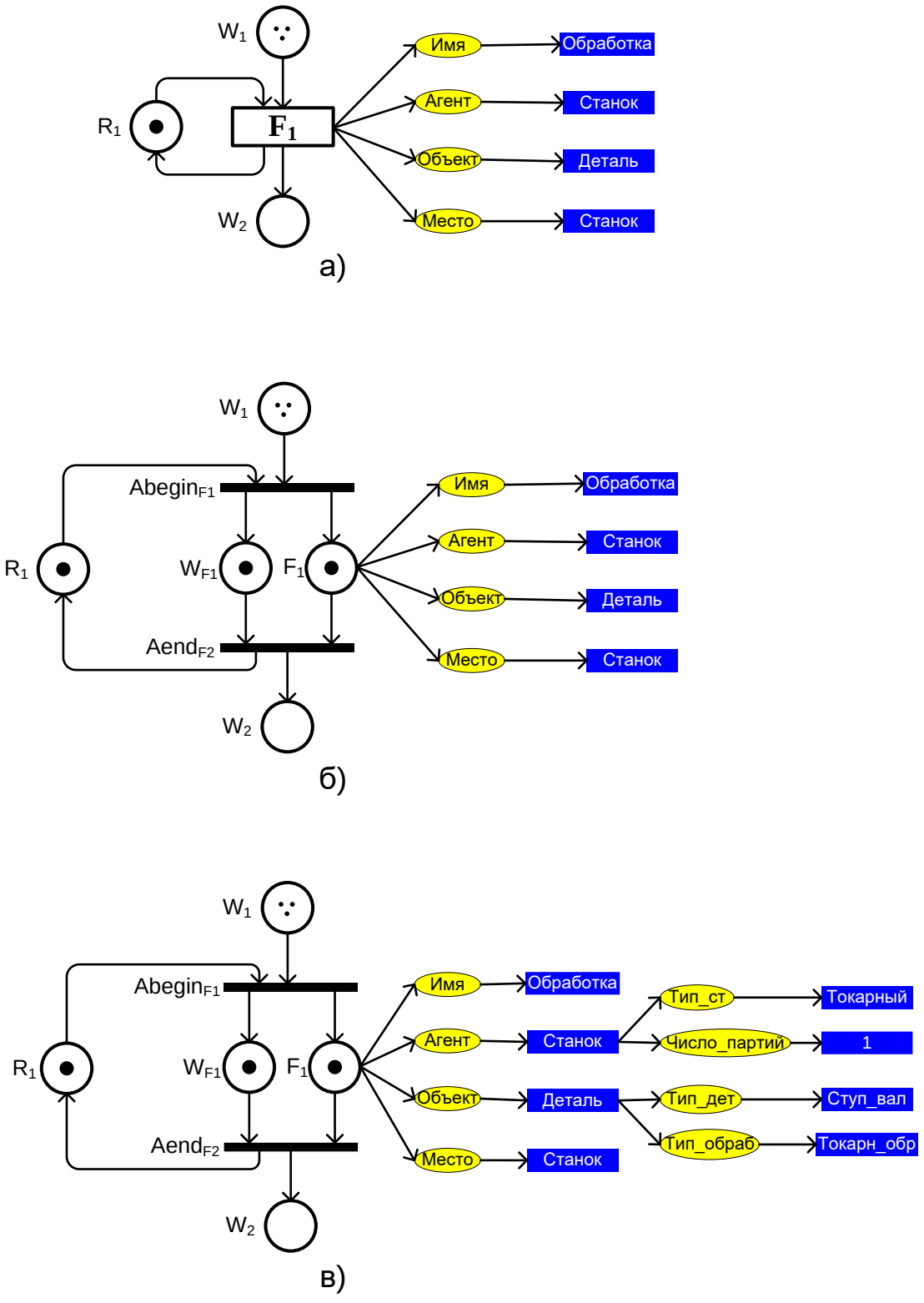


Рисунок 8 – Детализация представления фрагмента концептуальной сети Петри

9. Разработка функциональной модели беспроводной сети MANET, управляющей работой участка роботизированного производства

Управление производственными процессами ГАП возможно реали-зовать при помощи беспроводных мобильных ad hoc сетей MANET с пе-ременной топологией (Mobile Ad hoc NETworks). В ad hoc сетях каждый узел может выполнять функции маршрутизатора и способен ретранслировать пакеты. Узлы сети типа MANET предназначены для работы на подвижных объектах и могут перемещаться независимо друг от друга. В сетях MANET возможно использовать как реактивные, так и проактивные протоколы маршрутизации. Для рассматриваемых в настоящей работе производственных процессов, управляемых беспроводной сетью MANET, возможно ограничиться реактивным протоколом AODV (Ad hoc On-Demand Distance Vector protocol). Возможно использовать другие виды протоколов маршрутизации; при выборе протокола следует учитывать частое изменение топологии сети.

Уточним размещение подвижных и стационарных объектов при организации роботизированного производства (рисунок 9). Оператор (S_0), автомат (R_0), станок (R_1), робот 1 (R_2) и робот 2 (R_3) размещены на участке ГАП и не выходят из зоны радиовидимости сети. Штабелер (R_5), склад (S_1), робот 3 (R_6) и автопогрузчик (R_7) находятся на участке логистики и также не покидают зону радиовидимости сети. На всех перечисленных объектах размещены узлы сети MANET. Для обеспечения непрерывной связности этих узлов сети MANET в ее составе используются транзитные узлы T_1, T_2 и T_3, число которых определяется расстоянием между участками ГАП и логистики.

Мобильные узлы сети MANET, размещенные на робокаре (R_4) и на автомобиле (R_8), время от времени выходят из зоны радиовидимости сети: робокар выполняет перевозку готовых изделий в кассетах между участками, а автомобиль доставляет контейнеры с произведенной продукцией потребителю. На рисунке 9 вр*е*менные связи мобильных узлов с сетью MANET представлены пунктирными стрелками. Штрих-пунктирными линиями условно показаны направления перемещения мобильных объектов с установленными на них узлами сети MANET. Работа распределенного программного обеспечения сети определяется в целом сетями Петри (рисунки 6, 7).

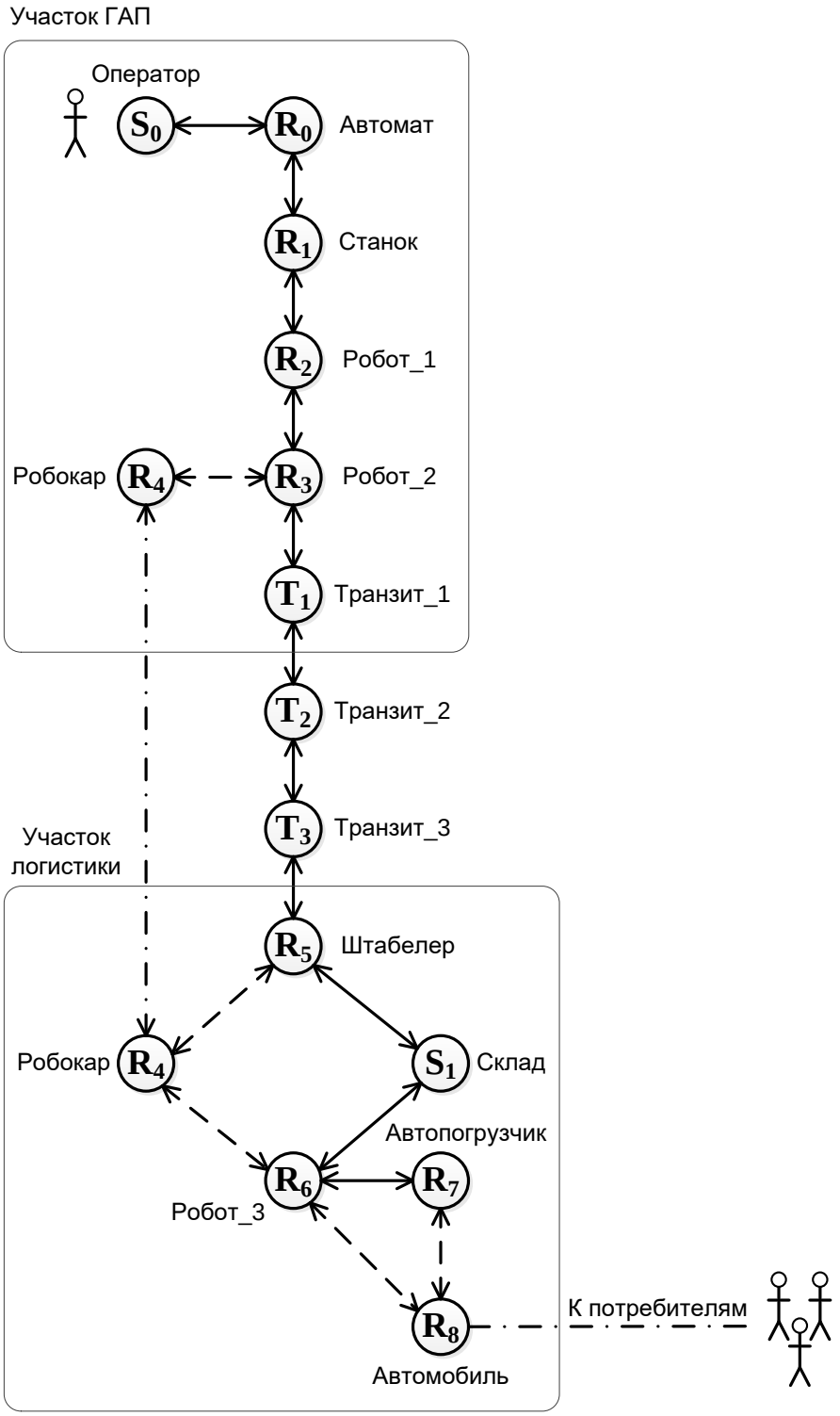


Рисунок 9 – Схема развертывания беспроводной управляющей сети MANET
в производственной среде

10. О реализации вычислительной и коммуникационной компонент киберфизической системы

В киберфизических системах вычислительная компонента, как правило, распределена по всей физической системе и связана с её составляющими элементами. Вычислительная и коммуникационная платформы являются основой для построения большинства современных эргатических и киберфизических систем. Эти платформы реализуют необходимые для прикладной функциональности алгоритмы. Взятые вместе, эти платформы называются сетевыми. Для классификации компьютерных сетей используются различные признаки, но чаще всего сети делят на типы по территориальному признаку, то есть по величине территории, которую покрывает сеть.

В настоящее время существует большое разнообразие вычислительных сетей, которые нашли свое применение в промышленности и на транспорте (https://aveon.ru). Например, на основе коммерческих беспроводных сетей стандартов IEEE 802.11 a/b/g/n появились промышленные сети Industrial Wi-Fi, для которых разрабатываются и тестируются устройства промышленного iWifi стандарта IEEE 802.11ac. Устройства для промышленных Wi-Fi сетей, предназначенные для работы в тяжелых промышленных условиях, могут характеризоваться следующими признаками: иметь прочный металлический корпус с защитой от вибраций, пыли и влаги; выполняться с учетом промышленного рабочего диапазона температур от -40 до +75°C; иметь гальваническую развязку интерфейсов и вводов/выводов для защиты от высокого уровня электромагнитных помех; обладать резервированием по электропитанию; иметь коннекторы, которые обеспечивают физическую защиту Ethernet-портов при работе в условиях агрессивной внешней среде и на движущихся объектах, которые препятствуют самопроизвольному разъединению и попаданию в устройство пыли и влаги.

Индустриальные беспроводные Mesh-сети – это масштабируемые, защищенные, управляемые ячеистые сети, обеспечивающие Wi-Fi покрытия больших территорий. Все узлы Mesh-сетей автоматически конфигурируются, при выходе любого узла сети MESH абоненты переключаются на соседние узлы, а сами узлы переконфигурируются без потери связи. Трафик от абонентов такой сети автоматически перенаправляется по лучшему маршруту. Узлы Mesh-сетей могут получать начальную конфигурацию и от контроллера, как и обычные точки доступа.

Решения Industrial Wifi нашли широкое применение на многих производственных площадках, особенно на предприятиях в нефтегазовой, энергетической и горнодобывающей отраслях, на транспорте, железной дороге и метрополитене.

11. Элементы киберфизической системы для роботизированного производства

Выбранные и размещенные ниже рисунки 10 – 19 иллюстрируют применяемые различными фирмами компоненты абстрактного роботизированного производства. Эти роботы и роботизированные устройства могли бы использоваться на участках ГАП и логистики рассмотренного примера.

Ближе всего по функциональной организации к рассмотренному примеру подходит, например, гибкая производственная система (ГПС) фирмы Shin Nippon Koki Ko, Ltd (Япония) для обработки крупногабаритных деталей. Она предназначена для обработки деталей судовых дизелей шести наименований с максимальными размерами по длине 2200 мм, ширине 900 мм, высоте 900 мм и массой до 2000 кг. В состав ГПС входят механический цех, в котором установлено три многоцелевых станка и имеются позиции накопления приспособлений с роботом перегрузки (робокаром 1), робот загрузки-разгрузки и переналадки, автомат накопления поддонов-спутников заготовок, а также

робот-перегрузчик заготовок (робокар 2). Для транспортирования заготовок и приспособлений используются два промышленных робота (робокара), совершающие продольное и поперечное движения с помощью установленной под полом ведущей индукционной антенны. Робокар 2 предназначен для транспортирования заготовок и поддонов общей массой до 4000 кг. Он транспортирует поддоны с заготовками к механизмам автоматической смены заготовок многоцелевых станков, а также транспортируют поддоны с обработанными деталями на позицию накопления поддонов-спутников. Робокар 3 (мини-робот) предназначен для перевозки приспособлений; он малогабаритный и высокоскоростной. Он предназначен для транспортирования приспособлений накопителей к станкам ГПС и обратно. Вся ГПС управляется из компьютерного пункта.

Рисунок 10 – Роботизированное производство под управлением беспроводной сети

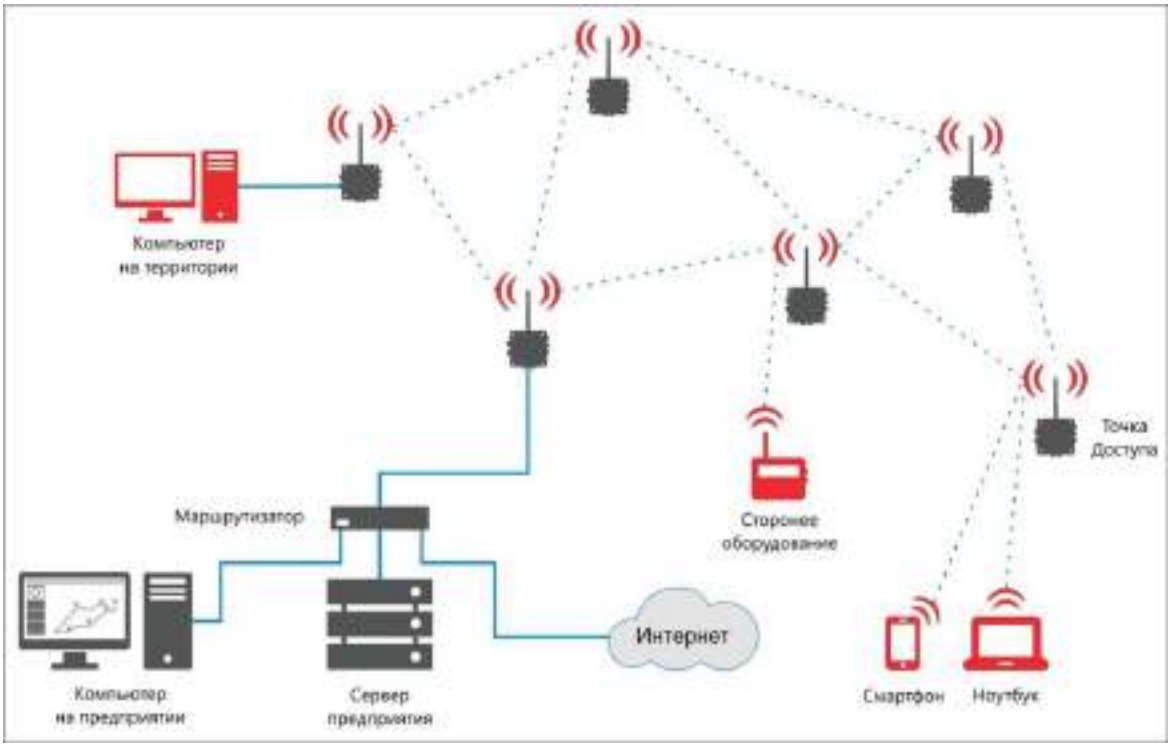


Рисунок 11 – Структурная схема промышленной сети Wi-Fi

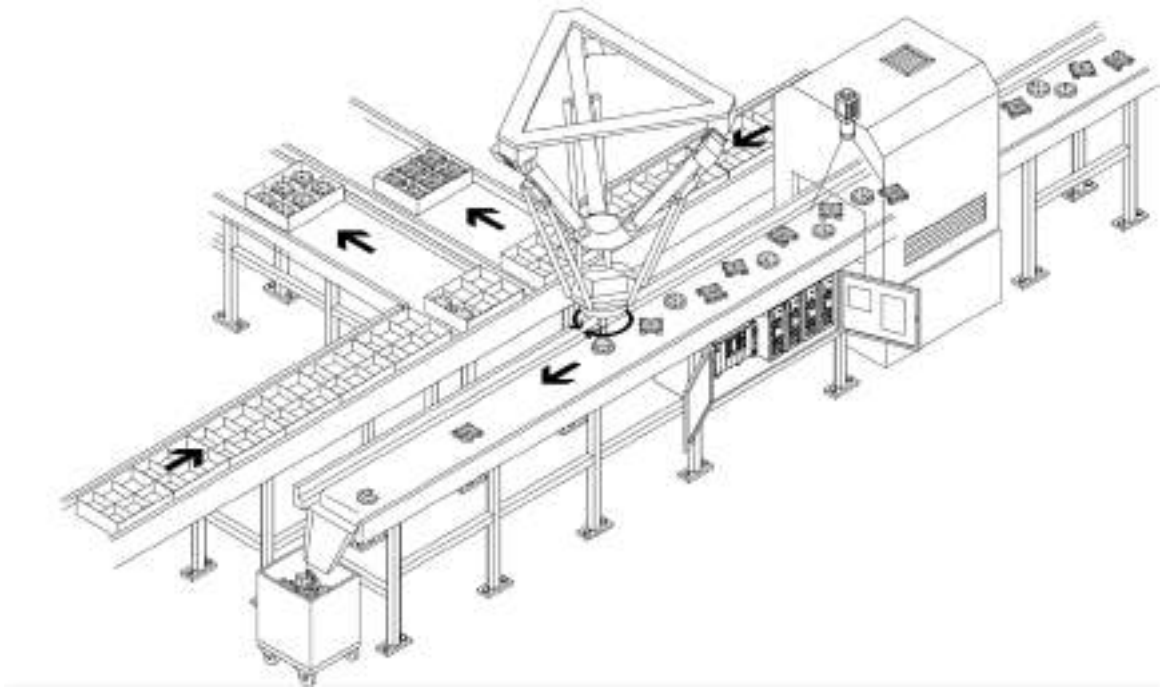


Рисунок 12 – Автомат для упаковки изделий на линии

Рисунок 13 – Робот-паллетайзер

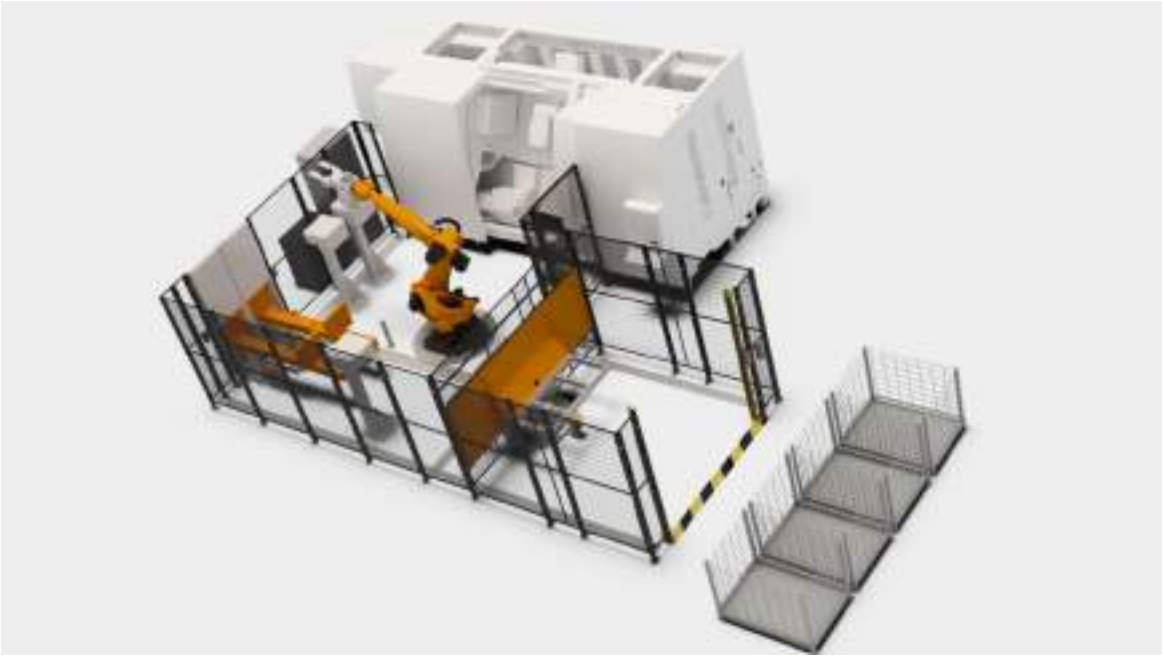


Рисунок 14 – Роботизированная ячейка ГАП

Рисунок 15 – Роботизированный модуль с передвижным подвесным роботом

Рисунок 16 – Промышленный робот-манипулятор

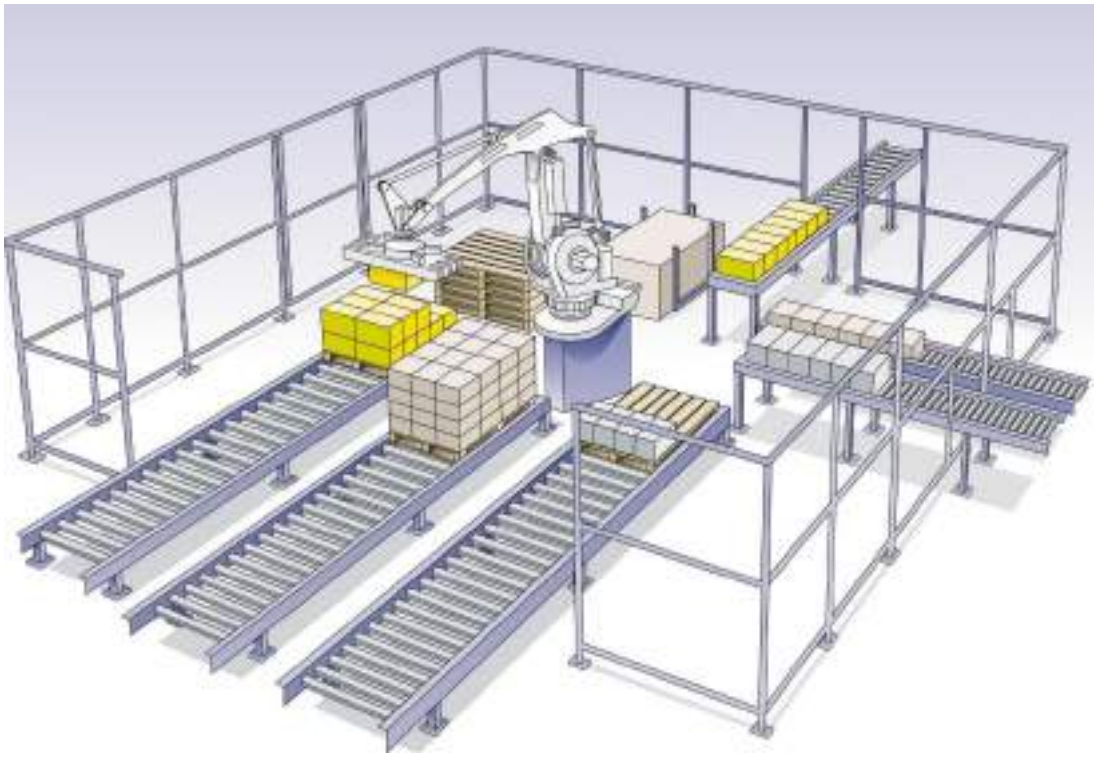


Рисунок 17 – Линия упаковки изделий

Рисунок 18 – Участок логистики: размещение контейнеров с изделиями на складе готовой продукции при помощи вилочного погрузчика-разгрузчика

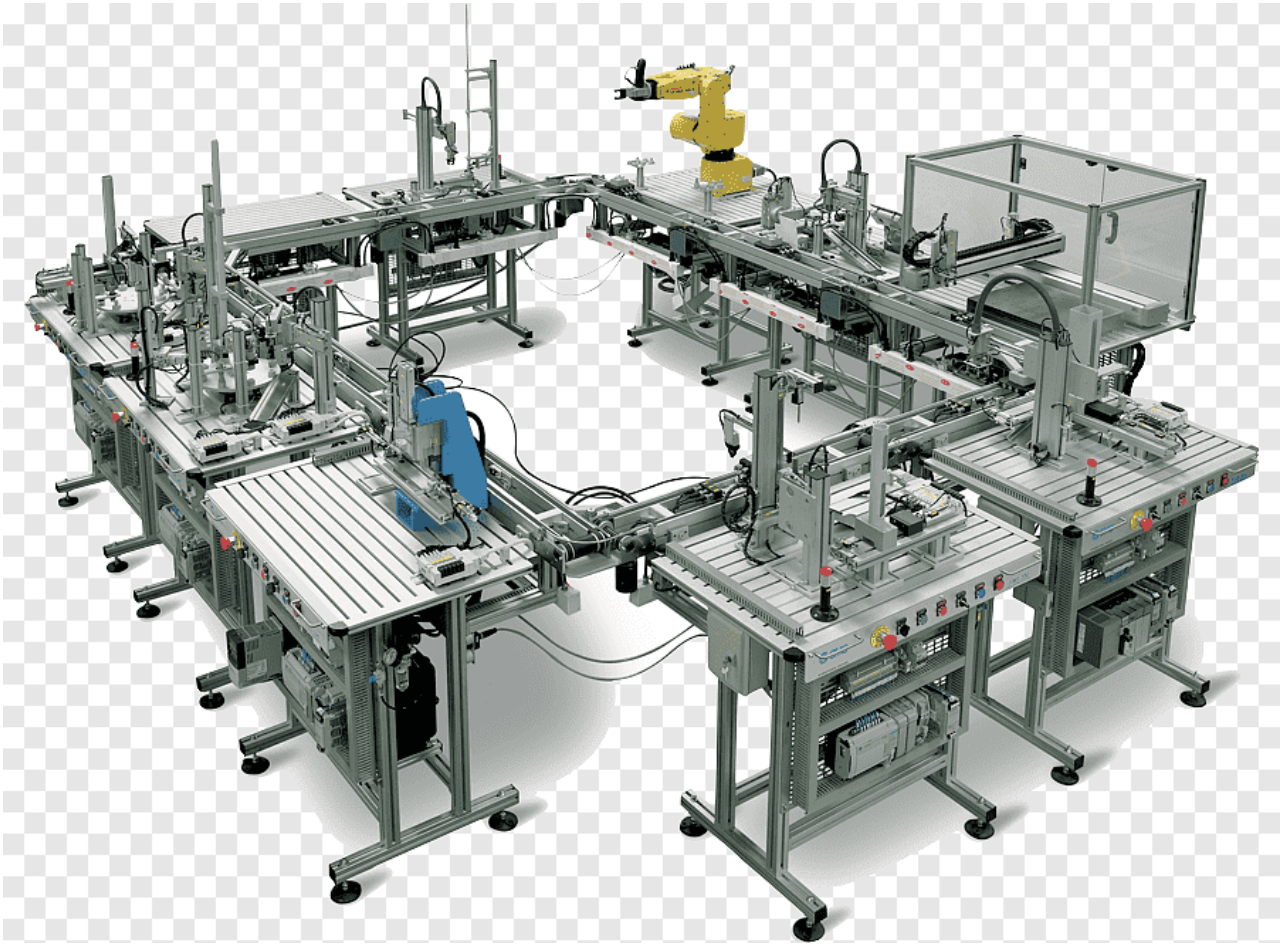


Рисунок 19 – Участок ГАП

Выводы

1. Показано, что семантические сети с событиями, ролевыми и каузальными связями между объектами (или сценарные сети) могут в достаточно полной степени представлять логическую, структурную и процедурную парадигмы развитых концептуальных моделей дискретно-событийных интеллектуальных систем, в том числе систем ситуационного управления в составе киберфизических систем.

2. Концептуальное представление сценариев позволяет естественным образом автоматизировать переход к построению базы знаний на основе реляционного подхода, что позволяет повысить оперативность внесения

изменений в концептуальную модель и базу знаний (например, при изменениях в предметной области).

3. Запросы к базе знаний, построенной на основе концептуального графа, могут быть формализованы на языке сетей абстрактных машин с использованием логики первого и второго порядков, что упрощает дальнейшее проектирование человеко-машинного интерфейса интеллектуальной системы на базе известных языков типа C++, C#, SQL, Python, Prolog и др.

Список литературы

1. Сидоркина, И. Г. Системы искусственного интеллекта : учебное пособие для вузов / И. Г. Сидоркина. – М. : Изд-во : КНОРУС, 2015. – 248 с.
2. Болотова, Л. С. Системы искусственного интеллекта: модели и технологии, основанные на знаниях : учебник для вузов / Л. С. Болотова. – М. : Финансы и статистика, 2012. – 664 с.
3. Рассел, С., Норвиг, П. Искусственный интеллект: современный подход. – М. : Издательский дом «Вильямс», 2007. – 1408 с.
4. Логический подход к искусственному интеллекту: от классической логики к логическому программированию / Тейз А. , Грибомон П , Луи Ж , и др. – М. : Изд-во Мир, 1990. – 429 с.
5. Искусственный интеллект. Применение в интегрированных производственных системах / Кьюсиак Э. (ред.). – М. : Изд-во : Машиностроение, 2018. – 544 с.
6. Бабкин, Э. А. Принципы и алгоритмы искусственного интеллекта : монография / Э. А. Бабкин, О. Р. Козырев, И. В. Куркина. – Н. Новгород : Изд-во : Нижегород. гос. техн. ун-та, 2006. – 132 с.
7. Бессмертный, И. А. Искусственный интеллект : учебное пособие для вузов / И. А. Бессмертный. – СПб : Изд-во : СПбГУ ИТМО, 2010. – 132 с.
8. Гаврилов, А. В. Интеллектуальные системы и основы теории интеллектуального управления : методические указания к лабораторным работам / А. В. Гаврилов. – Новосибирск : Изд-во : НГТУ, 2012. – 22 с.
9. Анисимов, В. В. Интеллектуальные информационные системы. Электронный ресурс. Доступ свободный. Время доступа 30.05.2021. URL: https://www.sites.google.com/site/anisimovkhv/learning/iis.
10. Хабаров, С.П. Интеллектуальные информационные системы. PROLOG – язык разработки интеллектуальных и экспертных систем: учебное пособие / С. П. Хабаров. – СПб. : Изд-во : СПб ГЛТУ, 2013. – 138 с.

11. Гаврилова, Т. А. Инженерия знаний. Модели и методы : учебник для вузов / Гаврилова Т. А., Кудрявцев Д. В., Муромцев Д. И. – СПб. : Изд-во : «Лань», 2016. – 324 с.
12. Conceptual Structures Home Page. Electronic resource. Free access. Access time 30.05.2021. URL: http://conceptualstructures.org.
13. CharGer – Conceptual Graph Editor 3.6 User's Guide 58 p. By Harry S. Delugach (The University of Alabama in Huntsville, Huntsville, AL, USA). Electronic resource. Free access. Access time 30.05.2021. URL: https://charger-conceptual-graph-editor.soft112.com.
14. Harry S. Delugach. Implementation and Visualization of Conceptual Graphs in CharGer // June 2014 International Journal of Conceptual Structures and Smart Applications 2(2):1-19 DOI:10.4018/IJCSSA.2014070101.
15. Филиппов, А. Н. Виртуальное строковое пространство технологических данных и знаний : учебное пособие для вузов / А. Н. Филиппов. – СПб : Изд-во : СПб ГУ ИТМО, 2015. – 81 с.
16. Wielemaker, J. SWI Prolog. Reference Manual. August 2015. Univ. of Amsterdam, the Netherlands. – 533 p. Electronic resource. Free access. Access time 30.05.2021. URL: http://www.swi-prolog.org.
17. Муромцев, Д. И. Введение в технологию экспертных систем : учебное пособие для вузов / Д. И. Муромцев. – СПб : Изд-во : СПб ГУ ИТМО, 2005. – 93 с.
18. Муромцев, Д. И. Онтологический инжиниринг знаний в системе Protégé : учебное пособие для вузов / Д. И. Муромцев. – СПб : Изд-во : СПб ГУ ИТМО, 2007. – 62 с.
19. Муромцев, Д. И. Концептуальное моделирование знаний в системе Concept Map : учебное пособие для вузов / Д. И. Муромцев. – СПб : Изд-во : СПб ГУ ИТМО, 2009. – 83 с.
20. Sowa J. F. Conceptual Graphs for a Data Base Interface // IBM J. Res. Develop. Vol. 20, July 1976. – P. 336–357.

21. Чери С., Готлоб Г., Танка Л. Логическое программирование и базы данных. Изд-во : М. : Мир, 1992. – 353 с.
22. Косяков, М. С. Введение в распределенные вычисления : учебное пособие для вузов / М. С. Косяков. – СПб : Изд-во : СПб ГУ ИТМО, 2014. – 155 с.
23. Бабичев, С. Л. Распределенные системы : учебное пособие для вузов / С. Л. Бабичев, К. А. Коньков. – М.: Изд-во Юрайт, 2019. – 507 с.
24. Шолле, Франсуа. Глубокое обучение на Python. – СПб. : Изд-во : «Питер», 2018. – 400 с.
25. Рашид, Тарик. Создаем нейронную сеть. – СПб. : Изд-во : ООО «Альфа-книга», 2017. – 272 с.
26. Гифт, Ной. Прагматичный искусственный интеллект. Машинное обучение и облачные технологии. – СПб. : Изд-во : «Питер», 2019. – 304 с.
27. Джоши, Пратик. Искусственный интеллект с примерами на Python. – СПб. : Изд-во : ООО «Диалектика», 2019. – 448 с.
28. Федоров, Д. Ю. Программирование на языке высокого уровня Python : учеб. пособие для прикладного бакалавриата / Д. Ю. Федоров. – М. : Изд-во : «Юрайт», 2019. – 161 с.
29. AI Chat Bot in Python with AIML. Electronic resource. Free access. Access time 05/30/2021. URL: https://www.devdungeon.com/content/ai-chat-bot-python-aiml/.
30. Мамиконов А. Г., Кульба В. В. Синтез оптимальных модульных систем обработки данных. – М.: Наука, 1986. – 276 с.
31. Карамышева, Н. С. Эволюционирующие агентно-ориентированные сети функциональных операторов / Н. С. Карамышева, В. Б. Механов, С. А. Зинкин // Информатизация образования и науки. – № 4. – 2013. – С. 95–112.
32. Карамышева, Н. С. Схематология фреймовых моделей представления знаний и синтез иерархических фреймовых структур / Н. С. Карамышева,

Н. П. Вашкевич // Актуальные проблемы образования: материалы XX научно-практической конференции. – Пенза, Пенз. гос. ун.-т., 2009. – С. 93–101.

33. Построение экспертных систем / Пер. с англ.; Под ред. Ф. Хейеса-Рота, Д. Уотермена, Д. Лената. –М.: Мир, 1987. – 441с.
34. Экспертные системы для персональных компьютеров: Методы, средства реализации: Справ. пособие / В. С. Крисевич и др. – Минск: Высш. шк., 1990. – 197 с.
35. Уотермен, Д. Руководство по экспертным системам / Пер. с англ. – М.: Мир, 1989. – 388 с.
36. Зинкин, С. А. Развитие информационно-коммуникационных инфраструктур распределенных вычислительных систем на основе концепции «Сеть – это компьютер» / С. А. Зинкин, М. С. Джафар // В журнале: «Известия Юго-Западного государственного университета». – 2018. – Т. 22. – №4 (79). – С. 75-93.
37. Зинкин, С. А. Интеграция методов концептуального и поведенческого моделирования дискретно-событийных систем: I. Синтез и анализ концептуальной модели / С. А. Зинкин, Д. В. Пащенко, У. Н. Пучкова, М. С. Джафар // В журнале: «Кибернетика и программирование», 2016, № 6. – С. 83-95.
38. Зинкин, С. А. Интеграция методов концептуального и поведенческого моделирования дискретно-событийных систем: II. Логико-алгебраические операционные модели и инфокоммуникационные технологии / С. А. Зинкин, Д. В. Пащенко, У. Н. Пучкова, М. С. Джафар // В журнале: «Кибернетика и программирование», 2017, № 1. – С. 75-93.
39. Зинкин, С. А. Организация функционирования распределенных вычислительных систем с переменной архитектурой в виде облачного сервиса, формируемого по запросу клиента (реализация изменяемой системной архитектуры) / С. А. Зинкин, М. С. Джафар, Н. С. Карамышева

// В журнале: «XXI век: итоги прошлого и проблемы настоящего *плюс*». – 2018. – Т. 7. – № 4 (44). – С. 54-60.

40. Зинкин, С. А. Организация функционирования распределенных вычислительных систем с переменной архитектурой в виде облачного сервиса, формируемого по запросу клиента (концептуальные графы распределенных алгоритмов) / С. А. Зинкин, М. С. Джафар, Н. С. Карамышева // В журнале: «XXI век: итоги прошлого и проблемы настоящего *плюс*». – 2018. – Т. 7. – № 4 (44). – С. 136-146.
41. Fedosin, M. E. Supporting tools for research projects in virtual information-computational laboratories / M. E. Fedosin // Modern Informatization Problems in the Technological and Telecommunication Systems Analysis and Synthesis : Proceedings of the XVII-th International Open Science Conference. – Lorman, MS, USA, 2013. – P. 367–371.
42. Зинкин, С. А. Описание запуска вычислительных заданий в веб-центрах на основе логико-алгебраических спецификаций / С. А. Зинкин, М. Е. Федосин, А. В. Савкина // Научно-технический вестник Поволжья. – 2013. – № 4. – С. 154–159.
43. Зинкин, С. А. Разработка программной и аппаратной архитектуры веб-центра с использованием логико-алгебраических моделей представления знаний / С. А. Зинкин, М. Е. Федосин, А. С. Федосин // Научно-технический вестник Поволжья. – 2013. – № 6. – С. 289–293.
44. Федосин, М. Е. Разработка программно-аппаратного комплекса для предоставления доступа к высокопроизводительному программному обеспечению в концепции облачных вычислений / М. Е. Федосин // Международный журнал прикладных и фундаментальных исследований. – 2013. – № 6. – С. 110–111.
45. Complete Overview of Petri Nets Tools Database. URL: http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/complete_db.html
46. About PIPE2. URL: http://pipe2.sourceforge.net/about.html

47. Робототехника и гибкие автоматизированные производства. – В 9 кн. Кн. 6. Техническая имитация интеллекта/ В. М. Назаретов, Д. П. Ким; под ред. И. М. Макарова. – М.: Высш. шк., 1986. – 144 с.
48. Многоуровневое структурное проектирование программ. Теоретические основы, инструментарий/ Е. Л. Ющенко, Г. Е. Цейтлин, В. П. Грицай, Т. К. Терзян. – М.: Финансы и статистика, 1989. – 208 с.
49. Gurevich Y. Abstract State Machines: An Overview of the Project // Foundations of Information and Knowledge Systems. Lect. Notes Comput. Sci. 2004. Vol. 2942. – P. 6-13.

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ ИНФОКОММУНИКАЦИОННЫЕ СИСТЕМЫ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

Рекомендации по выполнению лабораторного практикума
дисциплины Инфокоммуникации

Задание 1 " Моделирование и исследование механизмов поддержки качества обслуживания "

Материалы для выполнения задания по сетям с QoS надо взять с облачного диска ,,,,,,,,, из Папки «!! модели QOS в CPN Tools»:

1. Шаблоны моделей алгоритмов диспетчеризации очередей

DRR_basic_new - АЛГОРИТМ drr классический дефицитный циклический
WRR_basic_new - АЛГОРИТМ wrr классический взвешенный циклический
FPP_3 – АЛГОРИТМ Жестко приоритетный
DTSS - – АЛГОРИТМ с временной селекцией дефицитный
WTSS - – АЛГОРИТМ с временной селекцией взвешенный
wfq2_3 – АЛГОРИТМ справедливый взвешенный
DRR_band - АЛГОРИТМ дефицитный циклический с временной селекцией
WRR_band - АЛГОРИТМ взвешенный циклический с временной селекцией

2. Модель генератора трафика traffic_generator_exp_t_4frame_randdelay
3. Реализация трафика traffic_3335 raffic_3335_2.txt
4. Архив models с примерами результатов моделирования, включая обработку статистики, и примерами оформления отчетов.
5. Методические указания «Технология моделирования алгоритмов диспетчеризации очередей».
6. Электронные копии рекомендуемой литературы

Цель задания:

1. Изучение механизмов организации QoS в коммутаторах Ehternet компьютерных сетей.
2. Получение навыков моделирования телекоммуникационного оборудования с использованием аппарата цветных временных иерархических сетей Петри средствами пакета CPN Tools.

Порядок выполнения задания:

1. Изучить теоретический материал по системе поддержки качесива QOS в компьютерных сетях (см. книгу Олифер В .Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы 3-е изд. - СПб. : Питер, 2007. - 958 с)
2. Изучить по пособию Коннов Н.Н., Механов В.Б., Патунин Д.В. Алгоритмы и модели диспетчеризация очередей в телекоммуникациях (рукопись) принципы и алгоритмы Диспетчеризации очередей в коммутаторах.
3. Научиться работать с пакетом CPT TOOLS (см. пособие Д.А. Зайцев, Т.Р. Шмелева Д.А. Зайцев, Т.Р. Шмелева Моделирование телекоммуникационных систем в CPN Tools - –Одесса: ONAT, 2006. – С. 60). Пакет CPN Tools 4.0.1 можно скачать здесь http://cpntools.org/2018/01/15/windows/).
4. Изучить и настроить в пакете CPT TOOLS модель диспетчера по указанном преподавателем шаблону (см. файл Прочитай!!!). по пособию Коннов Н.Н., Механов В.Б., Патунин Д.В. Алгоритмы и модели диспетчеризация очередей в телекоммуникациях
5. Провести моделирование заданного алгоритма на 1 одном из наборов трафика, которые уже сгенерированы и лежит в отдельных файлах (выбрать по номеру зачетки: чет traffic_3335.txt, нечет traffic_3335_2.txt) или по специально сгенерированному трафику (см. методические указания «Технология моделирования алгоритмов диспетчеризации очередей».
6. Обработать статистику джиттера продвижения заданного трафика через коммутатор с поддержкой Qos (см/ Методические указания «Технология моделирования алгоритмов диспетчеризации очередей»).
7. Подготовить отчет

Задание 2. Мониторинг и анализ протоколов IP телефонии.

Материалы для выполнения задания по IP телефонии надо взять с облачного диска ,,,,,,,,, из Папки «VOIP»:

1. Электронные копии рекомендуемой литературы
2. Дистрибутив программы Wireshark
3. Образцы отчетов

Цель задания:

1. Изучение протоколов IP телефонии
2. Получение навыков настройки и работы с софтфонами и виртуальной АТС Asterisk
3. Получение навыков мониторинга итрафика и анализа работы компьютерной сети с помощью пакета Wireshark

Порядок выполнения задания:

1. Изучить принципы построения сетей IP телефонии на базе протокола SIP (см. Гольдштейн Б.С., Пинчук А.В., Суховицкий А.Л. IP+Телефония. — М.: Радио и связь, 2001. — 336 с)
2. Изучить основы мониторинга и анализа трафика с помощью пакета Wireshark (см. В. К. Конопелько, С. М. Лапшин, В. Ю. Цветков Конопелько, В. К. Измерение и анализ трафика IP-телефонии. – Минск :)БГУИР, 2011. – 56 с)
3. Настроить программы софтфонов и создать SIP пользователя цифровой АТС Asterisk
4. Сделать захват трафика различных сценариев звонков софтфон - софтфонов и выполнить его анализ с помощью пакета Wireshark (см. П.Н. Толмачев, Н.А. Ермакова, П.В. Подворный, С.А. Сапсай Анализатор протоколов Wireshark/: Учебно-методическое пособие для выполнения лабораторных работ. – М.: РУТ(МИИТ), 2016. – 38 с.)
5. Офотмить отчет (графическая схемы взаимодействия софтфонов и АТС, описание зарегистрированных SIP сообщений, Примеры форматов пакетов протоколов SIP и RTP)

Цель работы

Исследовать стек протоколов, используемый в VoIP телефонии. Обработать статистику джиттера.

Задание

Осуществить телефонный разговор между двумя различными машинами с использованием общего SIP сервера, перехватить передаваемые пакеты данных и проанализировать используемые при передаче голоса протоколы.

Анализ сетевой инфраструктуры

Компьютеры, с которых осуществлялся звонок, находятся в одной локальной сети, причем физическое соединение установлено через общий маршрутизатор — этот фактор позволяет заявить, что задержки от маршрутизации будут практически исключены. Учитывая тот факт, что на момент осуществления звонка в сети было активно не более 3х компьютеров (включая SIP сервер), то влияние на задержки от сетевой нагрузки также исключается.

Необходимо принять во внимание тот факт, что SIP сервер запущен на виртуальной машине, физически расположенной на компьютере одного из абонентов и имеет общий с ним физический Ethernet интерфейс — это приводит практически минимальной (порядка нескольких миллисекунд) задержке при установлении соединения; этот факт не оказывает влияние на производительность второго абонента, который выполнял перехват трафика VoIP.

Было осуществлено несколько звонков, при которых «захватывающая» сторона являлась вызывающей, вызываемой, удерживаемой, а также сценарий, когда выполнялся вызов по неверному номеру. Эти сценарии будут рассмотрены далее.

Сценарий 1 — вызывающий

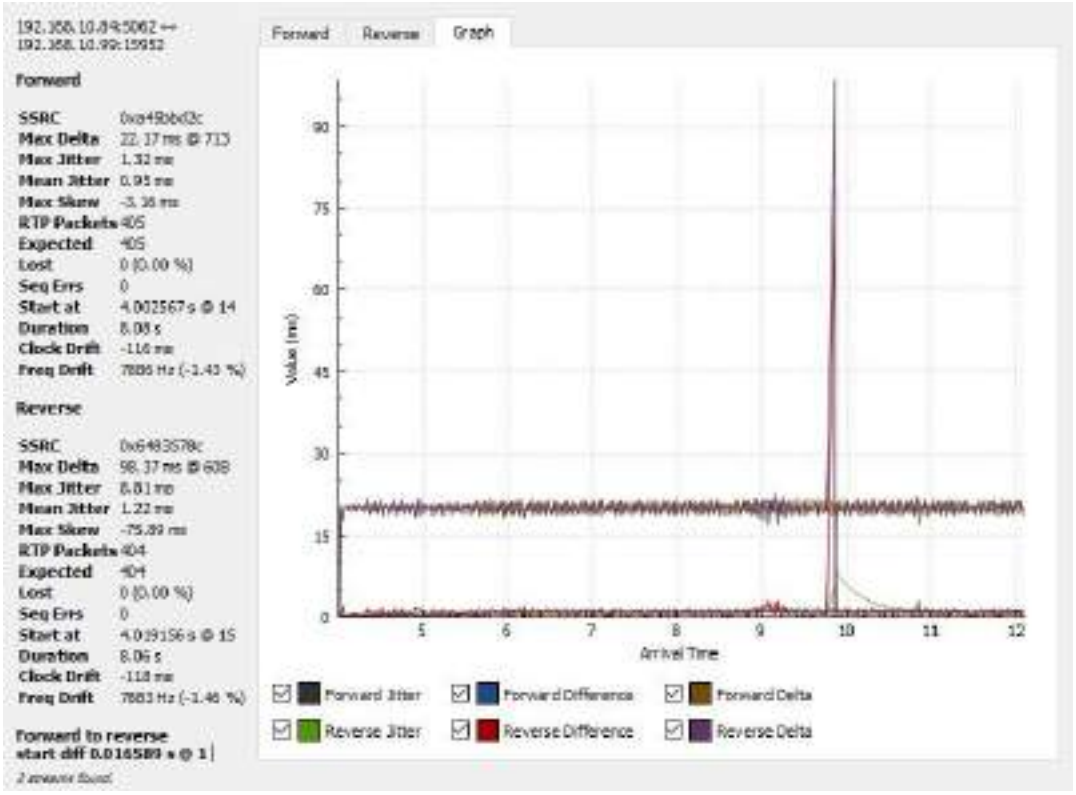
В данном сценарии вызов инициируется стороной, осуществляющей перехват трафика.

Первый пакет, направляемый серверу — это запрос на установление соединения. Он содержит номер вызываемого абонента, данные о вызывающем абоненте, набор поддерживаемых кодеков и другую служебную информацию. Как видно из последовательности обмена пакетами, первый запрос на соединение был отклонен, так как вызывающий не был авторизован. Обычно такая ошибка возникает, если пользователь неправильно ввел логин или пароль, или вовсе не зарегистрирован на сервере; но в данном случае приложение для осуществления звонков PhonerLite попыталось установить анонимное соединение (без проверки пароля), но сервер не поддерживает или запрещает такое. Следом идет попытка авторизоваться с помощью пароля, но без передачи пароля в открытом виде — используется простой протокол challenge-response, где пароль смешивается со случайным значением и серверу передается

результат (хеш) и использованное случайное число. Т.к. пароль в PhonerLite указан корректно, авторизация проходит успешно и сервер статусом Trying (попытка вызова, т.к. вызываемый номер существует), а затем несколькими статусами Ringing (вызывается, т.к. абонент в сети, но не поднимает трубку).

1.193547	5060	INVITE SDP (opus g711A...	5060	SIP INVITE From: "PhonerLite" <sip:124@192.16...
1.195049	5060	401 Unauthorized	5060	SIP Status 401 Unauthorized
1.195474	5060	ACK	5060	SIP ACK From: "PhonerLite" <sip:124@192.168.1...
1.195815	5060	INVITE SDP (opus g711A...	5060	SIP INVITE From: "PhonerLite" <sip:124@192.16...
1.199720	5060	100 Trying	5060	SIP Status 100 Trying
1.255772	5060	180 Ringing	5060	SIP Status 180 Ringing
1.270124	5060	180 Ringing	5060	SIP Status 180 Ringing
3.987708	5060	200 OK SDP (g711U g711...	5060	SIP Status 200 OK
3.989636	5060	ACK	5060	SIP Request INVITE ACK 200 CSeq:19
4.002567	5062	RTP (g711A)	15952	RTP, 405 packets. Duration: 8.080s SSRC: 0xA45...
4.019156	5062	RTP (g711A)	15952	RTP, 404 packets. Duration: 8.058s SSRC: 0x6483...
12.093366	5060	BYE	5060	SIP Request BYE CSeq:20
12.095150	5060	200 OK	5060	SIP Status 200 OK

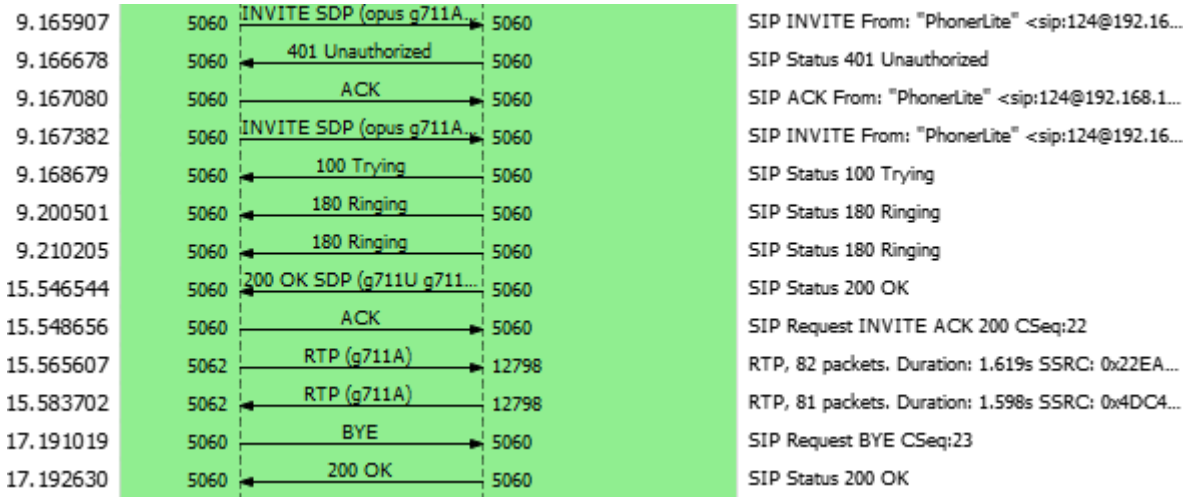
Как только вызываемый абонент берет трубку, сервер отвечает OK и устанавливается соединение по протоколу RTP для передачи звука. Необходимо отметить, что соединение остается клиент-серверным, тем самым IP адреса клиентов остаются скрытыми друг от друга. RTP соединение передает звук, закодированный кодеком Opus G.711 (согласуется по списку, передаваемому вызывающим вы вызываемым абонентами). RTP пакет включает непосредственно закодированные звуковые данные, порядковый номер пакета, относительное время, тип кодека и другую служебную информацию. Используя эти данные, а также время перехвата пакета можно собрать статистику джиттера:



\

Сценарий 1.1 — вызывающий с согласованием

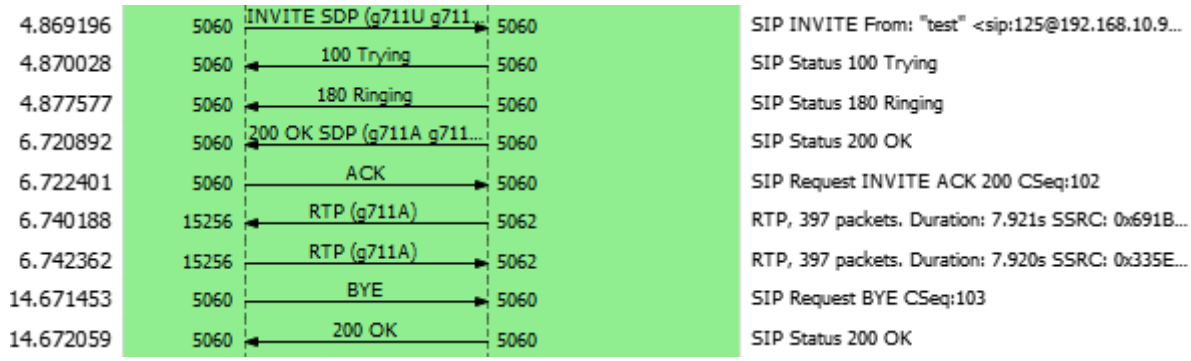
При указании TCP в качестве предпочитаемого протокола соединение осуществляется в той же последовательности. Конечный протокол, поверх которого работает RTP остается UDP, т.к. второй абонент не установил TCP в качестве предпочитаемого.



Сценарий 2 — вызываемый

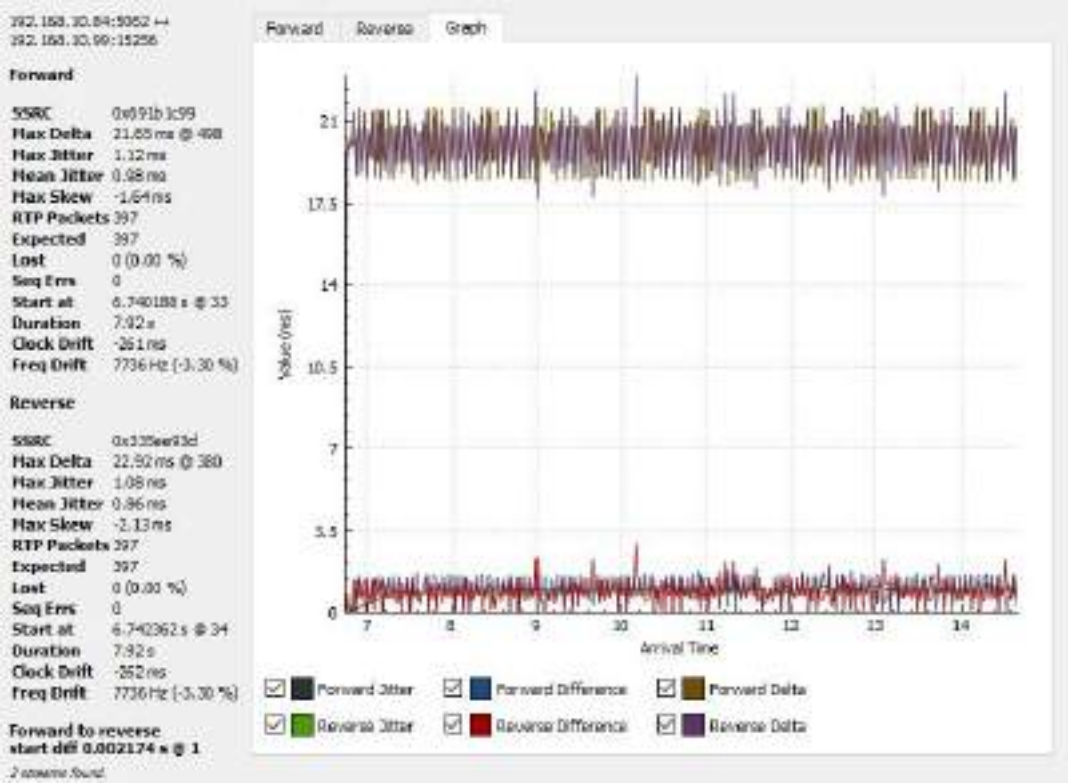
В данном сценарии перехватывающей стороне поступает входящий вызов. Сервер отправляет запрос на установление соединения вызываемому абоненту, который содержит номер вызывающего абонента, список кодеков и другую служебную информацию. В данном случае авторизация вызываемого абонента не осуществляется.

Вызываемый абонент отвечает серверу запросами Trying и Ringing, затем устанавливается RTP соединение.



Показатели задержки в данном сценарии практически идентичны сценарию 1, за исключением отсутствующей однократной задержки в 100мс.

\



Сценарий 3 — удерживаемый

В данном сценарии осуществляется «удержание» линии вызывающим абонентом. Момент начала удержания можно отследить по изменению номера порта, принимающего пакеты (11522) — команды по управлению каналами передачи данных передаются независимо от VoIP данных.

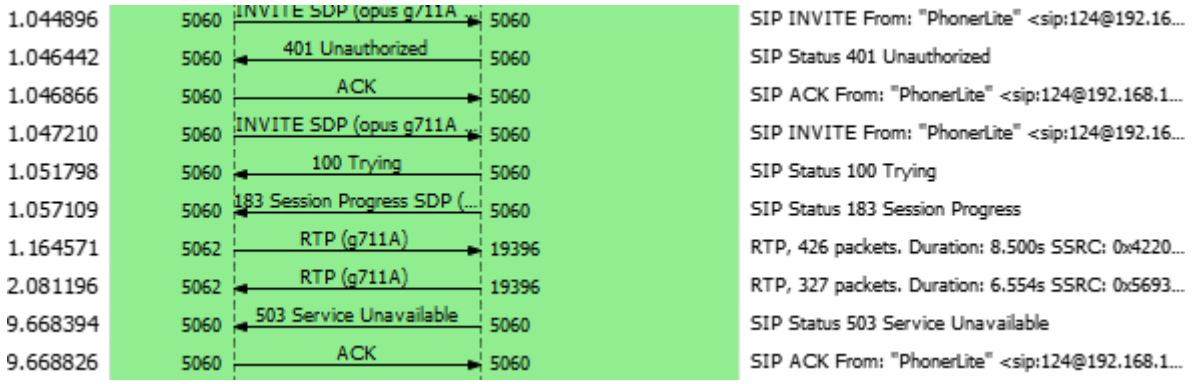
0.000000	5060	INVITE SDP (opus g711A...	5060	SIP INVITE From: "PhonerLite" <sip:124@192.16...
0.001374	5060	401 Unauthorized	5060	SIP Status 401 Unauthorized
0.001543	5060	ACK	5060	SIP ACK From: "PhonerLite" <sip:124@192.168.1...
0.001678	5060	INVITE SDP (opus g711A...	5060	SIP INVITE From: "PhonerLite" <sip:124@192.16...
0.005829	5060	100 Trying	5060	SIP Status 100 Trying
0.074226	5060	180 Ringing	5060	SIP Status 180 Ringing
0.083319	5060	180 Ringing	5060	SIP Status 180 Ringing
1.860535	5060	200 OK SDP (g711U g711...	5060	SIP Status 200 OK
1.862702	5060	ACK	5060	SIP Request INVITE ACK 200 CSeq:48
1.869935	5062	RTP (g711A)	11522	RTP, 148 packets. Duration: 2.939s SSRC: 0xD7A...
1.874447	5062	RTP (g711A)	11522	RTP, 148 packets. Duration: 2.940s SSRC: 0x6085...
4.822951	5060	INVITE SDP (opus g711A...	5060	SIP INVITE From: "PhonerLite" <sip:124@192.16...
4.824503	5060	100 Trying	5060	SIP Status 100 Trying
4.825443	5060	200 OK SDP (g711U g711...	5060	SIP Status 200 OK
4.826051	5060	ACK	5060	SIP Request INVITE ACK 200 CSeq:49
4.829561	5062	RTP (g711A)	11522	RTP, 152 packets. Duration: 3.020s SSRC: 0xD7A...
7.847328	5060	INVITE SDP (opus g711A...	5060	SIP INVITE From: "PhonerLite" <sip:124@192.16...
7.849625	5060	100 Trying	5060	SIP Status 100 Trying
7.849782	5060	200 OK SDP (g711U g711...	5060	SIP Status 200 OK
7.850978	5060	ACK	5060	SIP Request INVITE ACK 200 CSeq:50
7.855524	5062	RTP (g711A)	11522	RTP, 192 packets. Duration: 3.820s SSRC: 0x6085...
7.870248	5062	RTP (g711A)	11522	RTP, 192 packets. Duration: 3.819s SSRC: 0xD7A...
11.693909	5060	BYE	5060	SIP Request BYE CSeq:51
11.695486	5060	200 OK	5060	SIP Status 200 OK

\

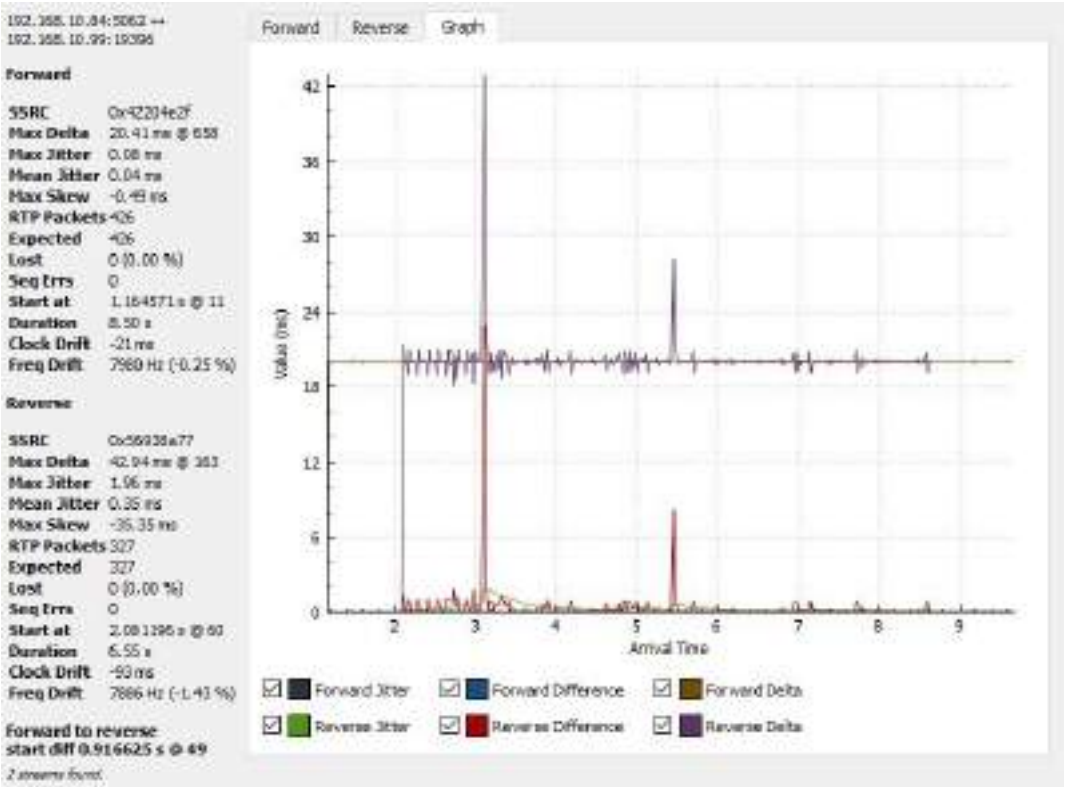
Сценарий 4 — неверный номер

В данном сценарии осуществлен вызов по несуществующему номеру. Результатом является передача сервером подготовленного голосового сообщения о несуществующем номере. Ожидалось, что задержка в данном вызове будет меньше, но т.к. сервер и вызываемый ранее абонент находились на одной физической машине, эта разница незаметна. Сервер ответил статусом «Недоступно».

При



Статистика джиттера со стороны сервера показывает, что было проиграно заранее подготовленное сообщение (джиттер 0.04мс против 0.98мс при реальном вызове).



Выводы

Рассмотрели работу стека протоколов SIP при нескольких сценариях звонков. Результаты перехвата трафика позволяют сделать вывод о характере физической сети, но не о поведении протоколов при потере пакетов данных. Невозможно сделать выводов о влиянии выбора TCP/UDP на работу VoIP.

\

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ МАШИННОЕ ОБУЧЕНИЕ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

1 Введение: задачи обучения по прецедентам

В этой вводной лекции даются базовые понятия и обозначения, которые будут использоваться на протяжении всего курса. Приводятся общие постановки задач обучения по прецедентам и некоторые примеры прикладных задач.

§1.1 Основные понятия и определения

Задано множество *объектов* X, множество *допустимых ответов* Y, и существует *целевая функция* (target function) $y^* : X \to Y$, значения которой $y_i = y^*(x_i)$ известны только на конечном подмножестве объектов $\{x_1, \ldots, x_\ell\} \subset X$. Пары «объект–ответk» (x_i, y_i) называются *прецедентами*. Совокупность пар $X^\ell = (x_i, y_i)_{i=1}^\ell$ называется *обучающей выборкой* (training sample).

Задача *обучения по прецедентам* заключается в том, чтобы по выборке X^ℓ *восстановить зависимость* y^*, то есть построить *решающую функцию* (decision function) $a: X \to Y$, которая приближала бы целевую функцию $y^*(x)$, причём не только на объектах обучающей выборки, но и на всём множестве X.

Решающая функция a должна допускать эффективную компьютерную реализацию; по этой причине будем называть её *алгоритмом*.

1.1.1 Объекты и признаки

Признак (feature) f объекта x — это результат измерения некоторой характеристики объекта. Формально признаком называется отображение $f : X \to D_f$, где D_f — множество допустимых значений признака. В частности, любой алгоритм $a : X \to Y$ также можно рассматривать как признак.

В зависимости от природы множества D_f признаки делятся на несколько типов.

Если $D_f = \{0, 1\}$, то f — *бинарный* признак;

Если D_f — конечное множество, то f — *номинальный* признак;

Если D_f — конечное упорядоченное множество, то f — *порядковый* признак;

Если $D_f = \mathbb{R}$, то f — *количественный* признак.

Если все признаки имеют одинаковый тип, $D_{f_1} = \cdots = D_{f_n}$, то исходные данные называются *однородными*, в противном случае — *разнородными*.

Пусть имеется набор признаков $f_1, \ldots, f_n$. Вектор $f_1(x), \ldots, f_n(x)$ называют *признаковым описанием* объекта $x \in X$. В дальнейшем мы не будем различать объекты из X и их признаковые описания, полагая $X = D_{f_1} \times \ldots \times D_{f_n}$. Совокупность признаковых описаний всех объектов выборки X^ℓ, записанную в виде таблицы размера $\ell \times n$, называют *матрицей объектов–признаков*:

$$F = \|f_j(x_i)\|_{\ell \times n} = \begin{pmatrix} f_1(x_1) & \ldots & f_n(x_1) \\ \ldots & \ldots & \ldots \\ f_1(x_\ell) & \ldots & f_n(x_\ell) \end{pmatrix}. \tag{1.1}$$

Матрица объектов–признаков является стандартным и наиболее распространённым способом представления исходных данных в прикладных задачах.

1.1.2 Ответы и типы задач

В зависимости от природы множества допустимых ответов Y задачи обучения по прецедентам делятся на следующие типы.

Если $Y = \{1, \ldots, M\}$ то это задача *классификации* (classification) на M непересекающихся классов. В этом случае всё множество объектов X разбивается на классы $K_y = \{x \in X : y^*(x) = y\}$ и алгоритм $a(x)$ должен давать ответ на вопрос «какому классу принадлежит x?k. В некоторых приложениях классы называют *образами* и говорят о задаче *распознавания образов* (pattern recognition).

Если $Y = \{0, 1\}^M$, то это задача *классификации на M пересекающихся классов*. В простейшем случае эта задача сводится к решению M независимых задач классификации с двумя непересекающимися классами.

Если $Y = \mathbb{R}$, то это задача *восстановления регрессии* (regression estimation).

Задачи *прогнозирования* (forecasting) являются частными случаями классификации или восстановления регрессии, когда $x \in X$ — описание прошлого поведения объекта x, $y \in Y$ — описание некоторых характеристик его будущего поведения.

1.1.3 Модель алгоритмов и метод обучения

Опр. 1.1. *Моделью алгоритмов называется параметрическое семейство отображений $A = \{g(x, \vartheta) \mid \vartheta \in \Theta\}$ где $g : X \times \Theta \to Y$ — некоторая фиксированная функция, Θ — множество допустимых значений параметра ϑ, называемое пространством параметров или пространством поиска (search space).*

Пример 1.1. В задачах с n числовыми признаками $f_j : X \to \mathbb{R}$, $j = 1, \ldots, n$ широко используются *линейные модели* с вектором параметров $\vartheta = (\vartheta_1, \ldots, \vartheta_n) \in \Theta = \mathbb{R}^n$:

$$g(x, \vartheta) = \sum_{j=1} \vartheta_j f_j(x) \quad \text{— для задач восстановления регрессии, } Y = \mathbb{R};$$

$$g(x, \vartheta) = \operatorname{sign} \sum_{j=1} \vartheta_j f_j(x) \quad \text{— для задач классификации, } Y = \{-1, +1\}.$$

Признаками могут быть не только исходные измерения, но и функции от них. В частности, многомерные линейные модели могут использоваться даже в задачах с единственным числовым признаком.

Пример 1.2. Один из классических подходов к аппроксимации функций одной переменной по заданным точкам $(x_i, y_i) \in \mathbb{R}^2$, $i = 1, \ldots, \ell$ заключается в построении *полиномиальной модели*. Если ввести n признаков $f_j(x) = x^{j-1}$, то функция $g(x, \vartheta)$ из примера 1.1 будет определять полином степени $n - 1$ над исходным признаком x.

Процесс подбора оптимального параметра модели ϑ по обучающей выборке X^ℓ называют *настройкой* (fitting) или *обучением* (training, learning)[1] алгоритма $a \in A$.

[1] Согласно английской терминологии алгоритм является обучаемым, учеником (learning machine), а выборка данных — обучающей, учителем (training sample).

Опр. 1.2. *Метод обучения (learning algorithm) — это отображение* $\mu\colon (X \times Y)^\ell \to A$, *которое произвольной конечной выборке* $X^\ell = (x_i, y_i)_{i=1}^\ell$ *ставит в соответствие некоторый алгоритм* $a \in A$. *Говорят также, что метод* μ *строит алгоритм* a *по выборке* X^ℓ. *Метод обучения должен допускать эффективную программную реализацию.*

Итак, в задачах обучения по прецедентам чётко различаются два этапа.

На этапе *обучения* метод μ по выборке X^ℓ строит алгоритм $a = \mu(X^\ell)$.

На этапе *применения* алгоритм a для новых объектов x выдаёт ответы $y = a(x)$.

Этап обучения наиболее сложен. Как правило, он сводится к поиску параметров модели, доставляющих оптимальное значение заданному функционалу качества.

1.1.4 Функционал качества

Опр. 1.3. *Функция потерь (loss function) — это неотрицательная функция* $\mathsf{L}(a, x)$, *характеризующая величину ошибки алгоритма* a *на объекте* x. *Если* $\mathsf{L}(a, x) = 0$, *то ответ* $a(x)$ *называется корректным.*

Опр. 1.4. *Функционал качества алгоритма* a *на выборке* X^ℓ:

$$Q(a, X^\ell) = \frac{1}{\ell} \sum_{i=1}^{\ell} \mathsf{L}(a, x_i). \tag{1.2}$$

Функционал Q называют также функционалом *средних потерь* или *эмпирическим риском* [4], так как он вычисляется по *эмпирическим данным* $(x_i, y_i)_{i=1}^\ell$.

Функция потерь, принимающая только значения 0 и 1, называется *бинарной*. В этом случае $\mathsf{L}(a, x) = 1$ означает, что алгоритм a допускает ошибку на объекте x, а функционал Q называется *частотой ошибок* алгоритма a на выборке X^ℓ.

Наиболее часто используются следующие функции потерь, при $Y \subseteq \mathbb{R}$:

$\mathsf{L}(a, x) = [a(x) = y^*(x)]$ — индикатор ошибки, обычно применяется в задачах классификации[2];

$\mathsf{L}(a, x) = |a(x) - y^*(x)|$ — отклонение от правильного ответа; функционал Q называется *средней ошибкой* алгоритма a на выборке X^ℓ;

$\mathsf{L}(a, x) = (a(x) - y^*(x))^2$ — квадратичная функция потерь; функционал Q называется *средней квадратичной ошибкой* алгоритма a на выборке X^ℓ; обычно применяется в задачах регрессии.

Классический метод обучения, называемый *минимизацией эмпирического риска* (empirical risk minimization, ERM), заключается в том, чтобы найти в заданной модели A алгоритм a, доставляющий минимальное значение функционалу качества Q на заданной обучающей выборке X^ℓ:

$$\mu(X^\ell) = \arg\min_{a \in A} Q(a, X^\ell). \tag{1.3}$$

Пример 1.3. В задаче восстановления регрессии ($Y = \mathbb{R}$) с n числовыми признаками $f_j : X \to \mathbb{R}$, $j = 1, \ldots, n$, и квадратичной функцией потерь метод минимизации эмпирического риска есть ничто иное, как метод наименьших квадратов:

$$\mu(X^\ell) = \arg\min_{\vartheta} \sum_{i=1}^{\ell} \left(g(x_i, \vartheta) - y_i\right)^2.$$

[2] Квадратные скобки переводят логическое значение в число по правилу [ложь] = 0, [истина] = 1.

1.1.5 Вероятностная постановка задачи обучения

В задачах обучения по прецедентам элементы множества X — это не реальные объекты, а лишь доступные данные о них. Данные могут быть *неточными*, поскольку измерения значений признаков $f_j(x)$ и целевой зависимости $y^*(x)$ обычно выполняются с погрешностями. Данные могут быть *неполными*, поскольку измеряются не все мыслимые признаки, а лишь физически доступные для измерения. В результате одному и тому же описанию x могут соответствовать различные объекты и различные ответы. В таком случае $y^*(x)$, строго говоря, не является функцией. Устранить эту некорректность позволяет *вероятностная постановка задачи*.

Вместо существования неизвестной целевой зависимости $y^*(x)$ предположим существование неизвестного вероятностного распределения на множестве $X \times Y$ с плотностью $p(x, y)$, из которого случайно и независимо выбираются ℓ наблюдений $X^\ell = (x_i, y_i)_{i=1}^\ell$. Такие выборки называются *простыми* или *случайными одинаково распределёнными* (independent identically distributed, i.i.d.).

Вероятностная постановка задачи считается более общей, так как функциональную зависимость $y^*(x)$ можно представить в виде вероятностного распределения $p(x, y) = p(x)p(y|x)$, положив $p(y|x) = \delta(y - y^*(x))$, где $\delta(z)$ — дельта-функция.

Принцип максимума правдоподобия. При вероятностной постановке задачи вместо модели алгоритмов $g(x, \vartheta)$, аппроксимирующей неизвестную зависимость $y^*(x)$, задаётся модель совместной плотности распределения объектов и ответов $\phi(x, y, \vartheta)$, аппроксимирующая неизвестную плотность $p(x, y)$. Затем определяется значение параметра ϑ, при котором выборка данных X^ℓ максимально правдоподобна, то есть наилучшим образом согласуется с моделью плотности.

Если наблюдения в выборке X^ℓ независимы, то совместная плотность распределения всех наблюдений равна произведению плотностей $p(x, y)$ в каждом наблюдении: $p(X^\ell) = p\,(x_1, y_1), \ldots, (x_\ell, y_\ell) = p(x_1, y_1) \cdot \ldots \cdot p(x_\ell, y_\ell)$. Подставляя вместо $p(x, y)$ модель плотности $\phi(x, y, \vartheta)$, получаем *функцию правдоподобия* (likelihood):

$$L(\vartheta, X^\ell) = \prod_{i=1}^{\ell} \phi(x_i, y_i, \vartheta).$$

Чем выше значение правдоподобия, тем лучше выборка согласуется с моделью. Значит, нужно искать значение параметра ϑ, при котором значение $L(\vartheta, X^\ell)$ максимально. В математической статистике это называется *принципом максимума правдоподобия*. Его формальные обоснования можно найти, например, в [13].

После того, как значение параметра ϑ найдено, искомый алгоритм $a_\vartheta(x)$ строится по плотности $\phi(x, y, \vartheta)$ несложно.

Связь максимизации правдоподобия с минимизацией эмпирического риска. Вместо максимизации L удобнее минимизировать функционал $-\ln L$, поскольку он аддитивен (имеет вид суммы) по объектам выборки:

$$-\ln L(\vartheta, X^\ell) = -\sum_{i=1}^{\ell} \ln \phi(x_i, y_i, \vartheta) \to \min_{\vartheta}. \tag{1.4}$$

Этот функционал совпадает с функционалом эмпирического риска (1.2), если определить *вероятностную функцию потерь* $\mathsf{L}(a_\vartheta, x) = \mathcal{L} \ln \phi(x, y, \vartheta)$. Такое определение потери вполне естественно — чем хуже пара (x_i, y_i) согласуется с моделью ϕ, тем меньше значение плотности $\phi(x_i, y_i, \vartheta)$ и выше величина потери $\mathsf{L}(a_\vartheta, x)$.

Верно и обратное — для многих функций потерь возможно подобрать модель плотности $\phi(x, y, \vartheta)$ таким образом, чтобы минимизация эмпирического риска была эквивалентна максимизации правдоподобия.

Пример 1.4. Пусть задана модель $g(x, \vartheta)$. Примем дополнительное вероятностное предположение, что ошибки $\varepsilon(x, \vartheta) = g(x, \vartheta) - y^*(x)$ имеют нормальное распределение $\mathsf{N}(\varepsilon; 0, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{\varepsilon^2}{2\sigma^2}\right)$ с нулевым средним и дисперсией σ^2. Тогда модель плотности имеет вид

$$\phi(x, y, \vartheta) = p(x)\phi(y \mid x, \vartheta) = p(x)\mathsf{N}\left(g(x, \vartheta) - y^*(x); 0, \sigma^2\right).$$

Отсюда следует, что вероятностная функция потерь совпадает с квадратичной с точностью до констант C_0 и C_1, не зависящих от параметра ϑ:

$$-\ln \phi(x, y, \vartheta) = -\ln p(x)\mathsf{N}\left(g(x, \vartheta) - y^*(x); 0, \sigma^2\right) = C_0 + C_1\left(g(x, \vartheta) - y^*(x)\right)^2.$$

Таким образом, существуют два родственных подхода к формализации задачи обучения: первый основан на введении функции потерь, второй — на введении вероятностной модели порождения данных. Оба в итоге приводят к схожим (иногда даже в точности одинаковым) оптимизационным задачам. Обучение — это оптимизация.

1.1.6 Проблема переобучения и понятие обобщающей способности

Минимизацию эмпирического риска следует применять с известной долей осторожности. Если минимум функционала $Q(a, X^\ell)$ достигается на алгоритме a, то это ещё не гарантирует, что a будет хорошо приближать целевую зависимость на произвольной *контрольной выборке* $X^k = (x'_i, y'_i)_{i=1}^k$.

Когда качество работы алгоритма на новых объектах, не вошедших в состав обучения, оказывается существенно хуже, чем на обучающей выборке, говорят об эффекте *переобучения* (overtraining) или *переподгонки* (overfitting). При решении практических задач с этим явлением приходится сталкиваться очень часто.

Легко представить себе метод, который минимизирует эмпирический риск до нуля, но при этом абсолютно не способен обучаться. Получив обучающую выборку X^ℓ, он запоминает её и строит алгоритм, который сравнивает предъявляемый объект x с обучающими объектами x_i из X^ℓ. В случае совпадения $x = x_i$ алгоритм выдаёт правильный ответ y_i. Иначе выдаётся произвольный ответ. Эмпирический риск принимает наименьшее возможное значение, равное нулю. Однако этот алгоритм не способен восстановить зависимость вне материала обучения. Отсюда вывод: для успешного обучения необходимо не только запоминать, но и обобщать.

Обобщающая способность (generalization ability) метода μ характеризуется величиной $Q(\mu(X^\ell), X^k)$ при условии, что выборки X^ℓ и X^k являются представительными. Для формализации понятия «представительная выборкаk обычно принимается стандартное предположение, что выборки X^ℓ и X^k — простые, полученные из одного и того же неизвестного вероятностного распределения на множестве X.

Опр. 1.5. *Метод обучения μ называется состоятельным, если при заданных достаточно малых значениях ε и η справедливо неравенство*

$$\mathsf{P}_{X^\ell, X^k} \{ Q(\mu(X^\ell), X^k) > \varepsilon \} < \eta. \tag{1.5}$$

Параметр ε называется точностью, параметр $(1 - \eta)$ — надёжностью.

Допустима также эквивалентная формулировка: для любых простых выборок X^ℓ и X^k оценка $Q(\mu(X^\ell), X^k) \leq \varepsilon$ справедлива с вероятностью не менее $1 - \eta$.

Получение оценок вида (1.5) является фундаментальной проблемой статистической теории обучения. Первые оценки были получены в конце 60-х годов В. Н. Вапником и А. Я. Червоненкисом [5, 6, 7]. В настоящее время статистическая теория развивается очень активно [34], однако для многих практически интересных случаев оценки обобщающей способности либо неизвестны, либо сильно завышены.

Эмпирические оценки обобщающей способности применяются в тех случаях, когда не удаётся воспользоваться теоретическими.

Пусть дана выборка $X^L = (x_i, y_i)_{i=1}^L$. Разобьём её N различными способами на две непересекающиеся подвыборки — обучающую X_n^ℓ длины ℓ и контрольную X_n^k длины $k = L - \ell$. Для каждого разбиения $n = 1, \ldots, N$ построим алгоритм $a_n = \mu(X_n^\ell)$ и вычислим значение $Q_n = Q(a_n, X_n^k)$. Среднее арифметическое значений Q_n по всем разбиениям называется оценкой *скользящего контроля* (cross-validation, CV):

$$\mathrm{CV}(\mu, X^L) = \frac{1}{N} \sum_{n=1}^{N} Q(\mu(X_n^\ell), X_n^k). \tag{1.6}$$

Возможны различные варианты скользящего контроля, отличающиеся способами разбиения выборки X^L [48]. В простейшем варианте разбиения генерируются случайным образом, число N берётся в диапазоне от 20 до 100. Стандартом «де фактоk считается методика *t×q-кратного скользящего контроля* (t×q-fold cross-validation), когда выборка случайным образом разбивается на q блоков равной (или почти равной) длины, каждый блок по очереди становится контрольной выборкой, а объединение всех остальных блоков — обучающей. Выборка X^L по-разному t раз разбивается на q блоков. Итого получается $N = tq$ разбиений. Данная методика даёт более точные оценки за счёт того, что все объекты ровно по t раз встречаются в контроле.

Недостатками скользящего контроля являются: вычислительная неэффективность, высокая дисперсия, неполное использование имеющихся данных для обучения из-за сокращения длины обучающей выборки с L до ℓ.

§1.2 Примеры прикладных задач

Прикладные задачи классификации, регрессии и прогнозирования встречаются в самых разных областях человеческой деятельности, и их число постоянно растёт.

1.2.1 Задачи классификации

Пример 1.5. В задачах *медицинской диагностики* в роли объектов выступают пациенты. Признаки характеризуют результаты обследований, симптомы заболевания

и применявшиеся методы лечения. Примеры бинарных признаков — пол, наличие головной боли, слабости, тошноты, и т. д. Порядковый признак — тяжесть состояния (удовлетворительное, средней тяжести, тяжёлое, крайне тяжёлое). Количественные признаки — возраст, пульс, артериальное давление, содержание гемоглобина в крови, доза препарата, и т. д. Признаковое описание пациента является, по сути дела, формализованной историей болезни. Накопив достаточное количество прецедентов, можно решать различные задачи: классифицировать вид заболевания (*дифференциальная диагностика*); определять наиболее целесообразный способ лечения; предсказывать длительность и исход заболевания; оценивать риск осложнений; находить синдромы — наиболее характерные для данного заболевания совокупности симптомов. Ценность такого рода систем в том, что они способны мгновенно анализировать и обобщать огромное количество прецедентов — возможность, недоступная человеку.

Пример 1.6. Задача *оценивания заёмщиков* решается банками при выдаче кредитов. Потребность в автоматизации процедуры выдачи кредитов впервые возникла в период бума кредитных карт 60-70-х годов в США и других развитых странах. Объектами в данном случае являются заёмщики — физические или юридические лица, претендующие на получение кредита. В случае физических лиц признаковое описание состоит из анкеты, которую заполняет сам заёмщик, и, возможно, дополнительной информации, которую банк собирает о нём из собственных источников. Примеры бинарных признаков: пол, наличие телефона. Номинальные признаки — место проживания, профессия, работодатель. Порядковые признаки — образование, занимаемая должность. Количественные признаки — возраст, стаж работы, доход семьи, размер задолженностей в других банках, сумма кредита. Обучающая выборка составляется из заёмщиков с известной кредитной историей. В простейшем случае принятие решений сводится к классификации заёмщиков на два класса: «хорошихk и «плохихk. Кредиты выдаются только заёмщикам первого класса. В более сложном случае оценивается суммарное число баллов (score) заёмщика, набранных по совокупности информативных признаков. Чем выше оценка, тем более надёжным считается заёмщик. Отсюда и название — *кредитный скоринг* (credit scoring). На стадии обучения производится синтез и отбор информативных признаков и определяется, сколько баллов назначать за каждый признак, чтобы риск принимаемых решений был минимален. Следующая задача — решить, на каких условиях выдавать кредит: определить процентную ставку, срок погашения, и прочие параметры кредитного договора. Эта задача также сводится к обучению по прецедентам.

Пример 1.7. Задача *предсказания ухода клиентов* (churn prediction) возникает у крупных и средних компаний, работающих с большим количеством клиентов, как правило, с физическими лицами. Особенно актуальна эта задача для современных телекоммуникационных компаний. Когда рынок приходит в состояние, близкое к насыщению, основные усилия компаний направляются не на привлечение новых клиентов, а на удержание старых. Для этого необходимо как можно точнее выделить сегмент клиентов, склонных к уходу в ближайшее время. Классификация производится на основе информации, хранящейся у компании: клиентских анкет, данных о частоте пользования услугами компании, составе услуг, тарифных планах, регулярности платежей, и т. д. Наиболее информативны данные о том, что именно изменилось в поведении клиента за последнее время. Поэтому объектами, строго говоря, явля-

ются не сами клиенты, а пары «клиент x_i в момент времени t_ik. Требуется предсказать, уйдёт ли клиент к моменту времени $t_i + \Delta t$. Обучающая выборка формируется из клиентов, о которых доподлинно известно, в какой момент они ушли.

1.2.2 Задачи восстановления регрессии

Пример 1.8. Термин «регрессияk был введён в 1886 году антропологом Фрэнсисом Гальтоном при изучении статистических закономерностей наследственности роста. Повседневный опыт подсказывает, что в среднем рост взрослых детей тем больше, чем выше их родители. Однако Гальтон обнаружил, что сыновья очень высоких отцов часто имеют не столь высокий рост. Он собрал выборку данных по 928 парам отец-сын. Количественно зависимость неплохо описывалась линейной функцией $y = \frac{2}{3}x$, где x — отклонение роста отца от среднего, y — отклонение роста сына от среднего. Гальтон назвал это явление «регрессией к посредственностиk, то есть к среднему значению в популяции. Термин *регрессия* — движение назад — намекал также на нестандартный для того времени ход исследования: сначала были собраны данные, затем по ним угадана модель зависимости, тогда как традиционно поступали наоборот: данные использовались лишь для проверки теоретических моделей. Это был один из первых случаев моделирования, основанного исключительно на данных. Позже термин, возникший в частной прикладной задаче, закрепился за широким классом методов восстановления зависимостей.

Огромное количество регрессионных задач возникает в физических экспериментах, в промышленном производстве, в экономике.

Пример 1.9. Задача *прогнозирования потребительского спроса* решается современными супермаркетами и торговыми розничными сетями. Для эффективного управления торговой сетью необходимо прогнозировать объёмы продаж для каждого товара на заданное число дней вперёд. На основе этих прогнозов осуществляется планирование закупок, управление ассортиментом, формирование ценовой политики, планирование промоакций (рекламных кампаний). Специфика задачи в том, что количество товаров может исчисляться десятками или даже сотнями тысяч. Прогнозирование и принятие решений по каждому товару «вручнуюk просто немыслимо. Исходными данными для прогнозирования являются временны́е ряды цен и объёмов продаж по товарам и по отдельным магазинам. Современные технологии позволяют получать эти данные от кассовых аппаратов и накапливать в едином хранилище данных. Для увеличения точности прогнозов необходимо учитывать различные внешние факторы, влияющие на спрос: рекламные кампании, социально-демографические условия, активность конкурентов, праздники, и даже погодные условия. В зависимости от целей анализа в роли объектов выступают либо товары, либо магазины, либо пары «магазин–товарk. Ещё одна особенность задачи — несимметричность функции потерь. Если прогноз делается с целью планирования закупок, то потери от заниженного прогноза, как правило, существенно выше, чем от завышенного.

Пример 1.10. Задача *предсказания рейтингов* решается интернет-магазинами, особенно книжными, видео и аудио. Приобретая товар, клиент имеет возможность выставить ему рейтинг, например, целое число от 1 до 5. Система использует информацию о всех выставленных рейтингах для *персонализации* предложений. Когда

клиент видит на сайте страницу с описанием товара, ему показывается также ранжированный список схожих товаров, пользующихся популярностью у схожих клиентов. Основная задача — прогнозировать рейтинги товаров, которые данный клиент ещё не приобрёл. Роль матрицы объектов–признаков играет матрица клиентов–товаров, заполненная значениями рейтингов. Как правило, она сильно разрежена и может иметь более 99% пустых ячеек. Фиксированного целевого признака в этой задаче нет. Алгоритм должен предсказывать рейтинги для любых незаполненных ячеек матрицы. Данный тип задач выделяют особо и называют задачами *коллаборативной фильтрации* (collaborative filtering).

О трудности и актуальности этой задачи говорит следующий факт. В октябре 2006 года крупнейшая американская компания Netflix, занимающаяся видеопрокатом через Internet, объявила международный конкурс с призом в 1 миллион долларов тому, кто сможет на 10% улучшить точность прогнозирования рейтингов, по сравнению с системой Netflix Cinematch (см. http://www.netflixprize.com). Примечательно, что прогнозы самой Cinematch были лишь на те же 10% точнее элементарных прогнозов по средним рейтингам фильмов. Компания крайне заинтересована в увеличении точности прогнозов, поскольку около 70% заказов поступают через рекомендующую систему. Конкурс успешно завершился только через два с половиной года.

1.2.3 Задачи ранжирования

Задачи ранжирования (ranking) возникают в области информационного поиска. Результатом поиска по запросу может оказаться настолько длинный список ответов, что пользователь физически не сможет его просмотреть. Поэтому ответы упорядочивают по убыванию *релевантности* — степени их соответствия запросу. Критерий упорядочения в явном виде неизвестен, хотя человек легко отличает более релевантные ответы от менее релевантных или совсем нерелевантных. Обычно для разметки выборки пар «запрос, ответk привлекают команду экспертов (асессоров), чтобы учесть мнения различных людей, которые зачастую противоречат друг другу. Затем решают задачу обучения ранжированию (learning to rank).

Пример 1.11. Задача *ранжирования текстовых документов,* найденных в Интернете по запросу пользователя, решается всеми современными поисковыми машинами. Объектами являются пары «запрос, документk, ответами — оценки релевантности, сделанные асессорами. В зависимости от методологии формирования обучающей выборки оценки асессоров могут быть бинарными (релевантен, не релевантен) или порядковыми (релевантность в баллах). Признаками являются числовые характеристики, вычисляемые по паре «запрос, документk. Текстовые признаки основаны на подсчёте числа вхождений слов запроса в документы. Возможны многочисленные варианты: с учётом синонимов или без, с учётом числа вхождений или без, во всём документе или только в заголовках, и т. д. Ссылочные признаки основаны на подсчёте числа документов, ссылающихся на данный. Кликовые признаки основаны на подсчёте числа обращений к данному документу.

Пример 1.12. Задачу *предсказания рейтингов* из примера 1.10 на практике лучше ставить как задачу ранжирования. Пользователю выдаётся список рекомендаций, поэтому важно обеспечить высокую релевантность относительно небольшого числа

товаров, попадающих в вершину списка. Среднеквадратичная ошибка предсказания рейтингов, которую предлагалось минимизировать в условии конкурса Netflix, в данном случае не является адекватной мерой качества. Заметим, что в этой задаче в роли асессоров выступают все пользователи рекомендательного сервиса. Покупая товар или выставляя оценку, пользователь пополняет обучающую выборку и тем самым способствует улучшению качества сервиса.

1.2.4 Задачи кластеризации

Задачи кластеризации (clustering) отличаются от классификации (classification) тем, что в них не задаются ответы $y_i = y^*(x_i)$. Известны только сами объекты x_i, и требуется разбить выборку на подмножества (кластеры) так, чтобы каждый кластер состоял из схожих объектов, а объекты разных кластеров существенно отличались. Для этого необходимо задавать функцию расстояния на множестве объектов. Число кластеров также может задаваться, но чаще требуется определить и его.

Пример 1.13. Основным инструментом *социологических и маркетинговых исследований* является проведение опросов. Чтобы результаты опроса были объективны, необходимо обеспечить представительность выборки респондентов. С другой стороны, требуется минимизировать стоимость проведения опроса. Поэтому при *планировании опросов* возникает вспомогательная задача: отобрать как можно меньше респондентов, чтобы они образовывали *репрезентативную выборку* , то есть представляли весь спектр общественного мнения. Один из способов это сделать состоит в следующем. Сначала составляются признаковые описания достаточно большого числа точек опроса (это могут быть города, районы, магазины, и т. д.). Для этого используются недорогие способы сбора информации — пробные опросы или фиксация некоторых характеристик самих точек. Затем решается задача кластеризации, и из каждого кластера отбирается по одной представительной точке. Только в отобранном множестве точек производится основной, наиболее ресурсоёмкий, опрос.

Задачи кластеризации, в которых часть объектов (как правило, незначительная) размечена по классам, называются задачами *с частичным обучением* (semi-supervised learning). Считается, что они не сводятся непосредственно к классификации или кластеризации, и для их решения нужны особые методы.

Пример 1.14. Задача *рубрикации текстов* возникает при работе с большими коллекциями текстовых документов. Допустим, имеется некоторый иерархический рубрикатор, разработанный экспертами для данной предметной области (например, для спортивных новостей), или для всех областей (например, универсальный десятичный классификатор УДК). Имеется множество документов, классифицированных по рубрикам вручную. Требуется классифицировать по тем же рубрикам второе множество документов, которое может быть существенно больше первого. Для решения данной задачи используется функция расстояния, сравнивающая тексты по составу терминов. Терминами, как правило, являются специальные понятия предметной области, собственные имена, географические названия, и т. д. Документы считаются схожими, если множества их терминов существенно пересекаются.

1.2.5 Задачи поиска ассоциаций

Задача *поиска ассоциативных правил* (association rule induction) вынесена в отдельный класс и относится к задачам обучения без учителя, хотя имеет много общего с задачей классификации.

Пример 1.15. Задача *анализа рыночных корзин* (market basket analysis) состоит в том, чтобы по данным о покупках товаров в супермаркете (буквально, по чекам) определить, какие товары часто совместно покупаются. Эта информация может быть полезной для оптимизации размещения товаров на полках, планирования рекламных кампаний (промо-акций), управления ассортиментом и ценами. В данной задаче объекты соответствуют чекам, признаки являются бинарными и соответствуют товарам. Единичное значение признака $f_j(x_i) = 1$ означает, что в i-м чеке зафиксирована покупка j-го товара. Задача состоит в том, чтобы выявить все наборы товаров, которые часто покупают вместе. Например, «если куплен хлеб, то с вероятностью 60% будет куплено и молокоk. Во многие учебники по бизнес-аналитике вошёл пример, когда система поиска ассоциативных правил обнаружила неочевидную закономерность: вечером перед выходными днями возрастают совместные продажи памперсов и пива. Разместив дорогие сорта пива рядом с памперсами, менеджеры смогли увеличить продажи в масштабах всей розничной сети, что окупило внедрение системы анализа данных. Позже маркетологи и социологи предложили разумное объяснение данному явлению, однако обнаружено оно было именно путём анализа данных.

Пример 1.16. Задача *выделения терминов* (term extraction) из текстов, решаемая перед задачей рубрикации (см. пример 1.14), может быть сведена к поиску ассоциаций. Терминами считаются отдельные слова или устойчивые словосочетания, которые часто встречаются в небольшом подмножестве документов, и редко — во всех остальных. Множество часто совместно встречающихся терминов образует тему, скорее всего, соответствующую некоторой рубрике.

1.2.6 Методология тестирования обучаемых алгоритмов

Пока ещё не создан универсальный метод обучения по прецедентам, способный решать любые практические задачи одинаково хорошо. Каждый метод имеет свои преимущества, недостатки и границы применимости. На практике приходится проводить численные эксперименты, чтобы понять, какой метод из имеющегося арсенала лучше подходит для конкретной задачи. Обычно для этого методы сравниваются по скользящему контролю (1.6).

Существует два типа экспериментальных исследований, отличающихся целями и методикой проведения.

Эксперименты на модельных данных. Их цель — выявление границ применимости метода обучения; построение примеров удачной и неудачной его работы; понимание, на что влияют параметры метода обучения. Модельные эксперименты часто используются на стадии отладки метода. Модельные выборки сначала генерируются в двумерном пространстве, чтобы работу метода можно было наглядно представить на плоских графиках. Затем исследуется работа метода на многомерных данных, при различном числе признаков. Генерация данных выполняется либо с помощью

датчика случайных чисел по заданным вероятностным распределениям, либо детерминированным образом. Часто генерируется не одна модельная задача, а целая серия, параметризованная таким образом, чтобы среди задач оказались как заведомо «лёгкиеk, так и заведомо «трудныеk; при такой организации эксперимента точнее выявляются границы применимости метода.

Эксперименты на реальных данных. Их цель — либо решение конкретной прикладной задачи, либо выявление «слабых местk и границ применимости конкретного метода. В первом случае фиксируется задача, и к ней применяются многие методы, или, возможно, один и тот же метод при различных значениях параметров. Во втором случае фиксируется метод, и с его помощью решается большое число задач (обычно несколько десятков). Специально для проведения таких экспериментов создаются общедоступные репозитории реальных данных. Наиболее известный — репозиторий UCI (университета Ирвина, Калифорния), доступный по адресу http://archive.ics.uci.edu/ml. Он содержит около двух сотен задач, в основном классификации, из самых разных предметных областей [30].

Полигон алгоритмов классификации. В научных статьях по машинному обучению принято приводить результаты тестирования предложенного нового метода обучения в сравнении с другими методами на представительном наборе задач. Сравнение должно производиться в равных условиях по одинаковой методике; если это скользящий контроль, то при одном и том же множестве разбиений. Несмотря на значительную стандартизацию таких экспериментов, результаты тестирования одних и тех же методов на одних и тех же задачах, полученные разными авторами, всё же могут существенно различаться. Проблема в том, что используются различные реализации методов обучения и методик тестирования, а проведённый кем-то ранее эксперимент практически невозможно воспроизвести во всех деталях. Для решения этой проблемы разработан *Полигон алгоритмов классификации*, доступный по адресу http://poligon.MachineLearning.ru. В этой системе реализована унифицированная расширенная методика тестирования и централизованное хранилище задач. Реализация алгоритмов классификации, наоборот, децентрализована. Любой пользователь Интернет может объявить свой компьютер вычислительным сервером Полигона, реализующим один или несколько методов классификации. Все результаты тестирования сохраняются как готовые отчёты в базе данных системы и могут быть в любой момент выданы по запросу без проведения трудоёмких вычислений заново.

Конкурсы по решению задач анализа данных. В последние годы компании, заинтересованные в решении прикладных задач анализа данных, всё чаще стали обращаться к такой форме привлечения научного сообщества, как открытые конкурсы с денежными премиями для победителя. В каждом таком конкурсе публикуется обучающая выборка с известными ответами, тестовая выборка, ответы на которой известны только организатору конкурса, и критерий, по которому алгоритмы претендентов сравниваются на данных тестовой выборки. Информацию о текущих конкурсах можно найти на сайтах http://www.kaggle.com, http://tunedit.org. Существуют также сайты, на которых можно тестировать различные алгоритмы на различных наборах данных: http://poligon.MachineLearning.ru, http://mlcomp.org.

1.2.7 Приёмы генерации модельных данных

Данный раздел носит справочный характер. В нём перечислены некоторые сведения, полезные при генерации модельных выборок данных.

Моделирование случайных данных. Следующие утверждения позволяют генерировать случайные выборки с заданными распределениями [10]. Будем предполагать, что имеется стандартный способ получать равномерно распределённые на отрезке $[0, 1]$ случайные величины.

Утв. 1. Если случайная величина r равномерно распределена на $[0, 1]$, то случайная величина $\xi = [r < p]$ принимает значение 1 с вероятностью p и значение 0 с вероятностью $1 - p$.

Утв. 2. Если случайная величина r равномерно распределена на $[0, 1]$, и задана возрастающая последовательность $F_0 = 0, F_1, \ldots, F_{k-1}, F_k = 1$, то дискретная случайная величина ξ, определяемая условием $F_{\xi-1} \leq r < F_\xi$, принимает значения $j = 1, \ldots, k$ с вероятностями $p_j = F_j - F_{j-1}$.

Утв. 3. Если случайная величина r равномерно распределена на $[0, 1]$, и задана возрастающая на R функция $F(x)$, $0 \leq F(x) \leq 1$, то случайная величина $\xi = F^{-1}(r)$ имеет непрерывную функцию распределения $F(x)$.

Утв. 4. Если r_1, r_2 — две независимые случайные величины, равномерно распределённые на $[0, 1]$, то *преобразование Бокса-Мюллера*

$$\xi_1 = \sqrt{-2 \ln r_1} \sin 2\pi r_2;$$
$$\xi_2 = \sqrt{-2 \ln r_1} \cos 2\pi r_2;$$

даёт две независимые нормальные случайные величины с нулевым матожиданием и единичной дисперсией: $\xi_1, \xi_2 \in \mathsf{N}(0, 1)$.

Утв. 5. Если ξ — нормальная случайная величина из $\mathsf{N}(0, 1)$, то случайная величина $\eta = \mu + \sigma\xi$ имеет нормальное распределение $\mathsf{N}(\mu, \sigma^2)$ с матожиданием μ и дисперсией σ^2.

Утв. 6. Пусть n-мерный вектор $x = (\xi_1, \ldots, \xi_n)$ составлен из независимых нормальных случайных величин $\xi_i \sim \mathsf{N}(0, 1)$. Пусть V — невырожденная $n \times n$-матрица, $\mu \in \mathsf{R}^n$. Тогда вектор $x' = \mu + V^{\mathsf{T}}x$ имеет многомерное нормальное распределение $\mathsf{N}(\mu, \Sigma)$ с вектором матожидания μ и ковариационной матрицей $\Sigma = V^{\mathsf{T}}V$.

Утв. 7. Пусть на вероятностном пространстве X заданы k плотностей распределения $p_1(x), \ldots, p_k(x)$. Пусть дискретная случайная величина ξ принимает значения $1, \ldots, k$ с вероятностями $w_1, \ldots, w_k$. Тогда случайный элемент $x \in X$, полученный согласно распределению $p_\xi(x)$, подчиняется *смеси распределений* $p(x) = \sum_{j=1}^{k} w_j p_j(x)$. На практике часто используют смеси многомерных нормальных распределений.

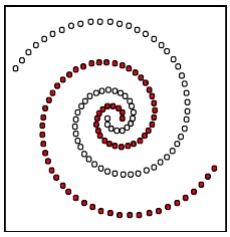


Рис. 1. Модельная выборка «спиралис.

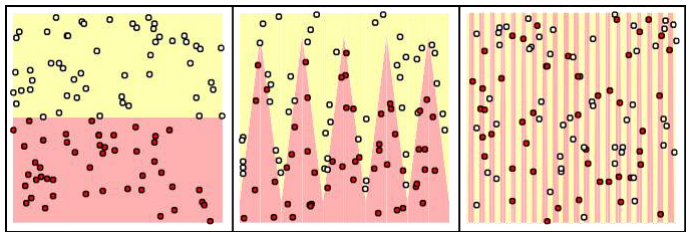


Рис. 2. Серия модельных выборок «пилас.

Утв. 8. Предыдущий случай обобщается на континуальные смеси распределений. Пусть на вероятностном пространстве X задано параметрическое семейство плотностей распределения $p(x, t)$, где $t \in \mathbb{R}$ — параметр. Пусть значение $\tau \in \mathbb{R}$ взято из распределения с плотностью $w(t)$. Тогда случайный элемент $x \in X$, полученный согласно распределению $p(x, \tau)$, подчиняется распределению $p(x) = \int_{-\infty}^{+\infty} w(t)p(x, t)\,dt$. Этот метод, называемый *методом суперпозиций*, позволяет моделировать широкий класс вероятностных распределений, представимых интегралом указанного вида.

Утв. 9. Пусть в $\mathbb{R}^n$ задана прямоугольная область $\Pi = [a_1, b_1] \times \ldots \times [a_n, b_n]$ и произвольное подмножество $G \subset \Pi$. Пусть $r = (r_1, \ldots, r_n)$ — вектор из n независимых случайных величин r_i, равномерно распределённых на $[a_i, b_i]$. *Метод исключения* состоит в том, чтобы генерировать случайный вектор r до тех пор, пока не выполнится условие $r \in G$. Тогда результирующий вектор r равномерно распределён на G. Этот метод вычислительно неэффективен, если объём G много меньше объёма Π.

Неслучайные модельные данные позволяют наглядно продемонстрировать, в каких случаях одни методы работают лучше других.

Один из классических примеров — две спирали на Рис. 1. Эта выборка хорошо классифицируется методом ближайших соседей, но непреодолимо трудна для линейных разделяющих правил. Если витки спиралей расположить ближе друг к другу, задача станет трудна и для метода ближайших соседей. Некоторые кусочно-линейные разделители справляются с задачей и в этом случае.

Обычно при создании модельных данных, как случайных, так и неслучайных, вводится параметр, плавно изменяющий задачу от предельно простой до предельно трудной. Это позволяет исследовать границы применимости метода. На Рис. 2 показана серия модельных задач классификации с двумя классами, обладающая таким свойством относительно метода ближайших соседей и некоторых других алгоритмов.

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ МЕТОДЫ ОПТИМИЗАЦИИ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

Введение

Под математическим программированием понимается совокупность методов решения задач оптимизации, связанных с нахождением наилучшего (с точки зрения выбранных критериев) значения некоторой целевой функции F(X) многих переменных $X=\{x_1, x_2,\ldots, x_n\}$ на заданном множестве точек L(x) евклидова пространства E_n, а также плана $X^*=\{x_1, x_2,\ldots, x_n\}$, при котором достигается соответствующее оптимальное значение $F(X^*\}$. При разработке РЭА проектировщик сталкивается с проблемой получения оптимальной или близкой к оптимальной конструкции устройства. Для решения задач оптимизации необходимо, прежде всего, уметь формулировать критерий оптимальности.

Численные методы программирования определяются видом целевой функции и ограничений математической модели, используемой в задаче. В зависимости от аналитических свойств исследуемой модели различают методы линейного и нелинейного программирования. При решении широкого класса нелинейных дискретных задач применяют методы, в основу которых положен многошаговый поиск (рекуррентный подход), объединяемые под общим названием «динамическое программирование».

Лабораторная работа №1
Симплекс-метод

Цель работы: изучить симплексный метод решения задач линейного программирования, его геометрический и аналитический алгоритмы решения.

Теоретическая часть

В широком классе технических задач показатель качества выражают линейно через параметры проектируемой системы, а условия, которым должны удовлетворять искомые параметры, записывают в виде линейных равенств и неравенств. Оптимизация подобных линейных математических моделей составляют предмет линейного программирования.

Математическую задачу линейного программирования записывают следующим образом.

Определить значение переменных $X^*=\{x_1^*, x_2^*,\ldots, x_n^*\}$, удовлетворяющих системе ограничений

$$\sum_{i=1}^{n} a_{ij} x_i \, R_j \, b_j, \qquad j=1,2,\ldots,m,$$

при которых достигается максимум (минимум) целевой функции

$$F(X) = \sum_{i=1}^{n} c_i x_i ,$$

где R_j – один из знаков =, $\geq$, $\leq$, a_{ij}, c_i, b_j – заданные действительные числа.

Любой упорядоченный набор значений $X=\{x_1, x_2,\ldots, x_n\}$, удовлетворяющий заданной системе ограничений, называется планом задачи. План, которому соответствует максимальное значение целевой функции, называют оптимальным решением (планом) или просто решением задачи линейного программирования.

Для решения задач линейного программирования используется несколько методов. Наиболее эффективным является метод, разработанный в 1951 году американским математиком Дж. Данцигом (симплекс-метод).

Симплекс-метод позволяет, отправляясь от известного опорного плана задачи, за конечное число шагов получить ее решение (оптимальный план). Каждый из этих шагов или итераций состоит в нахождении нового плана, которому соответствует большее значение линейной формы, чем значение этой же формы при предшествующем плане. Процесс повторяется до тех пор, пока не будет получен оптимальный план, т.е. не будет достигнут максимум целевой функции.

Все задачи на поиск экстремума полилинейной функции могут быть сведены к единой канонической форме, т.е. все переменные должны быть положительными, а ограничения иметь вид равенств

$$\begin{cases} a_{11}x_1+a_{12}x_2+\ldots.+a_{1n}x_n=b_1 \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ a_{m1}x_m+a_{m2}x_2+\ldots.+a_{mn}x_n=b_m \end{cases}$$

где $x_i \geq 0$ (i=1,…,n).

Строится целевая функция $F=a^0{}_1x_1+a^0{}_2x_2+\ldots+a^0{}_nx_n$. Требуется найти максимум или минимум этой функции в зависимости от заданных ограничений.

Симплекс-метод гарантирует окончание процесса поиска оптимального решения задачи линейного программирования за конечное число шагов, исходя из геометрической интерпретации симплекс-метода.

Геометрическая интерпретация метода

Дано:

Целевая функция $F(x_1, x_2,\ldots, x_n)$;

Ограничения, заданные в виде системы неравенств

$$\begin{cases} a_{11}x_1+a_{12}x_2+\ldots+a_{1n}x_nR_jb_1 \\ \ldots\ldots\ldots\ldots\ldots \\ a_{m1}x_m+a_{m2}x_2+\ldots+a_{mn}x_nR_jb_m \end{cases}$$

где R_j – один из знаков =, $\geq$, $\leq$

Найти: оптимальное решение целевой функции $F(x_1, x_2,\ldots, x_n)$, при заданных ограничениях.

Решение:

Каждому неравенству в системе ограничений соответствует полуплоскость, ограниченная соответствующей прямой.

Пересечением всех полуплоскостей является область D, которая и будет решением системы неравенств, называемой *областью допустимых значений*.

Необходимо найти максимальное (минимальной) значение целевой функции F, удовлетворяющей системе ограничений. Для этого строится целевая функция F, проходящая через начало координат, которая смещается по нормали (в направлении градиента функции или в направлении возрастания функции).

Пересечения этой прямой с областью D в ее вершинах будут решениями (планами) задачи. Если направление градиента указывает в сторону области допустимых значений, то первое пересечение с вершинами области будет являться минимумом целевой функции при заданных ограничениях и будет называться *опорным планом*, а прямая проходящая через эту точку называется *опорной прямой*. Вершина в которой функция F принимает максимальное значение, называется *оптимальным планом*. Т.е. движение целевой функции необходимо производить в направлении возрастания функции (в направлении градиента функции), первая точка области допустимых значений будет являться минимумом, а последняя – максимумом целевой функции при заданных ограничениях.

Пример:

Дано:

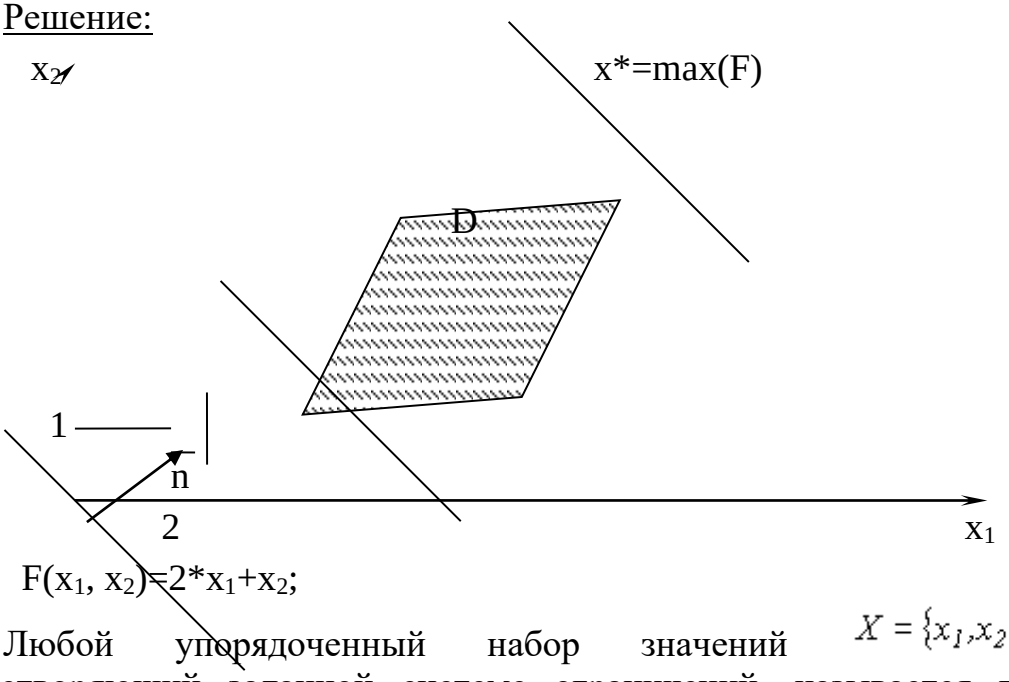
Целевая функция $F(x_1, x_2)=2*x_1+x_2$;

Система ограничений

$$\begin{cases} 3x_1+x_2\leq 2 \\ 6x_1+x_2\geq 1 \\ x_1-2x_2\leq 5 \\ 10x_1-x_2\geq 1 \end{cases}$$

Найти: максимум целевой функции $F(x_1, x_2)$ при заданных ограничениях (рис. 1).

Решение:



$F(x_1, x_2)=2*x_1+x_2;$

Любой упорядоченный набор значений $X = \{x_1, x_2, \ldots, x_n\}$, удовлетворяющий заданной системе ограничений, называется планом задачи. План, которому соответствует максимальное (минимальное) значение целевой функции, называют оптимальным решением (планом) или просто решением задачи линейного программирования.

Аналитическое решение задачи симплекс-методом

Различают 3 формы постановки задачи линейного программирования:

Стандартная форма

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \le b_1$$

.....

$$a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \le b_m$$

$$x_1, x_2, \ldots, x_n \ge 0$$

В векторной форме запись стандартной задачи будет иметь следующий вид:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{bmatrix}, \quad C = \begin{bmatrix} c_1 & c_2 & \ldots & c_n \end{bmatrix}, \quad B = \begin{bmatrix} b_1 \\ b_2 \\ \ldots \\ b_m \end{bmatrix}, \quad A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{m1} & a_{m1} & \ldots & a_{m1} \end{bmatrix}$$

$f(x) = CX$

$AX \le B$

$X \ge 0$

2. Каноническая форма

В канонической форме целевая функция стремиться к min, а ограничения представляют собой равенства. Именно эта форма используется при решении задачи геометрическим и аналитическим методами, так как решение задачи линейного программирования лежит на границе области допустимых решений

$$f(x) = CX \quad \rightarrow \min$$
$$AX = B$$
$$X \ge 0$$

3. Общая форма

В общей форме задачи линейного программирования часть ограничений записывается в виде равенств, а часть в виде неравенств, кроме того условие неотрицательности накладывается не на все искомые переменные.

$$f(x) = c_1x_1 + c_2x_2 + \ldots + c_nx_n \rightarrow \max(\min)$$

$a_{11}x_1+\ldots+a_{1n}x_n \le b_1$

$a_{k1}x_1+\ldots+a_{kn}x_n \le b_k$

$a_{k+11}x_1+\ldots+a_{k+1n}x_n = b_{k+1}$

$a_{m1}x_1+\ldots+a_{mn}x_n = b_m$

$x_r \ge 0$

$r < n$

$k < m$

Все три формы эквивалентны в том смысле, что каждую из них можно преобразовать к любой из двух остальных. Поэтому имея алгоритм решения для канонической формы, можно получить решение любой задачи, приведя её к этой форме.

Для решения задач линейного программирования используется несколько методов. Наиболее эффективным является метод, разработанный в 1951 году американским математиком Дж. Данцигом (симплекс–метод).

Симплекс-метод позволяет, отправляясь от известного опорного плана задачи, за конечное число шагов получить ее решение (оптимальный план). Каждый из этих шагов или итераций, состоит в нахождении нового плана, которому соответствует большее значение целевой функции, по сравнению со значением целевой функции при предшествующем плане.

Процесс повторяется до тех пор, пока не будет получен оптимальный план, т.е. будет достигнут максимум целевой функции.

Для решения задачи на поиск экстремума полилинейной функции Симплекс-методом математическая модель данной задачи должна быть приведена к канонической форме, т.е. все переменные должны быть положительными, а ограничения иметь вид равенств.

$a_{11}x_1+\ldots+a_{1n}x_n+x_{n+1} = b_1$

$a_{k1}x_1+..+a_{kn}x_n+x_{n+m} = b_m$

где $x_i \geq 0$, $i = \overline{1,n}$.

Строится целевая функция. $f(x) = c_1x_1 + c_2x_2 + \ldots + c_nx_n$

Требуется найти минимум этой функции в зависимости от заданных ограничений.

Симплекс-метод гарантирует окончание процесса поиска оптимального решения задачи линейного программирования за конечное число шагов, исходя из геометрической интерпретации симплекс-метода.

Основные этапы Симплекс-метода

Дано:

1) Целевая функция $F = F(x) = a_1x_1 + \ldots + a_nx_n$;

2) система ограничений

$$\begin{cases} a_{11}x_1 + \cdots + a_{1k}x_k \leq b_1 \\ \ldots \\ a_{m1}x_1 + \cdots + a_{mk}x_n \leq b_m \end{cases}$$

. где $x_i \geq 0$, $i = \overline{1,n}$.

Найти:

оптимальное решение, удовлетворяющее системе ограничений, при котором значение функции F(x) принимает максимальное значение.

Решение:

1. Система неравенств преобразуется в систему равенств путем введения дополнительных переменных в каждое из m уравнение.

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1k}x_k + x_{n+1} = b_1; \\ \ldots \\ a_{1m}x_1 + a_{2m}x_2 + \cdots + a_{mk}x_n + x_{n+m} = b_m. \end{cases} \quad (5)$$

2. Выражение целевой функции приводится к виду ,пригодному для решения данной задачи на минимум..

$$F(x) = x_0 = c_1 x_1 + \ldots + c_n x_n \; c_1 x_1 + \ldots + c_n x_n \Rightarrow$$

$$\Rightarrow x_0 - (c_1 x_1 + \ldots + c_n x_n) = 0 \Rightarrow x_0 - c_1 x_1 - \ldots - c_n x_n = 0$$

3. Создается первое базисное решение

$$\{x_{n+1} = b_1, \ldots, x_{n+m} = b_{m1}, \quad x_1 = \ldots = x_n = 0\}$$

где:

$x_1, \ldots, x_n$ - свободные переменные, переменные,

$x_{n+1}, \ldots, x_{n+m}$ - - базисные переменные, относительно которых разрешается эта система уравнений.

При данном выборе базисного решения целевая функция принимает нулевое значение.

4. Для нахождения оптимального решения необходимо составить симплекс-таблицу (таб 1), в строке целевой функции которой стоят коэффициенты при χ_j, а в i-х строках таблицы (i=1…m) стоят коэффициенты при χ_j в i-ом уравнении системы ограничений (5).

Критерием оптимальности для данного метода является отсутствие отрицательных элементов в строке целевой функции.

5. Проверяется план на оптимальность. Для этого просматриваем значения в строке целевой функции. Если среди них нет ни одного отрицательного значения, то полученное решение - оптимальное. Иначе выполняется п. 6..

таблица 1

Итерация	Базис	Значения	x_1	...	x_r	...	x_m	x_{m+1}	...	x_s	...	x_n
к	x_1	b_1	1	...	0	...	0	$a_{1,m+1}$	...	$a_{1,s}$	...	$a_{1,n}$
	...	...	...	...	...	...	...	...	...	...	...	...
	x_r	b_r	0	...	1	...	0	$a_{r,m+1}$	...	$a_{r,s}$	...	$a_{r,n}$
	...	...	...	...	...	...	...	...	...	...	...	...
	x_m	b_m	0	...	0	...	1	$a_{m,m+1}$	...	$a_{m,s}$	...	$a_{m,n}$
	z	z_0	0	0	0	0	0	c_{m+1}	...	c_s	...	c_n

6. Выбирается разрешающий столбец. Это столбец, который в строке целевой функции имеет отрицательный коэффициент. Если таких значений несколько, то выбирается коэффициент максимальный по модулю.

7. Выбирается разрешающий элемент в выбранном столбце для определения разрешающей строки. Для этого находят соотношения для всех значений $\alpha_{is} > 0$. Из всех полученных значений выбираем минимальное. Оно определяет разрешающую строку. Элемент, находящийся на пересечении разрешающей строки и разрешающего столбца, называется разрешающим элементом.

8. Переменная, соответствующая разрешающему столбцу переводится в базисные переменные, а базисная переменная соответствующая разрешающей строке в свободные, т.е. происходит смена базиса или переход к следующей вершине области допустимых решений (геометрическая интерпретация).

9. Строится новая симплекс-таблица. Для этого сначала вычисляются элементы разрешающей строки новой симплекс-таблицы, путем деления их на разрешающий элемент.

$$b'_r = b_r / a_{rs}$$

$$a'_{r,j} = a_{r,j} / a_{r,s}$$

Все остальные элементы новой симплекс-таблицы рассчитываются по правилу треугольника, согласно которому два значения из соотношений () берутся в старой симплекс-таблице, а одно из новой.

(6)

Значения b_i, $a_{i,j}$,c_j , берутся из старой симплекс-таблицы, а b'_r и $a'_{r,j}$ из разрешающей строки новой симплекс-таблицы.

Далее переходят к пункту 5.

Пример:

Составим математическую модель следующей задачи:

Фирма производит три вида продуктов А и В ,и С, рынок сбыта которых неограничен. Каждый из продуктов должен быть обработан каждой из машин 1-го,2-го и 3.-го вида. Время обработки в минутах для изделия А – 18 , 6 , 5 мин. для каждой машины соответственно. Время обработки в минутах для изделия В – 15 , 4 , 3 .мин. для каждой машины соответственно. Время обработки в минутах для изделия С – 12 , 8, 3 .мин. для каждой машины соответственно. Время работы машин 1-вой машины - 6 часов в неделю, 2-ой – 3 часа 12 мин.в неделю, 3-ей - 3 часа. в неделю. Прибыль от изделия А – 9$, от изделия В - 10$ от изделия С - 16$. Фирме надо определить недельные нормы выпуска изделий А и В и С, максимизирующие прибыль.

Искомый выпуск изделий А обозначим через χ_1 , изделий В - через χ_2, изделий С - через χ_3. Поскольку имеются ограничения на выделенный предприятию фонд машинного времени для машин каждого вида, переменные χ_1 χ_2 χ_3, , должны удовлетворять следующей системе неравенств.

$$\begin{cases} 18x_1 + 15x_2 + 12x_3 \le 360 \\ 6x_1 + 4x_2 + 8x_3 \le 192 \\ 5x_1 + 3x_2 + 3x_3 \le 180 \end{cases}$$

$$F = 9x_1 + 10x_2 + 16x_3$$

Общая стоимость произведенной предприятием продукции

2. Приводим задачу к канонической форме.

$$F = -9x_1 - 10x_2 - 16x_3 \to \min$$

$$\begin{cases} 18x_1 + 15x_2 + 12x_3 + x_4 = 360 \\ 6x_1 + 4x_2 + 8x_3 + x_5 = 192 \\ 5x_1 + 3x_2 + 3x_3 + x_6 = 180 \end{cases}$$

3. Составляем симплекс таблицу.

Итерация	**Базис**	**Значения**	**x1**	**x2**	**x3**	**x4**	**x5**	**x6**
0	x4	360	18	15	12	1	0	0
	x5	192	6	4	8	0	1	0
	x6	180	5	3	3	0	0	1
	z0	0	-9	-10	-16	0	0	0
1	x4	72	9	9	0	1	-1,5	0
	x3	24	0,75	0,5	1	0	0,125	0
	x6	108	2,75	1,5	0	0	-0,375	1
	z1	384	3	-2	0	0	2	0
2	x2	8	1	1	0	0,1	-0,17	0
	x3	20	0,25	0	1	-0,05	0,21	0
	x6	96	1,25	0	0	-0,15	-0,12	1
	z2	400	5	0	0	0,2	2,34	0

Находим разрешающий столбец : из отрицательных элементов строки z0 выбираем максимальный по модулю. Элемент (-16) определяет разрешающий столбец x3.

Находим разрешающую строку: элементы, находящиеся в столбце "значения", делим на элементы, расположенные в разрешающем столбце. Из полученных значений выбираем минимальное. Оно определяет разрешающую строку. В нашем примере x5- разрешающая. строка.

Находим разрешающий элемент: на пересечении разрешающего столбца и разрешающей строки расположен разрешающий элемент. Для нашего случая - это 8.

Переходим к новому базису: рассчитываем значения элементов разрешающей строки новой Симплекс – таблицы путем деления каждого элемента разрешающей строки на разрешающий элемент.

Остальные значения рассчитываем по приведенным в теории формулам.

Для строки x4: 360-12*24=72; 18-12*0,75=9; 15-12*0,5; 12-12*1=0; 1-12*0=1; 0-12*0,125=-1,5;0-12*0=0.

Для строки x6: 180-3*24=108; 5-3*0,75=2,75; 3-3*0,5=1,5; 3-3*1=0; 0-3*0=0; 0-3*0,125=-0,375; 1-3*0=1.

Для строки z1: 0-(-16)*24=384; -9-(-16)*0,75=3; -10-(-16)*0,5=-2; -16-(-16)*1=0; 0-(-16)*0=0; 0-(-16)*0,125=2; 0-(-16)*0=0;

д) Так как после первой итерации в строке z1 осталось отрицательное значение (это говорит о том, что план не оптимален), переходим ко второй итерации. Снова выполним пункты а), б), в), г),

только расчет ведем относительно значений, полученных на первой итерации.

После второй итерации получили оптимальный план, так как все значения в строке z2 положительные.

Экономическая интерпретация симплекс-таблицы.

Рассмотрим последнюю итерацию:

а) В базис вошли искомые переменные *x2=8* и *x3=20*, отвечающие за выпуск продукции В и С, от реализации которой получим максимальную прибыль *z2=400*. Переменная *x1*, отвечающая за выпуск продукции А, не вошла в базис, а значит производство продукции А нерентабельно для фирмы.

б) Машинное время третьей машины, за которое отвечает переменная *x6*, полностью не израсходовано и осталось в количестве 96 мин..

Порядок выполнения работы

Изучить аналитический симплекс- метод решения задач.

Построить математическую модель задачи и решить ее аналитическим симплекс-методом вручную. .

Ввести исходные данные и решить задачу в Excel с помощью средства поиск решения.

По результатам полученного решения создать отчеты по результатам, по устойчивости и по пределам.

Написать выводы по полученному решению и оформить отчет.

Контрольные вопросы

Сформулируйте постановку задачи линейной оптимизации.

Какие существуют формы задачи линейного программирования.

Дайте определение опорного и оптимального планов.

Определите критерий оптимальности симплекс-метода при решении аналитическим способом.

Объясните аналитический метод решения задачи симплекс-методом.

Назовите области применения метода.

Лабораторная работа №2
Транспортная задача

Цель работы: изучить методы решения транспортной задачи, построения опорных планов, построения оптимального плана методом потенциалов, провести сравнительный анализ этих методов.

Теоретическая часть

Многие классы широко распространенных на практике задач линейного программирования обладают особенностями, выражающимися в специфическом строении матрицы A коэффициентов системы ограничений, которые позволяют существенно упростить метод решения применительно к данной задаче. При этом ускоряется процесс получения оптимального плана. Одним из примеров задачи со специальной структурой матрицы ограничений является транспортная задача. Математическая модель транспортной задачи имеет следующий вид:

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij},$$

$$\sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} = a_j,\ x_{ij} = b_j$$

где $x_i \geq 0$, i=1,…,m, j=1,…,n.

т.е. $$\sum_{i=1}^{m} a_j = \sum_{j=1}^{n} b_j.$$

Для нахождения оптимального решения транспортной задачи необходимы два этапа: нахождение опорного плана и нахождение оптимального плана на основе опорного плана.

Нахождение опорного плана

Дано:

Количество товара (ресурсы) в m пунктах распределения $\{a_i\}$, i=1,…,m;

Потребности в данном товаре в n пунктах потребления $\{b_j\}$, j=1,…,n;

Матрица стоимости перевозки из пунктов распределения в пункты потребления $C=\{c_{ij}\}$, в которой записаны расстояния между i и j пунктами потребления.

Матрица 1

10	12	24	50	12	330
13	22	49	66	32	270
26	27	35	67	63	350
220	170	210	150	200	

Найти: распределить товар из пунктов поставки в пункты потребления оптимальным образом.

Решение:

Строится матрица, удовлетворяющая следующим условиям:

сумма элементов в заполненных клетках i- строки равна a_i;

сумма элементов в заполненных клетках j-столбца равна b_j;

сумма элементов последнего столбца равна сумме элементов последней строки (матрица 2).

Нахождение оптимального плана необходимо начать с построения опорного плана. Рассмотрим два метода построения опорного плана.

Метод «северо-западного угла»

Матрица стоимости в этом методе во внимание не принимается. Клетка в «северо-западном» углу матрицы соответствует переменной x_{ij}. Положим $x_{ij}=\min\{a_1,b_1\}$. Если $x_{11}=a_1$, то означает, что весь запас товара из первого пункта хранения направляется в первый пункт потребления. Значит $x_{ij}=0$ при $j=2,\ldots,n$. Получается матрица 3. Матрица 4 образуется в том случае, если $b_1 \leq a_1$, тогда первый пункт потребления получает требуемое количество товара, поэтому $x_{ij}=0$ при $i=2,\ldots,m$.

Матрица 2 Матрица 3 Матрица 4

				a_1		a_1	0	…	0	0		b_1				a_1-b_1
				a_2						a_2		0				a_2
				…						…		…				…
				a_m						a_m		0				a_m
b_1	b_2	…	b_n			b_1-a_1	b_2	…	b_n			0	b_2	…	b_n	

К подматрице незаполненных клеток применяется тот же прием. Получается одна из четырех матриц (5-8).

Матрица 5 Матрица 6

a_1	0	…	0	0		a_1	0	…	0	0
a_2	0	…	0	0		(b_1-$a_{1)}$)				(a_2-b_1+a_1)
				…						….
				a_m						a_m
(b_1-a_1-b_2)	b_2	…	b_n			0	b_2	…	b_n	

(a_2<b_1-a_1) (a_2>b_1-a_1)

Матрица 7 Матрица 8

b_1	0	…	0	0		b_1	b_2	…	0	(a_1-b_1-b_2)
0	0	…	0	a_2		(b_1-$a_{1)}$)				a_2
….				…						….
0				a_m						a_m
0	(b_2-(a_1-b_1))	…	b_n			0	0	…	b_n	

(b_2<a_1-b_1) (b_2>a_1-b_1)

Построение распределения выполняется до тех пор, пока не будут исчерпаны все пункты поставки и насыщены все пункты потребления.

Полученный план является опорным планом, на основе которого стоится оптимальный.

Пример:

Дано: a_1=6, a_2=8, a_3=3, a_4=9, a_5=14, b_1=10, b_2=18, b_3=7, b_4=5.

$$\sum_{i=1}^{5} a_j = \sum_{j=1}^{4} b_{j.} = 40.$$

Исходная матрица

					$\mathbf{a_i}$
	0	0	0	0	**6**
	0	0	0	0	**8**
	0	0	0	0	**3**
	0	0	0	0	**9**
	0	0	0	0	**14**
$\mathbf{b_j}$	**10**	**18**	**7**	**5**	**40**

Полученный опорный план

					$\mathbf{a_i}$
	6	0	0	0	**6**
	4	4	0	0	**8**
	0	3	0	0	**3**
	0	9	0	0	**9**
	0	2	7	5	**14**
$\mathbf{b_j}$	**10**	**18**	**7**	**5**	**40**

Метод минимальной стоимости

Необходимо найти такой план закрепления потребителей за поставщиками однородного груза, чтобы общие запросы перевозки были минимальны.

Нахождение опорного плана методом минимальной стоимости выполняется следующим образом:

Выбирается незаполненная клетка с минимальной стоимостью c_{ij}.

Сравнивается количества груза в a_i и потребностью b_j.

Если $a_i>b_j$, то в выбранную клетку заносится величина b_j, что означает, что пункт потребления b_j насытился, а в пункте потребления осталось груза (a_i-b_j), j-й столбец вычеркивается из рассмотрения. Начинается просмотр i-й строки в поисках незаполненной клетки с минимальной стоимостью. Это клетка со стоимостью c_{ij} (пусть j=k). Выполняется переход к п.2.

Если $a_i=b_j$, то вычеркивается i-я строка и j- столбец. Это означает, что пункт потребления b_j насыщен, а в пункте потребления a_i нет больше груза. Выполняется переход к п.6.

Если $a_i<b_j$, то в выбранную клетку заносится величина a_i что означает, что весь груз в пункте поставки a_i распределен в пункт потребления b_j. В пункте потребления b_j осталось доставить груз в количестве (b_j-a_i). Вычеркивается из рассмотрения i-я строка. Начинается просмотр j-го столбца в поисках незаполненной клетки с минимальной стоимостью. Это клетка со стоимостью c_{kj} (пусть i=k). Выполняется переход к п.2.

Проверяется, есть ли еще ненасыщенные пункты потребления b_j. Если есть, то выполняется переход к п.1, иначе заканчивается процесс построения опорного плана.

Построение оптимального плана для решения транспортной задачи

После нахождения опорного плана начинается построение оптимального плана методом потенциалов. Проверка плана на оптимальность осуществляется в соответствии со следующим критерием оптимальности.

Критерий оптимальности

Для того, чтобы решение транспортной задачи было оптимальным, необходимо и достаточно, чтобы существовала система чисел T_i и H_j, называемых потенциалами (где i – номера пунктов поставки, j – номера пунктов потребления), удовлетворяющих условиям:

$$T_i+H_j=c_{ij} \text{ – для занятых клеток} \qquad (1)$$

$T_i+H_j \le c_{ij}$ – для свободных клеток (2)

где c_{ij} – стоимость перевозки из i-го пункта поставки в j- пункт потребления;

T_i- потенциал пункта потребления;

H_j- потенциал пункта поставки.

Следствие

Если холя бы для одной из свободных клеток сумма потенциалов превосходит соответствующую стоимость, то план не является оптимальным.

Таблица потенциалов строится в соответствии с формулой (1) и $T_1=0$. Для нашего примера получается следующая таблица потенциалов (табл.5).

T_i	H_j				
	$H_1=10$	$H_2=12$	$H_3=39$	$H_4=71$	$H_5=22$
$T_1=0$	10 200	12 110	24	50	42
$T_2=10$	13	22 60	49 10	66	32 200
$T_3=-4$	26	27	35 200	67 150	63

Для данной таблицы потенциалы находятся следующим образом:

$T_1+H_1=10 \Rightarrow H_1=10$

$T_1+H_2=12 \Rightarrow H_2=12$

$T_2+H_2=22 \Rightarrow T_2=10$

$T_2+H_3=49 \Rightarrow H_3=39$

$T_2+H_5=32 \Rightarrow H_5=22$

$T_3+H_3=35 \Rightarrow T_3=-4$

$T_3+H_4=67 \Rightarrow H_4=71.$

После построения таблицы потенциалов необходимо проверить план на оптимальность по критерию оптимальности, т.е. должны выполняться условия (1) и (2). Если условия не выполняются, то приступаем к построению оптимального плана.

Метод потенциалов

Составляется цикл перераспределения перевозок следующим образом:

Выбираются свободные клетки, для которых не выполняется условие (2) ($T_i+H_j \le c_i$), из этих клеток выбирается клетка с $\max(c_{ij}- (T_i+H_j))$. В случае неоднозначности выбирается клетка у которой меньше $c_{ij,}$

Из выбранной клетки проходим «шахматной ладьей» по занятым клеткам так, чтобы вновь вернуться в исходную клетку, причем после каждого шага надо поворачивать с горизонтали на вертикаль и наоборот. Исходная клетка помечается знаком «+», а потом знак клеток буде чередоваться (табл. 6).

Таблица 6

T_i	H_j				
	H_1=10	H_2=12	H_3=39	H_4=71	H_5=22
T_1=0	10 200	- 110	12 24	+ 50	42
T_2=10	13	+ 60	22 -	49 10	66	32 200
T_3=-4	26	27	+ 200	35 -	67 150	63

Среди вершин с отрицательным знаком выбирается клетка с наименьшей по величине перестановкой. В примере это клетка со стоимостью c_{23} (min{110,150,10}=10).

Перераспределяется груз в выбранной клетке цикла по следующему правилу: в клетках, имеющих знак «+», прибавляется тот груз, который находится в выбранной клетке (т.е. 10 тонн), а в клетках, имеющих знак «-», вычитается этот груз.

После этого перераспределения строится в соответствии с формулой (2) новая таблица потенциалов.

Проверяется план на оптимальность в соответствии по критерию оптимальности. Если план удовлетворяет критерию оптимальности, то он оптимален, иначе выполняется переход к п.1.

В результате выполнения нескольких итераций получается следующий оптимальный план перевозок (табл.7).

Таблица 6

10 150	12 30	24	50 150	42
13 70	22	49	66	32 200
26	27 140	35 210	67	63

Порядок выполнения работы

1. Изучить теоретическую часть.
2. Построить два опорных плана, используя два различных метода.

3. На основе этих планов методом потенциалов найти оптимальные планы и сделать сравнительный анализ результатов.

4. Получить решения транспортной задачи средствами Excel (поиск решения) и сравнить с полученными результатами.

5. Оформить отчет.

Контрольные вопросы

1. Как формулируется транспортная задача и какова область ее применения?

2. Как определяется опорный план и какие методы построения опорного планы существуют?

3. Какие этапы построения оптимального плана используются для решения транспортной задачи?

4. Каков критерий оптимальности для транспортной задачи?

5. В чем заключатся метод потенциалов?

Лабораторная работа №3
Задача линейного назначения

Цель работы: изучить задачу линейного назначения, различные алгоритмы решения этой задачи, сравнить результаты, полученные этими алгоритмами.

Теоретическая часть

Задача линейного назначения является частным случаем транспортной задачи, она применяется в методах и алгоритмах, используемых при конструкторском проектировании РЭА. Чаще всего эта задача встречается при размещении конструктивных элементов на коммутационной плате. Задача линейного назначения формулируется следующим образом. Пусть имеется n вакантных видов работ, на которые претендуют n работников, причем на каждом виде работ может быть использован только один из n претендентов. Эффективность использования i-го работника на j-м виде работ равна c_{ij}. На основании этих данных можно построить квадратную матрицу $\|c_{ij}\|_{n*n}$, в которой каждый элемент i-ой строки соответствует эффективности использования i-го работника на каждом из n видов работ, а в j-м столбце записывается эффективность использования каждого из n работников на j-м виде работ. Требуется так распределить n работников на n вакантных видах работ, чтобы суммарная эффективность их использования была максимальной.

Определим некоторую матрицу $X = \|x_{ij}\|_{n*n}$, в которой x_{ij} =1, если j-й вид работ выполняет i-й работник; $x_{ij} = 0$ - в противном случае.При этом любой план задачи о назначениях однозначно определяет матрицу X, а математическая модель имеет вид:

максимизировать $F = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$, при условии, что$\sum_{i=1}^{n} x_{ij} = 1, j = 1, n$, (на каждом j-м виде работ может использоваться только один работник);

$\sum_{j=1}^{n} x_{ij} = 1, i = 1, n$, (за каждым i-м работником может быть закреплено только одно вакантное место);

$x_{ij} = 0$ или 1 для всех i и j.

Требование целочисленности переменных можно заменить на $x_{ij} > 0$, что позволяет решать эту задачу с помощью венгерского метода и алгоритма.

Венгерский метод и алгоритм

Основная идея этого метода была впервые высказана венгерским математиком Е. Эгервари задолго до возникновения теории линейного программирования.

Суть метода заключается в том, что сначала строится первоначальный допустимый план, который может и не удовлетворить сразу всем условиям задачи. Затем осуществляется переход к новому плану, более близкому к оптимальному. По заданному критерию проверяем план на оптимальность. Если он не удовлетворяет критерию, то опять осуществляем переход к новому плану, более близкому к оптимальному. И так до тех пор, пока не получим оптимальный план.

Универсальность венгерского метода применительно к задаче о назначениях заключается в том, что если при заданных ограничениях линейной модели можно найти максимальное значение целевой функции F, которому соответствует некоторый оптимальный план $C = \{c_{ij}\}$, то этот же план является одновременно и решением задачи минимизации целевой функции

$$F'(C) = \sum_{i=1}^{n}\sum_{j=1}^{n} c'_{ij}\, x_{ij} = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}^{*'}$$

где $c'_{ij} = -c_{ij}$, при тех же линейных ограничениях. Это утверждение вытекает из принципа двойственности задач линейного программирования. Таким образом, венгерский метод можно применять как для поиска максимума, так и минимума целевой функции.

Рассмотрим алгоритм, реализующий этот метод и включающий в себя предварительный этап и не более (n-2) последовательно повторяющихся итераций формирования оптимального набора переменных. Систему нулевых элементов матрицы, обладающую тем свойством, что никакая пара из них не лежит одновременно в одной строке или одном столбце, называют *системой независимых нулей.* Как только число независимых нулей становится равным n, задача назначения считается решенной: оптимальный план определяется местом положения независимых нулей в последней из матриц.

1. Исходная матрица C преобразуется к матрице C^0 в два этапа:

а) В матрице $\| c_{ij} \|_{n*n}$ для каждого столбца находим максимальный

элемент $d_j = \max\{c_{ij}\}, j = 1, n.$

б) Формируем новую матрицу $C = \| c_{ij} \|_{n*n}$,

где $c_{ij} = \max\{c_{ij}\} - c_{ij} = d_j - c_{ij} \geq 0.$

В каждом столбце новой матрицы есть, по крайней мере, один нуль. При этом преобразовании, за счет изменения знака элементов матрицы эффективности, переходим к задаче минимизации целевой функции, оптимальный план которой совпадает с исходной задачей максимизации.

в) Создаем матрицу $C_0 = \| c_{ij}^0 \|_{n*n}$, где $c_{ij}^0 = c_{ij} - t_i = -c_{ij} + d_i - t_i$, $t_i = \min\{c_{ij}\} = \min\{-c_{ij} + \max\{c_{ij}\}\}$.

г) Образуем первоначальную систему независимых нулей. Для этого отыскиваем и помечаем звездочкой произвольный нуль в первом столбце матрицы C_0. Просматриваем элементы второго столбца, и если обнаруживаем нуль, не лежащий в одной строке с ранее отмеченным нулем, то его тоже помечаем звездочкой. Такие операции осуществляем для всех последующих столбцов матрицы. Полученная система помеченных звездочкой нулей является исходной для дальнейшего решения.

Подсчитываем число (k) независимых нулей матрицы C_0. Если k = n, то решение получено, в противном случае переходим к 3.

Столбцы, содержащие нуль со звездочкой (0*), выделим знаком "+".

Проверяем, есть ли среди невыделенных столбцов матрицы хотя бы один нуль. Если да, то переходим к 5, в противном случае к 6.

Проверяем, содержит ли строка с невыделенным нулем также (0*). Если да, то переходим к 7, в противном случае к 8.

Формируем новые невыделенные нули. Для этого среди невыделенных элементов матрицы C_k выбираем минимальный и вычитаем его из элементов, расположенных в невыделенных строках, и прибавляем к элементам, лежащим в выделенных столбцах. Получаемая матрица C_k' является эквивалентом матрицы C_k Переходим к 5.

Найденный невыделенный нуль отмечаем (0'), а содержащую этот нуль строку - знаком "+". Снимаем знак выделения "+" над столбцом, в котором расположен (0*), лежащий в только что выделенной строке. Переходим к 4.

Невыделенный нуль отмечаем штрихом.

Подготавливаем информацию для оценки возможности увеличения числа независимых нулей (0*) в матрице эффективности. Для этого, начиная с (0'), в одной строке с которым нет (0*), осуществляем построение цепочки элементов матрицы C_k по следующему правилу: выбирая исходный (0'), в цепочку включаем (0*), лежащий с (0') в одном столбце (если такой найдется). К нему прибавляем опять (0'), лежащий в одной строке с предшествующим (0*) и т.д. Построение цепочки по

правилу осуществляется однозначно. Число нулей в такой цепочке всегда нечетно, причем (0’) находятся на нечетных местах, а (0*)- на четных. Может получится так, что в одном столбце с исходным (0’) нет (0*). Тогда цепочка нулей получается вырожденной и состоит из одного исходного элемента.

Меняем знаки у нулей в построенной цепочке, причем звездочки у нулей уничтожаем, а штрихи заменяем на звездочки. Так как такая целая цепочка обязательно должна заканчиваться (0’), а число элементов в ней нечетно, то при замене (0’) на (0*) происходит увеличение числа последних на единицу. Возвращаемся к 2.

Пример:

Дано: Исходная матрица эффективности (рис. 2а).

Необходимо: Найти оптимальный план размещения работников по местам работы.

Решение: Определяем максимальный элемент в каждом столбце матрицы С.

$B_1=\max\{c_{i1}\}=5;\ B_2=\max\{c_{i2}\}=8;\ B_3=\max\{c_{i3}\}=12;\ B_4=\max\{c_{i4}\}=11.$

Вычитаем каждый элемент матрицы С из максимального элемента соответствующего столбца. Получаем матрицу С’ (рис. 2б).

Определяем минимальные элементы в каждой из строк матрице С’.

$A'_1=\min\{c'_{1j}\}=0;\ A'_2=\min\{c'_{2j}\}=4;\ A'_3=\min\{c'_{3j}\}=0;\ A'_4=\min\{c'_{4j}\}=4.$

Вычитаем из каждого элемента матрицы С’ минимальный элемент соответствующей строки. Получаем матрицу C_0 (рис. 2в).

Образуем систему независимых нулей. Отмечаем звездочками независимые нули (рис. 2,г). Подсчитываем число (k) полученных независимых нулей в матрице C_0. Так как $k=3<n$, - переходим к коррекции полученного решения. Выделим столбцы, содержащие (0*) (рис. 2д). Остается невыделенный столбец j=3. В этом столбце существует невыделенный нуль. Переходим к пункту 5 алгоритма. Строка с невыделенным нулем содержит также и (0*). Переходим к п.7 алгоритма. Отмечаем невыделенный нуль со штрихом. Получаем матрицу C_0 (рис. 2е). Снимаем знак выделения над вторым столбцом (j=2). Выделяем 3-ю строку знаком "+". Возвращаемся к пункту 4 алгоритма. Во 2-м столбце есть еще один невыделенный нуль, он располагается во второй строке, в которой также имеется (0*). Помечаем этот невыделенный нуль штрихом (0’). Над первым столбцом снимаем знак выделения. Получаем матрицу C_0 (рис. 2ж). Просматриваем нули первого столбца. Элемент $c_{31} = 0$ нельзя выделить, т.к. он находится в ранее выделенной строке. Поэтому выделяем

c_{41}=0, помечая нуль штрихом (0’). В 4-ой строке нет (0*), поэтому выделение нулей заканчивается (рис. 2з).

Строим цепочку нулей, начиная с последнего, выделенного (0’). Это элемент c_{41}. Включаем в цепочку (0*), расположенный в одном столбце с элементом c_{41}. Это элемент c_{21}. Далее ищем и включаем в цепочку (0’), расположенный в одной строке с c_{21} и т.д. В результате образуется цепочка $c_{41} \Rightarrow c_{21} \Rightarrow c_{22} \Rightarrow c_{32} \Rightarrow c_{33}$. (рис. 2и). Изменяем пометки у нулей. Уничтожаем (0*), а (0’) меняем на (0*). Убираем все пометки строк и столбцов матрицы C_0 (рис. 2к). Во втором столбце есть и другой невыделенный нуль, лежащий в 4-й строке. Однако этот нуль находится в строке, которая не содержит (0*). В этом случае цепочка $0' \Rightarrow 0 \Rightarrow 0'$ быстро оборвется, поэтому невыгодно брать этот нуль.

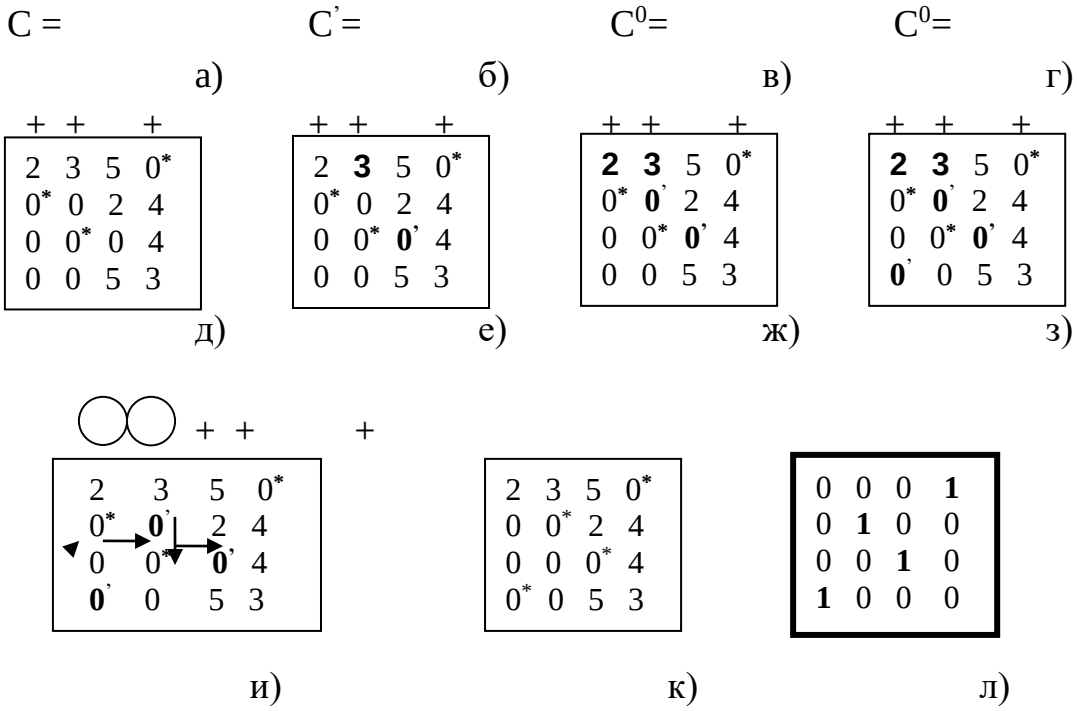


Рисунок 2

Подсчитаем число независимых нулей k=4=n. Задача решена. Получаем оптимальный план (рис2,л):

$$F(x) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}\, x_{ij} = 1 + 4 + 12 + 11 = 28$$

Универсальность этого метода заключается в том, что при заданных ограничениях линейной модели можно найти максимальное значение целевой функции и план $C = \| c_{ij} \|_{n*n}$. Этот же план является одновременно решением задачи *минимизации целевой функции.*

$F'(C\ ) = \sum_{i=1}^{n}\sum_{j=1}^{n} c'_{ij}\, x_{ij} = \sum_{i=1}^{n}\sum_{j=1}^{n} c^{*'}_{ij}$,где $c'_{ij} = -c_{ij}$ при тех же линейных ограничениях.

Алгоритм назначения на основе принципа максимина

В основу алгоритма положен принципа максимина, заключающийся в том, что в каждой строке и каждом столбце матрицы $A = \| a_{ij} \|_{n*m}$, где n - число строк, m- число столбцов (n < m), определяются минимальные элементы, и из всех найденных минимумов выбирается максимальное число. Строка и столбец, в которых расположен максимум, включаются в решение. После чего матрица цен корректируется: из нее удаляются найденные строка и столбец.

Начало алгоритма.

Присваиваем s=0, где s - суммарная цена назначения.

В каждой строке i матрицы A определяем минимальный элемент a_{ij0}, где j0 - номер столбца в котором расположен минимальный элемент.

В каждом столбце j матрицы A определяем минимальный элемент a_{i0j}, где i0- номер строки, в которой находится минимальный элемент.

Определяем максимальный элемент $a_{i*j*}=\max(a_{ij0}, a_{i0j})$.

Элемент a_{i*j*} определяет назначение строки i* столбцу j*. Удаляем строку i* и столбец j* из матрицы (записываем в них символ " ~ ").

Определяем $s=s+ a_{i*j*}$.

Если удалены все строки и столбцы, то переходим к 9, в противном случае к 3.

Конец алгоритма.

Пример:

Дано: Матрица цен A (рис 3,а).

Решение: Определяем минимальные элементы в каждой строке и в каждом столбце матрицы A (рис. 3,а).

A			$a_{i\,j0}$	j0
5	7	10	3	5
4	11	7	3	5
3	5	**6**	3	1
4	7	6	4	1
5	5	7	5	1

A1			$a_{i\,j0}$	j0
5	7	~	3	5
4	11	~	3	5
~	~	~	~	~
4	7	~	4	1
5	5	~	5	1

3	5	**6**	6	3

4	5	~	**6**	3

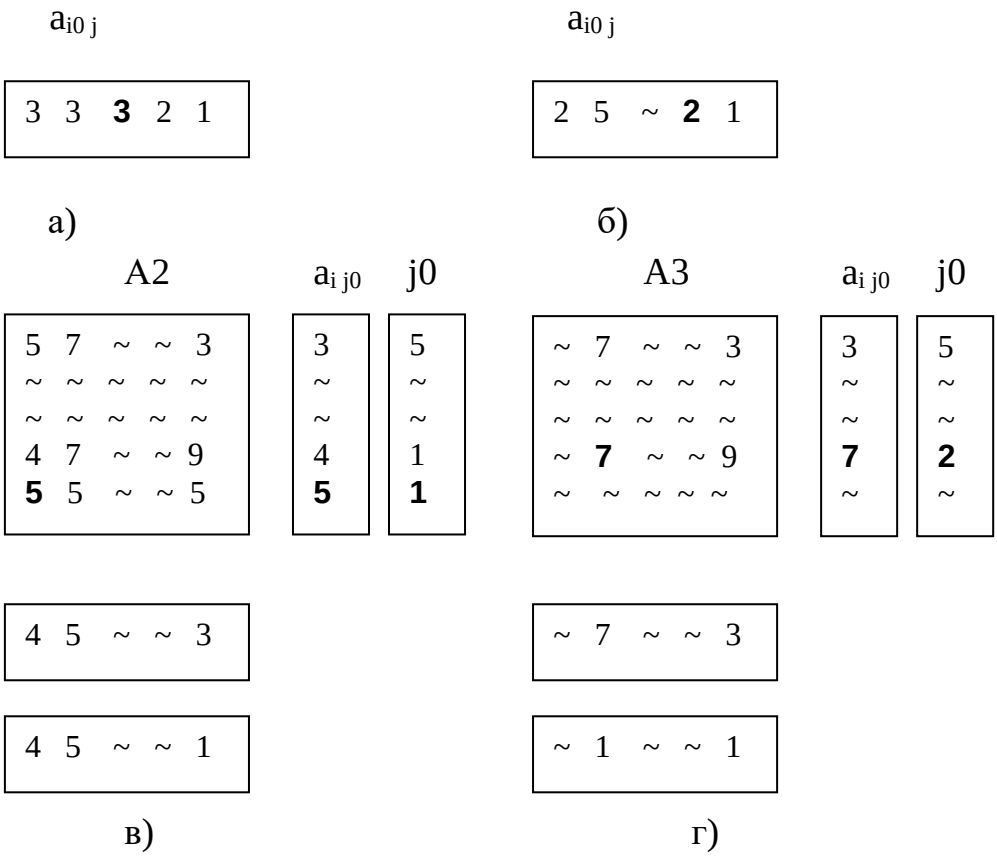


Рисунок 3

Определяем $a_{i^*j^*}$= max(a_{ij0}, a_{i0j})=6. Это элемент a_{33}. Таким образом, строка 3 назначается столбцу 3. Вычисляем s=s+a_{33}= 0+6=6. Удаляем 3-ю строку и 3-й столбец. Определяем минимальные элементы в оставшихся строках и столбцах, (рис. 3,б).

Определяем максимум из всех найденных минимальных элементов $a_{i^*j^*}$= max(a_{ij0}, a_{i0j})=6. Это элемент a_{24}. Строка 2 назначается столбцу 4. Вычисляем s=s+a_{24}=6+6=12. Удаляем 2-ю строку и 4-й столбец. Определяем минимальные элементы в оставшихся строках и столбцах (рис. 3,в).

Определяем максимум из всех найденных минимальных элементов $a_{i^*j^*}$= max(a_{ij0}, a_{i0j})=5. Это элемент a_{51}. Строка 5 назначается столбцу 1. Вычисляем s=s+a_{51}=12+5=17. Удаляем 5-ю строку и 1-й столбец. Определяем минимальные элементы в оставшихся строках и столбцах (рис. 3,г).

Определяем максимум из всех найденных минимальных элементов $a_{i^*j^*}$= max(a_{ij0}, a_{i0j})=7. Это элемент a_{42}. Строка 4 назначается столбцу 2. Вычисляем s=s+a_{42}=17+7=24. Удаляем 4-ю строку и 2-й столбец. После чего в матрице остается один элемент – a_{15}=3. Строка 1 назначается столбцу 5. Вычисляем s=s+a_{15}=24+3=27.

Решение получено. Суммарная цена назначения s=27.

Окончательный результат представлен в табл. 8.

Таблица 8

Номер строки	Номер столбца	Цена назначения
3	3	6
2	4	6
5	1	5
4	2	7
1	5	3

Алгоритм назначения на основе принципа минимального риска

Устанавливаем суммарную цену s=0.

Определяем в каждой строке i два минимальных элемента a_{i1}, a_{i2} и разность между ними Δa_i.

Определяем в каждом столбце j два минимальных элемента a_{1j}, a_{2j} и разность между ними Δa_j.

Определяем $\max(\Delta a_i, \Delta a_j)$ и номер строки (i*) или столбца (j*), в котором находится максимальное значение разности.

Если $\Delta a_i \geq \Delta a_j$, то i*=i, а j* выбираем равным адресу столбца, в котором находится первый минимум строки i*.

Если $\Delta a_j > \Delta a_i$, то j*=j, а i* выбираем равным адресу строки, в которой находится первый минимум столбца j*.

Вычисляем $s=s+a_{i^*j^*}$. Строка i* назначается столбцу j*. Удаляем строку i* и столбец j* матрицы A (записываем "~").

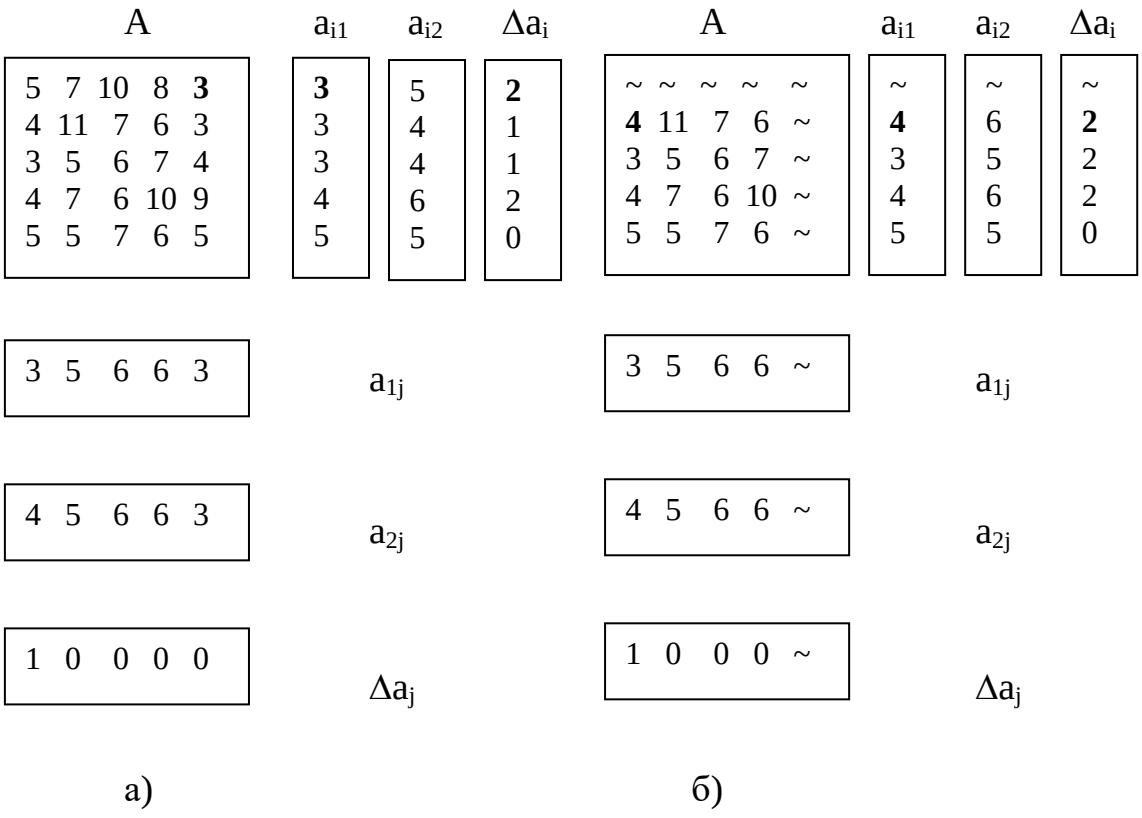
Если в матрице есть не удаленные строки, то переходим к 2, в противном случае - к 7.

Конец алгоритма.

Пример:

Рассмотрим работу алгоритма на том же примере (рис. 3,а), что и в алгоритме на основе принципа максимина.

Решение: Определяем минимальные элементы в строках и столбцах, а также разности между ними (рис. 4,а).


A
a_{i1}
a_{i2}
Δa_i
5 7 10 8 **3**
4 11 7 6 3
3 5 6 7 4
4 7 6 10 9
5 5 7 6 5
3 3 3 4 5
5 4 4 6 5
2 1 1 2 0
3 5 6 6 3
a_{1j}
4 5 6 6 3
a_{2j}
1 0 0 0 0
Δa_j
а)
A
a_{i1}
a_{i2}
Δa_i
~ ~ ~ ~ ~
4 11 7 6 ~
3 5 6 7 ~
4 7 6 10 ~
5 5 7 6 ~
~ **4** 3 4 5
~ 6 5 6 5
~ **2** 2 2 0
3 5 6 6 ~
a_{1j}
4 5 6 6 ~
a_{2j}
1 0 0 0 ~
Δa_j
б)

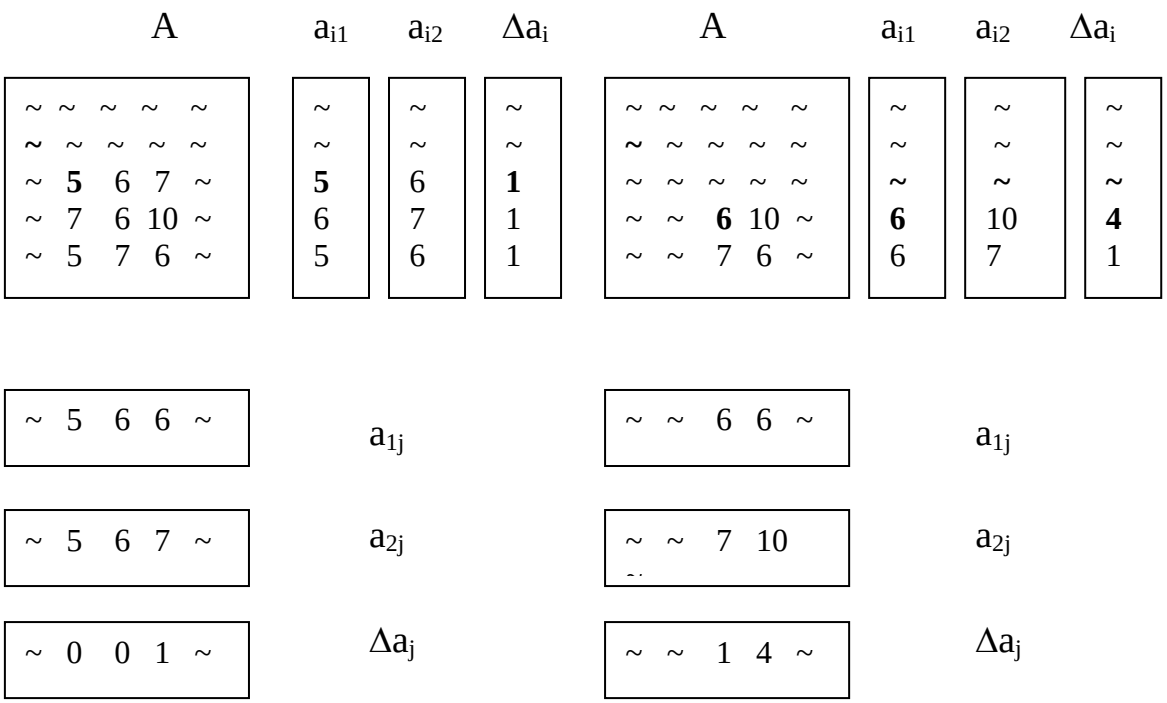


Рисунок 4

Определяем $\max(\Delta a_i, \Delta a_j) = 2$, что соответствует строке $i^*=1$. Первый минимум строки 1 располагается в столбце 5, поэтому $j^*=5$. Включаем в решение элемент матрицы $a_{15}=3$. Вычисляем $s=s+a_{15}=0+3=3$. Строка 1 назначается столбцу 5. Удаляем строку 1 и столбец 5, заменяя числовые значения знаком "~ " (рис.4,б).

Определяем минимальные элементы в оставшихся строках и столбцах, а также разности между ними (рис. 4,б). Определяем $\max(\Delta a_i, \Delta a_j) = 2$, , что соответствует строке $i^*=2$. Первый минимум строки 2 располагается в столбце 1, поэтому $j^*=1$. Включаем в решение элемент матрицы $a_{21}=4$. Вычисляем $s=s+a_{21}=3+4=7$. Строка 2 назначается столбцу 1. Удаляем строку 2 и столбец 1.

Определяем минимальные элементы в оставшихся строках и столбцах, а также разности между ними (рис. 4,в). Определяем $\max(\Delta a_i, \Delta a_j) =1$, что соответствует строке $i^*=3$. Первый минимум строки 3 располагается в столбце 2, поэтому $j^*=2$. Включаем в решение элемент матрицы $a_{32}=5$. Вычисляем $s=s+a_{32}=7+5=12$. Строка 3 назначается столбцу 2. Удаляем строку 3 и столбец 2.

Определяем минимальные элементы в оставшихся строках и столбцах, а также разности между ними (рис. 4,г). Определяем $\max(\Delta a_i, \Delta a_j) =4$, что соответствует строке $i^*=4$. Первый минимум строки 4 располагается в столбце 3, поэтому $j^*=3$. Включаем в решение элемент

матрицы a_{43}=6. Вычисляем s=s+a_{43}=12+6=18. Строка 4 назначается столбцу 3. Удаляем строку 4 и столбец 3.

В матрице остается один элемент a_{54}=6. Включаем его в решение. Вычисляем s=s+a_{54}=18+4=22. Суммарная цена назначения s=22.

Результат назначения представлен в таблице 9.

Таблица 9

Номер строки	Номер столбца	Цена назначения
1	5	3
2	1	4
3	2	5
4	3	6
5	4	8

Порядок выполнения

1. Изучить теоретическую часть.
2. Получить задание на лабораторную работу у преподавателя.
3. Решить вручную поставленную задачу с помощью приведенных алгоритмов.
4. Подготовить файлы с исходными данными.
5. Решить на ПЭВМ все варианты с помощью реализованных алгоритмов, сделать выводы об оптимальности полученных решений.
6. Оформить отчет.

Контрольные вопросы

1. Как формулируется задача линейного назначения?
2. Какие методы решения Вы знаете?
3. Какова область применения этой задачи?
4. В чем суть "Венгерского алгоритма"?
5. В чем суть алгоритма минимального риска?
6. В чем суть алгоритма на основе принципа максимина?

Лабораторная работа №4

Нахождение экстремума в условия неопределенности. Принцип гарантированного результата

Цель работы: изучить алгоритмы нахождения экстремума в условиях неопределенности для унимодальных функций.

Теоретическая часть

Еще многие задачи не решены в общем виде. Примеры таких задач можно найти при исследовании систем, деятельность которых не поддается формальному описанию. Обратимся к изучению задач, для которых характерно следующее:

целевая функция не имеет аналитического выражения (например, это критерий, которому должна удовлетворять задача);

значения этой функции при тех или иных значениях X могут быть получены посредством проведения некоторой операции, которая называется экспериментом;

представляющее интерес максимальное или минимальное значение целевой функции в соответствующей точке x_0 может быть получено путем проведения ряда однородных экспериментов, число которых ограничено.

Указанные действия составляют процесс поиска экстремума z=f(x) в рассматриваемых специфических условиях. Правила, по которым ведется поиск, называется стратегиями. Для поиска экстремума необходимо также правильно выбрать стратегию.

Обратимся к анализу детерминированных схем, чтобы понять принцип поиска экстремума.

Предположим, что $X=(x_0)$ и целевая функция обладает такими особенностями:

имеет только один максимум на интервале определения (свойство унимодальности);

может не удовлетворять требованиям непрерывности, существования производной во всех точках и т.п.

Будем рассматривать задачу на интервале [0,1]. При обобщении задачи всегда можно прейти к другому интервалу.

Чтобы иметь возможность использовать свойство унимодальности, обратимся к определению. Пусть x_1 и x_2 – значения x в двух

экспериментах, дающих соответственно z_1 и z_2, а x^* - значение x, при котором достигается максимум $z=z^*$ (полагаем, что $x_2>x_1$). Функция z=f(x) является унимодальной, если из условия $x_1>x^*$ следует $z_1>z_2$, а из условия $x_1<x^*$ следует $z_1<z_2$.Таким образом, если точки x_1 и x_2 расположены по одну сторону от x^*, то более близкой к x^* точке отвечает большая величина z.

Отсюда следует вывод: результат любой пары экспериментов должен указывать такой интервал значений x, который по своей величине меньше исходного интервала [0,1] и обязательно содержит x^*.

Исходя из определения унимодальности можно организовать пошаговый процесс сужения указанных подынтервалов с целью приближения к x^* следующим образом: если $x_1<x_2$, то возможны соотношения между z_1 и z_2:

$z_1>z_2$: в этом случае точка x^* не может лежать правее x_2 и подынтевал $[x_2,1]$должен быть исключен из рассмотрения;

$z_1<z_2$: в этом случае точка x^*не может лежать левее x_1 и подынтевал $[0,x_1]$ должен быть исключен из рассмотрения;

$z_1=z_2$: в этом случае: в этом случае точка x^* лежит между x_1 и x_2.

Таким образом, пусть проведено N экспериментов и варианту с номером q ($1\leq q\leq N$) соответствует предположению о том, что набольшее значение Z найдено в точке x_q. Определение интервала, на который должно приходиться значение x^*,происходит так же, как в примере с $N=2x_{q-1}<x^*<x_{q+1}/$

Это неравенство позволяет ввести меру эффективности поиска и определить на этой основе рациональные схемы его организации. Назовем интервал $[x_{q-1}, x_{q+1}]$ интервалом неопределенности, его длина $L_N=x_{q+1}-x_{q-1}$. Из всех l_n выбираем $\max(x_{q+1}-x_{q-1})$ при (1,q,N) – это ориентация на наихудший случай, который дает гарантию от неопределенных осложнений. Если из всех стратегий, которые были использованы, выбираем наименьшую L_N, то выбираем таким образом единственную наилучшую стратегию, называемую минимаксную ($L_N=\min \max (x_{q+1},x_{q-1})$), отклонение от которой приведет к наихудшему результату.

Существуют два вида стратегий:

пассивные, в которых еще до начала эксперимента назначают все $x_1,\ldots,x_N$;

активные, в которых выбор очередных значений x зависит от результатов предшествующих экспериментов (имеет место накопление и активное использование информации о свойствах целевой функции).

Активные стратегии поиска более прогрессивные, чем пассивные.

Активные стратегии поиска.
Метод дихотомии (половинного деления)

Суть этого метода заключается в следующем: из экспериментов, находящихся в распоряжении исследователя, выбираются два (x_1,x_2) и размещаются на исходном единичном отрезке наилучшим образом, т.е. симметрично относительно середины отрезка на расстоянии ε/2 от нее. В результате сравнения полученных величин и становится возможным указать гарантированный интервал неопределенности, т.е. из определения унимодальности после сравнения значений функций в точках x_1 и x_2 выбираем следующий интервал неопределенности, и процесс нахождения максимума функции продолжается до тех пор, пока не найдем последние x_N и x_{N-1} и выбираем последний интервал неопределенности, которому принадлежит x^*.

Пример:

Найти точку экстремума (максимума) унимодальной функции

$$z=f(x)=\begin{cases} e^{3x/2}, & 0\le x\le 0{,}45; \\ 3-2{,}3*x, & 0{,}45\le x\le 0{,}7; \\ 1-x^2, & 0{,}7\le x\le 1. \end{cases}$$

методом дихотомии при ε=0,004 и N=8.

Решение:

Выбираем точки x_1 и x_2:

$x_1=0{,}5-\varepsilon/2=0{,}48$, $x_2=0{,}5+\varepsilon/2=0{,}52$.

Находим $z_1=f(x_1)=1{,}896$ и $z_2=f(x_2)=1{,}804$.

Выбираем отрезок [0;0,52] с центром в точке x=0,26.

Выбираем точки x_3 и x_4:

$x_3=0{,}26-\varepsilon/2=0{,}24$, $x_4=0{,}26+\varepsilon/2=0{,}28$.

Находим $z_3=f(x_3)=1{,}433$ и $z_4=f(x_4)=1{,}522$.

Выбираем отрезок [0,24;0,52] с центром в точке x=0,38.

Выбираем точки x_5 и x_6

$x_5=0{,}38-\varepsilon/2=0{,}40$, $x_6=0{,}38+\varepsilon/2=0{,}36$.

Находим $z_5=f(x_5)=1{,}716$ и $z_6=f(x_6)=1{,}822$.

Выбираем отрезок [0,36;0,52] с центром в точке x=0,44.

Выбираем точки x_7 и x_8:

$x_7=0{,}44-\varepsilon/2=0{,}42$, $x_8=0{,}44+\varepsilon/2=0{,}46$.

Находим $z_7=f(x_7)=1{,}877$ и $z_8=f(x_8)=1{,}993$.

Выбираем отрезок [0,42;0,52], которому принадлежит x^*.

Метод чисел Фибоначчи

Реализация этого метода связана с использованием последовательности чисел, открытой итальянским математиком Леонардо Пизанским (Фибоначчи) в начале XIII века. Числа формируются по следующей формуле: $F_N=F_{N-2}+F_{N-1}$, где N=2,3,4,…., а $F_0=F_1=1$.

N	F_N	N	F_N
0	1	9	55
1	1	10	89
2	2	11	144
3	3	12	233
4	5	13	377
5	8	14	610
6	13	15	987
7	21	16	1597
8	34	….	

Алгоритм:

Задается функция z=f(x) и ε;

L_1 – исходный единичный интервал неопределенности, $L_1=1$;

Находим $N_{пред}$ для данной ε из формулы $F_{Nпред+1}\leq 1/\varepsilon$.

Находим $L_2 = [F_{N-1}+\varepsilon*(F_{N-1}*F_{N-2}-F_N*F_{N-3})]/F_N$;

На интервале L_1 отмеряем L_2 подсчитанное по формуле, и строим точку x_1. Относительно середины L_1 симметрично строим точку x_2. Выбираем из этих точек ту, для которой z больше. В зависимости от местоположения выбираем интервал, содержащий x_2 (или x_1), и на этом интервале строим точку x_3, симметричную x_2 (или x_1) относительно середины выбранного интервала неопределенности, и т.д. до тех пор, пока не будет проведено N экспериментов или расстояние между точками не станет меньше ε.

Т.е. каждый раз мы выбираем интервал, содержащий точку с большим значением функции. В выбранном отрезке строим следующую точку, которая будет симметрична относительно середины нового интервала неопределенности оставшейся точке. Последняя исследуемая точка - это $x_{Nпред.}$.

Метод золотого сечения

Существует пропорция деления данного отрезка на два, которая получила название золотого сечения. Зная его, можно построить последовательность экспериментов, начиная с $x_1=1-1/t=0,382$, и прийти к

интервалу L=1/tN-1. Производить эксперименты здесь можно до тех пор, пока выполняется совместно условия $L_{N-1}=tL_N$ и $L_N \geq 0{,}5*(L_{N-1}+\varepsilon)$.

Алгоритм:

Находим $N_{пред}$ для данного ε по формуле $N \leq 1+[\lg(2-t)-\lg(\varepsilon)]/\lg(t)$, где t=1,618.

$x_1=1-1/t=0{,}382/$

Строим x_1 на L_1.

Симметрично относительно середины единичного отрезка строим x_2.

Сравниваем $z_1=f(x_1)$ и $z_2=f(x_2)$ и в зависимости от этого выбираем интервал неопределенности по принципу гарантированного результата. Строим следующую точку, симметрично относительно середины отрезка, и опять сравниваем значения функций в этих точках и выбираем следующий интервал неопределенности и так до тех пор, пока не достигнем $N_{пред}$.

Порядок выполнения работы

1. Изучить теоретическую часть.
2. Построить график заданной функции средствами Excel, график распечатать и сделать вывод об унимодальности функции.
3. Найти максимум заданной функции методами дихотомии, чисел Фибоначчи, золотого сечения. Проанализировать полученные результаты и сделать выводы.
4. Оформить отчет.

Контрольные вопросы

1. Дайте определение унимодальной функции и схему нахождения максимума или минимума на унимодальных функциях.
2. Дайте определение принципа гарантированного результата.
3. Дайте определение стратегий и их классификаций.
4. Дайте схему нахождения экстремума с помощью метода половинного деления.
5. Дайте схему нахождения экстремума с помощью метода чисел Фибоначчи.
6. Дайте схему нахождения экстремума с помощью метода золотого сечения.

Библиографический список

1. *Ашманов С.А.* Линейное программирование. – М.: Наука, 1981.
2. *Аоки М.* Введение в методы оптимизации. – М.: Наука, 1977.
3. *Дегтярев Ю.И.* Методы оптимизации. – М.: Сов. радио, 1980.
4. *Деньдобренько Б.Н., Малика А.С.* Автоматизации конструирования РЭА. – М.: Высш. шк., 1980.
5. *Гасс С.* Линейное программирование. – М.: Гос. изд-во физ.-мат. лит., 1961.
6. *Акулич И.Л.* Математическое программирование в примерах и задачах. – М.: Высш. шк., 1986.

Содержание

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ МИРОВЫЕ ТЕНДЕНЦИИ РАЗВИТИЯ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

Квантовые компьютеры

1. Начало квантовой эры

Точкой отсчёта квантовой эры принято считать 1900 год, когда Макс Планк выдвинул гипотезу, заключающуюся в том, что энергия при тепловом излучении испускается и поглощается отдельными порциями (квантами). Квантовая физика привнесла в нашу жизнь такие изобретения, как, например, лазер, транзисторы, ядерное оружие и прочее. В 1980 году советский учёный Юрий Манин высказал идею квантовых вычислений. В 1981 году Ричард Фейнман предложил модель квантового компьютера, способную произвести моделирование эволюции квантовой системы.

Основные вехи в истории создания квантовых компьютеров:

- [1994]. П.Шор. Разработан квантовый алгоритм факторизации чисел
- [1998]. Создан первый 2-х кубитный квантовый компьютер
- [2001]. IBM представил выполнение алгоритма Шора для разложения числа 15
- [2007-2016]. D-Wave создает и развивает компьютер с 128-2000 кубитов
- [2012]. В Университете Калифорнии реализовали алгоритм Шора для числа 21
- [2016]. Google смоделировал молекулу водорода на 9-и кубитном компьютере
- [2017]. IBM смоделировала гидрид бериллия BeH2 (три атома)
- [2019]. IBM Q System One. 20-и кубитный компьютер в облаке
- [2019]. Google Sycamore. 53-х кубитный компьютер. Квантовое превосходство?

2. Основные направления развития квантовых технологий

Перспективные направления квантовой физики способны предоставить человечеству множество новых возможностей и технологий, таких как:

- ***Квантовые сенсоры***, превосходящие все современные аналоги по точности при гораздо меньших размерах. Их использование в медицине позволят изучить тело человека на клеточном уровне; в атомных часах, основанных на использовании цезия-133, позволяющих измерять не только время, причём с точностью, в 37 раз превосходящей современные аналоги, но и гравитацию; квантовые гравиметры, помогающие шахтёрам находить месторождения и опасные пустоты под землёй.
- ***Квантовая криптография***, позволяющая обнаружить попытки перехвата сигнала в ***квантовой сети***.

Применительно к разработке квантовых компьютеров, можно выделить два основных направления:

- ***Специализированные квантовые компьютеры***
- ***Универсальные квантовые компьютеры***

3. Принципы работы квантового компьютера

Квантовый компьютер (в отличие от обычного) в качестве носителей информации использует ***квантовые объекты***, а для проведения вычислений квантовые объекты должны быть соединены в квантовую систему.

Квантовый объект — объект микромира (квантового мира), который проявляет квантовые свойства:

- Имеет определенное состояние с двумя граничными уровнями
- Находится в суперпозиции своего состояния до момента измерения
- Запутывается с другими объектами для создания квантовых систем
- Выполняет теорему о запрете клонирования (нельзя скопировать состояние объекта)

Квантовая система — система запутанных квантовых объектов, обладающая следующими свойствами:

- Квантовая система находится в суперпозиции всех возможных состояний объектов, из которых она состоит
- Нельзя узнать состояние системы до момента измерения
- В момент измерения система реализует один из возможных вариантов своих граничных состояний

Следствие для квантовых программ:

- Квантовая программа имеет заданное состояние системы на входе, суперпозицию внутри, суперпозицию на выходе
- На выходе программы после измерения имеем вероятностную реализацию одного из возможных конечных состояний системы (плюс возможные ошибки)
- Любая квантовая программа имеет архитектуру дымоходной трубы (вход -> выход. Нет циклов, нельзя посмотреть состояние системы в середине процесса.)

В обычном компьютере минимальной единицей информации является ***бит***. Ввиду того, что бит принимает одно из двух детерминированных значений, он не подходит для реализации квантового объекта. В качестве минимальной единицы информации реализован ***кубит*** (квантовый бит). В

граничных состояниях он реализует похожие на 0 и 1 состояния |0> и |1> (***бра*** и ***кет***), а в суперпозиции представляет собой ***вероятностное распределение над своими граничными состояниями*** |0> и |1>:

```
a|0> + b|1>, такое, что a^2+b^2=1
```

a и b при этом представляют собой амплитуды вероятностей, а квадраты их модулей — собственно вероятности получить именно такие значения граничных состояний |0> и |1>, если схлопнуть кубит измерением прямо сейчас.

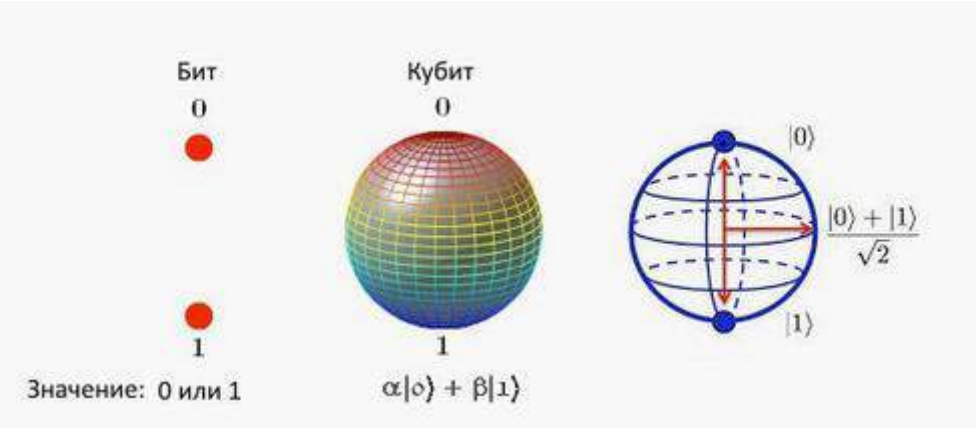


Рисунок 1 - бит и кубит

Физическим носителем информации для квантового компьютера является состояние квантового объекта (направление поляризации, спин и т. д.), которое может находиться в состоянии суперпозиции.

Таблица 1 - сравнение обычного и квантового компьютера

	Обычный компьютер	**Квантовый компьютер**
Логика	0/1	`a|0> + b|1>, a^2+b^2=1`
Физика	Полупроводниковый транзистор	Квантовый объект
Носитель информации	Уровни напряжения	Поляризация, спин,…
Операции	NOT, AND, OR, XOR над битами	Вентили: CNOT, Адамара,…
Взаимосвязь	Полупроводниковый чип	Запутанность между собой
Алгоритмы	Стандартные (см. Кнут)	Специальные (Шор, Гровер)
Принцип	Цифровой, детерминированный	Аналоговый, вероятностный

Типичным ***примером*** квантового объекта может служить обычная монета. Два граничных уровня – «орёл» и «решка». Когда мы подбрасываем монету, она летит и вращается в воздухе – находится в суперпозиции своего состояния. Однако, когда мы ловим монету, она принимает одно из двух граничных значений – «орёл» или «решка». Прихлопывание монеты в данном случае и есть измерение. «Квантовая система» из монет образуется, когда мы одновременно подбрасываем несколько монет, которые соударяются друг с другом и изменяют состояние друг друга. Пока монеты летят и вращаются, система не может быть скопирована никаким образом.

4. Квантовые вентили и алгоритмы квантовых вычислений

Для реализации логических схем на обычном компьютере используются всем нам хорошо известные логические операции, для операций над кубитами пришлось придумывать совершенно иную систему операций, называемую квантовыми вентилями. Вентили бывают однокубитные и двухкубитные, в зависимости от того, над сколькими кубитами производится преобразование.

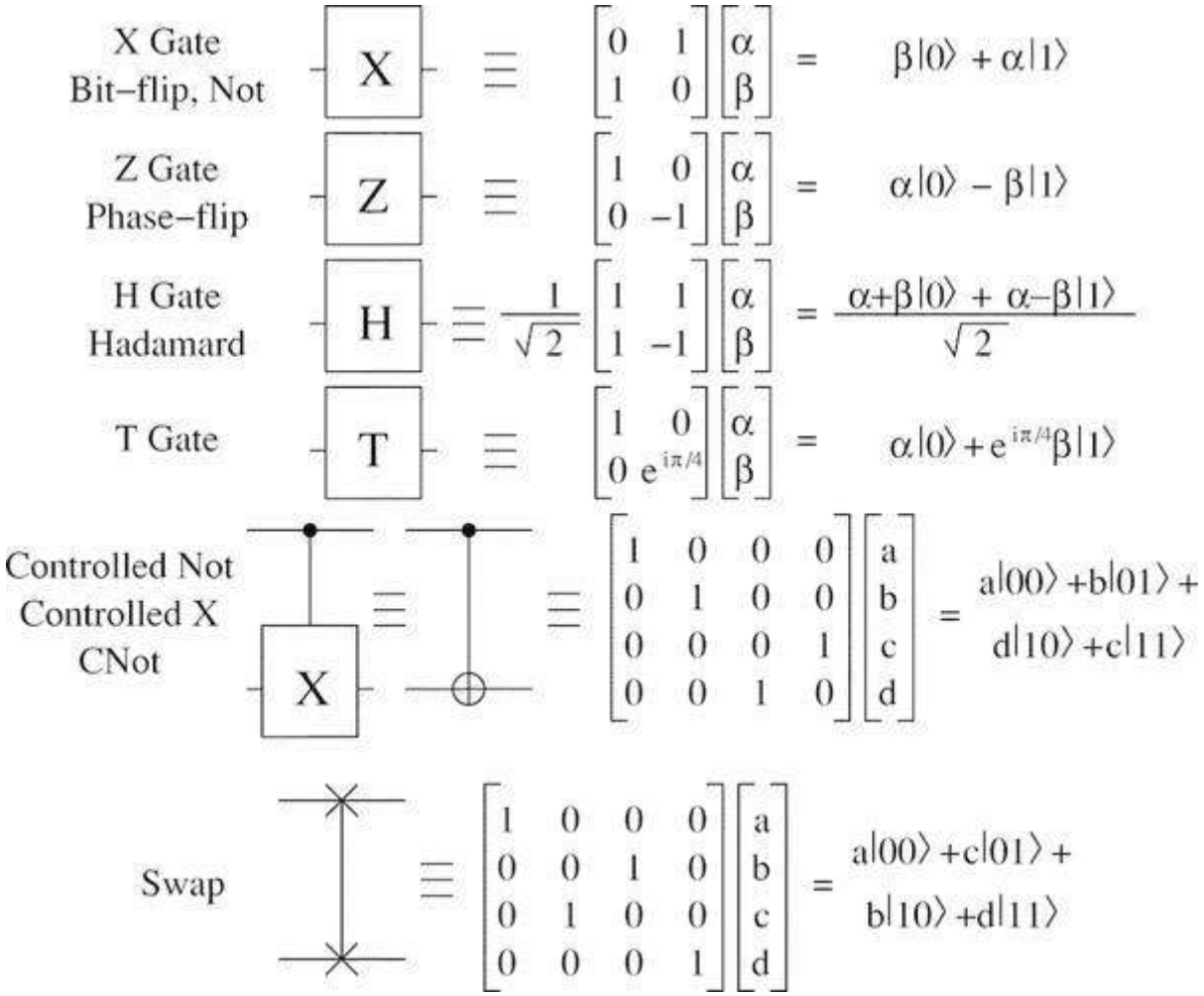


Рисунок 2 – квантовые вентили

Есть понятие ***универсального набора вентилей***, которых достаточно для выполнения любого квантового вычисления. Например, универсальным является набор, включающий вентиль Адамара, вентиль фазового сдвига, вентиль CNOT и вентиль $\pi/8$. С их помощью можно выполнить любое квантовое вычисление на произвольном наборе кубитов.

- Операции над квантовыми объектами требуют создания новых логических операторов (квантовых вентилей)
- Квантовые вентили бывают однокубитные и двухкубитные
- Существуют универсальные наборы вентилей, с помощью которых можно выполнить любое квантовое вычисление

Стандартные алгоритмы, которые накопило человечество к текущему моменту, совершенно не подходят для реализации на квантовом компьютере. Квантовые компьютеры, основанные на вентильной логике над кубитами, требуют создания совершенно иных алгоритмов, квантовых алгоритмов. Из наиболее известных квантовых алгоритмов можно выделить три:

- Алгоритм Шора (факторизация)
- Алгоритм Гровера (быстрый поиск в неупорядоченной базе данных)
- Алгоритм Дойча-Йожи (ответ на вопрос, постоянная или сбалансированная функция)

И самое главное отличие — это ***принцип работы***. У стандартного компьютера это цифровой, жестко детерминированный принцип, основанный на том, что если мы задали какое-то начальное состояние системы и пропустили его через заданный алгоритм, то результат вычислений будет один и тот же, сколько бы раз мы это вычисление не запускали. Собственно, такое поведение — это именно то, что мы от компьютера и ждем.

Квантовый компьютер работает на аналоговом, вероятностном принципе. Результат работы заданного алгоритма на заданном начальном состоянии представляет собой выборку из вероятностного распределения конечных реализаций алгоритма плюс возможные ошибки.

Такая вероятностная природа квантовых вычислений обусловлена самой вероятностной сутью квантового мира. “Бог не играет в кости со вселенной”, — говорил старик Эйнштейн, но все эксперименты и наблюдения пока (в текущей научной парадигме) подтверждают обратное.

5. Преимущества квантовых вычислений

Итак, представим, что у нас есть следующая задача:

Есть группа из трех человек: (A)ндрей, (B)олодя и (C)ережа. Есть два такси (0 и 1).

Известно также, что:

- (A)ндрей, (B)олодя — друзья
- (A)ндрей, (C)ережа — враги
- (B)олодя и (C)ережа — враги

Задача: разместить народ по такси так, чтобы Max(друзья) и Min(враги)

Оценка: L = (кол-во друзей) — (кол-во врагов) для каждого варианта размещения

ВАЖНО: предположим, что эвристик нет, оптимального решения нет. В этом случае задача решается только полным перебором вариантов.

Рисунок 3 – решение задачи

Решение на обычном компьютере

Как решать эту задачу на обычном (супер)компьютере (или кластере) — понятно, что надо перебрать в цикле все возможные варианты. Если у нас мультипроцессорная система, то можно распараллелить расчет решений на несколько процессоров и потом собрать результаты.

У нас 2 возможных варианта размещения (такси 0 и такси 1) и 3 человека. Пространство решений 2^3 = 8. Перебрать 8 вариантов можно даже на калькуляторе, это не проблема. А теперь усложним задачу — у

нас 20 человек и два автобуса, пространство решений 2^20 = 1 048 576. Тоже ничего сложного. Увеличим количество людей в 2.5 раза — возьмем 50 человек и два поезда, пространство решений теперь 2^50 = 1.12 x 10^15. У обычного (супер)компьютера уже начинаются серьезные проблемы. Увеличим кол-во людей в 2 раза, 100 человек дадут нам уже 1.2 x 10^30 возможных вариантов.

Все, за разумное время эту задачу не посчитать.

Как мы видим, **при увеличении размерности исходных данных пространство решений растет по степенному закону**, в общем случае для N битов у нас есть 2^N возможных вариантов решения, которые при сравнительно небольших N (100) дают нам непросчитываемое (на текущем технологическом уровне) пространство решений.

Возьмем мешок и положим в него 1000 белых и 1000 черных шаров. Будем проводить эксперимент — вынимать шар, записывать цвет, возвращать шар в мешок и перемешивать шары в мешке.

Провели эксперимент 10 раз, вытащили 10 черных шаров. Возможно? Вполне. Дает нам эта выборка какое-то разумное понятие об истинном распределение в мешке? Очевидно, что нет. Что надо сделать — правильно, повторить эксперимент миллион раз и рассчитать частоты выпадения черных и белых шаров. Получим, например 49.95% черных и 50.05% белых. В этом случае уже более-менее понятна структура распределения из которого мы семплируем (вынимаем один шарик).

Главное, что надо понять, что сам эксперимент имеет вероятностную природу, одним семплом (шариком) мы не узнаем истинную структуру распределения, нам надо многократно повторить эксперимент и усреднить результаты.

Добавим в наш мешок 10 красных и 10 зеленых шаров (ошибки). Повторим эксперимент 10 раз. Вытащили 5 красных и 5 зеленых. Возможно? Да. Можем что-то сказать об истинном распределении — Нет. Что надо сделать — ну вы поняли.

Для получения понимания о структуре вероятностного распределения надо многократно просемплировать единичные исходы из этого распределения и усреднить результаты.

Теперь вместо черных и белых шаров давайте возьмём бильярдные шары, и положим в мешок 1000 шаров с номером 2, 1000 с номером 7 и 10 шаров с другими номерами. Представим себе экспериментатора, который обучен простейшим действиям (достать шар, записать номер, положить шар обратно в мешок, перемешать шары в мешке) и делает он это за 150

микросекунд. Ну такой экспериментатор на спидах. Тогда за 150 секунд он сможет провести наш эксперимент 1 миллион раз и предоставить нам результаты усреднения.

Усадили экспериментатора, дали мешок, отвернулись, подождали 150 секунд — получили:

номер 2 — 49.5%, номер 7 — 49.5%, остальные номера в сумме — 1%.

Да, все верно, наш мешок — это квантовый компьютер с алгоритмом, решающим нашу задачу, а шары — возможные варианты решения. Поскольку правильных решений два, то квантовый компьютер будет выдавать нам равновероятно любое из этих возможных решений, и 0.5% (10/2000) ошибок, о которых мы поговорим позднее.

Для получения результата работы квантового компьютера надо многократно запустить квантовый алгоритм на одном и том же входном наборе данных и усреднить результат.

Масштабируемость квантового компьютера

Теперь представим себе, что для задачи, в которой участвуют 100 человек (пространство решений 2^100 мы помним об этом), правильных решений тоже только два. Тогда, если взять 100 кубитов и написать алгоритм, вычисляющий нашу целевую функцию (L, см. выше) над этими кубитами, то мы получим мешок, в котором будет 1000 шаров с номером первого правильного ответа, 1000 с номером второго правильного ответа и 10 шаров с другими номерами. И наш экспериментатор за те же 150 секунд выдаст нам оценку вероятностного распределения правильных ответов.

Время выполнения квантового алгоритма (с некоторыми допущениями) можно считать константным O(1) по отношению к размерности пространства решений (2^N).

И вот именно это свойство квантового компьютера — константность времени выполнения по отношению к возрастающей по степенному закону сложности пространства решений и является ключевым.

6. Перспективы развития квантовых вычислений

Исследовательская компания IDC прогнозирует, что к 2023 году 25% компаний из списка Fortune 500 будут использовать квантовые вычисления.

В то время как квантовые вычисления еще не учитывались в решении глобальных проблем, таких как пандемия коронавируса, именно этот тип проблем имеет потенциал для решения с помощью квантовых вычислений. По мнению одного из разработчиков квантовых компьютеров, компании IBM, именно в этом десятилетии квантовые вычисления шагнут в реальный мир.

По этой причине сегодня важно придавать квантовым технологиям публичность, а не держать их похороненными в исследовательских учреждениях. Такие компании, как Google, Microsoft, D-Wave и Regetti, также стремятся продвигать квантовые технологии, и, основываясь на недавнем докладе IDC «тенденции внедрения квантовых вычислений: результаты опроса 2020 года», эта тенденция набирает обороты.

Согласно ответам 520 ИТ-специалистов и специалистов в области бизнеса, бюджеты на внедрение квантовых вычислений в ближайшие 2 года продолжат увеличиваться. Половина всех респондентов, участвовавших в опросе IDC, сообщили, что средства, выделяемые на квантовые вычисления, составляют всего 0-2% от годовой ИТ-инфраструктуры в 2019 году, но в ближайшие 24 месяца составят 7-10%. Для компаний с численностью сотрудников более 10 000 человек увеличение расходов является более значимым: половина респондентов собираются потратить на квантовые технологии от 9% до 14% в течение следующих двух лет.

Респондентам, участвовавшим в опросе IDC, было ясно, на чем они сосредоточивают свое внимание: 65% респондентов планируют использовать облачные мощности квантовых компьютеров, а ещё 45% планируют внедрить или уже используют квантовые алгоритмы (включая симуляторы, оптимизацию, искусственный интеллект, машинное обучение и глубокое обучение). Согласно опросу IDC, в первую пятерку входят квантовые сети (44%), гибридные квантовые вычисления (40%) и квантовая криптография (33%).

7. Что нужно для работы с квантовыми вычислениями

Программирование

Чтобы начать работу с квантовыми вычислениями, вам потребуется знание основ программирования. Уже существуют некоторые языки программирования, предназначенные только для квантовых компьютеров, но вам не нужно их знать на начальном этапе.

Одним из наиболее часто используемых языков программирования для существующих квантовых компьютеров является ***Python***.

Если вы захотите попробовать запрограммировать квантовый компьютер, я бы посоветовал вам начать с инструментов с открытым исходным кодом на платформе **Qiskit**, разработанную IBM. **Qiskit** не только позволит вам запрограммировать реальный квантовый компьютер IBM, но и поможет вам без труда изучить многие квантовые концепции. После освоения квантовых концепций вы сможете попробовать изучить квантовый язык программирования.

Математика

Математика, наверное, занимает первое месте среди причин, которые отпугивают от квантовых вычислений. Не стану врать —в этой сфере математика нужна.

Но математика нужна и в других технических областях, таких как машинное обучение, искусственный интеллект и обработка естественного языка. Фактически, для некоторых из этих *областей науки о данных нужна более сложная математика, чем для квантовых вычислений*. Я знаю, звучит неправдоподобно, но это правда.

Основной раздел математики, позволяющий квантовым вычислениям творить чудеса,—это ***линейная алгебра***. Всё в квантовых вычислениях — от представления кубитов и вентилей до функциональности схем — можно описать с помощью различных методов линейной алгебры.

Ещё один раздел математики, связанный с квантовыми вычислениями,—это основы ***теории вероятностей***.

Всё. Линейная алгебра и теория вероятностей. Два раздела, связанные со всеми подразделами науки о данных и большинством современных технических областей.

Нельзя сказать, что по мере продвижения в этой области ничего не будет усложняться. Но разве так не во *всех* сферах?

Физика / механика

Внутренние функции квантовых вычислений, такие как запутанность и квантовая суперпозиция, являются чисто физическими явлениями. Определённые законы физики контролируют и объясняют, почему они происходят.

Однако внутренние функции классических компьютеров такие же. Вам не нужно точно знать, как работает оборудование вашего компьютера, чтобы с его помощью создавать крутые вещи. Вам нужно только знать, как им пользоваться.

Такая же логика применима к квантовым вычислениям. Чтобы быть разработчиком квантового программного обеспечения, вам только нужно узнать, как работает квантовый компьютер, а затем использовать его для создания собственных приложений.

Если вы собираетесь работать над созданием оборудования для квантовых вычислений, тогда вам действительно понадобится разобраться в физике и механике работы квантового компьютера.

То же самое необходимо, если вы хотите построить классический процессор или графический процессор. Вы должны разобраться в физике, которая позволяет этому оборудованию работать корректно.

8. Вывод

Квантовые компьютеры и квантовые вычисления — очень многообещающая, очень молодая и пока малоприменимая в промышленном плане область информационных технологий.

Развитие квантовых вычислений позволит (когда-нибудь) решать задачи:

- Моделирования сложных физических систем на квантовом уровне
- Нерешаемые на обычном компьютере из-за вычислительной сложности

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ ОНТОЛОГИЯ ПРОЕКТИРОВАНИЯ ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

ГЛАВА 1. ПРЕДСТАВЛЕНИЕ ЗНАНИЙ В ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ СИСТЕМАХ

1.1. Данные и знания

Интеллектуальные информационные системы (ИИС) или системы, основанные на знаниях, предствляют комплекс программных, лингвистических и логико-математических средств для реализации основной задачи – осуществления поддержки деятельности человека и поиска информации в режиме продвинутого диалога на естественном языке. Системы, основанные на знаниях, применяются во многих предметных областях для решения сложных слабоструктурированных проблем. При изучении подобных систем возникает вопрос – что же такое знания и чем они отличаются от данных.

Данные – это поддающееся многократной интерпретации представление информации в формализованном виде, пригодном для хранения, передачи и обработки. Данные могут быть цифровыми (факты, результаты измерений), графическими, аудио, видео и т.п. Они могут описываться на различных языках (символьном, математическом, графическом и т.п.). Качественными мерами для данных являются своевременность, соответствие и точность.

Знания являются более сложной категорией информации по сравнению с данными. Под знаниями понимается форма представления информации, которой присущи такие особенности, как:

- Внутренняя интерпретируемость (каждая информационная единица имеет уникальное имя, по которому система находит ее, а также отвечает на запросы, в которых это имя упомянуто).
- Структурированность (включенность одних информационных единиц в состав других).

- Связность (возможность задания временных, причинных, пространственных или иного рода отношений).
- Семантическая метрика (возможность задания отношений, характеризующих ситуационную близость).
- Активность (выполнение программ инициируется текущим состоянием информационной базы).

По своей природе знания можно разделить на декларативные и процедурные. Декларативные знания содержат в себе лишь представление о структуре неких понятий. Эти знания приближены к данным. Декларативные знания представляют собой описания фактов и явлений, фиксируют наличие или отсутствие таких фактов, а также включают описания основных связей и закономерностей, в которые эти факты и явления входят. Например: высшее учебное заведение есть совокупность факультетов, а каждый факультет в свою очередь есть совокупность кафедр. Процедурные знания имеют активную природу. Процедурные знания – это описания действий, которые возможны при манипулировании фактами и явлениями для достижения намеченных целей. Процедурные знания определяют представления о средствах и путях получения новых знаний, проверки знаний. Примерами процедурных знаний являются алгоритмы разного рода.

По способу приобретения знания можно разделить на факты и эвристику (правила, которые позволяют сделать выбор при отсутствии точных теоретических обоснований). Первая категория знаний обычно указывает на хорошо известные в данной предметной области обстоятельства. Вторая категория знаний основана на собственном опыте эксперта, работающего в конкретной предметной области, накопленном в результате многолетней практики.

По типу представления знания делятся на факты и правила. Факты — это знания типа «А – это А», такие знания характерны для баз данных и сетевых моделей. Правила или продукции – это знания типа «ЕСЛИ А, ТО В». Кроме фактов и правил существуют еще метазнания — знания о знаниях.

Они необходимы для управления базой знаний и для эффективной организации процедур логического вывода.

Состав знаний в ИИС зависит от:

- предметной области;
- назначения и структуры;
- требований и целей пользователей;
- языка общения и способах организации диалога с пользователем.

Интеллектуальные информационные системы базируются на концепции использования базы знаний для генерации алгоритмов решения прикладных задач различных классов в зависимости от конкретных информационных потребностей пользователей. База знаний (БЗ) представляет собой особого рода базу данных, разработанную для оперирования знаниями. В отличие от базы данных, БЗ содержат в себе не только фактическую информацию, но и правила вывода, позволяющие делать автоматические умозаключения об уже имеющихся или вновь вводимых фактах и тем самым производить семантическую (осмысленную) обработку информации.

1.2. Концептуальная парадигма работы со знаниями

Концептуальная парадигма работы со знаниями утвердилась в связи и по мере развития концепции интеллектуальных информационных систем, согласно которой главной информационной единицей компьютерной обработки стало «знание». Согласно данной парадигме работа со знаниями рассматривается как процесс, в котором выделяют следующие подпроцессы:

- извлечение знаний;
- приобретение знаний;
- представление знаний;
- манипулирование знаниями.

В область аспектов работы со знаниями, кроме указанных подпроцессов, включают также методы и средства.

1. *Извлечение знаний* из различных источников включает два основных процесса:

- анализ исходной информации и формализацию качественных знаний;
- интеграцию знаний.

Первый процесс связан с созданием методов, позволяющих переходить от знаний, выраженных, в том числе в естественно-языковой форме, к их аналогам, пригодным для ввода в память знание-ориентированной информационной системы. Второй процесс связан с интеграцией знаний, получаемых от различных источников, в некоторую взаимосвязанную и непротиворечивую систему знаний о предметной области.

2. *Приобретение знаний*. Знаний, содержащихся в источниках информации, отчуждѐнных от специалиста, как правило, недостаточно. Значительную часть профессионального опыта эти специалисты не могут выразить словесно (профессиональное умение или интуиция). Поэтому для того, чтобы приобрести такие знания, нужны специальные приѐмы и методы. Они используются в инструментальных системах по приобретению знаний, создание которых – одна из задач инженерии знаний. Полученные от экспертов знания нужно оценить с точки зрения их соответствия ранее накопленным знаниям и формализовать для ввода в память системы. Кроме того, знания, полученные от различных экспертов, необходимо согласовать между собой, так как нередки случаи, когда эти знания оказывались внешне несовместимыми и даже противоречивыми. Рассматриваемый подпроцесс включает такие процессы как:

- организация работы с экспертами;
- оценка и формализация знаний;
- согласование знаний.

3. *Представление знаний*. Этот процесс предполагает разработку формальной научной теории, включающей построение *модели знаний*, *системы представления знаний* и *базы знаний*.

Представление знаний – это соглашение о том, как и в какой формальной теории описывать реальный мир. В естественных и технических науках принят следующий традиционный способ представления знаний. На естественном языке вводятся основные понятия и отношения между ними. Но при этом используются ранее определѐнные понятия и отношения, смысл которых уже известен. Далее устанавливается соответствие между характеристиками (чаще всего количественными) понятий и подходящей математической моделью. Основная цель представления знаний – строить математические модели реального мира и его частей. С этой целью созданы и используются в действующих системах различные модели представления знаний. Каждая модель знаний определяет форму представления знаний и является формализмом, призванным отобразить объекты, связи между ними и отношения, иерархию понятий предметной области и изменение отношений между объектами

Выделяют два типа моделей представления знаний: формальные и неформальные. К неформальным моделям относят сетевые, продукционные модели, фреймы. В отличие от формальных моделей, в основе которых лежит строгая математическая теория, неформальные модели такой теории не придерживаются. Каждая неформальная модель годится только для конкретной предметной области и поэтому не обладает универсальностью, которая присуща моделям формальным. Логический вывод в формальных системах строг и корректен, поскольку подчинен жестким аксиоматическим правилам. Вывод в неформальных системах во многом определяется самим исследователем, который и отвечает за его корректность. Однако в формальных моделях очень сложно учитывать динамические изменения в предметной области. Поэтому такой способ представления знаний

используются в тех предметных областях, которые хорошо локализуются и мало зависят от внешних факторов.

В инженерии знаний системы представления знаний включают совокупность процедур, необходимых для записи знаний, извлечение их из памяти и поддержки хранения знаний в рабочем состоянии. Системой представления знаний называют совокупность средств, позволяющих:

- описывать знания о предметной области с помощью языка представления знаний;
- организовать хранение знаний в системе (накопление, анализ, структурное обобщение и организация знаний);
- выводить новые знания из имеющихся и объединять их;
- находить требуемые знания;
- обновлять знания;
- осуществлять интерфейс между системой и пользователем.

Системы представления знаний оформляются как базы знаний.

4. *Манипулирование знаниями*. К этому процессу относятся такие процессы как

- пополнение знаний;
- классификация знаний;
- обобщение знаний;
- вывод на знаниях;
- рассуждения с помощью знаний;
- объяснения на знаниях;
- решение прикладных задач предметной области.

Новые знания, поступающие в БЗ, должны вместе с теми сведениями, которые уже были ранее записаны в неѐ, сформировать расширение поступивших знаний. Знания образуют упорядоченные структуры, что облегчает поиск нужных знаний и поддержание работоспособности БЗ. Для этого используются различные классифицирующие процедуры:

родовидовые, «часть–целое», ситуативные (когда в одно множество объединяются знания, релевантные некоторой типовой ситуации). В процессе классификации часто происходит абстрагирование от отдельных элементов описаний (фрагментов знаний об объектах или явлениях), появляются обобщѐнные знания, что приводит, в конце концов, к абстрактным знаниям, для которых нет прямого прообраза во внешнем мире.

Вывод на знаниях зависит от модели, которая используется для их предс тавления. Появления в памяти информационной системы новых фактов и сведений может повлиять на смену исходных аксиом в процессе вывода. Еще одна особенность вывода на знаниях – неполнота сведений о предметной области и протекающих в ней процессах, неточность входной информации. А это означает, что выводы в системах, основанных на знаниях, носят не абсолютно достоверный характер, как в традиционных логических системах, а приближенный, правдоподобный характер. Такие выводы требуют развитого аппарата вычисления оценок правдоподобия и методов оперирования ими.

Решающим аспектом разработки интеллектуальной системы, предопределяющим ее возможности, свойства и характеристики, является выбор модели представление знаний, С точки зрения человека, желательно, чтобы описательные возможности используемой модели были как можно выше. С другой стороны, сложное представление знаний требует специальных способов обработки (усложняется механизм вывода), что затрудняет проектирование и реализацию интеллектуальной подсистемы. Кроме того, используемый в интеллектуальной системе формализм представления знаний определяет характер их получения и накопления, в результате которого создается БЗ, ориентированная на определенную структуру представления, а не на сущность самих знаний. Выбор модели, неадекватной типам знаний, приводит к потере многих существенных деталей прикладной задачи и порождает тривиальный интеллект.

1.3. Представление знаний с использованием формальных логических систем

В основе логических моделей лежит формальная система взаимодействия ряда множеств, задаваемая выражением вида $M = \langle T, P, A, B \rangle$. Множеством T представлены базовые элементы различной природы, например, слова из некоторого ограниченного словаря и т. п. При этом для множества T существует некоторый способ определения принадлежности или непринадлежности произвольного элемента к этому множеству. Процедура проверки принадлежности П(T) должна за конечное число шагов дать положительный или отрицательный ответ на вопрос, является ли элемент x частью множества T.

Множество P определяет набор правил, при помощи которых из элементов множества T образуются синтаксически правильные совокупности. Например, из слов ограниченного словаря строятся синтаксически правильные фразы. Декларируется существование процедуры П(P), с помощью которой за конечное число шагов можно получить ответ на вопрос, является ли совокупность X синтаксически правильной.

Во множестве синтаксически правильных совокупностей выделяется некоторое подмножество A, элементы которого называются аксиомами. Для этого множества существует своя процедура П(A), с помощью которой для любой синтаксически правильной совокупности можно получить ответ на вопрос о принадлежности ее к множеству A.

Множество B определяет набор правил вывода. Применяя эти правила к элементам подмножества A, можно получать новые синтаксически правильные совокупности, к которым снова можно применять правила из B. Так формируется множество выводимых в данной формальной системе совокупностей. Если имеется процедура П(B), при помощи которой можно определить для любой синтаксически правильной совокупности, является ли она выводимой, то соответствующая формальная система называется

разрешимой. Это показывает, что именно правило вывода является наиболее сложной составляющей формальной системы.

Для знаний, входящих в базу знаний, можно считать, что множество *A* образуют все информационные единицы, которые введены в базу знаний извне, а при помощи правил вывода из них выводятся новые производные знания. Другими словами, формальная система представляет собой генератор порождения новых знаний, образующих множество выводимых в данной системе знаний. Это свойство логических моделей позволяет хранить в базе лишь те знания, которые образуют множество *A*, а все остальные знания получать из них по правилам вывода.

Основными преимуществами формальных логических моделей являются наличие формального аппарат вывода новых фактов (знаний) из известных фактов (знаний), возможность контроля целостности. Однако в данной модели знания трудно структурировать и поэтому к предметной области предъявляются высокие требования и ограничения.

1.4. Сетевые модели представления знаний

Сетевые модели формально можно задать в виде

$$H = \langle I, C_1, C_2, ..., C_n, G \rangle,$$

где I – множество информационных единиц; C_1, C_2 , ..., C_n — множество типов связей между информационными единицами; G задает отображение между информационными единицами, входящими в I, посредством связей из заданного набора их типов.

В зависимости от используемых в сетевых моделях типов связей различают классифицирующие сети, функциональные сети и сценарии.

В классифицирующих сетях используются отношения структуризации. Такие сети позволяют в базах знаний вводить разные иерархические отношения между информационными единицами.

Функциональные сети характеризуются наличием функциональных отношений. Их часто называют вычислительными моделями, т. к. они

позволяют описывать процедуры вычисления одних информационных единиц через другие.

В сценариях используются каузальные отношения, а также отношения типов «средство–результат», «орудие–действие» и т. п.

Если в сетевой модели допускаются связи различного типа, то ее обычно называют семантической сетью. *Семантическая сеть* – информационная модель предметной области, имеющая вид ориентированного графа, вершины которого соответствуют объектам предметной области, а дуги (рёбра) задают отношения между ними. Наиболее часто в семантических сетях используются следующие отношения:

- связи типа «часть – целое» («класс – подкласс», «элемент – множество» и т. п.);
- функциональные связи (определяемые обычно глаголами «производит», «влияет»...);
- количественные (больше, меньше, равно...);
- пространственные (далеко от, близко от, за, под, над...);
- временные (раньше, позже, в течение...);
- атрибутивные связи (иметь свойство, иметь значение);
- логические связи (И, ИЛИ, НЕ);
- лингвистические связи и др.

Для всех семантических сетей справедливо разделение по арности и количеству типов отношений.

По количеству типов отношений сети могут быть *однородными* и *неоднородными*. Однородные сети обладают только одним типом отношений. В неоднородных сетях количество типов отношений больше одного. Классические иллюстрации данной модели представления знаний представляют именно такие сети. Неоднородные сети представляют больший интерес для практических целей, но и большую сложность для исследования. Неоднородные сети можно представлять как переплетение древовидных многослойных структур.

По арности выделяют сети с *бинарными* и *n-арными* отношениями. Типичными являются сети с бинарными отношениями (связывающими ровно два понятия). Бинарные отношения очень просты и удобно изображаются на графе в виде стрелки между двух концептов. Кроме того, они играют исключительную роль в математике. На практике, однако, могут понадобиться отношения, связывающие более двух объектов – n-арные. При этом возникает сложность в изображении подобной связи на графе. Концептуальные графы снимают это затруднение, представляя каждое отношение в виде отдельного узла.

Проблема поиска решения в базе знаний типа семантической сети сводится к задаче поиска фрагмента сети, соответствующего некоторой подсети, отражающей поставленный запрос к базе.

Достоинство семантических сетей – наглядность представления знаний, с их помощью удобно представлять причинно-следственные связи между элементами (подсистемами), а также структуру сложных систем. Недостаток таких сетей – сложность вывода, поиска подграфа, соответствующего запросу.

1.5. Продукционные модели представления знаний

В продукционных моделях используются некоторые элементы логических и сетевых моделей. Из логических моделей заимствована идея правил вывода, которые здесь называются продукциями, а из сетевых моделей – описание знаний в виде семантической сети. В результате применения правил вывода к фрагментам сетевого описания происходит трансформация семантической сети за счет смены ее фрагментов, наращивания сети и исключения из нее ненужных фрагментов.

В общем случае продукционную модель можно представить в следующем виде:

$$i = \langle S, P, A \Rightarrow B, Q \rangle,$$

где i – имя продукции;

S – сфера применения продукции;

P – условие применимости ядра продукции;

$A \Rightarrow B$ – ядро продукции, в котором А – условие ядра (или антецендент), В – заключение ядра (или консеквент) , « $\Rightarrow$ » – знак логической секвенции (или следования);

Q – постусловие продукционного правила, актуализирующиеся при положительной реализации продукции.

В продукционных моделях процедурная информация явно выделена и описывается иными средствами, чем декларативная информация. Вместо логического вывода, характерного для логических моделей, в продукционных моделях появляется вывод на знаниях. Вывод на такой базе знаний бывает *прямой* или *обратный*. В системе продукций с обратными выводами с помощью правил строится дерево «И/ИЛИ», связывающее в единое целое факты (посылки) и доказываемое (опровергаемое) утверждение. Оценка этого дерева на основании фактов, имеющихся в базе данных, и есть логический вывод. Оценка заключается в том , что необходимо найти ту посылку, наличие или отсутствие которой в наибольшей степени подтвердит или опровергнет рассматриваемое утверждение. В системе продукций с прямым выводом известна посылка, нужно получить результат.

Наиболее распространенными являются системы продукций с прямым выводом. Они состоят из базы правил, включающей набор продукций (правил вывода), базы данных, в которой содержится множество фактов, и интерпретатора для получения логического вывода. База данных и база правил составляют базу знаний, а интерпретатор соответствует механизму логического вывода.

Продукционная модель чаще всего применяется в промышленных экспертных системах. Она привлекает разработчиков своей наглядностью, высокой модульностью, легкостью внесения дополнений и изменений и простотой механизма логического вывода.

Вместе с тем, продукционной модели присущ ряд недостатков. Основной причиной недостатков продукционных моделей является нехватка строгой теоретической основы. При задании модели предметной области в виде совокупности продукций нельзя быть уверенным в ее полноте и непротиворечивости. Так, при накоплении достаточно большого числа (порядка нескольких сотен) продукций они начинают вследствие необратимости дизъюнкций противоречить друг другу. В этом случае разработчики начинают усложнять систему, включая в неё модули нечёткого вывода или иные средства разрешения конфликтов, — правила по приоритету, правила по глубине, эвристические механизмы исключений, возврата и т. п. *Механизм исключений* означает, что вводятся специальные правила-исключения. Их отличает большая конкретность в сравнении с обобщёнными правилами. При наличии исключения основное правило не применяется. *Механизм возвратов* же означает, что логический вывод может продолжаться в том случае, если на каком-то этапе вывод привёл к противоречию. Просто необходимо отказаться от одного из принятых ранее утверждений и осуществить возврат к предыдущему состоянию.

Еще один недостаток продукционной модели – это несоответствие структуры знаний системы структуре знаний человека. В частности, структура базы знаний продукционной системы не позволяет описывать метазнания и свойственную человеческому мышлению нечеткую логику.

1.6. Фреймовые модели представления знаний

Фреймовая модель представления знаний основана на фреймовой теории, предложенной М.Минским в 1974 г., и представляющей собой систематизированную в виде единой теории психологическую модель памяти человека и его сознания. Важным моментом во фреймовой теории является понятие фрейма – однажды определенной структуры данных для представления некоторого концептуального объекта. Информация, относящаяся к конкретному фрейму, содержится в слоте. Все фреймы

объединяются в иерархическую структуру, интегрирующую в себе декларативные и процедурные знания. Данная структура отображает целостный образ знаний, которому свойственна иерархичность концептуального представления.

Основным отличием фреймовых моделей является жесткая фиксация структуры информационных единиц, которая называется протофреймом. В общем виде она выглядит следующим образом:

(Имя фрейма:
Имя слота 1 (значение слота 1)
Имя слота 2 (значение слота 2)
. . . . . .
Имя слота K (значение слота K)).

Значением слота могут быть числа или математические соотношения, тексты на естественном языке или программы, правила вывода или ссылки на другие слоты данного фрейма или других фреймов. Таким образом фреймы объединяются в сеть. Свойства фреймов наследуются сверху вниз, то есть от вышестоящих к нижестоящим.

При конкретизации фрейма ему и слотам присваиваются конкретные имена и происходит заполнение слотов. Незаполненный фрейм называется *протофреймом*, а заполненный – *фреймом-экземпляром*. Переход от исходного протофрейма к фрейму-экземпляру может быть многошаговым за счет постепенного уточнения значений слотов. Связи между фреймами задаются значениями специального слота с именем «Связь». Таким образом, во фреймовых моделях в представлении знаний практически объединены все основные особенности моделей остальных типов.

Слот может содержать не только конкретное значение, но и имя процедуры, позволяющей вычислить его по заданному алгоритму, а также одну или несколько продукций (эвристик), с помощью которых это значение определяется. В слот может входить не одно, а несколько значений. Иногда

этот слот включает компонент, называемый *фасетом,* который задает диапазон или перечень его возможных значений. Фасет указывает также граничные значения заполнителя слота.

Помимо конкретного значения в слоте могут храниться процедуры и правила, которые вызываются при необходимости вычисления этого значения. Среди них выделяют *процедуры-демоны* и *процедуры-слуги*. Первые запускаются автоматически при выполнении некоторого условия, а вторые активизируются только по специальному запросу.

Совокупность фреймов, моделирующая какую-либо предметную область, представляет собой иерархическую структуру, в которую фреймы собираются с помощью родовидовых связей. На верхнем уровне иерархии находится фрейм, содержащий наиболее общую информацию, истинную для всех остальных фреймов. Фреймы обладают способностью наследовать значения характеристик своих родителей, находящихся на более высоком уровне иерархии. Эти значения могут передаваться по умолчанию фреймам, находящимся ниже них в иерархии, но если последние содержат собственные значения данных характеристик, то в качестве истинных принимаются именно они. Это обстоятельство позволяет без затруднений учитывать во фреймовых системах различного рода исключения.

Различают *статические* и *динамические* системы фреймов. В системах первого типа фреймы не могут быть изменены в процессе решения задачи, а в системах второго типа это допустимо.

Основным преимуществом фреймов как модели представления знаний является то, что она отражает концептуальную основу организации памяти человека, а также ее гибкость и наглядность. Однако разрозненные части информации, объединенные во фрейм, не могут быть выстроены в последовательность высказываний. Иначе говоря, языки описания знаний во фреймовой модели не являются языками, родственными естественным, а ближе к изобразительным средствам. Также недостатком фреймовой модели является отсутствие специального механизма управления выводом, в связи с

чем разработчики должны реализовать данный механизм с помощью присоединенных процедур.

ГЛАВА 2. МОДЕЛИ И МЕТОДЫ ОНТОЛОГИЧЕСКОГО ИНЖИНИРИНГА

2.1. Предпосылки появления онтологической модели представления знаний и перспективы использования

Основным результатом исследований в области искусственного интеллекта является не только разработка различного вида интеллектуальных систем, но и создание технологий, позволяющих быстро разрабатывать интеллектуальные системы, имеющие практическую ценность. Составляющими таких технологий являются

- формальная теория искусственного интеллекта;
- методы проектирования интеллектуальных систем;
- средства автоматизации проектирования;
- средства информационной поддержки разработчиков;
- средства компьютерной поддержки управления коллективной разработкой.

Анализ современных технологий искусственного интеллекта показывает, что наряду с впечатляющими достижениями они имеют целый ряд серьезных недостатков. К числу таких недостатков, в частности, относится отсутствие общего унифицированного решения проблемы семантической совместимости информационных систем, порожденное разнообразием применяемых разработчиками моделей, методов, средств представления знаний. Это обусловливает высокую трудоемкость создания комплексных интегрированных ИС. Наиболее остро проблема совместимости ИС проявляет себя при разработке web-ориентированных систем. В данной области широко известны работы Томаса Грубера, который

занимался исследованиями в части создания механизма взаимодействия баз знаний, обеспечивающего интерпретацию знаний, импортированных из разных источников, на уровне семантики. В рамках решения данной проблемы он предложил подход, согласно которому формальные модели баз знаний должны были максимально просты и легко понимаемы не только их интерпретаторами, но и всеми разработчиками. Доступность семантики для пользователей можно обеспечить за счет выбора соответствующего формата представления знаний. Однако для прикладных программ семантика должна быть представлена в формальной, машинно-обрабатываемой форме. Поэтому Грубер разделил описание знаний на две составляющие:

- онтологию, описывающую предметную область в виде иерархической структуры, содержащей описание всех концептов предметной области, их связей и правил, принятых в предметной области.
- каноническую форму, описывающую знания на языке логики предикатов.

Каждая интеллектуальная система, ядром которой является база знаний, может предоставлять несколько таких описаний, соответствующих различным областям хранящихся в ней декларативных знаний, и выступать, таким образом, как хранилище библиотеки онтологий. При этом библиотеке онтологий уже не обязательно быть интеллектуальной системой, достаточно просто предоставлять сервис по передаче онтологий по требованию.

Составление описания декларативного знания обычно требует большой работы и определенных навыков. Для обозначения этой работы, а также ее результата, Грубер ввел в обиход специальный термин «концептуализация». Описание он называл «спецификацией». Таким образом, онтология по Груберу определяется как *спецификация концептуализации*.

Введенное Грубером разделение спецификаций знаний на две составляющие (каноническую форму и онтологию) не очень удобно, т.к. приходится описывать одни и те же знания два раза. Современные языки

описания онтологий позволяют совместить эти формы спецификаций в единое целое. Сейчас под онтологией понимается любое описание декларативных знаний, сделанное на формальном языке и снабженное некоторой классификацией специфицируемых знаний, позволяющей человеку удобно воспринимать их. Любое такое описание должно включать в себя представление декларативных знаний в виде иерархии объектов (классов), только в этом случае это описание может считаться онтологией. Такой способ описания знаний объединяет в себе другие известные модели представления знаний. Таким образом, онтология – это иерархическая структура конечного множества понятий, описывающих заданную предметную область, такая, что:

2) структура представляет собой онтограф, вершинами которого являются понятия, а дугами – семантические отношения между ними;

3) понятия и отношения интерпретируются в соответствии с общезначимыми функциями интерпретации, взятыми из источников знаний заданной предметной области;

4) определение понятий и отношений выполняется аксиомами и ограничениями области действия;

5) формально онтограф описывается на одном из языков описания онтологий;

6) функции интерпретации и аксиомы описаны в некоторой подходящей формальной теории.

Перспективы использования онтологий при описании баз знаний можно подразделить на следующие категории:

- улучшение взаимодействия;
- унификация обмена данными;
- формализация процессов спецификации;
- повышение надежности и обеспечения многократности использования.

Улучшение взаимодействия связано с ожиданием уменьшение терминологической и концептуальной путаницы и неоднозначности понимания на основе осуществления следующих мероприятий:

- Создание в конкретной области человеческой деятельности (среде) унифицирующего нормативного ядра понятий, которое позволило бы достичь однозначного семантического толкования основных объектов и процессов этой области. Предполагается, что это нормативное ядро является семантической основой для порождения, переопределения и интерпретации новых понятий.
- Создание однозначно понимаемого множества отношений между понятиями нормативного ядра, допускающего исследование динамических и статических аспектов среды, влияния на нее различных факторов, вывод и планирование ситуаций.
- Обеспечение совместимости онтологий, разработанных различными коллективами путем широкого использования полисинонимии и ретранслируемости.
- Обеспечение возможности коллективной работы по согласованию и унификации онтологий и их нормативного ядра.

Унификацию обмена данными связывают с созданием интегрированных инструментальных программных средств, построенных на основе использования нормативного ядра и множества отношений. Эти средства должны обеспечивать возможность обмена данными для созданных или создаваемых на базе онтологий систем моделирования сред и выступают как средство межъязыкового общения, как своеобразный декларативный язык, на который переводятся другие языки и из которого перевод осуществляется в индивидуальный язык пользователя. Эти средства должны иметь в своем составе поддерживаемые, развивающиеся и доступные для использования извне библиотеки онтологий.

Формализацию процессов спецификации, повышения надежности и обеспечения многократности использования связывают с ролью онтологий,

которую они призваны играть для развития систем моделирования сред. Язык онтологии выступает в этом случае как средство спецификации таких систем и является декларативным. Роль онтологии зависит от степени выразительности, формализованности и других свойств декларативного языка онтологии. Повышение надежности систем моделирования связывают с возможностью и удобством полуформального и формального анализа декларативного описания на языке онтологии. При этом, говоря о формальном анализе, полагают, что описание допускает формальный вывод (доказательство) наличия тех или иных свойств среды. Обеспечение многократности использования предполагает наличие в онтологии метауровня, позволяющего настроить онтологию на конкретную задачу применения, определить степень ее пригодности для решения конкретной задачи и модифицировать или расширить ее, если это необходимо.

Наиболее характерной сферой применения онтологий в настоящее время является представление знаний в Интернете. Круг связанных с этим вопросов весьма широк и включает в себя мультиагентные системы, автоматическое извлечение знаний из текстов на естественном языке, поиск информации, интеллектуальное аннотирование, автоматическое составление авторефератов и прочее.

2.2. Формальная модель онтологии

Онтология описывает знания о некоторой предметной области с помощью набора понятий (классов), связанных определенным набором отношений (свойств), а также набора функций, необходимых для ограничения интерпретации и использования понятий. Формальной моделью онтологии $\mathbf{O}$ является упорядоченная тройка вида:

$$\mathbf{O} = < X, R, F >$$

где:

X – конечное непустое множество классов предметной области, которую представляет онтология $\mathbf{O}$;

$\boldsymbol{R}$ – конечное множество отношений между классами;

$\boldsymbol{F}$ – конечное множество функций интерпретации (аксиоматизации), заданных на классах и/или отношениях онтологии $\mathbf{O}$

Классы онтологии представляют собой понятия предметной области. Формально класс интерпретируется как множество объектов предметной области – экземпляров классов. Например, для предметной области «Университет» такими классами являются «Факультет», «Группа», «Студент», «Дисциплина» и др.

Между классами могут быть установлены определенные связи, называемые свойствами. Формально свойства представляют собой бинарные отношения на классах. Базовыми типами свойств являются «класс-подкласс», «часть-целое», «экземпляр-класс», «причина-следствие», «похоже на» и т. п. Например, Студент_Группа, Группа_Факультет и т.д.

Роль функции интерпретации может играть словесное пояснение термина (аннотация), формула для вычисления значения термина, алгоритмическое описание, а также определение в виде логической формулы. Аксиомы используются для моделирования утверждений, которые всегда являются истинными. Аксиомы могут быть включены в онтологию для разных целей, например, для определения комплексных ограничений на аргументы отношений, для проверки корректности информации, описанной в онтологии, или для вывода новой информации.

Рассмотрим частные случаи онтологии.

Пусть $\boldsymbol{R} = \emptyset$ и $\boldsymbol{F} = \emptyset$. Тогда онтология **O** трансформируется в простой словарь, т.е. конечный список терминов:

$$\mathbf{O} = \boldsymbol{V} = < \boldsymbol{X}, \emptyset, \emptyset >.$$

Такая вырожденная онтология может быть полезна для спецификации, пополнения и поддержки словарей предметной области, но онтологии-словари имеют ограниченное использование, поскольку не вводят эксплицитно смысла терминов. Хотя в некоторых случаях, когда используемые термины принадлежат очень узкому (например, техническому)

словарю и их смыслы уже заранее хорошо согласованы в пределах определенного (например, научного) сообщества, такие онтологии применяются на практике. Известными примерами онтологии этого типа являются индексы машин поиска информации в сети Интернет.

Пусть $\boldsymbol{R} = \{ \boldsymbol{is\_a} \}$ – отношение строгого порядка на множестве X, устанавливающее иерархию классов предметной области; $\boldsymbol{F} = \emptyset$. Тогда онтология превращается в таксономию – иерархическую систему понятий

$$\mathbf{O} = \boldsymbol{T} = < X, \{is\_a\}, \emptyset > .$$

Отношение **is_a** имеет фиксированную заранее семантику и позволяет организовывать структуру понятий онтологии в виде дерева. Такой подход имеет свои преимущества и недостатки, но в общем случае является адекватным и удобным для представления иерархии понятий.

По своей функциональной полноте и степени формальности различают три вида онтологий: простая, полная (или строгая) и множество промежуточных или неполных онтологий.

Простая онтология – это такая онтология, в которой $\boldsymbol{R} = \emptyset$ и $\boldsymbol{F} = \emptyset$. Она служит (в основном) для однозначного восприятия научным сообществом понятий в соответствующей прикладной области.

Строгая или полная *онтология* ($\boldsymbol{R} \neq \emptyset$ и $\mathbf{F} \neq \emptyset$) – это такая онтология, в которой множества классов и отношений максимально полные, а к функциям интерпретации добавляются аксиомы, определения и ограничения. При этом описания всех компонент представлены на некотором формальном языке, доступном для их интерпретации компьютером. Схема формальной модели полной онтологии описывается четвѐркой:

$$\mathbf{O} = \langle \boldsymbol{X}, \boldsymbol{R}, \boldsymbol{F}, \boldsymbol{A}(\boldsymbol{D},\boldsymbol{G}) \rangle$$

где:

$\boldsymbol{X}$ – конечное непустое множество классов;

$\boldsymbol{R}$ – конечное множество отношений между классами;

$\boldsymbol{F}$ – конечное множество функций интерпретации, заданных на классах и/или отношениях;

$\boldsymbol{A}$ – конечное множество аксиом, которые используются для записи всегда истинных высказываний (определений и ограничений);

$\boldsymbol{D}$ – множество дополнительных определений понятий;

$\boldsymbol{G}$ – множество ограничений, определяющих область действия понятийных структур.

Полная онтология является формальным выражением кон цептуальных знаний о предметной области и представляет собой понимают базу знаний специального типа, которую можно разделять, отчуждать, и самостоятельно использовать в рамках рассматриваемой предметной области.

Множество *промежуточных* или неполных *онтологий* ($\boldsymbol{R} = \emptyset$, $\boldsymbol{F} \neq \emptyset$; $\boldsymbol{R} \neq \emptyset$, $\boldsymbol{F} = \emptyset$) возникает, когда для каждого концепта (или их большей части) добавлены аксиомы и определения. Одним из распространѐнных вариантов неполной онтологии является структура вида $\boldsymbol{R} \neq \emptyset$, $\boldsymbol{F} = \emptyset$, где множество $\boldsymbol{F}$ в явном виде отсутствует в предположении, что концепты $\boldsymbol{x} \in \boldsymbol{X}$ общеизвестны (определены по умолчанию) либо (и) достаточно полно интерпретированы отношениями R .

2.3. Типы онтологий. Формальная модель онтологической системы

Выделение типов онтологий будет приведено на основе таких признаков, как универсальность и выразительность. Уровень универсальности определяет масштаб онтологии, а выразительность – детальность ее описания.

По *уровню универсальности* выделяют четыре типа онтологий:

- Онтологии верхнего уровня (метаонтологии);
- Онтологии, ориентированные на предметную область (предметные онтологии);
- Онтологии, ориентированные на конкретную задачу (онтологии задач);

- Прикладные онтологии.

Под *метаонтологией* (онтологией верхнего уровня) понимается самая общая онтология, которая является общей для всех областей знаний. Метаонтология описывает наиболее общие понятия и отношения, которые не зависят от предметных областей, например, «Объект», «Процесс», «Свойство», «Значение» и т.д. Считается, что такая онтология существует, и разные группы специалистов предъявляют свои варианты. Существует несколько крупных онтологий верхнего уровня: Cyc, DOLCE, SUMO. Метаонтологии связаны с мировоззрением, и, естественно, эта область близка к исследованиям в философии и лингвистике.

Предметная онтология описывает словарь, связанный с предметной областью за счет специализации терминов, введенных в метаонтологии. Предметная онтология содержит понятия, описывающие конкретную предметную область, отношения, семантически значимые для данной предметной области, и множество интерпретаций этих понятий и отношений. Предметная онтология обычно применяется для того, чтобы уточнить понятия, определённые в метаонтологии (если используется), и/или определить общую терминологическую базу предметной области. Предметные онтологии используются только внутри одной предметной области.

Онтология задач описывает концептуальную модель конкретной задачи. Такая онтология содержит в качестве понятий типы решаемых задач, а отношения специфицируют декомпозицию задач на подзадачи. Онтологии задач содержат наиболее специфичную информацию.

Прикладные онтологии описывают концепты, зависящие как от конкретной предметной области, так и от задач, которые в них решаются. Концепты в таких онтологиях часто соответствуют ролям, которые играют объекты в предметной области в процессе выполнения определенной деятельности.

Центральной идеей системно-онтологического подхода является разработка онтологических средств поддержки решения прикладных задач – полифункциональной онтологической системы. Под формальной моделью онтологической системы S_O понимают триплет вида:

$$S_o = \langle O^{meta}, \{O^{sub}, O^{task}\}, I \rangle,$$

где O^{meta} онтология верхнего уровня (метаонтология);

$\{O^{sub}, O^{task}\}$ – множество предметных онтологий O^{sub} и онтологий задач предметной области O^{task};

I – модель машины вывода, ассоциированной с онтологической системой S_O.

В модели S_O имеются три онтологические компоненты:

- метаонтология;
- предметная онтология;
- онтология задач.

Как указывалось выше, метаонтология оперирует общими концептами и отношениями, которые не зависят от конкретной предметной области. Тогда на уровне метаонтологии мы получаем интенсиональное описание свойств предметной онтологии и онтологии задач. Онтология метауровня является статической, что дает возможность обеспечить здесь эффективный вывод. Использование системы онтологии и специальной машины вывода позволяет решать в такой модели различные задачи. Расширяя систему моделей $\{O^{sub}, O^{task}\}$, можно учитывать предпочтения пользователя, а, изменяя модель машины вывода, вводить специализированные критерии релевантности получаемой в процессе поиска информации и формировать специальные репозитории накопленных данных, а также пополнять при необходимости используемые онтологии.

По *степени выразительности* выделяют следующие типы онтологий:

1. Контролируемый словарь – конечный список терминов, которые используются, чтобы пометить единицы информации так, чтобы они могли быть более легко восстановлены поиском. Контролируемые словари

уменьшают двусмысленность, врожденную от нормальных естественных языков, где тому же самому понятию можно дать различные имена и гарантировать последовательность.

2. Глоссарий, представляющий собой список терминов с их значениями. Значения описываются в виде комментариев на естественном языке. Это дает больше информации, поскольку люди могут прочесть такой комментарий и понять смысл термина. Интерпретации терминов могут быть многозначными.

3. Тезаурус – словарь терминов, в котором явно указаны отношения между терминами (род-вид, целое-часть, синонимические отношения и др.).

4. Таксономия – древовидная иерархия терминов, использующихся для классификации объектов. Эта разновидность онтологий включает точное определение отношения Подкласс_Класс (обозначаемого как is_a). В таких системах строго соблюдается транзитивность отношения is_a: если А является подклассом класса В, то каждый подкласс класса А также является подклассом класса В. Строгая иерархия классов необходима при использовании наследования для процедуры логического вывода.

5. Онтологии, в которых установлено отношение класс-экземпляр. Такие онтологии содержат на нижнем уровне конкретные объекты предметной области – экземпляры.

6. Онтологии, в которых установлены свойства типов данных – связи между характеристиками классов и значениями типов данных. Например, класс «Продукт» может иметь характеристику «цена» с типом данных «вещественное число». Свойства типов данных бывают особенно полезными, когда они определены на верхних уровнях иерархии и наследуются подклассами. Так, в потребительской иерархии класс «Продукт» может иметь свойство «цена», которое получат все его подклассы.

Большей выразительностью обладают онтологии, включающие ограничения на область значений свойств. Значения свойств берутся из некоторого предопределенного множества (целые числа, символьные

константы) или из подмножества концептов онтологии (множество экземпляров данного класса, множество классов). Можно ввести дополнительные ограничения на то, что может заполнять свойство. Например, для свойства «сделан из» класса «Предмет одежды» значения могут быть ограничены экземплярами класса «Материал».

7. Онтологии, содержащие описание классов с простыми логическими или математическими ограничениями на отношения.

В целом с необходимостью описывать более сложные факты выразительные средства онтологии (и ее структура) усложняются. Например, может потребоваться заполнить значение какого-либо свойства экземпляра, используя математическое выражение, основанное на значениях других свойств данного экземпляра или значениях свойств других экземпляров. Многие онтологии позволяют объявлять два и более класса дизъюнктивными (непересекающимися). Это означает, что у данных классов не существует общих экземпляров. Некоторые языки описания онтологий позволяют делать произвольные логические утверждения о концептах – аксиомы, а также фиксировать утверждения на языке логики предикатов первого порядка.

2.4. Характеристики качества онтологий интеллектуальных информационных систем

Цель создания онтологий – обеспечить поддержку деятельности по накоплению, разделению и повторному использованию знаний. Исходя из этой цели, существует ряд критериев качества, которым должна отвечать онтология:

1. Ясность – онтология должна ясно и однозначно передавать смысл введенных терминов. Определения должны быть объективными, хотя мотивация введения терминов может определяться ситуацией или требованиями вычислительной эффективности. Для объективизации определений должен использоваться четко

фиксированный формализм, при этом целесообразно задавать определения в виде логических аксиом.

2. Согласованность – все определения должны быть логически непротиворечивы, а все утверждения, выводимые в онтологии, не должны противоречить аксиомам.
3. Расширяемость – онтология должна быть спроектирована так, чтобы обеспечивать использование разделяемых словарей терминов, допускающих возможность монотонного расширения и/или специализации без необходимости ревизии уже существующих понятий.
4. Минимальное влияние кодирования. В онтологической системе должен быть реализован принцип совместного использования онтологий, который предполагает спецификацию онтологии на уровне представления, а не символьного кодирования. Запись такой спецификации на общепринятом и платформонезависимом языке описания онтологий можно передать для использования любому программному агенту.
5. Минимальные онтологические соглашения. Онтология должна содержать только наиболее существенные предположения о моделируемой предметной области, но их должно быть достаточно для описания знаний, которые предполагается совместно использовать.

Для обеспечения успеха при разработке, внедрении и использовании онтологии, ее оценка должна проводиться на протяжении всего жизненного цикла по заранее определенным критериям, зависящим от назначения онтологии и средств ее поддержки. Оценка онтологии включает в себя сбор информации об определенных её параметрах, проверку соответствия этой информации некоторым требованиям и оценку пригодности онтологии для конкретной задачи. Некоторые свойства онтологий не связаны с конкретной задачей, другие требуют оценки отношений между онтологией и её

предметной областью, окружением или особыми вариантами её использования и могут оцениваться исключительно в контексте сценария использования. Разнообразие потенциальных вариантов использования онтологий не позволяет создать универсальный набор критериев оценки. Таким образом, не существует одного метода оценки, применимого ко всем онтологиям. Несмотря на это, можно выделить некоторые методы оценки, необходимые для большинства онтологий. Оценка качества онтологии предполагает её оценку в качестве модели предметной области, пригодной для понимания человеком, модели для машинной обработки и онтологии, как части сложной программной системы. Главными характеристиками здесь являются следующие:

1. Разборчивость – оценивается пригодность онтологии к пониманию её человеком.
2. Точность – оценивается, насколько точно онтология представляет предметную область.
3. Мастерство – оценивается, насколько хорошо построена онтология, и в какой мере соблюдены оригинальные организационные решения.
4. Степень соответствия – оценивается, насколько подходит онтологическая модель под решаемую задачу.
5. Простота развертывания – оценивается, отвечает ли онтология требованиям системы, частью которой является.

Для обеспечения разборчивости недостаточно того, чтобы онтология была читаема специалистами-онтологами. Все целевые пользователи системы должны быть в состоянии интерпретировать её содержимое (экземпляры, классы, отношения и т.д.), важное для их деятельности. Разборчивость не предполагает прямого распознавания онтологии пользователями, однако документация на нее должна быть понятна всей целевой аудитории. Для этого может требоваться наличие множественных определений для одного понятия (например, на разных языках).

Разборчивость особенно важна для онтологий, используемых в качестве управляемого словаря, однако и для онтологий, применяющихся в качестве внутренней структуры программных систем, также желательно обеспечение разборчивости, так как поддержка онтологий осуществляется людьми, не всегда причастными к её созданию.

Точность отражает корректность описания предметной области как в аксиомах, так и документации к онтологии. Мастерство отвечает за аккуратность исполнения онтологии от наличия синтаксических ошибок до вопросов правильности реализации философского базиса онтологии.

Требования к соответствию задачам и развертываемости онтологии зависят от сценария её использования. Онтология является моделью конкретной предметной области, которая разрабатывается для конкретной цели. Следовательно, не существует правильной онтологии определенной предметной области. Онтология представляет собой абстракцию конкретной области и всегда имеет жизнеспособные альтернативы. Что именно будет включено в эту абстракцию, определяется в зависимости от задач приложения, для которого она разрабатывается, и от будущих потенциальных вариантов ее использования. Лучшее решение зависит от предполагаемого приложения и ожидаемых расширений. Поэтому оценка должна производиться с учетом требований к системе, частью которой является онтология.

2.5. Жизненный цикл создания онтологии

Жизненный цикл (ЖЦ) любой онтологии состоит из ряда процессов, в ходе которых онтология зарождается, специфицируется, адаптируется, развертывается, используется и поддерживается. Происходят эти процессы параллельно или последовательно, однажды за ЖЦ или повторяются несколько раз, частично зависит от того, как выполнялось создание онтологии. Поскольку онтологии создаются для разных задач, отдельные стадии жизненного цикла могут отсутствовать для одних онтологий и

присутствовать для других. Данный факт не позволяет создать единую общую модель ЖЦ онтологии с четко обозначенной последовательностью этапов ЖЦ. ЖЦ конкретных онтологий являются упрощенными частными случаями, зависящими от специфики создания конкретной онтологии. Оценка онтологий производится на всех стадиях ЖЦ, что позволяет установить, насколько онтология удовлетворяет требованиям последующей стадии. Рассмотрение упрощенного ЖЦ позволяет выделять стадии, общие для всех онтологий:

1. Стадия формирования требований.
2. Стадия онтологического анализа.
3. Стадия проектирования онтологии.
4. Стадия проектирования информационной системы, основанной на онтологии.
5. Стадия разработки онтологии.
6. Стадия разработки информационной системы, основанной на онтологии.
7. Стадия развертывания.
8. Стадия промышленной эксплуатации

1. *Стадия формирования требований*. На данной стадии рассматриваются и анализируются все возможные сценарии применения будущей онтологии, на основе анализа формулируются начальные требования. Как правило, сценарий использования понятен из задач онтологии. На начальном этапе требования могут быть представлены фрагментарно и касаться только отдельных задач. Одним из способов формулирования требований является использование вопросов проверки компетенции, т.е. вопросов, на которые должна отвечать онтология. Эти вопросы формулируются на естественном языке и соответствуют запросам, которые должна поддерживать онтология в выбранных сценариях использования. Результатом этапа формирования требований является документ, который должен отвечать на следующие вопросы:

1. Зачем нужна данная онтология? (причина появления, ожидаемая польза)
2. Каковы предполагаемые сценарии использования?
3. Какие группы пользователей должны быть в состоянии понимать определенные части онтологии?
4. Каков масштаб онтологии?
5. Есть ли существующие онтологии и стандарты, пригодные для использования?
6. Каковы вопросы компетенции?
7. Отражают ли вопросы компетенции все сценарии использования онтологии?
8. Каковы требования рабочей среды?
9. Какие ресурсы стоит принять во внимании при создании онтологии? (Например, имеющиеся базы данных, модели данных, глоссарии, словари, схемы, таксономии, онтологии, стандарты, доступ к экспертам предметной области).

2. *Стадия онтологического анализа.* Задачей стадии онтологического анализа является выделение ключевых сущностей онтологии (экземпляров, классов и отношений между ними), а так же отождествление их с терминологией выбранной предметной области. Итоги этой работы, как правило, представляются в неформальном виде, пригодном для восприятия как для онтологов, так и для экспертов в предметной области (например, в виде таблиц и диаграмм). Результат онтологического анализ должен определять следующую информацию:

- Важные сущности в пределах предполагаемой предметной области.
- Важные характеристики сущностей, включая отношения между ними, неоднозначность описаний и свойства, важные для предметной области в рамках выбранного сценария использования.

- Терминологию, используемую для обозначения этих сущностей и предоставление достаточного количества контекстуальной информации для устранения неоднозначности многозначных терминов.

Результат анализа дает исходную информацию для проектирования и создания онтологии. Общая схема этапов онтологического анализа представлена на рис. 1

Рис. 1 – Стадия онтологического анализа

Следует заметить, что процесс построения онтологии здесь распадается на серию подпроцессов по созданию промежуточных представлений. При этом выполнение отдельных подпроцессов не последовательное, а определяется полнотой и точностью уже накопленных знаний.

Сначала строится глоссарий терминов, включающий все термины (концепты и их экземпляры, атрибуты, действия и т. п.), важные для предметной области, и их естественно-языковые описания. Понятия в онтологии должны быть близки к объектам (физическим или логическим) и

отношениям в интересующей предметной области. Наиболее часто это существительные (объекты) или глаголы (отношения) в предложениях, которые описывают предметную область. Когда глоссарий терминов достигает существенного объема, строятся деревья классификации концептов. Как правило, при этом используются отношения типа Класс_Подкласс и некоторые другие таксономические отношения. Таким образом, идентифицируются основные таксономии предметной области, а каждая таксономия дает в конечном счете онтологию.

Следующим шагом является построение диаграмм бинарных отношений, целью создания которых является фиксация отношений между концептами одной или разных онтологий. В дальнейшем эти диаграммы могут послужить исходным материалом для интеграции разных онтологий.

После построения представлений, фиксированных выше, для каждого дерева классификации концептов строятся:

1. Словарь концептов, содержащий все концепты предметной области, экземпляры таких концептов, атрибуты экземпляров концептов, отношения, источником которых является концепт, а также (опционально) синонимы и акронимы концепта.

2. Таблица бинарных отношений для каждого отношения, исходный концепт которого содержится в классификационном дереве. Для каждого отношения фиксируется его имя, имена концепта-источника и целевого концепта, инверсное отношение и т. п. характеристики.

3. Таблица атрибутов экземпляра для каждого экземпляра из словаря концептов. Основные характеристики здесь следующие: имя атрибута, тип значения, единица измерения, точность, диапазон изменения, значение «по умолчанию», атрибуты, которые могут быть выведены с использованием данного, формула или правило для вывода атрибута и др.

4. Таблица атрибутов класса для каждого класса из словаря концептов с аналогичными характеристиками.

5. Таблица логических аксиом, в которой даются определения концептов через всегда истинные логические выражения. Определение каждой аксиомы включает ее имя, естественно-языковое описание, концепт, к которому аксиома относится, атрибуты, используемые в аксиоме, логическое выражение, формально описывающее аксиому, и др.

6. Таблица констант, где для каждой константы указывается ее имя, естественно-языковое описание, тип значения, само значение, единица измерения, атрибуты, которые могут быть выведены с использованием данной константы, и т. п.

7. Таблица формул для каждой формулы, включенной в таблицу атрибутов экземпляра. Каждая таблица этого типа, помимо собственно формулы, должна специфицировать ее имя, атрибут, выводимый с помощью этой формулы, естественно-языковое описание, точность, ограничения, при которых возможно использовать формулу, и др.

8. Деревья классификации атрибутов, которые графически показывают соответствующие атрибуты и константы, используемые для вывода значения корневого атрибута и формулы, применяемые для этого. По сути дела, эти деревья используются для проверки того, что все атрибуты, представленные в формуле, имеют описания и ни один из атрибутов не пропущен.

9. Таблица экземпляров для каждого входа в словарь концептов. Здесь специфицируется имя экземпляра, его атрибуты и их значения.

Выходные параметры стадии онтологического анализа оцениваются в соответствии со следующими критериями:

1. Задокументированы ли все важные термины предметной области?
2. Выделены ли все сущности, важные в масштабе онтологии?
3. Согласны ли эксперты в предметной области с результатами онтологического анализа?
4. Является ли документация односмысленной в степени, достаточной для согласованного использования терминологии?

3. *Стадия проектирования онтологии*. На данной стадии разрабатывается проект онтологии – определяются принципы организации онтологии, проектируется структура онтологии, выбираются языки построения онтологии и язык запросов. Одним из способов организации онтологии является использование существующих онтологий, которые доступны в электронном виде и могут быть импортированы в используемую среду проектирования онтологии. Структура определяет разбиение онтологии на модули и то, как эти модули будут взаимодействовать между собой. Имеющиеся онтологии могут быть использованы в структуре в качестве модулей новой онтологии. Поведение модулей может быть описано по ответам на вопросы компетенции. Эти вопросы, специфичные для конкретных модулей, как правило, получают из вопросов для целой онтологии.

Следует отметить, что результаты проектировочных решений могут привести к конфликту требований выразительности и производительности онтологии. Данное противоречие можно разрешить путем создания отдельных справочных и операционных онтологий. Справочная онтология описывает предметную область во всей полноте, необходимой для решения задачи. Операционная онтология создается на её основе с возможным введением некоторых упрощений с целью увеличения производительности.

Оценка результатов фазы проектирования онтологий состоит в формировании документа, отвечающего на следующие вопросы:

1. Достаточны ли описательные возможности языка онтологии для удовлетворения требований к онтологии?
2. Достаточно ли выразителен язык запросов для формализации вопросов компетентности?
3. Поддерживает ли выбранный язык все необходимые возможности онтологии (например, если онтология оперирует вероятностями, язык должен описывать вероятностную информацию)?

4. Является ли каждый добавленный в онтологию класс или концепт подклассом или экземпляром класса верхнего уровня?
5. Определены ли правила именования концептов и соблюдаются ли они?
6. Требует ли проект создания нескольких отдельных онтологических модулей? Если да, то описывают ли модули в совокупности потребную предметную область.
7. Описано ли в проекте, будут ли повторно использоваться созданные онтологии и как?
8. Все ли модули онтологии имеют определенные (неформально) вопросы компетенции?
9. Определено ли для каждого модуля, какие типы сущностей в нем представлены?
10. Определено ли для каждого модуля, как он будет оцениваться и кто за это будет отвечать?
11. Позволяет ли структура (конструкция) онтологии избегать добавления возможностей или содержимого, не относящегося к удовлетворению требований к онтологии?

4. *Стадия проектирования информационной системы, основанной на онтологии*. На данной стадии принимаются решения, влияющие на внедряемость онтологии в информационную систему. Оценка результатов стадии состоит в ответах на следующие вопросы:

1. Какие операции будут выполняться с использованием онтологий?
2. Какие компоненты будут выполнять эти операции?
3. Как бизнес-требования, разработанные на стадии определения требований, применимы к этим специфическим операциям и компонентам?
4. Будут ли и, если будут, то какие изменения и добавления в онтологии после развертывания системы?

5. Какие интерфейсы будут задействованы во внесении добавлений? Как будут тестироваться эти интерфейсы относительно измененной онтологии? Каким требованиям нужно будет отвечать?
6. Какие источники данных будут использоваться совместно с онтологией? Через какие интерфейсы будет происходить обмен информацией?
7. Как будет создаваться, оцениваться и поддерживаться онтологий? Какие для этого необходимы инструменты?
8. Если онтология будет иметь модульную структуру и/или создаваться распределенными разработчиками, как это будет поддерживаться?

5. *Стадия разработки онтологии*. Стадия включает четыре главных процесса:

- концептуальное моделирование,
- формализация вопросов компетенции,
- формальное моделирование,
- операционная адаптация.

Эти процессы обычно повторяются в цикле для отдельных модулей и для онтологии в целом. На практике они часто выполняются без четких границ между ними, тем не менее, важно понимать их концептуальные различия.

В процессе концептуального моделирования происходит доработка результатов онтологического анализа. Для каждого модуля происходит увязка терминологии с основными онтологическими концептами. Оценка результатов концептуального моделирования проводится на основе ответов на следующие вопросы:

1. Находятся ли в модели исключительно сущности выбранной предметной области?
2. Хорошо ли определены все концепты (например, не ссылаются сами на себя)?

3. Хорошо ли документирована интерпретация неопределенных экземпляров, классов и отношений?
4. Пригодна ли документация для понимания её экспертами предметной области?

Сценарии и вопросы компетенции формализуются на основе результатов концептуального моделирования. Вопросы, предназначенные для оценки результатов данного процесса, следующие:

1. Охватывают ли вопросы компетенции все сценарии использования?
2. Отражает ли формализованный вопрос цель вопроса изначального?

В процессе формального моделирования содержимое информационной модели записывается на каком-либо онтологическом языке, а затем конкретизируется аксиомами. Онтология, созданная или выбранная для повторного использования, оценивается по трем критериям:

- точность отображения предметной области;
- техническое совершенство (качество выполнения онтологии и то, насколько она отвечает требованиям, сформулированным в фазе онтологического проектирования);
- адекватность (насколько представление онтологии отвечает требованиям к её использованию).

В силу того, что оценка точности отображения предметной области зависит от понимания предметной области, оценка требует проверку содержания онтологии экспертами предметной области. Оценка технического совершенства происходит на основе оценки проектных и методологических решений стадии онтологического проектирования. В Поскольку проектирование онтологий – это молодая отрасль, существует слишком мало универсальных критериев оценки. Одним из способов проверки соответствия онтологии требованиям предметной области являются формализованные вопросы компетентности и сценарии. Соответствие онтологии требованиям предметной области также можно оценить путем проведения тестов.

В ходе операционной адаптации справочная онтология адаптируется к операционным задачам для получения операционной онтологии. Главным вопросом является, сможет ли новая онтология обеспечить требуемую производительность. Это может требовать упрощения онтологии или других оптимизационных процедур (например, реструктурирования). В некоторых случаях операционная онтология пишется на другом языке и с другой семантикой, нежели ссылочная.

6. Стадия разработки информационной системы, основанной на онтологии. На этом этапе происходит интеграция онтологии и других компонентов в подсистемы и систему в целом в соответствие с планом, разработанном на стадии проектирования. Стадия разработки и интеграции отнесена к ЖЦ онтологии по причине того, что, как правило, результат от использования онтологии получается только при её взаимодействии с другими компонентами информационной системы. Оценка результатов происходит, в частности, по ответам на следующие вопросы:

1. Успешно ли внедрена онтология?
2. Достигнут ли результат от внедрения онтологии, описанный в документации?

7. Стадия развертывания. На этом этапе онтология переходит от развития и интеграции к работе. Развертыванию обычно предшествует несколько циклов доработки, несмотря на это, она все равно может подвергаться дополнительным тестам перед внедрением. Оценка на этом этапе может проводиться с привлечением третьей стороны или включать в себя полную симуляцию работы системы. Целью таких испытаний является исключение негативного влияния внедрения онтологии на бизнес-процессы. Когда все испытания пройдены, онтология вводится в эксплуатацию и становится доступной для использования. Вопросы на этапе развертывания:

1. Отвечает ли онтология всем требованиям стадии разработки?
2. Существуют ли риски внедрения онтологии?

3. Использовались ли вопросы компетентности предыдущих этапов для создания регрессионных тестов?
4. Были ли проведены регрессионные тесты для оценки возможности снижения операционных показателей от внедрения системы? Если некоторое снижение прогнозируется, будет ли оно компенсировано положительным эффектом от внедрения онтологии?

8. *Стадия промышленной эксплуатации*. Этот этап фокусируется на поддержании имеющихся функций, а не на добавлении новых. Когда онтология (или ее версия) находится на стадии эксплуатации и технического обслуживания, происходит сбор информации о результатах оперативного использования онтологии. При выявлении проблем или фактов снижения операционных показателей могут проводиться микро-доработки для устранения возникших проблем. Одновременное выявление новых случаев использования, желаемых улучшений и новых требований, которое может произойти в течение того же периода использования, не следует рассматривать как часть технического обслуживания деятельности; скорее они являются предпосылками для разработки требований к будущей версии, расширению онтологии или созданию нового модуля. Вопросы к этапу промышленной эксплуатации:

1. Все ли регрессивные тесты пройдены успешно? Если нет, то какие меры принимаются?
2. Существуют ли проблемы функционирования системы? Если да, то вызваны ли они онтологией или проблемы в другом?
3. Если проблема в онтологии, может ли она быть решена без серьезного изменения онтологии?
4. Если проблема не может быть устранена без серьезной доработки онтологии, стоит ли продолжать её внедрение?

Как показывает анализ стадий ЖЦ онтологий, все они хорошо коррелируют с теми стадиями, которые выделены и используются при построении баз знаний. И это не случайное совпадение, а закономерность,

связанная с тем, что онтология — это, по существу, БЗ специального вида. Поэтому, как и в случае построения баз знаний, здесь используется концепция быстрого прототипирования, а специфика проявляется в тех конкретных процессах, которые реализуют рассмотренные выше процедуры. При этом:

- планирование выполняется до начала собственно разработки;
- контроль и гарантии качества осуществляются в процессе разработки;
- большая часть операций по накоплению знаний и их оценке выполняется на стадии концептуализации для того, чтобы предотвратить распространение ошибок на фазу реализации;
- интеграция не должна рассматриваться как интеграция на стадии реализации. Напротив, она выполняется в процессе разработки.

Как правило, онтология в процессе своего ЖЦ проходит через выделенные стадии более одного раза. Идентификация стадий ЖЦ онтологии позволяет кластеризировать действия вокруг целей, входов и выходов узнаваемого типа. Более того, модель ЖЦ наглядно демонстрирует зависимость одних стадий ЖЦ от других, например, качество онтологии напрямую зависит от того, насколько грамотно были сформулированы требования к ней. Зависимости между стадиями инвариантны для всех онтологий, несмотря на их различия между собой.

2.6. Особенности проектирования предметных онтологий

Предметная онтология является формальным выражением концептуальных знаний о предметной области и по своей значимости сопоставима с базой знаний интеллектуальной информационной системы. Онтология определяет общеупотребительные, семантически значимые «понятийные единицы знаний», которыми оперируют исследователи и разработчики знаниеориентированных информационных систем. Предметная

онтология – это концептуальная модель реального мира и её компоненты должны отражать эту реальность.

Пусть $\mathbf{O} =< \boldsymbol{X}, \boldsymbol{R}, \boldsymbol{F} >$ – предметная онтология, где:

$\boldsymbol{X}$ –множество классов предметной области, которую представляет онтология $\mathbf{O}$;

$\boldsymbol{R}$ – множество свойств онтологии;

$\boldsymbol{F}$ – множество функций интерпретации (аксиоматизации), заданных на классах и/или свойствах онтологии $\mathbf{O}$.

Построение множества $\boldsymbol{X}$ считается наиболее важным моментом при разработке онтологии предметной области. Оно должно быть обязательно не пустым. Для хорошо проработанных предметных областей за основу элементов множества *X* может быть взято содержимое подходящих словарей. В противном случае следует составить полный список терминов, в котором указать

- чем является каждый термин – понятием-классом предметов или конкретным понятием;
- указать для каждого термина возможные существенные отношения с другими терминами из списка;
- описать возможные существенные свойства понятий.

Известно, что в любой предметной области существуют термины-синонимы. Для них в онтологии отводится только одно понятие, в аксиомах которого может быть указан синонимический ряд терминов. Другими словами, синонимы одного и того же понятия не представляют различные классы. Далее уточняется и определяется окончательный список классов-понятий, имена которых будут входить в разрабатываемую онтологию и являться вершинами онтографа. При этом следует придерживаться единых правил присваивания имён понятиям и их свойствам. В результате должен быть получен полный список существенных для заданной предметной области (и предполагаемых приложений) понятий и их машинно-интерпретируемых формулировок.

Следующим шагом является определение свойств онтологии и построение онтологического графа. Построение онтографа является специальным видом классификации понятий предметной области – онтологической классификацией. В процессе соотнесения классов и построения иерархии следует учитывать, что:

- прямые подклассы в иерархии должны располагаться на одном уровне обобщения;
- класс может быть подклассом нескольких классов, и тогда он может наследовать свойства от всех этих классов;
- если класс имеет только один прямой подкласс, то, возможно, при моделировании допущена ошибка или онтология неполная;
- если у данного класса есть более 7-10 подклассов, то, возможно, необходимы дополнительные промежуточные классы.

В заключение данного подэтапа следует соотнести разработанные классы и их иерархии с результатами анализа предметной области. В частности, уточнить зависимости для конкретных пар экземпляров.

Множество аксиом онтологии состоит из множества определений и множества ограничений на понятия. Определения записываются в виде тождественно истинных высказываний, которые могут быть взяты, в частности, из словарей предметной области. В них могут быть указаны дополнительные взаимосвязи между классам. Ограничения – это аксиомы, которые ограничивают множество экземпляров, которые могут являться членами класса.

Следует учесть, что из полного списка отобранных в онтологию терминов не все представляют понятия. Существуют термины, которые соответствуют свойствам определённых классов-понятий. Такие свойства следует привязать к описанию самого общего класса, обладающего ими. А подклассы этого класса будут наследовать указанное свойство. Свойства понятий имеют определённые значения, такие как тип значений, мощность значений, разрешённые значения (для данного класса) и другие. Например,

мощность значений можно описать: с единичной мощностью, мощностью без ограничений и мощностью с некоторым допустимым интервалом.

На основе построенных множеств можно синтезировать концептуальную модель предметной области, получить формальное описание разработанной онтологии на одном из языков описания онтологий, а также графическое представление онтографа. При этом следует помнить, что в настоящее время не существует единственного правильного способа или методологии разработки онтологий. Онтология всегда отражает взгляд аналитика, т.е. всегда субъективна.

2.7. Отображение онтологий

Повторное использование существующих онтологий на сегодняшний день стало стандартным этапом в процессе разработки новой онтологии. Практически никто не ведет разработку онтологий «с нуля». Причем при разработке конкретного приложения чаще всего используется не одна существующая онтология, а их комбинация.

Интеграция онтологий представляет собой деятельность по созданию новой онтологии или фрагмента онтологии из двух и более исходных онтологий. Задача отображения онтологий возникла из необходимости интеграции онтологий, разработанных независимо друг от друга и имеющих, таким образом, свой собственный словарь. Также отображение онтологий является неотъемлемой частью большинства задач согласования онтологий, таких как выравнивание онтологий, модификация одной онтологии для достижения однородности с другой и так далее.

Отображение онтологий – это установление соответствия между концептами нескольких онтологий, или, другими словами, нахождение семантических связей подобных элементов из разных онтологий. Две (или более) онтологии могут по-разному описывать одну и ту же предметную область или близкие предметные области с точки зрения разных сообществ. Онтология задаёт подразумеваемую семантику для понятий предметной

области и определят онтологический контекст, в котором работает сообщество.

С наиболее общей точки зрения важность задачи отображения онтологий обусловлена тем фактом, что установление факта подобия сущностей в разных онтологиях означает извлечение из этих онтологий дополнительных знаний. Разные контексты использования онтологий, созданных разными сообществами, отражаются на особенностях подходов к спецификации понятий. В результате, семантика понятий в контекстах, описанных разными онтологиями, может быть сходной при различных подходах к описанию их структуры: составу, ограничениям и степени детализации. Проблема отображения онтологий заключается в том, что, во-первых, сущности, имеющие одинаковые имена, могут иметь разный смысл, а во-вторых, сущности, имеющие одинаковый смысл, могут иметь разные имена.

Отображение онтологий разделяется на две подзадачи:

1. Локальное отображение сущностей, подразумевающее независимое установление соответствий между двумя сущностями рассматриваемых онтологий.

2. Глобальное отображение сущностей, под которым подразумевается пересмотр локальных отображений с учетом отображений всех остальных элементов.

Современные методы установления отображения онтологий носят междисциплинарный характер. Выделяют лингвистические, статистические, структурные и логические методы. Для обеспечения максимальной точности отображения сущностей используют все четыре метода.

Лингвистические методы определяют сходство между сущностями исходной и целевой онтологии на основе сравнения их имен путем оценки количества совпадающих символов, нахождения общих частей слов и др., или на основе анализа синонимичных терминов. Для выявления

синонимичных терминов могут использоваться существующие словари общей и профессиональной лексики, тезаурусы.

Структурные методы включают в себя:

1. Анализ внутренней структуры исходной и целевой онтологии. Сущностей с похожими областями определения и областями значений может быть достаточно много, поэтому данные методы используются только для формирования кластеров сходных понятий и требуют сочетания с другими методами.

2. Анализ внешней структуры, включающий анализ сходства по иерархическим связям и анализ сходства по перекрестным связям.

Рассмотрим анализ сходства по иерархическим связям. Оценка схожести двух сущностей двух онтологий может быть основана на позициях данных сущностей в иерархии классов. Если две сущности двух онтологий схожи, то их «соседи» также как-то схожи. Такое утверждение может использоваться по-разному и порождает ряд возможных критериев (признаков) для сходства двух сущностей:

- их прямые супер-сущности (или все супер-сущности) уже являются схожими;
- их сущности-братья (или все их сущности-братья) уже являются схожими;
- их прямые сущности-потомки (или все их сущности-потомки) уже являются схожими;
- все их сущности-листья (сущности, не имеющие потомков, находящиеся в дереве, корнем которой является рассматриваемая сущность) уже являются схожими;
- все (или большинство) сущности на пути от корня к рассматриваемой сущности уже являются схожими.

Определение сходства между сущностями может быть основано также на анализе перекрестных связей сущностей. Если класс A_1 связан с классом B_1 свойством типа R_1 в одной онтологии, а класс A_2 связан с B_2 свойством

типа R_2 в другой онтологии, и если известно, что B_1 и B_2 – схожи, R_1 и R_2 – схожи, можно предположить схожесть A_1 и A_2. Подобным образом можно говорить и сходстве типов свойств R_1 и R_2, если известно, что A_1 и A_2 – схожи, B_1 и B_2 – схожи.

Для оценки экстенсионального соответствия классов используются существующие экземпляры классов. Для установки соответствия между сущностями используются диагностические правила, устанавливающие эквивалентность классов и отношение включения подкласса в класс. Анализ экстенсионала позволяет также идентифицировать классы-роли, когда возникает два разных класса для описания одного экстенсионала.

Логический анализ основан на выявлении родовых классов сопоставляемых классов и анализе наложенных на них ограничений. Например, в одной онтологии может существовать класс «Микро-компания», который является видовым классом для класса «Компания» с наложенным ограничением на «Число сотрудников» <5. В другой онтологии может существовать класс «Малое предприятие», который является видовым классом для класса «Фирма» с наложенным ограничением на «Число работников» <10. При анализе соответствия между классами «Микро-компания» и «Малое предприятие» выявляются родовые классы «Компания» и «Фирма». При наличие информации о соответствие данных классов производится сравнение ограничений наложенных на данные родовые классы. Для этого сравниваются свойства классов «Число сотрудников» и «Число работников», если данные свойства схожи, то проводится сравнение наложенных ограничений <5 (сотрудников) и <10 (работников). В результате делается заключение, что «Микро-фирма» $\subset$ «Малое предприятие». Ограничением данного метода является потребность в «якорях» – сущностях, которые либо заведомо эквивалентны в двух сопоставляемых онтологиях, либо являются разделяемыми сущностями в некоторой сторонней онтологии. После получения локальных соответствий между сущностями определяется глобальное соответствие между сущностями.

Практические рекомендации по расстановке приоритетов между результатами различных способов локального анализа:

- При наличие баз знаний, включающих в себя экземпляры отображаемых онтологий, приоритетное значение имеют результаты экстенсионального анализа.
- При наличие «якорей» в отображаемых онтологиях приоритетное значение имеют результаты логического анализа.

Однако следует помнить, что результаты любого анализа следует согласовывать с результатами, полученными с использованием других видов анализа. Особенно важно такое согласование при установке соответствия между классами ролями, исполнители которых (экстенсионал) могут выполнять одновременно несколько ролей.

2.8. Онтологии как основа семантического веба

Количество информации, которую создает мировое сообщество, растет с каждым годом. Открытость информационного поля теоретически обеспечивает свободный и быстрый доступ к данным. Однако у такой всеобщей доступности есть и обратная сторона – чтобы получить информацию, ее нужно сначала найти.

Получение информации об интересующем объекте в подавляющем большинстве случаев сводится к использованию интернет ресурсов поисковых систем (ПС). Поиск информации поисковым роботом представляет собой процесс выявления в индексированном множестве ПС релевантных документов, т.е. таких, которые удовлетворяют заранее определенному запросу. Основная задача состоит в том, чтобы на конкретный запрос пользователя ПС провела обработку информации с последующим ранжированием найденных веб-ресурсов по их релевантности. Пользователь не может описать системе признаки искомого объекта, поскольку принцип поиска ПС базируется на тексте и ключевых словах. Фактически пользователю сложно найти данные, о которых он еще не знает,

а для их получения необходимо ввести в строку запроса информацию, содержащуюся в ответе. Приходится различать формальную релевантность и содержательную релевантность, причем, если формальная релевантность, повторяющая в листе выдачи ПС форму запроса, но не передающая изначально заданной содержательной сути сегодня достижима, то реализация содержательного соответствия документа смыслу запроса является в большинстве случаев нерешенной. Таким образом, основной проблемой нахождения смыслового соответствия документа пользовательскому запросу является разработка и реализация подходов, основанных на семантическом поиске.

Семантический поиск является одним из методов информационного поиска и представляет собой процесс поиска документов по их смысловому содержанию. Основой семантического поиска служат заранее установленные отношения между символами и объектами, которые они обозначают. Можно выделить два основных вида семантического поиска. Полнотекстовый поиск – поиск по всему содержимому документа с использованием предварительно построенных индексов. Поиск по метаданным – это поиск по неким атрибутам документа, которые описывают определенные объекты, поддерживаемые системой. Например, автор, адрес, название организации и т. д. Именно использование метаданных сегодня широко применяется в относительно новой концепции развития интернет под названием Семантический веб.

Концепцию Семантического веба (Semantic Web) выдвинул Тим Бернерс-Ли, один из основоположников Worl-Wide Web и председатель WWW-консорциума (W3C) на международной конференции XML-2000, прошедшей в 2000 году в Вашингтоне. Основная идея этого проекта заключается в организации такого представления данных в сети, чтобы допускалась не только их визуализация, но и их эффективная автоматическая обработка программами разных производителей. Путем таких радикальных преобразований концепции уже традиционного Веб-пространства

предполагается превращение его в систему семантического уровня. По замыслу создателей семантический веб должен обеспечить «понимание» информации компьютерами, выделение ими наиболее подходящих по тем или иным критериям данных, и уже после этого – предоставление информации пользователям. При автоматической обработке информации в рамках семантического веба взаимодействующие друг с другом сервисы на основе анализа смысловых связей между объектами и понятиями, хранящимися в сети Интернет, должны отбирать лишь ту информацию, которая будет реально полезна пользователям.

Таким образом, семантический веб – это расширение существующей Веб-сети, в котором информации приписывается точное, формально определенное значение, что дает возможность компьютерам «понимать» хранящуюся в Сети информацию и обрабатывать ее на семантическом уровне. Семантический веб формируется на базе Веб-сети путём стандартизации представления информации в виде, пригодном для машинной обработки.

Основной акцент концепции семантического веба делается на работе с метаданными, однозначно характеризующими свойства и содержание веб-ресурсов, вместо используемого в настоящее время текстового анализа документов. Эта концепция была принята и продвигается Консорциумом W3C. Для ее внедрения предполагается создание сети документов, содержащих метаданные о веб-ресурсах. Тогда как сами ресурсы предназначены для восприятия человеком, метаданные используются поисковыми роботами (агентами) для проведения однозначных логических заключений о свойствах этих ресурсов.

Семантический веб в математической форме представляет собой разновидность графа, где роль вершин выполняют понятия базы знаний, а направленные дуги задают отношения между ними. Таким образом, строится семантическая сеть, которая отражает семантику предметной области в виде понятий и отношений. Идея состоит в том, чтобы глобальной семантической

сетью было подмножество систем, которые замкнуты на специфичных путях достижения достаточного удобства для агентов.

Семантический веб основан на следующих принципах проектирования:

1) обеспечение доступности в вебе структурированных и полуструктурированных данных, представленных в стандартных форматах;

2) обеспечение доступности в вебе не только наборов данных, но и отдельных элементов данных и отношений между ними;

3) описание предполагаемой семантики таких данных с помощью формализма, обеспечивающего возможность машинной обработки этой семантики.

Данные принципы проектирования реализованы с помощью следующих семантических технологий:

1. В качестве модели данных для описания объектов и отношений между ними используются размеченные графы. Объекты представляются как вершины графа, а отношения между ними – как дуги. В качестве формализма для представления таких графов используется фреймворк описания ресурсов RDF (Resource Description Framework).
2. Для идентификации отдельных элементов данных и отношений между ними, которые включаются в наборы данных, используются веб-идентификаторы URI (Uniform Resource Identifier). Использование веб-идентификаторов отражено в стандарте языка RDF. В последних версиях семантического веба для идентификации ресурсов используется IRI (Internationalized Resource Identifier) – международный идентификатор ресурса, представляет собой Unicode-строку и соответствует синтаксису, определенному в стандарте URI.
3. В качестве модели данных, позволяющей формально представить предполагаемую семантику данных, используются онтологии.

Данная модель представляется с помощью таких формализмов, как язык RDF Shema и язык веб-онтологий OWL (Web Ontology Language), в которых URI (IRI) используются для идентификации терминов и их свойств.

Существование стандартов для описания данных (RDF) и их атрибутов (RDF Shema) позволяет создавать инструменты обработки информации из многочисленных источников. То, насколько глубоко различные приложения могут обмениваться данными и использовать их, принято называть синтаксическим взаимодействием сетей. Чем более стандартизированными и распространенными являются эти инструменты работы с данными, тем выше степень синтаксического взаимодействия сетей.

Синтаксическое взаимодействие сетей требует определенного преобразования между терминами, для чего, в свою очередь, необходим контентный анализ. Два поисковых агента могут использовать различные идентификаторы для обозначения одного и того же понятия и им необходимо объяснить, что два конкретных термина используются ими для обозначения одного и того же. Такой контентный анализ требует формальных и подробных спецификаций моделей доменов, которые определяют используемые термины и их связи. Подобные формальные модели доменов принято называть онтологиями. Они определяют модели данных в терминах классов, подклассов и свойств. Проще говоря, онтология – это документ, формально задающий отношения между терминами.

Наиболее типичными видами онтологий в вебе являются таксономия и набор правил вывода. Таксономия определяет классы объектов и от ношения между ними. Например, понятие адрес может быть определено как разновидность понятия местонахождение, а код города можно задавать применительно лишь к местонахождениям и так далее. Большое количество отношений между инди видами можно задать путем приписывания классам определенных свойств и позволяя подклассам наследовать эти свойства. Правила вывода, задаваемые в онтологиях, дают еще больше возможностей.

В рамках онтологии можно записать такое правило: «Если объект А соответствует некоторому объекту В, а в объекте С фигурирует объект А, то этому объекту С тоже соответствует объект В». Поисковый робот не «понимает» в полном смысле этого слова ничего из всей этой информации, но теперь он уже может манипулировать терминами гораздо более эффективно с тем, чтобы стать полезным и осмысленным для пользователя.

Онтологический язык Web (Web Ontology Language), рекомендуемый W3C, добавляет больше словарных возможностей для описания свойств и классов, чем RDF или схема RDF. В частности, он позволяет описывать связи между классами, мощность множества, равенство, более богатую типологию свойств и их характеристики.

Рабочей группой W3C по доступу к данным разработан язык запросов SPARQL. SPARQL имеет SQL-подобный синтаксис, определяет запросы в терминах шаблонов графа, которые сравниваются с направленным графом, представляющим данные RDF. SPARQL предоставляет возможности для запроса необходимых и необязательных шаблонов, а также для их объединения и разделения. Результат сравнения также может быть использован для конструирования нового графа RDF с использованием отдельного шаблона. Используя такие точки доступа SPARQL, агенты могут запрашивать удаленные RDF данные и формировать новые RDF графы без какой-либо локальной обработки.

Одним из первых серьезных и популярных проектов, основанным на принципах семантической паутины, стал проект «Дублинское ядро», реализуемый инициативной организацией Dublin Core Metadata Initiative (DCMI). Это открытый проект, цель которого разработать стандарты метаданных в формате RDF, независящие от платформ и подходящие для широкого спектра задач.

В то время как совокупность ресурсов и их метаданных можно считать статической частью семантической паутины, ее динамическую часть представляют так называемые семантические веб-сервисы – законченные

элементы программной логики с однозначно описанной семантикой, доступные через интернет и пригодные для поиска, композиции и выполнения. Консорциум W3C предполагает использование для описания веб-сервисов тех же языков разметки, что и для статической части семантической паутины, а также онтологии OWL-S, описывающей базовую терминологию предметной области. Онтология OWL-S состоит из четырех онтологий – онтологии сервиса, онтологии модели сервиса, онтологии процесса и онтологии базы.

Потенциальная выгода от использования семантических веб-сервисов заключается в возможности автоматического поиска программными агентами подходящих сервисов для решения поставленных задач. Тем не менее, сложность этой задачи в ее общей формулировке пока позволяет добиваться некоторых положительных результатов только в узкоспециализированных отраслях, явным образом выигрывающих от внедрения сервисо-ориентированной архитектуры (SOA), например, в интеграции корпоративных приложений.

Несмотря на очевидную актуальность Semantic Web существуют сложности ее практической реализуемости. Во-первых, необходимость описания метаданных приводит к дублированию информации. Правда, этот недостаток семантической паутины был главным толчком к созданию микроформатов, с помощью которых можно семантически размечать сведения о разнообразных сущностях непосредственно в коде HTML или XHTML. Во-вторых, в семантической паутине для получения ответа на некоторые вопросы совсем не обязательно переходить по ссылке на сайт. Доля поискового трафика на сайты может значительно снизиться, т.к. поисковые системы будут сами отбирать и предоставлять нужную пользователю информацию. Сответственно отпадает необходимость посещать сайт, на котором публикованы материалы и реклама, а значит, коммерческая выгода от привлечения пользователей на сайт уменьшается в разы. Здесь же можно сказать о том, что сохранение анонимности или же

авторских прав на текстовую информацию становится весьма проблематичным. Полнотекстовый семантический поиск является альтернативным подходом к концепции семантической паутины. В настоящее время разрабатываются алгоритмы, которые самостоятельно анализируют содержание интернета и переводят его из текстового представления в объектное.

ГЛАВА 3. ДЕСКРИПТИВНЫЕ ЛОГИКИ КАК ФОРМАЛЬНЫЕ МОДЕЛИ ОНТОЛОГИЙ

3.1. Базовые формализмы дескрипционных логиках

Построение, использование и эволюция онтологий в значительной степени зависит от их семантических свойств и инструментов логического вывода. Важное значение имеют при этом формальные модели описания онтологий и языки представления онтологии, которые бы поддерживали эту модель. Дескрипционные логики (ДЛ) являются семейством языков представления знаний, которые могут быть использованы для записи знаний предметной области формальным способом. ДЛ описывают знания прикладной проблемной области, вначале вводя подходящие понятия (его терминологию), а затем используюя эти понятия для точного описания свойств объектов и экземпляров. Ключевой особенностью ДЛ, отличающей их от предшественников, таких, как семантические сети и фреймы, является то, что это логики, т.е. формальные языки с хорошо определенной семантикой. Другой их отличительной особенностью является наличие логического вывода, который позволяет выявить неявно представленные знания из знаний, которые явно содержатся в базе знаний. Разрешимость и сложность задач вывода зависит от выразительной силы конкретной ДЛ. С одной стороны, очень экспрессивные ДЛ сталкиваются с задачами вывода высокой сложности, которые могут быть даже неразрешимыми. С другой стороны, очень слабые ДЛ (с эффективными процедурами вывода) могут не обладать достаточной выразительностью для представления важных понятий конкретной прикладной задачи. Определение баланса между выразительностью ДЛ и сложностью их задач вывода является одной из наиболее важных проблем в исследованиях ДЛ.

Дескрипционные логики оперируют понятиями концепт и роль, соответствующими в других разделах математической логики понятиям одноместный предикат (или множество, класс) и двуместный предикат (или

бинарное отношение). Концепты используются для описания классов некоторых объектов, например, «Человек», «Женщина». Роль используются для описания двуместных отношений между объектами, например, на множестве людей имеется двуместное отношений «являться родителем».

Концепты и роли ДЛ представляют собой инструмент для записи знаний об описываемой предметной области. Эти знания подразделяются на общие знания о понятиях и их взаимосвязях (так называемые *интенсиональные знания*) и знания об индивидуальных объектах, их свойствах и связях с другими объектами (так называемые *экстенсиональные знания*). Первые более стабильны и постоянны, тогда как вторые более подвержены модификациям. В соответствии с этим делением знания, записываемые с помощью ДЛ, подразделяются на:

- набор терминологических аксиом или TBox. TBox описывает словарь интересующей предметной области, содержащий концепты и роли;
- набор утверждений об индивидах или ABox. ABox содержит утверждения о конкретных представителях (экземплярах) в терминах словаря предметной области.

Вместе они составляют так называемую базу знаний $B = \text{TBox} \cup \text{ABox}$.

Язык для построения описаний является характерной особенностью каждой системы ДЛ, и различные системы различаются их языком описания. Дескриптивный язык имеет теоретико-модельную семантику. Поэтому утверждения в TBox и в ABox могут отождествляться с формулами в логиках первого порядка или, в некоторых случаях, незначительными их расширениями. Системы ДЛ позволяют не только описать терминологию и утверждения, но также предоставляют возможности по выполнению логического вывода с их помощью.

3.2. Синтаксис и семантика логики *ALC*

Дескрипционная логика ALC (от Attributive Language with Complement) была введена в 1991 году и является одной из базовых ДЛ, на основе которой строятся многие другие ДЛ.

Для того чтобы задать какую-либо ДЛ, необходимо задать ее синтаксис и семантику. Синтаксис описывает, какие выражения (концепты, роли, аксиомы и т.п.) считаются правильно построенными в данной логике. Семантика указывает, как интерпретировать эти выражения, т.е. придает им формальный смысл.

Пусть $AC = \{A_1,...,A_m\}$ и $AR = \{R_1,...,R_n\}$ – конечные непустые множества атомарных концептов и атомарных ролей. Множество концептов логики ALC задается следующим индуктивным определением.

- выражения $\top$ (Thing, истина) и $\bot$ (NoThing, ложь) – концепты;
- всякий атомарный концепт A является концептом;
- если C – концепт, то выражение $\neg C$ является концептом и называется *дополнением* концепта C;
- если C и D – концепты, выражения то $C \sqcap D$ и $C \sqcup D$ являются концептами и называются *пересечением* и *объединением*;
- если C – концепт, а R – атомарная роль, то выражения $\exists R.C$ и $\forall R.C$ – концепты;
- никакие другие выражения не являются концептами.

В дальнейшем для формулировки синтаксиса будем использовать более краткую запись. Так, синтаксис для концептов логики ALC в этой записи выглядит следующим образом.

$$\top \mid \bot \mid A \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists R.C \mid \forall R.C,$$

где A – атомарный концепт, R – атомарная роль, C, D – произвольные концепты.

Семантика логики ALC задается с помощью понятия интерпретации. Интерпретация есть пара $I = (\Delta, \cdot^I)$, состоящая из непустого множества Δ,

называемого областью данной интерпретации (доменом), и интерпретирующей функции $\cdot^I$, которая сопоставляет:

- каждому атомарному концепту $A \in AC$ – произвольное подмножество $A^I \subseteq \Delta$;
- каждой атомарной роли $R \in AR$ – произвольное подмножество $R^I \subseteq \Delta \times \Delta$.

Если пара экземпляров принадлежит интерпретации некоторой роли R, т.е. $(e,d) \in R^I$, то говорят, что экземпляр d является R-*последователем* экземпляра e.

Интерпретирующая функция распространяется на множество всех концептов логики *ALC* индукцией по построению концепта:

- $\top$ интерпретируется как весь домен: $\top^I = \Delta$;
- $\bot$ интерпретируется как пустое множество: $\bot^I = \varnothing$;
- дополнение концепта интерпретируется как дополнение множества: $(\neg C)^I = \Delta \setminus C^I$;
- пересечение концептов интерпретируется как пересечение множеств: $(C \sqcap D)^I = C^I \cap D^I$;
- объединение концептов интерпретируется как объединение множеств: $(C \sqcup D)^I = C^I \cup D^I$;
- выражение $\exists R.C$ интерпретируется как множество тех экземпляров, у которых имеется R -последователь, принадлежащий интерпретации концепта C:

  $(\exists R.C)^I = \{e \in \Delta \mid$ существует $d \in \Delta$ такой, что $(e,d) \in R^I$ и $d \in C^I\}$;
- выражение $\forall R.C$ интерпретируется как множество тех экземпляров, у которых все R -последователи принадлежат интерпретации концепта C:

$(\forall R.C)^I = \{e \in \Delta \mid$ для всех $d \in \Delta$ таких, что $(e,d) \in R^I$, выполнено $d \in C^I\}$

Пусть домен Δ – множество людей. Атомарным концептам Женщина, Мужчина, Родитель сопоставим соответственно множество женщин, множество мужчин и множество родителей. Атомарную роль иметь_ребенка интерпретируем двуместным отношением, связывающим всякого родителя с его ребенком. Другими словами, пара (e,d) принадлежит отношению иметь_ребенка, если d является ребенком e. Тогда можно дать интерпретацию следующим концептам:

- Женщина ⊔ Мужчина – множество всех людей;
- Женщина ⊓ Родитель – множество матерей;
- ∃ иметь_ребенка.⊤ – множество людей, кто имеет хотя бы одного ребенка;
- ∀ иметь_ребенка.Женщина – множество людей, все дети которых являются девочками.

Следующие определения применимы не только к логике ALC, но и к любой ДЛ:

- Концепт C называют *выполнимым*, если существует такая интерпретация I, что $C^I \neq \varnothing$. При этом I называют моделью концепта C.
- Концепты C и D называют *эквивалентными* и обозначают $C \equiv D$, если в любой интерпретации I имеем $C^I = D^I$.
- Концепт C называют *вложенным* в концепт D и обозначают $C \sqsubseteq D$, если в любой интерпретации I имеем $C^I \subseteq D^I$.
- Концепты C и D называют *непересекающимися*, если в любой интерпретации I имеем $C^I \cap D^I = \varnothing$.

Очевидно, что справедливы эквивалентности и вложения концептов, являющиеся аналогами законов, выполняющихся для пересечения,

объединения и дополнения множеств. Концепт $\exists R.\bot$ в любой интерпретации обозначает множество элементов, у которых существует R-последователь, принадлежащий пустому множеству $\bot^I = \varnothing$. Так как пустому множеству никакой элемент принадлежать не может, то значит концепт $\exists R.\bot$ обозначает множество элементов, обладающих невыполнимым свойством, т.е. пустое множество. С другой стороны, $\bot$ тоже всегда интерпретируется как пустое множество. Отсюда следует, что имеет место эквивалентность: $\exists R.\bot \equiv \bot$ Аналогично, $\forall R.\top \equiv \top$.

На практике обычно возникают задачи по заданному концепту выяснить его выполнимость, а также по заданным двум концептам выяснить, имеет ли место их эквивалентность, вложение или непересекаемость. Однако нет необходимости создавать независимые алгоритмы для решения этих задач, так как все они сводятся друг к другу, поскольку для любых концептов C и D справедливо:

- C невыполним $\Leftrightarrow C \sqsubseteq \bot$;
- $C \equiv D \Leftrightarrow C \sqsubseteq D \; u \; D \sqsubseteq C$;
- C и D не пересекаются $\Leftrightarrow C \sqcap D \sqsubseteq \bot$.

С другой стороны, очевидно, что для любых концептов C и D справедливо:

- $C \sqsubseteq D \Leftrightarrow$ концепт $C \sqcap \neg D$ невыполним;
- $C \equiv D \Leftrightarrow$ концепты $C \sqcap \neg D \; u \; D \sqcap \neg C$ оба невыполнимы;
- C и D не пересекаются $\Leftrightarrow$ концепт C ⊓ D невыполним.

3.3. Терминология логики ALC

Аксиомой называется выражение вида $C \sqsubseteq D$ или $C \equiv D$, где C и D – произвольные концепты. *Терминологией* (или *TBox*) называется произвольный конечный набор аксиом данного вида.

Следующая совокупность аксиом является примером терминологии:

Человек ≡ Мужчина ⊔ Женщина

Учитель ⊑ Человек

Мать ⊑ Человек ⊓ ∃ иметь_ребенка.⊤

Интуитивно эти аксиомы говорят, что всякий человек является мужчиной или женщиной; быть матерью означает быть человеком и иметь ребенка.

Семантика терминологии определяется естественным образом. Пусть дана интерпретация *I*:

- Аксиома $C \sqsubseteq D$ *истинна* в интерпретации *I*, если $C^I \subseteq D^I$; при этом интерпретацию *I* называют *моделью* данной аксиомы и пишут $I \models C \sqsubseteq D$.
- Аксиома $C \equiv D$ *истинна* в интерпретации *I*, если $C^I = D^I$; при этом интерпретацию *I* называют *моделью* данной аксиомы и пишут $I \models C \equiv D$.
- Интерпретацию *I* называют *моделью* терминологии *T* и пишут $I \models T$, если *I* является моделью всех аксиом из *T*.
- Терминология *TBox* называется *выполнимой,* если она имеет модель.

Пусть *TBox* – произвольная терминология, *C* и *D* – произвольные концепты. Тогда:

- Концепт *C выполним* в терминологии *TBox*, если существует модель *I* терминологии *TBox*, такая что $C^I \neq \varnothing$.
- Концепты *C* и *D эквивалентны* в терминологии *TBox* (обозначение: $TBox \models C \equiv D$), если в любой модели *I* терминологии *TBox* имеем $C^I = D^I$.
- Концепт *C вложен в D в терминологии TBox* (обозначение: $TBox \models C \sqsubseteq D$), если в любой модели *I* терминологии *TBox* имеем $C^I \subseteq D^I$.
- Концепты *C* и *D* называют *непересекающимися* в *TBox*, если в любой модели *I* терминологии *TBox* имеем $C^I \cap D^I = \varnothing$.
- Аксиома α *следует из TBox* (обозначение: $TBox \models \alpha$), если α истинна в любой модели терминологии *TBox*.

- Две терминологии $TBox_1$ и $TBox_2$ называют *эквивалентными* (обозначение: $TBox_1 \equiv TBox$), если у них одни и те же модели, т.е. для любой интерпретации I она является моделью $TBox_1$ тогда и только тогда, когда она является моделью $TBox_2$.

Пусть I – терминология из предыдущего примера. Тогда концепт Женщина $\sqcap$ $\neg$Человек невыполним в I, т.к. ввиду первой аксиомы имеем: $I \models$ Женщина $\sqsubseteq$ Человек. В то же время концепт Мужчина $\sqcap$ Женщина выполним в I. Чтобы этот концепт стал невыполнимым в I, нужно добавить в $TBox$ дополнительные аксиомы.

На практике эксперты в какой-либо предметной области создают терминологию, в которой в виде аксиом фиксируют взаимосвязи основных понятий (концептов и ролей), имеющихся в данной области знаний. После того, как такая система аксиом сформулирована, возникают задачи вывода новых (или, как говорят, неявных) знаний из знаний, заданных явно в терминологии. Одной из первых задач является проверка того, что терминология вообще имеет хотя бы одну модель (т.е. совместна). Далее проверяют, что все атомарные концепты являются выполнимыми в данной терминологии. Если какой-то атомарный концепт оказывается невыполнимым, это обычно является признаком того, что в аксиомах терминологии допущены ошибки. В дальнейшем терминология используется для вывода новых включений и эквивалентностей концептов из уже имеющихся. Соответствующие задачи сводимы друг к другу. Для любых концептов C и D и терминологии $TBox$ справедливы следующие эквивалентности:

- C невыполним в $TBox \Leftrightarrow TBox \models C \sqsubseteq \bot$;
- $TBox \models C \equiv D \Leftrightarrow TBox \models C \sqsubseteq D$ и $TBox \models D \sqsubseteq C$;
- C и D не пересекаются в $TBox \Leftrightarrow TBox \models (C \sqcap D) \sqsubseteq \bot$.

Для любых концептов C и D и терминологии $TBox$ справедливы следующие эквивалентности:

- $TBox \models C \sqsubseteq D \Leftrightarrow$ концепт $C \sqcap \neg D$ невыполним в *TBox*;
- $TBox \models C \equiv D \Leftrightarrow$ концепты $C \sqcap \neg D$ и $D \sqcap \neg C$ оба невыполнимы в *TBox*;
- C и D не пересекаются в $TBox \Leftrightarrow$ концепт $C \sqcap \mathrm{D}$ невыполним в *TBox*.

3.4. Система фактов логики ALC

Терминологии позволяют записывать общие знания о концептах и ролях. Однако обычно требуется также записать знания о конкретных экземплярах: к какому классу (концепту) они принадлежат, какими отношениями (ролями) они связаны друг с другом. Это делается в той части базы знаний, которая называется системой фактов об индивидах или *ABox* (от англ. assertional box). С этой целью, помимо множества *AC* атомарных концептов и множества *AR* атомарных ролей, т.е. имен для классов и отношений, вводится также конечное множество *IN* имён экземпляров. Факты об экземплярах бывают двух видов:

- утверждение о принадлежности индивида a концепту C – записывается как $a{:}C$;
- утверждение о связи двух индивидов a и b ролью R – записывается как aRb.

Определим синтаксис *ABox*. *Системой фактов* называется конечное множество *ABox* утверждений вида $a{:}\ C$ и aRb, где $a,b \in IN$ есть индивиды, C – произвольный концепт, R – роль.

Чтобы задать семантику для *ABox*, надо взять произвольную интерпретацию $I = (\Delta, \cdot^I)$ и расширить ее на экземпляры: каждому экземпляру $a \in IN$ сопоставить элемент области интерпретации $a^I \in \Delta$.

Положим, что факт $a{:}C$ или aRb *верен* в интерпретации I, если $a^I \in C^I$ или $\langle a^I, b^I \rangle \in R^I$ соответственно. При этом I называется *моделью* этого факта, что записывается как $I \models a{:}C$ и $I \models aRb$ соответственно.

Интерпретация *I* называется *моделью* системы фактов *ABox,* если *I* является моделью всех фактов из *ABox*. *ABox* называется *выполнимой* (в терминологии *TBox*), если *ABox* имеет модель (являющуюся одновременно моделью терминологии *TBox*).

Следующая совокупность является системой фактов в языке логики *ALC*:

Мария: Женщина ⊓ Учитель Мария

иметь_ребенка Александр

Александр: Учитель ⊓ ∀иметь_ребенка. ⊥

Здесь Мария и Александр есть имена экземаляров. Интуитивно (другими словами, при «естественной» семантике) эти утверждения означают, что Мария является женщиной, но не учителем, у нее есть ребенок Александр, причем Александр является учителем и не имеет детей.

Иногда дополнительно требуют, чтобы семантика ABox удовлетворяла так называемому «*соглашению об уникальности имен*», означающему, что разным именам индивидов интерпретация должна сопоставлять различные элементы из области интерпретации, то есть чтобы отображение $a \in IN \to a^I \in \Delta$ было инъективным. Это требование вполне естественно; например, именам Мария и Александр из последнего примера логично было бы сопоставлять разные элементы при интерпретации. Однако в дальнейшемпо умолчанию не будем требовать выполнения этого соглашения; если жеоно будет подразумеваться, то будем говорить об этом явно.

3.5. База знаний логики ALC

База знаний логики *ALC* – это пара *B* = (*TBox, ABox*), где *TBox* – произвольная терминология, а *ABox* – произвольная система фактов. Интерпретация *I* называют *моделью* базы знаний *B* и обозначают *I* |= *B*, если

I является одновременно моделью *TBox* и *ABox*. База знаний называется *выполнимой* (или *совместной*), если она имеет модель.

Говорят, что факт α (вида *a: C* или *aRb*) *следует* из *B* и пишут $B \models \alpha$, если α выполняется в любой модели *I* базы знаний *B*.

Дескрипционные логики строятся таким образом, чтобы базы знаний были *разрешимыми* теориями. Более того, обычно требуется, чтобы по теории (базе знаний) и утверждению можно было эффективно (т.е. алгоритмически) установить, является ли данное утверждение следствием данной теории. Сформулируем основные алгоритмические проблемы, которые обычно решаются на практике.

Если имеется лишь терминология *TBox*, то рассматривают следующие алгоритмические проблемы:

T0: совместность терминологии *TBox*;

T1: выполнимость концепта *C* в терминологии *TBox*;

T2: вложение концептов $C \sqsubseteq D$ в терминологии *TBox* ;

T3: классификация терминологии *TBox,* когда требуется для всех атомарных концептов *A, B,* встречающихся в *TBox,* проверить вложение $A \sqsubseteq B$ в *TBox* и в качестве результата выдать так называемую таксономию – частично упорядоченное множество всех атомарных концептов относительно вложения.

Основной проблемой является T1. Остальные проблемы к ней сводятся: T0 сводится к T1, так как совместность терминологии *TBox* равносильна выполнимости концепта $\top$ в *TBox*; T2 сводится к T1; T3 решается путем многократного решения проблемы T2.

Если же даны не только *TBox*, но и *ABox*, то рассматривают следующие логические проблемы:

B0: выполнимость базы знаний *B,* другими словами, выполнимость *ABox* относительно *TBox*;

B1: принадлежность экземпляра a концепту *C* относительно *B,* т.е. проверка того, что $B \models a: C$;

*B*2: выборка экземпляров класса (для заданного концепта *C* найти $\{a \in IN \mid B \models a: C\}$);

*B*3: классификация экземпляра (найти все минимальные по вложению атомарные концепты, содержащие заданный индивид *a* относительно *B)*. Более точно, найти все атомарные концепты *A*, такие что $B \models a: A$ и не существует атомарного концепта A', такого что $B \models a: A'$, $TBox \models A' \sqsubseteq A$ и $TBox \not\models A \sqsubseteq A'$.

Из этих проблем в качестве основной обычно выбирают *TBox B0*. Остальные к ней сводятся. Действительно, B$\models a : C$ тогда и только тогда, когда база знаний $(TBox, ABox \cup \{a : \neg C\})$ невыполнима; тем самым *B1* сведена к *B0*. Проблема *B*2 есть многократно решаемая проблема *B*1. Проблема *B*3 комбинирует в себе проблемы *B*1 и *T*2. Более того, проблема *T*1, а значит и все проблемы из первой группы, тоже сводится к *B*0. Действительно, очевидна эквивалентность: концепт *C* выполним относительно терминологии тогда и только тогда, когда база знаний (T, $\{a:C\}$) выполнима, где *a* есть новый (нигде не встречавшийся) индивид. Тем самым достаточно научиться решать проблему *B*0 выполнимости баз знаний.

3.6. Разрешимость дескрипционной логики. Понятие разрешающего алгоритма

Установим разрешимость логических проблем, перечисленных в предыдущем разделе, для логики *ALC*. Для этого опишем так называемый *табло-алгоритм* (tableau algorithm), который проверяет выполнимость баз знаний в этой логике. Пусть мы хотим проверить, верно ли вложение концептов:

$$\exists R.A \sqcap \neg \forall R.(\neg B \sqcap A) \sqsubseteq \exists R.B.$$

Для этого мы должны проверить на выполнимость концепт

$$C := \exists R.A \sqcap \neg \forall R.(\neg B \sqcap A) \sqcap \neg \exists R.B.$$

Если ответ на этот вопрос будет «да», то ответ на исходный вопрос будет «нет» и наоборот.

Нормализуем концепт C, то есть «пронесем» все отрицания внутрь с тем, чтобы получить эквивалентный концепт, в котором все отрицания стоят лишь перед атомарными концептами. Это всегда можно сделать, пользуясь законами де-Моргана ($\neg(C \sqcap D) \equiv (\neg C \sqcup \neg D)$ и т.п.), законами двойственности ($\neg\exists R.D \equiv \forall R.\neg D$ и т.п.) и законом снятия двойного отрицания ($\neg\neg C \equiv C$). В нашем случае получим нормализованный концепт:

$$C_0 := \exists R.A \sqcap \exists R.(B \sqcup \neg A) \sqcap \forall R.\neg B\,.$$

Будем строить модель, в которой этот концепт выполним (непуст). Создадим начальную точку x с условием

(0) $x : C_0$.

Далее из этого условия будем выводить новые условия на строящуюся модель и записывать их в «протокол», имеющий вид обычной *ABox*. Итак, изначально имеем $ABox := \{x : C_0\}$. Поскольку C_0 есть конъюнкция трех концептов, то x должен принадлежать всем трем, поэтому дописываем в *ABox* следующие факты:

(1) $x : \exists R.A$,

(2) $x : \exists R.(B \sqcup \neg A)$,

(3) $x : \forall R.\neg B$.

Ввиду условия **(1)** должна существовать точка y, связанная с x отношением R, в которой выполнен концепт A. Поэтому добавляем в *ABox* следующие факты:

(4) xRy,

(5) $y : A$.

Аналогично, из условия **(2)** следует существование точки z, связанной с x отношением R, в которой выполнен концепт $B \sqcup \neg A$. Поэтому добавляем в *ABox* следующие факты:

(6) xRz,

(7) $z : B \sqcup \neg A$.

Далее, ввиду условия **(3)** во всех точках, связанных с x отношением R (в нашем случае это точки y и z), должен быть выполнен концепт $\neg B$. Поэтому добавляем в *ABox* следующие факты:

(8) $y : \neg B$,

(9) $z : \neg B$.

Остается разобрать условие **(7)**. Так как точка z принадлежит дизъюнкции (объединению) концептов, то она принадлежит одному или другому концепту. Поэтому нужно рассмотреть два случая. Первый случай: **(7′)** $z : B$ – немедленно приводит к противоречию с условием **(9)**, согласно которому $z : \neg B$. Второй же случай: **(7″)** $z : \neg A$ – никаких явных противоречий не создает.

Итак, доведя все условия до элементарных, пришли к протоколу, в котором никаких явных противоречий нет. Можно показать, что эта *ABox*, а вместе с ним и исходный концепт C_0, имеет модель. Фактически сама *ABox* и представляет модель: нужно все созданные в процессе работы алгоритма точки считать элементами области интерпретации, а присутствующие в *ABox* элементарные факты (вида xRy или $y : A$) считать заданием интерпретации атомарных концептов и ролей:

$$\mathcal{I} = (\Delta, \cdot^I), \text{ где } \Delta = \{x, y, z\},\ A^I = \{y\},\ B^I = \varnothing,\ R^I = \{\langle x, y\rangle, \langle x, z\rangle\}.$$

В такой модели все факты из *ABox* будут верными. Следовательно, концепт C выполним, а исходное включение концептов – неверно.

В дескрипционной логике традиционно отсутствуют исчисления как таковые. Вместо этого для заданной логики пытаются построить алгоритм, проверяющий, например, истинность аксиом или выполнимость концептов. В ДЛ традиционно разрабатывают алгоритмы для проверки *выполнимости* концептов. Вложенность концептов $C \sqsubseteq D$ эквивалентна *не*выполнимости концепта C, соответственно, и ответы алгоритма будут противоположны тем, что давались при проверке вложенности концептов. Поэтому для проблемы

выполнимости вышеприведенные понятия «меняются местами», и приходим к следующему определению.

Алгоритм G является *разрешающей процедурой* для проблемы выполнимости концептов относительно терминологий для ДЛ, если выполнены следующие три условия:

Завершаемость**:** для любых (C, $TBox$) алгоритм G выдает ответ $G(C, TBox)$ через конечное время;

Корректность**:** для любых (C, $TBox$) если концепт C выполним относительно $TBox$, то $G(C, TBox) = 1$;

Полнота**:** для любых (C, $TBox$) если $G(C, TBox) = 1$, то концепт C выполним относительно $TBox$.

3.7. Табло-алгоритм для логики ALC

Пусть дан концепт C_0 логики ALC, выполнимость которого требуется выяснить. Без ограничения общности можно считать, что C_0 уже нормализован, т.е. все встречающиеся в нем символы отрицания стоят перед атомарными концептами. Строим начальную систему фактов $ABox_0\{x_0 : C_0\}$. Далее на каждом шаге к текущей системе фактов применяется правило и получается одна или две новых системы фактов. Список правил приведен в таблице 1.

Таблица 1: Правила табло-алгоритма для логики ALC.

Правило	**Условия применения**	**Действие**
$\sqcap$-правило	если 1. $x : (C \sqcap D) \in ABox$ 2. $x : C \notin ABox$ или $x : D \notin ABox$	то $ABox' := ABox \cup \{x : C, x : D\}$
$\sqcup$-правило	если 1. $x : (C \sqcup D) \in ABox$ 2. $x : C \notin ABox$ или $x : D \notin ABox$	то $ABox' := ABox \cup \{x : C\}$ $ABox'' := ABox \cup \{x : D\}$
$\exists$-правило	если 1. $x : \exists R.C \in ABox$ 2. нет такого y, что $xRy \in ABox$ и $y : C \in ABox$	то создать новую точку y и $ABox' := ABox \cup \{xRy, y : C\}$
$\forall$-правило	если 1. $x : \forall R.C \in ABox$ 2. есть такой y, что $xRy \in ABox$ и $y : C \notin ABox$	то $ABox' := ABox \cup \{y : C\}$

Порядок применения правил произволен. Следует заметить, правила для $\sqcap$, $\exists,\forall$ из текущей *ABox* создают одну новую систему фактов $ABox'$, тогда как $\sqcup$-правило из *ABox* создает двеновых системы фактов $ABox'$ и $ABox''$ и далее правила применяются к каждому из них независимо.

Таким образом, из исходной $ABox_0$ путем применения описанных правил будет построено дерево поиска, у которого в корне находится $ABox_0$ и у каждой *ABox* есть 0, 1 или 2 последователя. Применение правил прекращается, если к очередной *ABox* не применимо ни одно из правил, либо если в *ABox* содержится явное противоречие, так что применять дальнейшие правила не имеет смысла. Эти два вида *ABox* будут листьями дерева поиска, их называют *листовыми ABox*. Введем соответствующие термины:

- *ABox* называется *противоречивой,* если она содержит факты $x : A$ и $x : \neg A$ для некоторого экземпляра x и атомарного концепта A, либо он содержит факт $x :\bot$ для некоторого экземпляра x;
- *ABox* называется *полной непротиворечивой,* если оно не является противоречивым, но ни одно из правил табло-алгоритма к ней не применимо.

Если алгоритм встречает полную непротиворечивую *ABox,* то он выдает ответ 1, в этом случае существует модель и оставшиеся ветви дерева поиска алгоритм уже не обходит. Напротив, если алгоритм встречает противоречивую *ABox*, то это лишь означает, что данная ветвь дерева поиска не привела к модели, тогда алгоритм продолжает строить остальные ветви дерева поиска. Наконец, когда алгоритм обошел всё дерево поиска и оказалось, что все листовые *ABox* противоречивы, алгоритм выдает ответ 0. В этом случае моделей нет. Итак, по построению табло-алгоритм возваращает значение 1 тогда и только тогда, когда хотя бы одна из листовых *ABox* является полной непротиворечивой и возвращает 0 в противном случае.

Пусть задан концепт C_0 и терминология *TBox*. Требуется проверить, выполним ли C_0 относительно *TBox*. Без ограничения общности будкм полагать, что концепт C_0 нормализован (то есть в нем все отрицания стоят лишь перед атомарными концептами), а также терминология *TBox* состоит из единственной аксиомы $\top \sqsubseteq E$, где концепт E тоже нормализован.

Итак, даны концепты C_0 и E и требуется выяснить, существует ли интерпретация I, в которой $E^I = \Delta$ и $C^I{}_0 \neq \varnothing$. Как и раньше, берем начальную систему фактов $ABox_0 := \{x_0 : C_0\}$. Введем ряд определений. Говорят, что точка x *блокирует* точку y, если x есть предок y и $L(x) \supseteq L(y)$. Точку y будем называть *блокированной*, если она блокирована некоторой точкой x. Блокированные точки и их потомков будем называть *неактивными* точками, остальные –*активными*.

Теперь правила табло-алгоритма легко сформулировать – нужно к каждому правилу из таблицы 1 добавить дополнительное предусловие «точка x – активная», а также ввести новое правило, добавляющее концепт E в каждую точку. Получающаяся система правил приведена в таблице 2.

Таблица 2. Правила табло-алгоритма для логики *ALC+TBox*

Правило	**Условия применения**	**Действие**
$\sqcap$-правило	если 0. точка x – активная 1. $x : (C \sqcap D) \in ABox$ 2. $x : C \notin ABox$ или $x : D \notin ABox$	то $ABox' := ABox \cup \{x : C, x : D\}$
$\sqcup$-правило	если 0. точка x – активная 1. $x : (C \sqcup D) \in ABox$ 2. $x : C \notin ABox$ и $x : D \notin ABox$	то $ABox' := ABox \cup \{x : C\}$ $ABox'' := ABox \cup \{x : D\}$
$\exists$ -правило	если 0. точка x – активная 1. $x : \exists R.C \in ABox$ 2. нет такого y, что $xRy \in ABox$ и $y : C \in ABox$	то создать новую точку y и $ABox' := ABox \cup \{xRy, y : C\}$
$\forall$ -правило	если 0. точка x – активная 1. $x : \forall R.C \in ABox$ 2. нет такого y, что $xRy \in ABox$ и $y : C \notin ABox$	то $ABox' := ABox \cup \{y : C\}$
T -правило	если 0. точка x – активная	то $ABox' := ABox \cup \{x : E\}$

	1. $x : E \notin ABox$	

Как видно, сам табло-алгоритм остается прежним, за исключением того, что в строчку, где фигурируют правила ⊓, ∃, ∀, необходимо добавить упоминание *TBox*-правила, которое из *ABox* создает одну новую $ABox'$.

ГЛАВА 4. ЯЗЫКИ ОПИСАНИЯ ОНТОЛОГИЙ

4.1. Виды языков описания онтологий

Ключевым моментом в проектировании онтологий является выбор соответствующего языка спецификации онтологий. Для реализации различных онтологии необходимы языки их представления, обладающие достаточной выразительной мощностью и позволяющие избежать «низкоуровневых» проблем при проектировке онтологии. Выделяются три основные концепции создания языков описания онтологий: машинно-ориентированные языки, универсальные языки, языки веб-онтологий.

Специализированные машинные языки онтологического описания – это традиционные языки спецификации онтологий. По типу применяемой логики их классифицируют на языки фреймово-продукционные, дескриптивной логики и логики первого порядка. Широко распространёнными представителями машинных языков являются KIF (Knowledge Interchange Format), CycL (Cycorp Language), F-Logic

KIF – универсальный машинно-ориентированный язык для обмена данными в рамках выбранной предметной области. KIF имеет декларативную семантику и логическую всесторонность (т.е. предусматривает выражение произвольных предложений в исчислении предикатов первого порядка). Язык обеспечивает представление знаний, используется для описания объектов, функций и отношений. KIF – это язык продукционного типа, где каждая продукция записывается в виде импликации.

CycL – формальный язык, синтаксис которого базируется на логике первого порядка. CycL различает такие сущности, как экземпляры, классы, предикаты и функции. Словарь CycL состоит из термов. Множество термов можно разделить на константы, неатомарные термы и переменные. Термы используются при составлении значащих выражений CycL, из которых формируются утверждения. Совокупность утверждений составляет базу знаний. Исходный код CycL находится в свободном доступе по лицензии

OpenCyc, однако для работы большинства приложений Cycl необходимо дополнительно лицензировать сервер Apache.

F-Logic – онтологический язык, который базируется на логиках первого порядка, однако классы и свойства в нем представлены как термины, а не как предикаты. Язык создавался для осуществления взаимодействия между онтологиями, построенными на основе предикатов, и онтологиями, построенными на основе F-Logic. Создатели определили интуитивные трансляторы для преобразования знаний из предикатных онтологий в F-Logic онтологии и показали, что такой перевод сохраняет логические связи (preserves entailment) для большого количества онтологических языков.

Общим недостатком специализированных машинных языков онтологического описания является сложность их синтаксиса и необходимость в специальных инструментах для интерпретации. Этого недостатка лишены универсальные языки, которые, сохраняя описательную полноту для компьютерных систем, обеспечивают семантическую прозрачность для человека.

К универсальным языкам описания онтологий относится XML (Extensible Markup Language) и HTML (HyperText Markup Language). XML – рекомендованный W3C (World Wide Web Consortium – консорциум Всемирной паутины) язык разметки данных. Спецификация XML описывает XML-документы и частично описывает поведение XML-процессоров (программ, читающих XML-документы и обеспечивающих доступ к их содержимому). XML разрабатывался как язык создания машино-читаемых документов с простым формальным синтаксисом, обладающим семантической прозрачностью для комфортного восприятия человеком. Основной сферой применения языка XML является Интернет. HTML – стандартный язык разметки документов во Всемирной паутине. Большинство веб-страниц содержат описание разметки на языке HTML. Язык HTML интерпретируется браузерами и отображается в виде документа в удобной для человека форме. Язык XHTML (Extensible Hy-pertext Markup Language)

является более строгим вариантом HTML, он следует всем ограничениям XML и фактически XHTML можно воспринимать как приложение языка XML к области разметки гипертекста.

Более поздние языки, основанные на Web-стандартах, такие как XML, RDF, RDF Schema, DAML+OIL, OWL созданы специально для обмена онтологиями через Web.

RDF (Resource Description Framework) – язык описания метаданных ресурсов, использующий XML-синтаксис. В RDF используется базовая модель данных «объект – атрибут – значение», позволяющая ему сыграть роль универсального языка описания семантики и связей для различных ресурсов. Ресурсы описываются в виде ориентированного размеченного графа – каждый ресурс может иметь свойства, которые в свою очередь также могут быть ресурсами или их коллекциями. Все словари RDF используют базовую структуру, описывающую классы ресурсов и типы связей между ними. Это позволяет использовать разнородные децентрализованные словари, созданные для машинной обработки по разным принципам и методам. Важной особенностью стандарта является расширяемость: можно задать структуру описания источника, используя и расширяя такие встроенные понятия RDF-схем, как классы, свойства, типы, коллекции. Модель схемы RDF включает наследование классов и свойств.

RDF Schema (RDFS) – это семантическое расширение RDF, которое обеспечивает механизмы описания связанных ресурсов и их связей. Система классов и свойств RDF Schema похожа на систему типов языков объектно-ориентированного программирования (ООП) с некоторыми отличиями. Так, описательный язык словаря RDF определяет свойства в терминах того класса ресурсов, к которому эти свойства относятся, в отличие от языков ООП, описывающих класс в терминах свойств своих элементов. RDFS предоставляет хорошие базовые возможности для описания словарей типов предметных областей. Одно из ограничений – невозможность с помо щью

RDFS выразить аксиоматические знания, т.е. задать аксиомы и правила вывода, построенные на них.

DAML+OIL – семантический язык разметки веб-ресурсов, который расширяет стандарты RDF и RDF Schema за счет более полных примитивов моделирования. В последнюю версию DAML+OIL включен набор дополнительных конструкций для создания онтологий и разметки информации в легко интерпретируемом машиной виде. Онтология DAML+OIL состоит из:

- заголовков (headers);
- элементов классов (class elements);
- элементов свойств (property elements);
- экземпляров (instances).

OWL (Web Ontology Language) – язык представления онтологий следующего поколения после DAML+OIL. Обладает более богатым набором возможностей чем XML, RDF, RDF Schema и DAML+OIL. В OWL реализован мощный механизм семантического анализа. OWL обеспечивает более полную машинную обработку веб-контента, предоставляя наряду с формальной семантикой дополнительный терминологический словарь.

4.1. Язык OWL и его виды. Связь дескрипционных логик с OWL

Язык онтологий OWL разработан для использования приложениями, ко торые должны обрабатывать содержимое информации, а не только пред-ставлять эту информацию. В семантическом вебе онтология содержит описания классов, свойств и их реализацию в виде экземпляров классов. Формальная семантика OWL описывает, как с помощью такой онтологии сделать логические выводы, т.е. получить факты, которые не представлены в онтологии напрямую, но следуют из ее семантики. Эти выводы могут быть основаны на одном документе или множестве распределенных документов, связанных между собой определенным образом.

Язык OWL предполагает открытость, т.е. описания ресурсов не ограничены единственным файлом или областью видимости. Класс OWL, первоначально определенный в одной онтологии, может быть расширен в других онтологиях, причем расширение не должно отменять первоначальное определение.

Для языка OWL определено следующее пространство имен:

http://www.w3.org/2002/07/owl#.

Обычно для этого пространства используется префикс owl.

Поскольку существует много ДЛ, различающихся как по выразительной силе, так и по вычислительной сложности, это привело к тому, что в языке OWL имеется несколько вариантов. Виды OWL не являются независимыми языками, а построены на принципе расширения возможностей. Имеющиеся в ДЛ понятия концепт, роль, экземпляр и база знаний в OWL соответствуют понятиям класс, свойство, объект и онтология соответственно. Официальной рекомендацией W3C от 10 февраля 2004 года является версия языка OWL 1.0. Данная спецификация языка OWL подразделяется на следующие варианты:

- OWL Lite – соответствует дескрипционной логике SHIF(D). OWL Lite имеет наименьшую выразительную мощность из всех, но для решения простых задач его может быть достаточно. OWL Lite обладает важнейшим свойством разрешимостью. Именно разрешимость и относительно невысокая вычислительная сложность является главной причиной использования OWL Lite для создания многочисленных практических онтологий (в медицине, биоинформатике и т.п.).
- OWL DL – соответствует дескрипционной логике SHOIN(D). OWL DL обладает большей выразительной мощью, чем OWL Lite. OWL DL также обладает свойством разрешимости, однако

вычислительная сложность у него выше, чем OWL Lite. Разрешимость достигается, в частности, наложением ограничений на синтаксис языка. Так, в OWL DL классу запрещено быть экземпляром.

- OWL Full – не соответствует какой-либо ДЛ. Преимущество языка OWL Full заключается в том, что он базируется на семантике языка RDF, что позволяет говорить о полной как структурной, так и семантической совместимости с RDF: любой правильный документ на языке RDF является правильным документом на языке OWL2 Full, а любой правильный вывод на языке RDF Schema является правильным выводом и на языке OWL2 Full. Недостаток языка OWL Full заключается в том, что его большая выразительная мощность не гарантирует полной (эффективной) поддержки логического вывода

Каждый из этих диалектов (кроме OWL Lite) является расширением предыдущего. Как следствие, любая OWL Lite онтология является OWL DL онтологией, а любая OWL DL онтология является OWL Full онтологией.

4.2. Структура документа OWL

Документ OWL – это документ на языке RDF/XML, который может содержать заголовок OWL, а также содержит определения классов, свойств и сведений о представителях классов. Представители классов (individuals) по терминологии OWL – это экземпляры классов.

В качестве расширения файла с документом OWL можно использовать расширения .owl или .rdf. Класс owl:Ontology используется для описания заголовка OWL, который в языке RDF/XML имеет следующий синтаксис:

```
<owl:Ontology rdf:about="ресурс">
    …
</ >
```

В атрибуте rdf:about задается наименование онтологии или ссылка на онтологию. Если значение атрибута – пустая строка (""), то наименованием онтологии служит базовый URI (IRI) экземпляра класса owl:Ontology. Как правило, это URI (IRI) документа, содержащего онтологию. Исключение является случай, когда в элементе rdf:RDF задан атрибут xml:base, явно устанавливающий базовый URI (IRI) документа.

Классами, определяющими семантические свойства классов OWL, являются owl:AnnotationProperty и owl:OntologyProperty.

Свойство owl:imports является экземпляром класса owl:OntologyProperty. Он определяет URI (IRI) документа, импортируемого в данный документ. Импорт документов может быть вложенным.

Свойство owl:versionInfo является экземпляром класса owl:AnnotationProperty. Обычно его объектом является строка, содержащая сведения о версии данной онтологии.

Свойство owl:priorVersion является экземпляром класса owl:OntologyProperty. Это свойство содержит ссылку на другую онтологию, которая таким образом рассматривается как предыдущая версия данной онтологии.

Свойство owl:incompatibleWith является экземпляром класса owl:OntologyProperty. Это свойство содержит ссылку на другую онтологию, которая таким образом рассматривается как предыдущая версия данной онтологии, несовместимая с данной онтологией.

Свойство owl:backwardCompatibleWith является экземпляром класса owl:OntologyProperty. Это свойство содержит ссылку на другую онтологию, которая таким образом рассматривается как предыдущая версия данной онтологии, совместимая с данной онтологией.

Если в классе owl:Ontology задано свойство owl:backwardCompatibleWith, то документ OWL может содержать классы owl:DeprecatedClass и owl:DeprecatedProperty, описывающие классы и свойства предыдущей версии,

которые в данной версии отменены или заменены другими классами и свойствами.

Класс owl:DeprecatedClass является подклассом класса rdfs:Class, а класс owl:DeprecatedProperty – подклассом класса rdf:Property.

4.3. Описание классов OWL

Классы OWL обеспечивают группирование ресурсов со сходными характеристиками. Каждый класс связан с набором представителей класса, называемым *расширением класса*. Представители класса в расширении класса называются *экземплярами класса*. Каждый класс имеет некоторый содержательный смысл, который связан, но не эквивалентен расширению класса, т.е. два класса могут имеет одинаковые расширения, но являться разными классами.

В языках OWL Lite и OWL DL экземпляр класса не может в то же время классом. В языке OWL Full класс может функционировать как экземпляр другого класса.

Классы OWL описываются с помощью описаний класса, которые затем комбинируются в аксиомы класса. В языке OWL определены шесть типов описаний классов:

- с помощью идентификатора класса;
- с помощью перечисления представителей класса;
- с помощью ограничения свойств;
- с помощью пересечения двух и более описаний классов;
- с помощью объединения двух и более описаний классов;
- с помощью дополнения описания класса.

При использовании первого типа определения класс задается с определенным именем. В остальных типах класс задается как пустой узел со свойством rdf:type, чье значение равно owl:Class.

Описание класса с помощью идентификатора в нотации N3 задается следующим образом:

префикс:имя_класса rdf:type owl:Class,

где префикс – это префикс пространство имен, в котором определен новыйкласс с именем имя-класса.

В RDF/XML класс задается с помощью элемента owl:Class с атрибутом rdf:ID. Ниже приведен пример объявление класса с именем Office:

<owl:Class rdf:ID="Office"/>.

Класс owl:Class является подклассом класса rdfs:Class. Класс owl:Class в OWL Lite и OWL DL не может одновременно являться экземпляром своего класса. В OWL Full этого ограничения нет, и оба класса считаются эквивалентными.

Два класса с идентификаторами Thing и Nothing в языке OWL являются предопределенными. Расширением класса для owl:Thing является набор всех представителей этого класса, а расширением класса owl:Nothing является пустой набор. Соответственно каждый класс OWL является подклассом owl:Thing, а класс owl:Nothing является подклассом каждого класса.

Описание класса с помощью перечисления выполняется с помощью свойства owl:oneOf. Значением этого свойства является список представителей класса (его экземпляров). Объявить класс по перечислению можно только в языках OWL DL и OWL Full. Приведем пример объявление класса по перечислению OWL, представителями которого являются цвета RGB:

```
<owl:Class>
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#Red"/>
    <owl:Thing rdf:about="#Green"/>
    <owl:Thing rdf:about="#Blue"/>
  </owl:oneOf>
</owl:Class>.
```

Под описанием класса с помощью ограничения свойств понимается описание безымянного класса, все представители которого удовлетворяют заданным ограничениям. Ограничения определяются с помощью класса owl:Restriction, являющегося подклассом класса owl:Class. Значение ограничения (ресурс или литерал) задаются с помощью свойства owl:onProperty. Для ограничения может быть задано только одно свойство owl:onProperty. В OWL определены два вида ограничения свойств: ограничения по значению (value constraints) и ограничения по мощности (cardinality constraints).

Ограничения по значению задают диапазон свойства, используемого в описании данного класса. В OWL определены следующие свойства ограничения по значению: owl:allValuesFrom, owl:someValuesFrom, и owl:hasValue. В OWL Lite единственным типом описания класса, допустимым в качестве объекта в свойствах ограничения по значению является имя класса.

Свойство owl:allValuesFrom связывает ограничиваемый класс либо с описанием класса, либо с диапазоном данных. Это свойство используется для описания класса всех представителей, для которых все значения свойства либо описания классов, либо значения данных в заданном диапазоне. Например, описание ограничения класса hasColor, представителями которого являются только значения класса RGBColor (цвета RGB), может быть представлено следующим образом:

```
<owl:Restriction>
        <owl:onProperty rdf:resource="#hasColor"/>
        <owl:allValuesFrom rdf:resource="#RGBColor"/>
</owl:Restriction>.
```

Свойство owl:someValuesFrom связывает ограничиваемый класс либо с описанием класса, либо с диапазоном данных. Это свойство используется для описания класса всех представителей, для которых по крайней мере одно

значение свойства – экземпляр либо описания класса, либо значения данных в заданном диапазоне. Приведем пример описания ограничения класса hasColor, среди представителей которого имеется, по крайней мере, один представитель для белого цвета (whiteColor):

```
<owl:Restriction>
        <owl:onProperty rdf:resource="#hasColor"/>
        <owl:allValuesFrom rdf:resource="#whiteColor"/>
</owl:Restriction>.
```

Свойство owl:hasValue связывает ограничиваемый класс с заданным значением, которое является либо представителем класса, либо значением данных. Это свойство используется для описания класса всех представителей, для которых по крайней мере одно значение свойства – экземпляр либо описания класса, либо значения данных в заданном диапазоне. Ниже приведен пример описания ограничения класса hasColor, среди представителей которого имеется ссылка на представитель класса для черного цвета (blackColor):

```
<owl:Restriction>
        <owl:onProperty rdf:resource="#hasColor"/>
        <owl:hasValue rdf:resource="#blackColor"/>
</owl:Restriction>.
```

Ограничение по мощности задает количество значений, которое может иметь свойство. В OWL определены следующие свойства ограничения по мощности: owl:maxCardinality, owl:minCardinality и owl:Cardinality. В OWL Lite свойства ограничения по мощности могут иметь только значения "0" и "1".

Свойство owl:maxCardinality связывает ограничиваемый класс со значением данных, принадлежащим к пространству значений типа nonNegativeInteger схемы XML. Свойство описывает класс всех

представителей, который имеет самое большее N различных значений, где N – значение ограничения мощности. Описание ограничения класса SmallOfficeStaff, который имеет самое большее 10 представителей, можно представить следующим образом:

```
<owl:Restriction>
        <owl:onProperty rdf:resource="#SmallOfficeStaff"/>
        <owl:maxCardinality rdf:datatype=
        "&xsd;nonNegativeInteger">10</owl:maxCardinality>
</owl:Restriction>.
```

Свойство owl:minCardinality связывает ограничиваемый класс со значением данных, принадлежащим к пространству значений типа nonNegativeInteger схемы XML. Свойство описывает класс всех представителей, который имеет самое меньшее N различных значений, где N – значение ограничения мощности. Опишем ограничения класса BigOfficeStaff, который имеет самое меньшее 100 представителей, следующим образом:

```
<owl:Restriction>
        <owl:onProperty rdf:resource="#SmallOfficeStaff"/>
        <owl:minCardinality rdf:datatype=
        "&xsd;nonNegativeInteger">100</owl:minCardinality>
</owl:Restriction>.
```

Свойство owl:cardinality связывает ограничиваемый класс со значением данных, принадлежащим к пространству значений типа nonNegativeInteger схемы XML. Свойство описывает класс всех представителей, который имеет точно N различных значений, где N – значение ограничения мощности. Например, ограничения класса SomeOfficeStaff, который имеет точно 8 представителей, может быть описано так:

```
<owl:Restriction>
        <owl:onProperty rdf:resource="#SmallOfficeStaff"/>
        <owl:cardinality rdf:datatype=
        "&xsd;nonNegativeInteger">8</owl:cardinality>
</owl:Restriction>.
```

Описание класса с помощью пересечения (операция AND – логическое И) выполняется с помощью свойства owl:intersectionOf, связывающего класс со списком описаний классов. Это свойство описывает безымянный класс, в котором расширение содержит только те представители, которые содержатся во всех расширениях в описании классов в списке. В OWL Lite значения свойство owl:intersectionOf могут быть только идентификаторами класса и/или ограничениями свойств. Ниже приведен пример описания ограничений, являющихся пересечением двух коллекций цветов, для класса:

```
<owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
                <owl:Class>
                        <!-- Коллекция из двух элементов:
                        черного и красного цвета -->
                <owl:oneOf rdf:parseType="Collection">
                        <owl:Thing rdf:about="#blackColor"/>
                        <owl:Thing rdf:about="#redColor"/>
                </owl:oneOf>
                </owl:Class>
                <owl:Class>
                        <!-- Коллекция из четырех элементов:
                        красного, зеленого, синего
                        и белого цветов -->
                <owl:oneOf rdf:parseType="Collection">
                        <owl:Thing rdf:about="#redColor"/>
                        <owl:Thing rdf:about="#greenColor"/>
                                <owl:Thing rdf:about="#blueColor"/>
```

```
                    <owl:Thing rdf:about="#whiteColor"/>
            </owl:oneOf>
            </owl:Class>
        </owl:intersectionOf>
    </owl:Class>.
```

Этот класс будет иметь только одного представителя – redColor (красный цвет), поскольку только этот представитель содержится в обоих списках.

Описание класса с помощью объединения (операция OR – логическое ИЛИ) выполняется с помощью свойства owl:unionOf, связывающего класс со списком описаний классов. Это свойство описывает безымянный класс, в котором расширение содержит все представители, которые содержатся хотя бы в одном из расширений класса в описании классов в списке. Свойство owl:unionOf можно использовать только в OWL DL и OWL Full. Приведем пример описание ограничений, являющихся объединением двух коллекций цветов, для класса:

```
<owl:Class>
    <owl:unionOf rdf:parseType="Collection">
        <owl:Class>
                <!-- Коллекция из двух элементов:
                черного и красного цвета -->
        <owl:oneOf rdf:parseType="Collection">
            <owl:Thing rdf:about="#blackColor"/>
            <owl:Thing rdf:about="#redColor"/>
        </owl:oneOf>
        </owl:Class>
        <owl:Class>
                <!-- Коллекция из четырех элементов:
                красного, зеленого, синего
                и белого цветов -->
        <owl:oneOf rdf:parseType="Collection">
```

```
                    <owl:Thing rdf:about="#redColor"/>
                    <owl:Thing rdf:about="#greenColor"/>
                        <owl:Thing rdf:about="#blueColor"/>
                        <owl:Thing rdf:about="#whiteColor"/>
                </owl:oneOf>
                </owl:Class>
            </owl:unionOf>
        </owl:Class>.
```

Этот класс содержит пять представителей: blackColor (черный цвет), redColor (красный цвет), greenColor (зеленый цвет), blueColor (синий цвет) и whiteColor (белый цвет), т.е. представителями класса является сумма представителей обоих списков (представитель redColor есть в обоих списках, но в суммарном списке он будет представлен только один раз).

Описание класса с помощью дополнения (операция NOT – логическое НЕ) выполняется с помощью свойства owl:complementOf, описывающего класс, для которого содержит только те представители, которые не принадлежат к расширению класса, являющегося объектом предложения. Свойство owl:complementOf можно использовать только в OWL DL и OWL Full. Опишем ограничение, являющееся дополнением для класса:

```
<owl:Class>
  <owl:complementOf>
    <owl:Class rdf:about="#greyColor"/>
  </owl:complementOf>
</owl:Class>.
```

Расширение этого класса содержит всех тех представителей, которые не принадлежат классу greyColor.

4.4. Аксиомы классов

Описания классов образуют компоненты для определения классов с помощью аксиом классов. Простейшей формой аксиомы класса является описание класса с помощью идентификатора, однако обычно аксиомы содержат дополнительные компоненты, задающие необходимые и/или достаточные характеристики классов. Для комбинирования описания класса в аксиому класса используются следующие свойства: rdfs:subClassOf, owl:equivalentClass и owl:disjointWith. Свойство rdfs:subClassOf задает, что расширение класса в описании класса является подмножеством расширения класса в описании другого класса:

```
<owl:Class rdf:ID="RGBColor">
        <!-- Класс RGBColor является подклассом
        класса Color -->
    <rdfs:subClassOf rdf:resource="#Color"/>
        <!-- Ограничения на свойство hasValue
        класса RGBColor: один из трех цветов:
        красный, зеленый или синий -->
    <rdfs:subClassOf>
        <owl:Restriction>
        <owl:onProperty rdf:resource="#hasValue"/>
        <owl:someValuesFrom>
            <owl:Class>
            <owl:oneOf rdf:parseType=
                    "Collection">
                    <owl:Thing rdf:about=
                        "#RedColor"/>
                        <owl:Thing rdf:about=
                        "#GreenColor"/>
                        <owl:Thing rdf:about=
                        "#BlueColor"/>
            </owl:oneOf>
            </owl:Class>
```

```
            </owl:someValuesFrom>
            </owl:Restriction>
        </rdfs:subClassOf>
            <!-- Класс RGBColor является дополнением
            к классу CMYKColor -->
        <rdfs:subClassOf>
            <owl:Class>
            <owl:complementOf rdf:resource="#CMYKColor"/>
            </owl:Class>
        </rdfs:subClassOf>
</owl:Class>
```

Свойство owl:equivalentClass связывает описание данного класса с описанием другого класса. Использование этого свойства в аксиоме означает, что оба класса имеют одинаковое расширение, т.е. оба расширения класса содержат одинаковый набор представителей. Например:

- Объявление именованного класса BaseColor эквивалентным именованному классу RGBColor:

```
<owl:Class rdf:about="#BaseColor">
    <owl:equivalentClass rdf:resource="#RGBColor"/>
</owl:Class>
```

- Объявление именованного класса CMYKColor эквивалентным безымянному вложенному классу, содержащему расширение из четырех представителей: CyanColor (циановый цвет), MagentaColor (фиолетовый цвет), YellowColor (желтый цвет) и BlackColor (черный цвет):

```
<owl:Class rdf:ID="CMYKColor">
    <owl:equivalentClass>
        <owl:Class>
```

```
            <owl:oneOf rdf:parseType="Collection">
                <owl:Thing rdf:about="#CyanColor"/>
                <owl:Thing rdf:about="#MagentaColor"/>
                <owl:Thing rdf:about="#YellowColor"/>
                    <owl:Thing rdf:about="#BlackColor"/>
            </owl:oneOf>
            </owl:Class>
        </owl:equivalentClass>
</owl:Class>
```

- Можно задать эквивалентность определяемого класса и вложенного класса в сокращенной форме (без использования элемента owl:equivalentClass). Предыдущий пример при использовании сокращенной формы будет иметь следующий вид:

```
<owl:Class rdf:ID="CMYKColor">
        <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#CyanColor"/>
        <owl:Thing rdf:about="#MagentaColor"/>
        <owl:Thing rdf:about="#YellowColor"/>
                <owl:Thing rdf:about="#BlackColor"/>
    </owl:oneOf>
</owl:Class>
```

Свойство owl:disjointWith связывает описание данного класса с описанием другого класса. Использование этого свойства в аксиоме означает, что расширения двух классов не имеют общих представителей. Например, объявление об отсутствии общих представителей в классах CMYKColor и RGBColor можно описать так:

```
<owl:Class rdf:about="#CMYKColor">
        <owl:disjointWith rdf:resource="#RGBColor"/>
</owl:Class>
```

4.5. Свойства OWL

В языке OWL определены следующие категории свойств:

- свойства онтологий (ontology properties);
- свойства аннотаций (annotation properties);
- свойства объектов (object properties);
- свойства типизированных данных (datatype properties).

Классами, определяющими свойства онтологий и аннотаций, являются owl:AnnotationProperty и owl:OntologyProperty. Свойства объектов в OWL определяются как экземпляры класса owl:ObjectProperty, а свойства типизированных данных – как экземпляры класса owl:DatatypeProperty. Оба этих класса являются подклассами класса rdf:Property. В языке OWL Full свойства объектов и свойства типизированных данных не являются взаимоисключающими, поскольку значения данных можно рассматривать как представления. Поэтому в OWL Full класс owl:ObjectProperty эквивалентен классу rdf:Property.

Аксиома свойства определяет характеристики свойства. В простейшем случае аксиома свойства просто задает существование свойства, например

```
<owl:ObjectProperty rdf:ID="hasValue"/>
```

Однако чаще аксиомы свойства задают дополнительные характеристики свойств. В языке OWL поддерживаются следующие основные конструкции для характеристик свойств: свойства RDFS; отношения к другим свойствам; глобальные ограничения мощности; характеристики логических свойств.

В OWL DL субъект и объект подсвойства должны быть оба либо свойствами типизированных данных, либо свойствами объектов. В OWL Lite значением свойства rdfs:domain и rdfs:range должен быть только идентификатор класса. Для задания отношения к другим свойствам в языке OWL используются свойства owl:equivalentProperty и owl:inverseOf.

Свойство owl:equivalentProperty используется для указания того, что два свойства имеют одинаковое расширение. Например, объявление об эквивалентности свойств hasColorValue и hasValue можно сделать так:

```
<owl:ObjectProperty rdf:ID="hasColorValue">
        < owl:equivalentProperty rdf:resource="#hasValue"/>
</owl:ObjectProperty>.
```

Свойство имеет направление от домена к диапазону. Иногда необходимо определить отношения в обоих направления. Например, для отношения человек_владеет_автомашиной обратным является отношение автомашина_принадлежит_человеку.

Свойство owl:inverseOf формирует обратное отношение для свойства, например, объявление об том, что у свойства hasDescendant (имеет потомка) существует обратное свойство hasAncestor (имеет предка) описано следующим образом:

```
<owl:ObjectProperty rdf:ID="hasDescendant">
         <owl:inverseOf rdf:resource="#hasAncestor"/>
</owl:ObjectProperty>.
```

Глобальные ограничения мощности задаются с помощью классов owl:FunctionalProperty и owl:InverseFunctionalProperty. Эти ограничения называются глобальными, потому что, к какому бы классу не применялось свойство, ограничения будут выполнены. Класс owl:FunctionalProperty определяет функциональное свойство. Функциональным свойством может быть как свойство объекта, так и свойство типизированного данного. Класс owl:FunctionalProperty является специальным подклассом класса rdf:Property. Приведем пример объявления об том, что свойство объекта fillColor (цвет фона) является функциональным, т.е. для ресурса свойство может принимать одно из значений, определенных в классе RGBColor:

```
<owl:ObjectProperty rdf:ID="fillColor">
        <rdf:type    rdf:resource="&owl;FunctionalProperty"/>
        <rdfs:domain rdf:resource="#Shape"/>
        <rdfs:range rdf:resource="#RGBColor"/>
</owl:ObjectProperty>.
```

Если свойство объявляется обратно функциональным, то объект в утверждении для свойства однозначно определяет субъект (некоторое представление класса). Более точно, если задано утверждение, что *P* – это обратное функциональное свойство, оно предполагает, что значение *y* может быть значением *P* для одного экземпляра *x*, т.е. не может быть двух различных экземпляров *x*1 и *x*2 таких, что обе пары (*x*1,*y*) и (*x*2,*y*) являются экземплярами *P*. Обратное функциональное свойство соответствует понятию ключа в базе данных.

Аксиома обратного функционального свойства задается объявлением свойства как экземпляра класса owl:InverseFunctionalProperty, который является подклассом класса owl:ObjectProperty. По своему определению обратными функциональными свойствами могут быть только свойства объектов. Поскольку в OWL Full свойства типизированных данных являются подклассами свойств объектов, обратное функциональное свойство может быть определено и для свойств типизированных данных. Такое определение недопустимо в OWL Lite и OWL DL. Наприме, зададим объявление о том, что каждый объект предложений birthRegion (представитель класса Person) может однозначно идентифицировать субъект (представитель класса Region) :

```
<owl:InverseFunctionalProperty rdf:ID="birthRegion">
        <rdfs:domain rdf:resource="#Region"/>
        <rdfs:range rdf:resource="#Person"/>
</owl:InverseFunctionalProperty>.
```

Характеристики логических свойств (транзитивное или симметричное свойство) задаются с помощью классов owl:TransitiveProperty и owl:SymmetricProperty. Если свойство *P* задается как транзитивное, это означает, что если пара (*x*,*y*) является экземпляром *P* и пара (*y*,*z*) также является экземпляром *P*, то пара (*x*,*z*) – тоже экземпляр *P*. Аксиома транзитивного свойства задается объявлением свойства как экземпляра класса owl:TransitiveProperty, который является подклассом класса owl:ObjectProperty. В OWL DL для транзитивного свойства нельзя задавать ограничения мощности (ни локальные, ни глобальные) ни для свойств или их суперсвойств, ни для обратных свойств или их суперсвойств. К примеру, объявление транзитивного свойства descendantOf (потомок) можно описать следующщим образом:

```
<owl:TransitiveProperty rdf:ID="descendantOf">
        <rdfs:domain rdf:resource="#Person"/>
        <rdfs:range rdf:resource="#Person"/>
</owl:TransitiveProperty>.
```

Пусть Иванов_Иван_Петрович, Иванов_Петр_Александрович и Иванов_Александр_Николаевич являются представителями класса Person. Тогда, если для экземпляра Иванов_Петр_Александрович значение свойства descendantOf (потомок) равно Иванов_Иван_Петрович, и для экземпляра Иванов_Александр_Николаевич значение свойства descendantOf (потомок) равно Иванов_Петр_Александрович, то тогда, в силу транзитивности свойства descendantOf при семантической обработке документа можно сделать вывод, что Иванов_Иван_Петрович является потомком экземпляра Иванов_Александр_Николаевич.

Симметричное свойство – это свойство, для которого предполагается, что если пара (*x*,*y*) является экземпляром *P*, тогда пара (*y*,*x*) – также экземпляр *P*. Аксиома симметричного свойства задается объявлением

свойства как экземпляра класса owl:SymmetricProperty, который является подклассом класса owl:ObjectProperty. Для симметричного свойства значения свойств должны быть одинаковы. Например, объявление транзитивного свойства brotherOf (брат) имеет вид:

```
<owl:SymmetricProperty rdf:ID="brotherOf">
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="#Person"/>
</owl:SymmetricProperty>.
```

Пусть для экземпляра Иванов_Иван_Петрович значение свойства brotherOf равно Иванов_Константин_Петрович. Тогда, в силу симметричности свойства brotherOf при семантической обработке документа можно сделать вывод, что и Иванов_Константин_Петрович является братом экземпляра Иванов_Иван_Петрович.

4.6. Описание экземпляров классов в OWL

Представители классов (экземпляры) в языке OWL определяются с помощью специальных аксиом, также называемых фактами. В OWL для представителей определены два типа фактов:

- факты о принадлежности к классу и значениях свойств;
- факты о тождественности представителей.

Факты о принадлежности к классу и значении свойств задают имя представителя, класс, экземпляром которого является представитель и набор значений свойств представителя. Экземпляр класса не обязательно должен иметь имя, он может быть и безымянным. В этом случае в элементе определения представителя не задается атрибут rdf:ID.

Факты о тождественности представителей задают утверждения относительно того, являются ли какие-либо представителя тождественными друг другу, т.е. относятся ли они к одному и тому же ресурсу. В языке OWL определены три конструкции для задания факта тождественности представителей: owl:sameAs, owl:differentFrom и owl:AllDifferent. Свойство

owl:sameAs связывает двух представителей отношением тождественности, т.е. ссылки URI (IRI) в них указывают на один и тот же ресурс. Понятие тождественности отличается от понятия эквивалентности следующим образом. Эквивалентность означает, что расширения двух классов одинаковы, но не подразумевает, что эти классы указывают на один и тот же ресурс. В OWL Full класс рассматривается одновременно и как свой представитель, поэтому в нем можно задавать тождественность классов. Таким образом, два класса считаются тождественными, если они реализованы по-разному, но имеют одинаковое содержательное значение. Ниже приведен пример оОбъявление тождественности ссылки на ресурс Moscow и ссылки на ресурс CapitalOfRussua, поскольку Москва является столицей России:

```
<rdf:Description rdf:about="# Moscow">
        <owl:sameAs rdf:resource="# CapitalOfRussua"/>
</rdf:Description>.
```

Свойство owl:differentFrom указывает, что два представителя не тождественны друг другу, т.е. ссылки URI (IRI) в них указывают на разные ресурсы. Например, зададим представителя класса с именем studIvanov1, отличного от представителей studIvanov2 и studIvanov3:

```
        <!-- Задание представителя класса Student
        с именем studIvanov1 -->
<inst:Student rdf:ID="studIvanov1">
        <!-- Задание полного имени -->
        <inst:fullName>
        <inst:firstName rdf:datatype="&xsd;token">
        Иван</inst:firstName>
        <inst:surName rdf:datatype="&xsd;token">Иванович</inst:surName>
```

```
        <inst:secondName rdf:datatype="&xsd;token">
        Иванов</inst:secondName>
        </inst:fullName>
            <!-- Задание даты рождения -->
        <inst:birthDate rdf:datatype="&xsd;date">
        1991-05-17</birthDate>
            <!-- Представитель studIvanov1 отличается
            от представителей studIvanov2 и studIvanov3 -->
        <owl:differentFrom rdf:resource="#studIvanov2"/>
        <owl:differentFrom rdf:resource="#studIvanov3"/>
</inst:Student>.
```

В тех онтологиях, где предполагается уникальность имен, использование свойства owl:differentFrom может привести к большому количеству предложений, задающих отличие одного представления от другого. Для таких случаев в OWL введен класс owl:AllDifferent, для которого определено свойство owl:distinctMembers, связывающее экземпляр owl:AllDifferent со списком представителей (это свойство может использоваться только как субъект owl:AllDifferent). В этом списке указываются все представители, отличающиеся друг от друга. Зададим всех отличных друг от друга представителей класса Student, указанных в примере выше, в одном списке в элементе owl:AllDifferent:

```
<owl:AllDifferent>
        <owl:distinctMembers rdf:parseType="Collection">
            <inst:Student rdf:about="#studIvanov1"/>
            <inst:Student rdf:about="#studIvanov2"/>
            <inst:Student rdf:about="#studIvanov3"/>
         </owl:distinctMembers>
</owl:AllDifferent>.
```

4.5. Профили языка OWL

Спецификация OWL включает в себя ряд так называемых профилей. Одни из них являются известными подмножествами языка OWL DL, а другие обладают большей выразительностью, чем OWL DL, но при этом не поддерживают полную семантику языка OWL Full. Многие существующие онтологии используют только ограни-ченный набор языковых конструкций, доступных в дескрипционных логиках. Именно это послужило поводом к разработке различных профилей языка OWL. Использование менее выразительного языка позволяет значительно увеличить производительность машин логи-ческого вывода. Поэтому стандартная библиотека профилей языка OWL2 с различными вариантами компромиссов между выразительностью и вычислительной сложностью очень полезна на практике.

Для любого профиля справедливо следующее:

- профиль ограничен синтаксисом, причем значение синтаксиса определяется спецификацией языка OWL DL;
- профиль определяется логикой, которая позволяет осуществить вывод за полиномиальное время в зависимости либо от количества фактов в онтологии, либо от размера онтологии в целом.

Рассмотрим профили, определенные в языке OWL, и области их применения.

OWL EL. Профиль EL является расширением дескрипционной логики EL. Его основная особенность – выполнение за полиномиальное время вывода для онтологий с большим количеством аксиом классов. Данный профиль разработан для поддержи выразительной мощности нескольких существующих крупных онтологий в сфере здравоохранения и наук о жизни (например, SNOMED-CT, Gene Ontology и GALEN). Основное преимущество профиля OWL EL заключается в работе с пересечениями

классов и ограничениями существования. Язык считается легким и поддерживает обоснованный и полный вывод за полиномиальное время.

OWL QL. Машины вывода, разработанные для языков OWL DL и OWL EL, оптимизированы для вывода на аксиомах классов и относительно неэффективны для работы с онтологиями, содержащими достаточно простые определения классов и большое количество утверждений об экземплярах. Профиль OWL QL был разработан для обеспечения эффективной обработки запросов именно к таким онтологиям. Профиль использует технологии управления реляционными базами данных. Он основан на семействе дескрипционных логик DL Lite и расширен такими выразительными языковыми примитивами, как аксиомы включения свойств, функциональные и обратно-функциональные объектные свойства.

OWL2 RL. Профиль OWL RL основан на так называемых программах дескрипционной логики (Description Logic Programs, DLP) и обеспечивает взаимодействие между дескрипционными логиками и правилами. Фактически профиль представляет собой крупнейший синтаксический фрагмент языка OWL DL, реализуемый с помощью правил. Это очень важная особенность, так как правила эффективно ыполняются параллельно, что позволяет разрабатывать масштабируемые реализации логического вывода. OWL RL отличается от профилей QL и EL тем, что он представляет собой некоторое промежуточное звено между языками OWL DL и OWL Full: Машины вывода, основанные на правилах, могут легко игнорировать ограничения OWL DL (такие как различие между классами и индивидами). Это означает, что реализации правил языка OWL RL можно расценивать как реализации подмножества языка OWL Full. Многие из наиболее масштабируемых машин вывода для семантического веба основаны на использовании языка OWL RL или очень похожего языка, называемого pD* или OWL-Horst. Набор правил, который должен быть

реализован в таких приложениях, опубликован в спецификации профиля OWL RL.

ГЛАВА 5. ОНТОЛОГИЧЕСКИЙ ИНЖИНИРИНГ В СИСТЕМЕ PROTEGE

5.1. Программный инструментарий для онтологического инжиниринга

При проектировании и разработке онтологий целесообразно пользоваться подходящими инструментами. Инструментальная поддержка одновременно важна как для процесса разработки, так и для использования онтологических моделей в различных областях. Будем называть инструментальные программные средства, созданные специально для проектирования, редактирования и анализа онтологий, редакторами онтологий. Основная функция любого редактора онтологий состоит в поддержке процесса формализации знаний и представлении онтологии как спецификации (точного и полного описания). В большинстве своем современные редакторы онтологий предоставляют средства кодирвоания формальной модели в том или ином виде. Некоторые дают дополнительные возможности по анализу онтологии, используют механизм логического вывода. Рассмотрим основные характеристики редакторов онтологий.

Поддерживаемые редактором формализмы и форматы представления. Под формализмом понимается теоретический базис, лежащий в основе способа представления онтологических знаний. Примерами формализмов могут служить логика предикатов (First Order Logic - FOL), дескриптивная логика, фреймовые модели (Frames), концептуальные графы и т.п. Формализм, используемый редактором, может не только существенно влиять на внутренние структуры данных, но и определять формат представления или даже пользовательский интерфейс.

Формат представления онтологии в редакторе. Формат представления онтологии задает вид хранения и способ передачи онтологических описаний. Под форматами подразумеваются языки представления онтологий: RDF, OWL, KIF, SCL. Таким образом, некоторая

формальная модель представляется в формализме FOL и может быть выражена средствами языка KIF. Редакторы онтологий обычно поддерживают работу с несколькими формализмами и форматами представления, но часто только один формализм является «родным» (native) для данного редактора.

Функциональность редактора онтологий. Важной характеристикой является функциональность редактора, т.е. множество сценариев его использования. Базовый набор функций обеспечивает:

- работу с одним или более проектами:
- сохранение проекта в нужном формализме и формате (экспорт);
- открытие проекта;
- импорт из внешнего формата;
- редактирование метаданных проекта (в широком смысле: от настройки форм редактирования и представления данных до поддержки версий проекта);
- редактирование онтологии. Набор возможных действий обычно включает создание, редактирование, удаление понятий, отношений, аксиом и прочих структурных элементов онтологии, редактирование таксономии.

К дополнительным возможностям редакторов относят поддержку языка запросов (для поиска нетривиальных утверждений), анализ целостности, использование механизма логического вывода, поддержку многопользовательского режима, поддержку удаленного доступа через Интернет.

Наличие сложных инструментальных средств. Эти средства нужны для того, чтобы не только вводить и редактировать онтологическую информацию, но и анализировать ее, выполняя типичные операции над онтологиями, например, выравнивание, отображение, объединение онтологий.

Рассмотрим наиболее известные инструменты инженерии онтологий.

Ontolingua. Система Ontolingua была разработана в KSL (Knowledge Systems Laboratory) Стенфордского университета и стала первым инструментом инженерии онтологий. Она состоит из сервера и языка представления знаний. Сервер Ontolingua организован в виде набора онтологий, относящихся к Web-приложениям, которые надстраиваются над системой представления знаний Ontolingua. Редактор онтологий – наиболее важное приложение сервера Ontolingua является Web-приложением на основе форм HTML. Кроме редактора онтологий Сервер Ontolingua включает сетевое приложение Webster (получение определений концептов), сервер OKBC (доступ к онтологиям Ontolingua по протоколу OKBC) и Chimaera (анализ, объединение, интегрирование онтологий). Все приложения, кроме сервера OKBC, реализованы на основе форм HTML. Система представления знаний реализована на Lisp. Сервер Ontolingua также предоставляет архив онтологий, включающий большое количество онтологий различных предметных областей, что позволяет создавать онтологии из уже существующих. Сервер поддерживает совместную разработку онтологии несколькими пользователями, для чего используются понятия пользователей и групп. Система включает графический браузер, позволяющий просмотреть иерархию концептов, включая экземпляры. Ontolingua обеспечивает использование принципа множественного наследования и богатый набор примитивов. Сохраненные на сервере онтологии могут быть преобразованы в различные форматы для использования другими приложениями, а также импортированы из ряда языков в язык Ontolingua.

OntoEdit. Редактор OntoEdit первоначально был разработан в институте AIFB (Institute of Applied Informatics and Formal Description Methods) и обеспечивает проверку, просмотр, кодирование и

модификацию онтологий. В настоящее время OntoEdit поддерживает языки представления: FLogic, включая машину вывода, OIL, расширение RDFS и внутреннюю, основанную на XML, сериализацию модели онтологии используя OXML - язык представления знаний OntoEdit (OntoEdit"s XML-based Ontology representation Language). К достоинствам инструмента можно отнести удобство использования; разработку онтологии под руководством методологии и с помощью процесса логического вывода; разработку аксиом; расширяемую структуру посредством плагинов, а также очень хорошую документацию. OntoEdit - автономное Java-приложение, которое можно локально установить на компьютере, но его коды закрыты. Существует две версии OntoEdit: свободно распространяемая OntoEdit Free (ограничена 50 концептами, 50 отношениями и 50 экземплярами) и лицензированная OntoEdit Professional (нет ограничений на размер). Естественно, что OntoEdit Professional имеет более широкий набор функций и возможностей (например, машину вывода, графический инструмент запросов, больше модулей экспорта и импорта, графический редактор правил, поддержка баз данных JDBC и т.д.).

OilEd. OilEd – автономный графический редактор онтологий, разработан в Манчестерском университете в рамках европейского IST проекта On-To-Knowledge. Инструмент основан на языке OIL (сейчас адаптирован для DAML+OIL, в перспективе - OWL), который сочетает в себе фреймовую структуру и выразительность дескриптивной логики (Description Logics) с сервисами рассуждения. Из недостатков можно выделить отсутствие поддержки экземпляров. Существующая версия не обеспечивает полную среду разработки – не поддерживается разработка онтологий большого масштаба, миграция и интеграция онтологий, контроль версий и т.д. OilEd свободно распространяется по общедоступной лицензии GPL.

WebOnto. Редактор *WebOnto* предназначен для поддержки совместного просмотра, создания и редактирования онтологий. Его достоинства – простота использования, предоставление средств масштабирования для построения больших онтологий. Для моделирования онтологий WebOnto использует язык OCML (Operational Conceptual Modeling Language). В WebOnto пользователь может создавать структуры, включая классы с множественным наследованием, что можно выполнять графически. Все слоты наследуются корректно. Инструмент проверяет вновь вводимые данные контролем целостности кода OCML. Инструмент имеет ряд полезных особенностей: сохранение структурных диаграмм, раздельный просмотр отношений, классов, правил и т.д. Другие возможности включают совместную работу нескольких пользователей над онтологией, использование диаграмм, функций передачи и приёма и др.

KADS22. KADS22 - инструмент поддержки проектирования моделей знаний согласно методологии CommonKADS. Модели CommonKADS определены в CML (Conceptual Modeling Language). KADS22 представляет интерактивный графический интерфейс для CML со следующими функциональными возможностями: синтаксический анализ файлов CML, печать, просмотр гипертекста, поиск, генерация глоссария и генерация HTML.

Protégé. Protégé – локальная, свободно распространяемая Java-программа, разработанная группой медицинской информатики Стенфордского университета. Программа предназначена для построения (создания, редактирования и просмотра) онтологий прикладной области. Её первоначальная цель – помочь разработчикам программного обеспечения в создании и поддержке явных моделей предметной области и включение этих моделей непосредственно в программный код. Protégé включает редактор онтологий, позволяющий проектировать онтологии, разворачивая иерархическую структуру абстрактных или

конкретных классов, свойств. Структура онтологии сделана аналогично иерархической структуре каталога. На основе сформированной онтологии Protégé может генерировать формы получения знаний для введения экземпляров классов и подклассов. Инструмент имеет графический интерфейс, удобный для использования неопытными пользователями, снабжен справками и примерами. Protégé основан на фреймовой модели представления знания OKBC (Open Knowledge Base Connectivity) и снабжен рядом плагинов, что позволяет его адаптировать для редактирования моделей хранимых в разных форматах (стандартный текстовый, в базе данных JDBC, UML, языков XML, XOL, SHOE, RDF и RDFS, DAML+OIL, OWL).

5.1. Создание нового проекта в PROTEGE

Запустите Protégé. После запуска программы появится окно проекта Protégé. Пользовательский интерфейс состоит из главного меню и нескольких вкладок. Новый проект открывается на вкладке «Active Ontology», представленной на рис. 5.1.

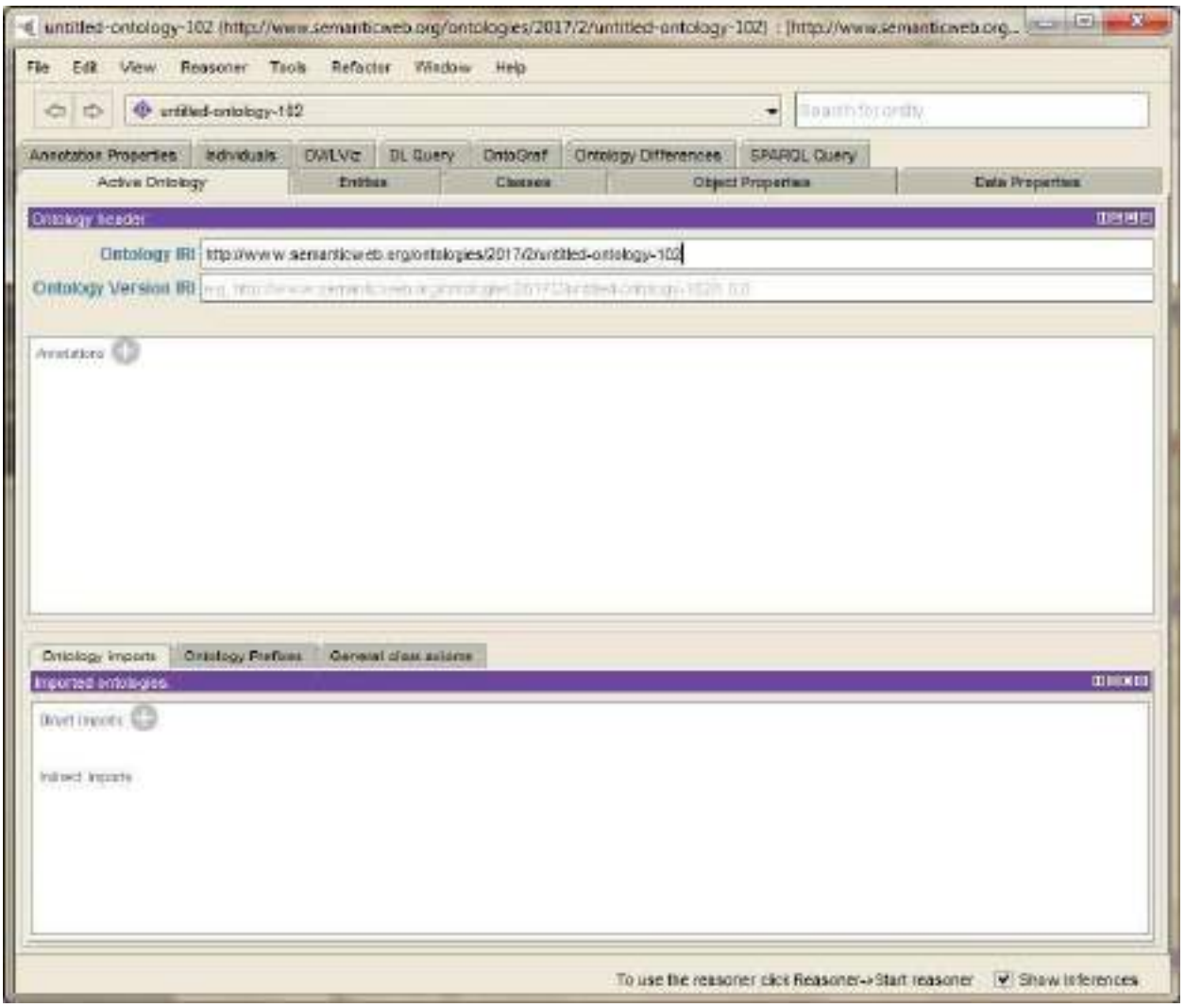


Рисунок 5.1. Вкладка «Active Ontology»

Protégé позволяет работать с несколькими онтологиями сразу. На вкладке «Active Ontology» представлена информация о той онтологии, с которой пользователь работает в настоящий момент, а именно:

- IRI (Internationalized Resource Identifier, международный идентификатор ресурса) текущей онтологии и IRI ее версии соответственно, присваиваемые по умолчанию;
- комментарии пользователя к текущей онтологии.

Создание стандарта IRI имеет целью преодоление ограниченных возможностей стандарта Uniform Resource Identifier (URI). Ограниченность возможностей идентификации ресурсов, предусмотренных стандартом URI, заключается в том, что такие идентификаторы могут содержать только латинские символы и знаки

препинания из набора символов ASCII. В случае использования символов национальных алфавитов в URI потребуется кодировать их в Unicode, а символы Unicode, в свою очередь, представлять в URI в виде специальных комбинаций символов ASCII. В результате идентификатор URI становится нечитаемым. Использование IRI позволяет применять символы Unicode. IRI определяет возможности использования для идентификации ресурсов Internet удобочитаемых строк литер. Эти строки могут быть, в частности, мнемоничными и содержать литеры национальных алфавитов.

Замените заданный по умолчанию идентификатор онтологии на http://www.semanticweb.org/ontologies/Lab (рис. 5.2):

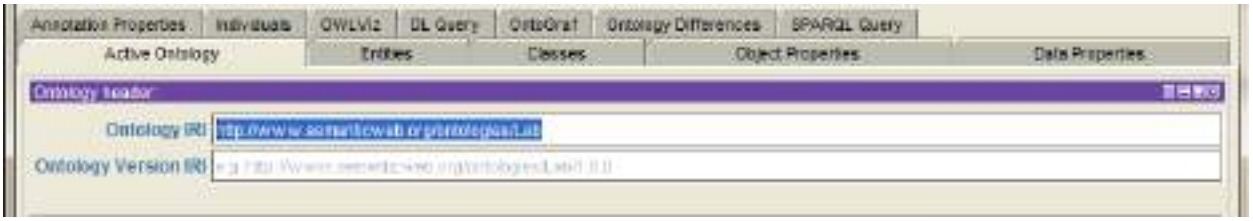


Рисунок 5.2. Изменение IRI онтологии

Вкладка «Active Ontology» имеет несколько вкладок: «Ontology Imports», «Ontology Prefixes», «General class axioms».

Вкладка «Ontology Imports» содержит средства для импорта других онтологий и их слияния с текущей. Импорт другой онтологии переносит весь набор утверждений, обеспеченных в той онтологии, в текущую онтологию.

Вкладка «Ontology Prefixes». Каждому объекту, созданному в Protégé, присваивается международный идентификатор ресурса (IRI), который достаточно длинный. Данная вкладка предназначена для указания аббревиатур (префиксов) для используемых IRI.

Вкладка «General class axioms» предназначена для задания общих (генеральных) аксиом класса.

5.2. Создание комментариев к онтологии

Protégé позволяет аннотировать (пояснять) классы, свойства, экземпляры классов и саму онтологию с помощью фрагментов информации (методанных). Ограничения на использовании аннотации:

- в аннотации должны находиться либо литералы данных (например, «Иван», 18, 3.14), либо ссылка URI или экземпляры классов;
- свойства аннотации не могут быть использованы в аксиомах свойств.

Для создания комментария на вкладке «Active Ontology» щелкните значок (+) рядом с «Annotations». Появится окно редактирования аннотации «Create Annotation» (рис. 5.3).

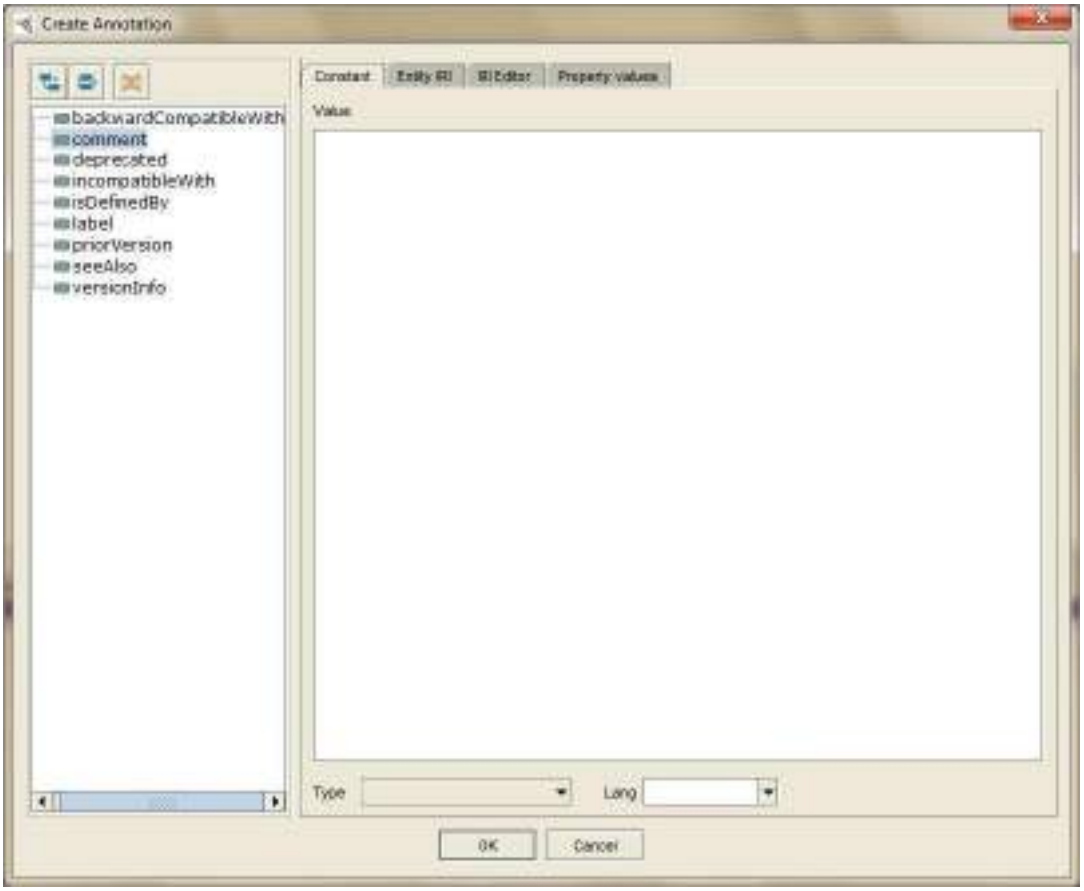


Рисунок 5.3. Окно «Create Annotation»

В Protégé имеется несколько предопределенных свойств аннотации, которые могут быть использованы для аннотирования классов, свойств, экземпляров:

- **comment** – свойство предназначено для пояснения смысла элемента онтологии, имеет диапазон значений «строка»;
- **versionInfo** – значение свойства хранит информацию о версии, диапазон значений атрибута – «строка»;
- **backwardCompatibleWith** – значение свойства определяет предыдущую версию онтологии, совместимую с текущей версией;
- **inconpatibleWith** – значение свойства определяет предыдущую версию онтологии, не совместимую с текущей.
- **label** – свойство используется для придания смысла именам элементов онтологии и имеет диапазон значений «строка»;
- **seeAlso** – свойство имеет значение «URI-ссылка» и используется для идентификации соответствующих ресурсов;
- **isDefinedBy** – свойство имеет значение «URI-ссылка» и используется для ссылки на онтологию;
- **priorVersion** – свойство предназначено для идентификации предыдущей версии онтологии;

Выберите из списка в левой части окна пункт «comment» и введите текст «Онтология профессиональной деятельности врача» в текстовое поле в правой части окна.

Нажмите кнопку «OK», чтобы сохранить комментарий. На вкладке «Active Ontology» появится введенный комментарий (рис. 5.4)

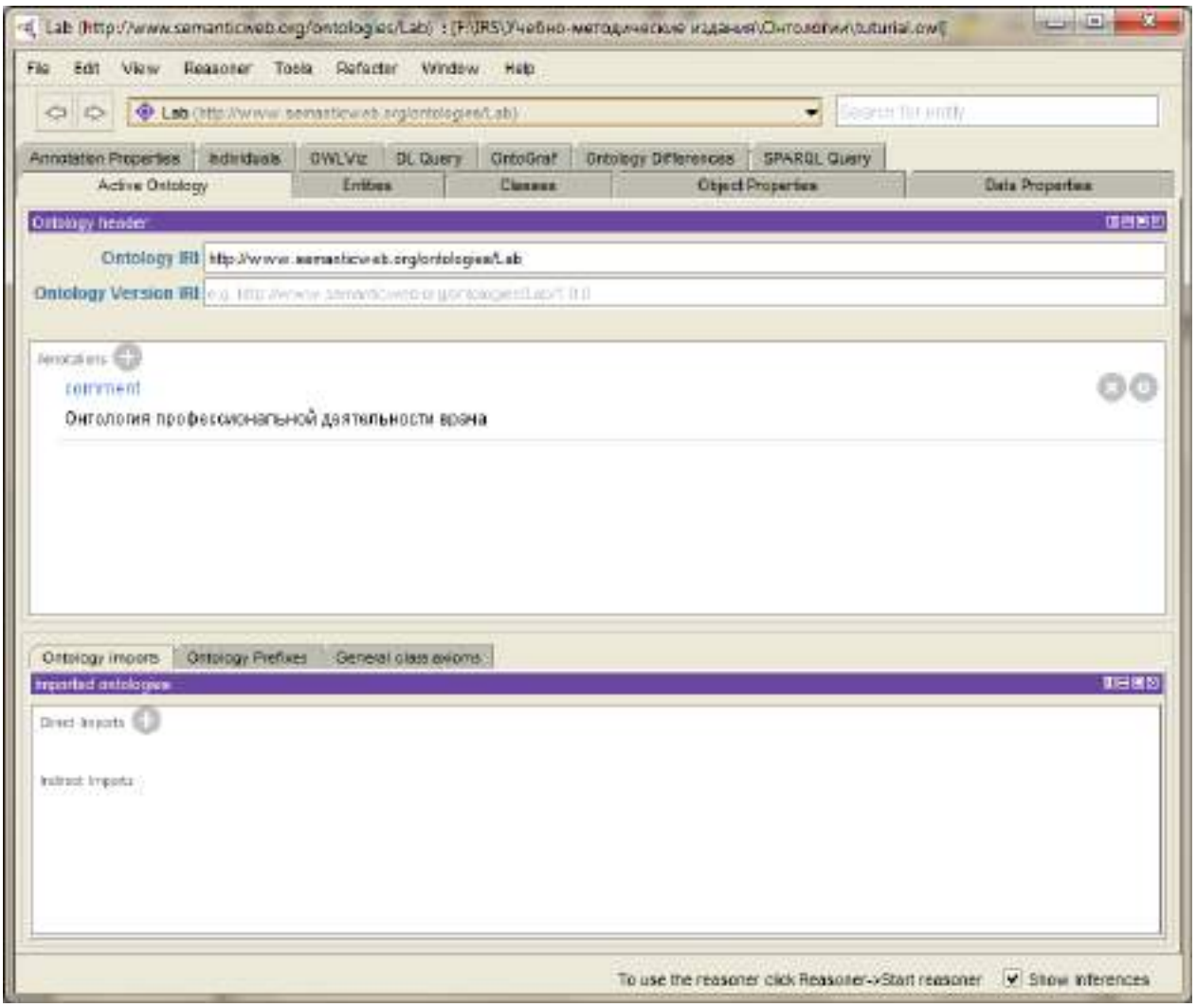


Рисунок 5.4. Вкладка «Active Ontology»

5.3. Сохранение проекта

Для сохранения промежуточных изменений выберите пункт меню «File» ⇒ «Save as». В открывшемся диалоговом окне (рис. 5.5) выберите формат сохранения онтологии «RDF/XML» и нажмите кнопку «OK».

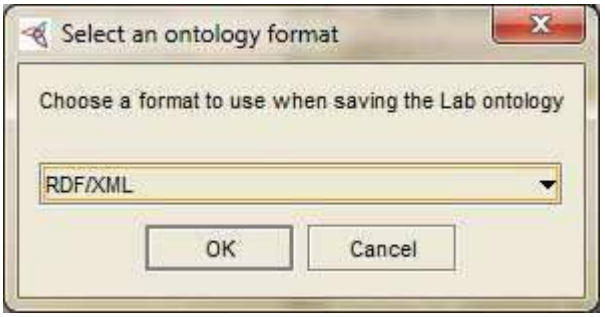


Рисунок 5.5. Окно выбора формата сохранения онтологии

В открывшемся окне (рис. 5.6) укажите место, куда будет сохранен проект, и введите имя файла проекта (например, «tutorial»). Нажмите кнопку **Save.**

Рисунок 5.6. Окно сохранения проекта

5.4. Способы описания классов

Класс онтологии интерпретируется как некоторое множество объектов предметной области. Иерархия классов описывается логической формулой – импликацией $A \to B$.

Класс может быть описан несколькими способами:

- идентификатором класса (IRI);
- перечислением всех экземпляров класса;
- ограничением на значение свойства;
- с помощью теоретико-множественных операций (пересечение2-х и более классов, объединение, дополнением).

Только первый способ определяет именованный класс OWL. Все оставшиеся определяют анонимный (безымянный) класс через ограничение его экстенсионала. Второй способ явно перечисляет

экземпляры класса, третий ограничивает экстенсионал только теми экземплярами, которые удовлетворяют данному свойству. Четвертый способ используют теоретико-множественные операции (объединение, пересечение и дополнение) над экстенсионалами соответствующих классов, чтобы определить экстенсионал нового класса.

Классы образовывают иерархию обобщения (родительский класс – подкласс), иначе называемую таксономией. В Protege существует класс owl:Thing, содержащий все объекты. Данный класс является родительским по отношению ко всем остальным именованным классам. Средства автоматической категоризации на основании онтологии проверяют непротиворечивость иерархии классов и достраивают ее.

5.5. Создание именованного класса

Перейдите на вкладку «Classes» (рис. 5.7). На вкладке «Classes» найдите область «Class Hierarchy» (в окне Protégé слева). Эта область отображает иерархию классов с выделенным текущим выбранным классом. Пустая онтология содержит один класс с именем **THING**. Класс **THING** – это универсальное множество, содержащее все объекты предметной области. По этой причине все классы являются подклассами **THING**.

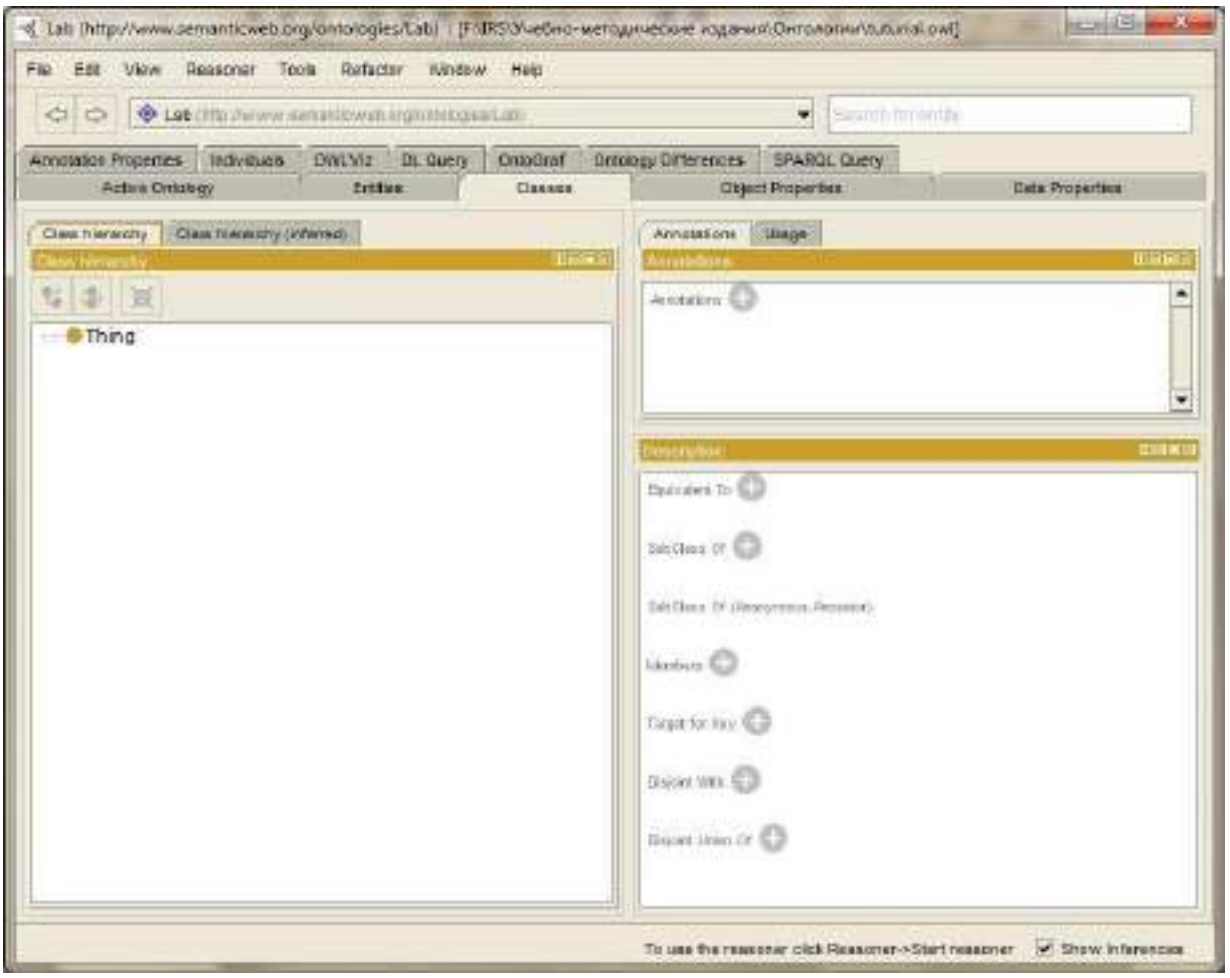


Рисунок 5.7. Вкладка «Classes»

Выделите класс **THING** и нажмите кнопку – «Add subclass» («Создать подкласс») (рис. 7)

Рис. 5.8. Создание подкласса

Появится окно редактора классов (рис. 5.9). В системе Protégé приняты правила наименования: все имена классов должны начинаться с

прописной буквы и не содержать пробелов. Введите имя класса – Врач и нажмите кнопку «OK».

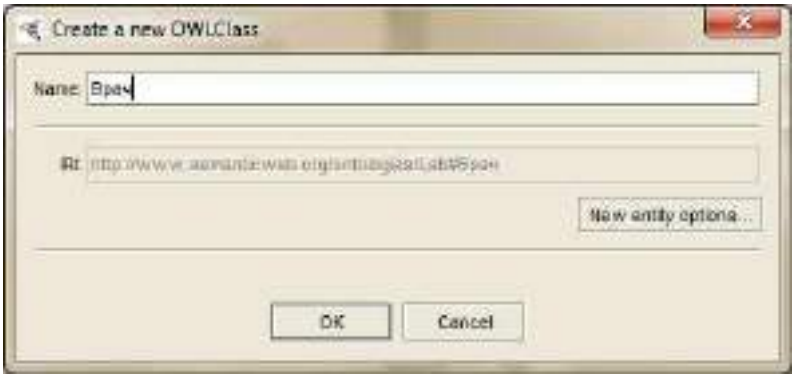


Рисунок 5.9. Окно создания нового класса

Дерево иерархии классов примет вид, представленный на рис. 5.10. Переименовать созданный класс можно с помощью команды меню «Refractor» ⇒ «Rename entity».

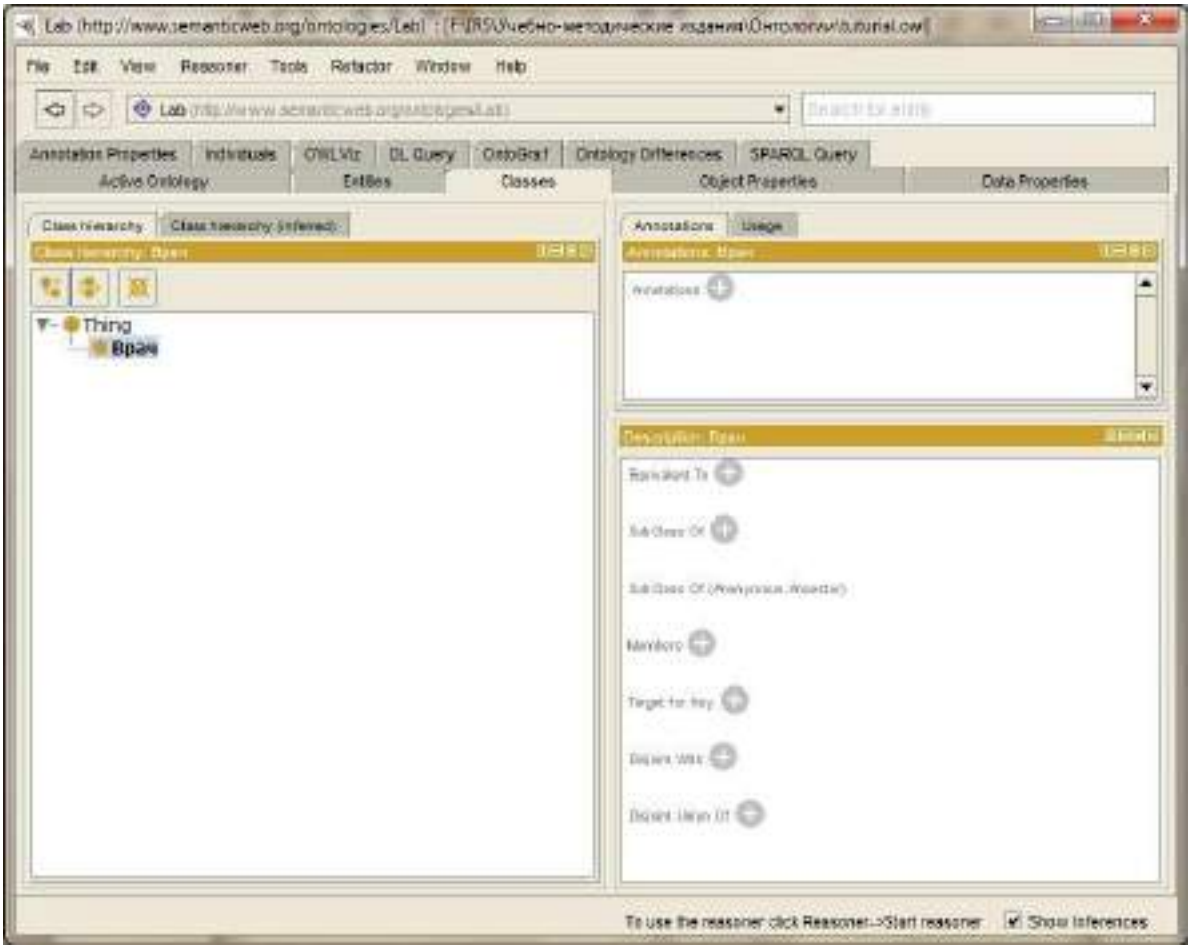


Рисунок 5.10. Иерархия классов

На вкладке «Classes» найдите вкладку «Annotations» (в окне Protégé справа). Выделите класс Врач и щелкните значок (+) рядом с

«Annotations». Появится окно редактирования аннотации к классу Врач. Выберите из списка в левой части окна пункт «comment» и введите текст «Специалист с законченным высшим медицинским образованием» в текстовое поле в правой части окна. Нажмите кнопку «OK», чтобы сохранить комментарий. На экране отобразится созданный комментарий (рис. 11)

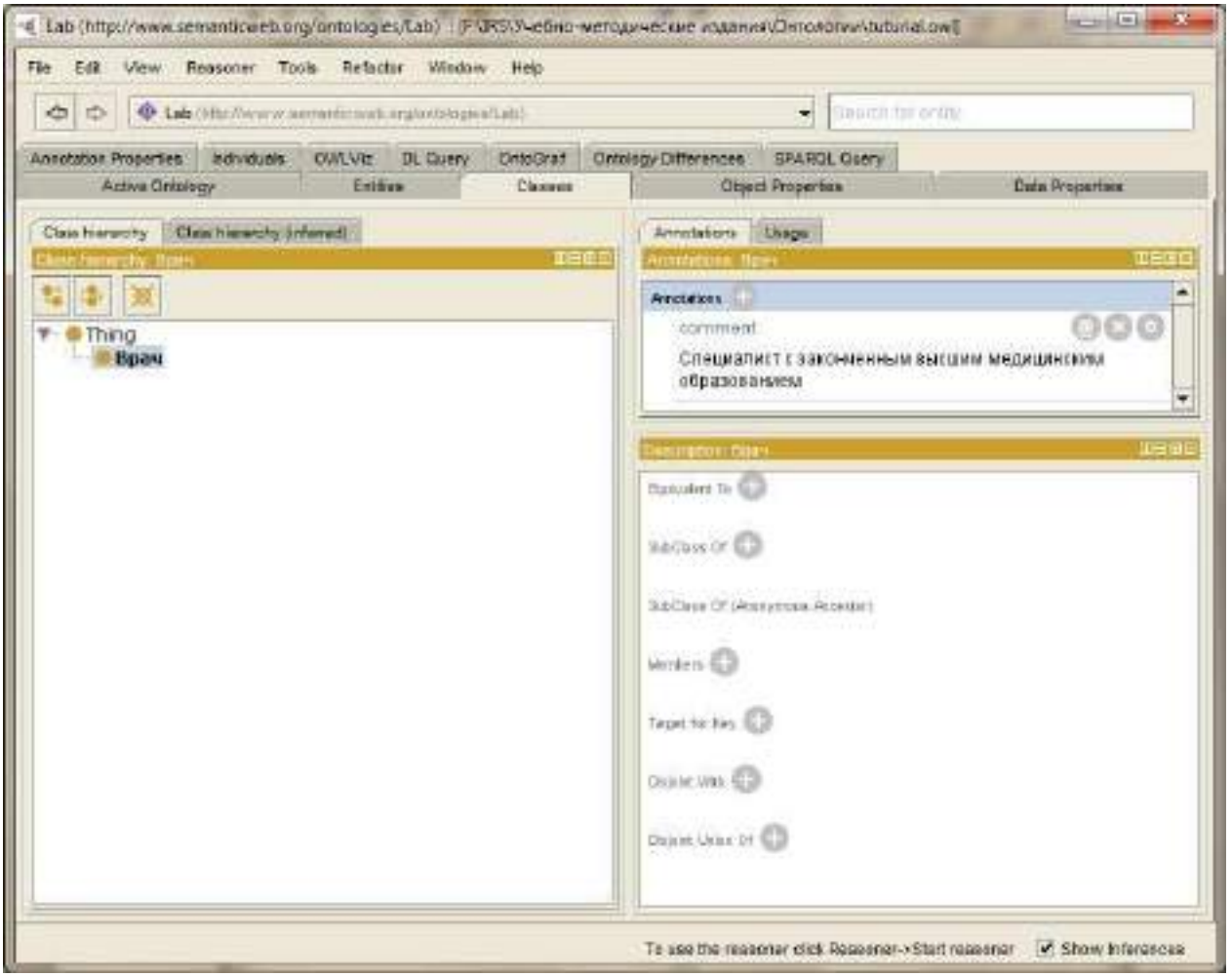


Рисунок 5.11. Создание комментария к классу

Создадим класс Пациент. Для создания нового класса выделите класс THING. Если этого не сделать, то новый класс будет подклассом класса **Врач**. Нажмите кнопку «Add subclass»  . В открывшемся окне наберите с клавиатуры имя класса: Пациент. Нажмите кнопку «OK» для завершения создания класса.

В представлении «Annotations: Пациент» добавьте комментарий к классу Пациент**:** «Физическое лицо, обратившееся за медицинской помощью, находящееся под медицинским наблюдением либо получающее медицинскую помощь». На экране отобразится созданный комментарий (рис. 12).

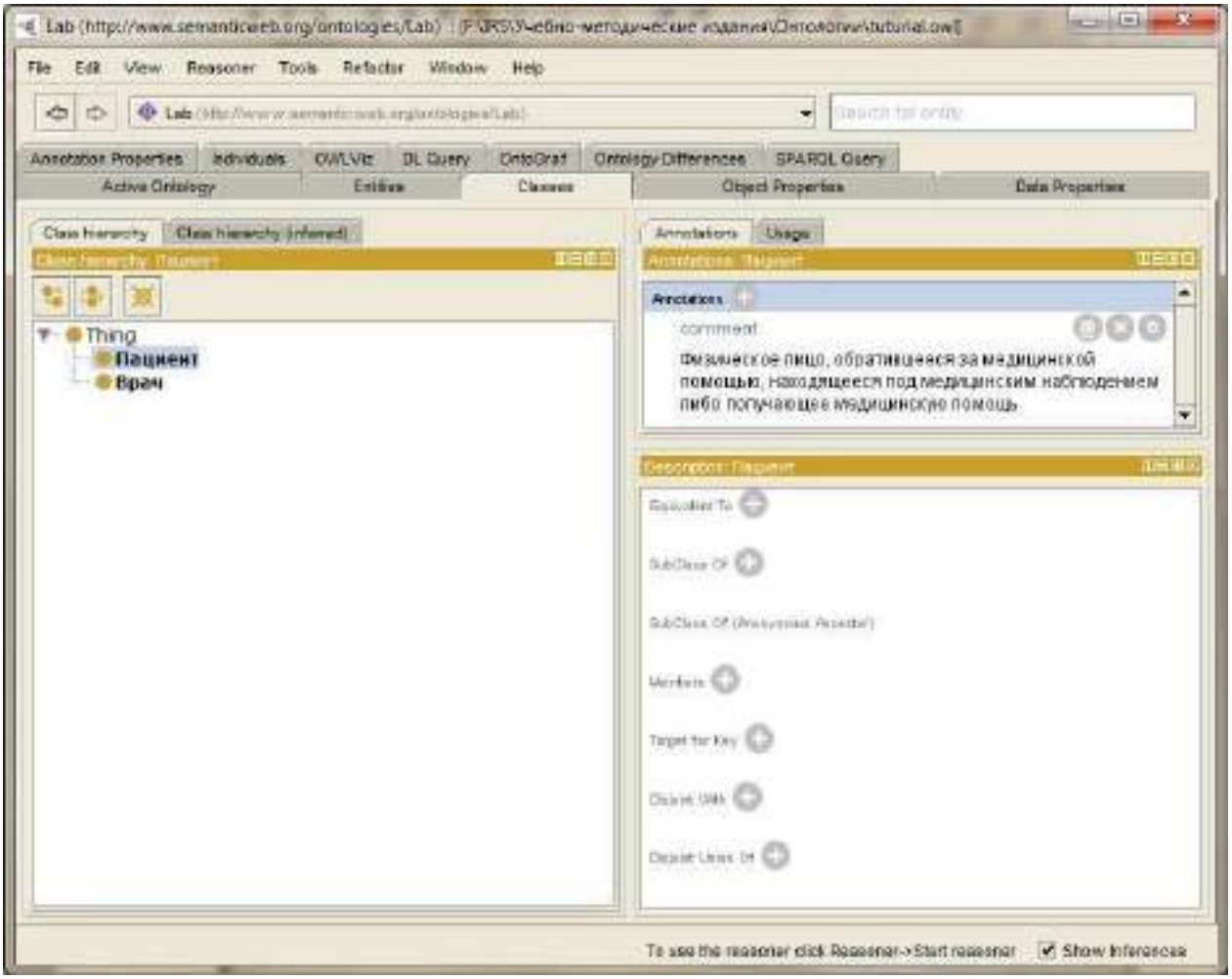


Рисунок 12. Создание класса Пациент

Для создания иерархии классов также можно использовать кнопку – «Add sibling class» («Создать класс одного уровня иерархии»). При нажатии на эту кнопку будет создан класс, находящийся в дереве иерархии на одном уровне с выделенным классом.

Выделите класс Врач и нажмите кнопку – «Add sibling class». В окне редактирования классов введите: Диагноз. Нажмите кнопку «ОК».

Введите аннотацию: «Заключение о сущности болезни и состоянии пациента, выраженное в принятой медицинской терминологии и основанное на всестороннем систематическом изучении пациента». Иерархия классов примет вид, представленный на рис. 13.

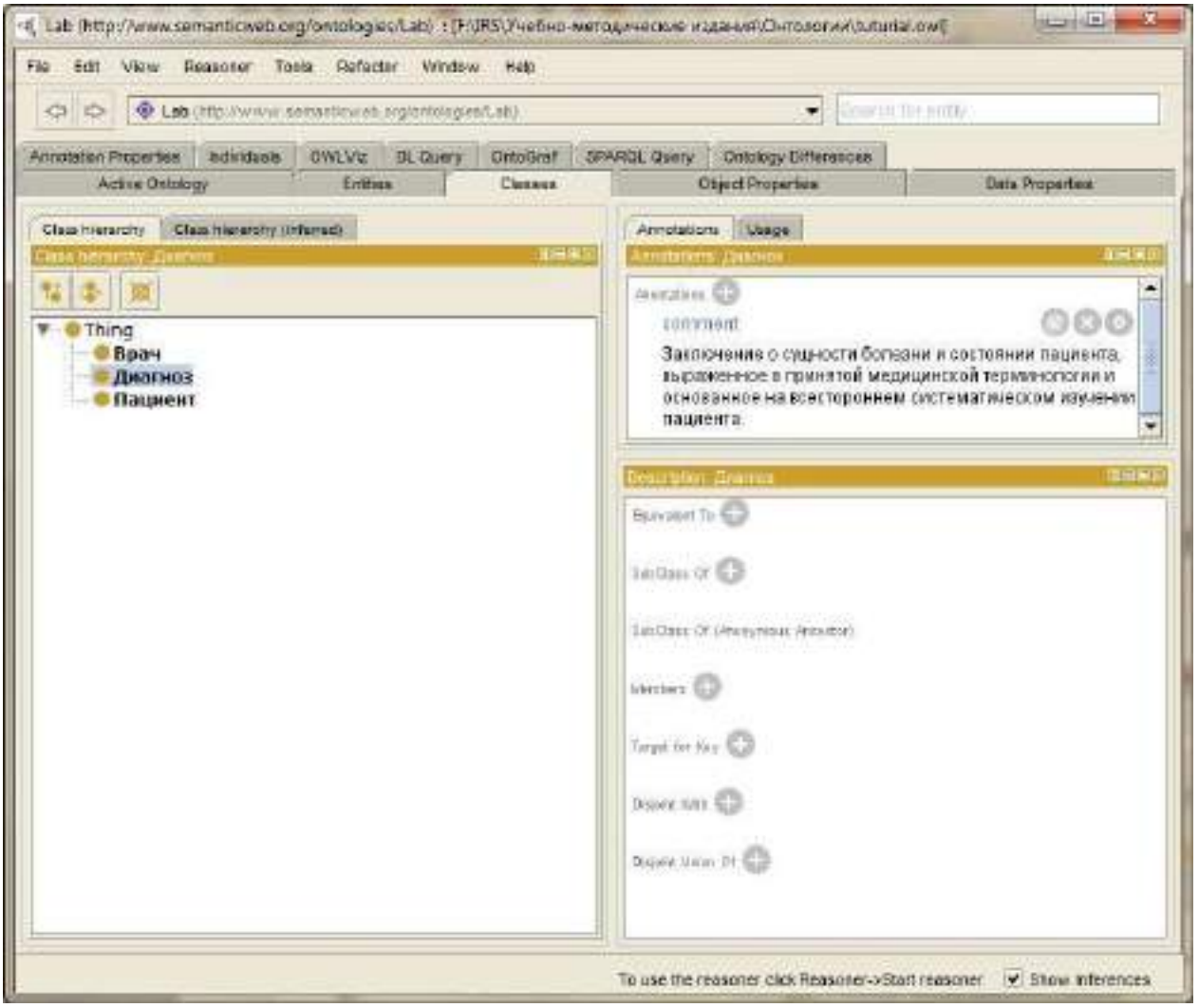


Рисунок 13. Создание класса Диагноз

По умолчанию классы онтологии могут пересекаться. Для того чтобы разделить классы, их необходимо явно сделать непересекающимися.

Объявим, что классы Врач и Диагноз не пересекаются. Для этого выделите класс Врач в иерархии классов и щелкните значок (+) рядом с пунктом «Disjoint with» («Не пересекается с»), который расположен в нижней части области описания класса «Description» (рис. 5.14).

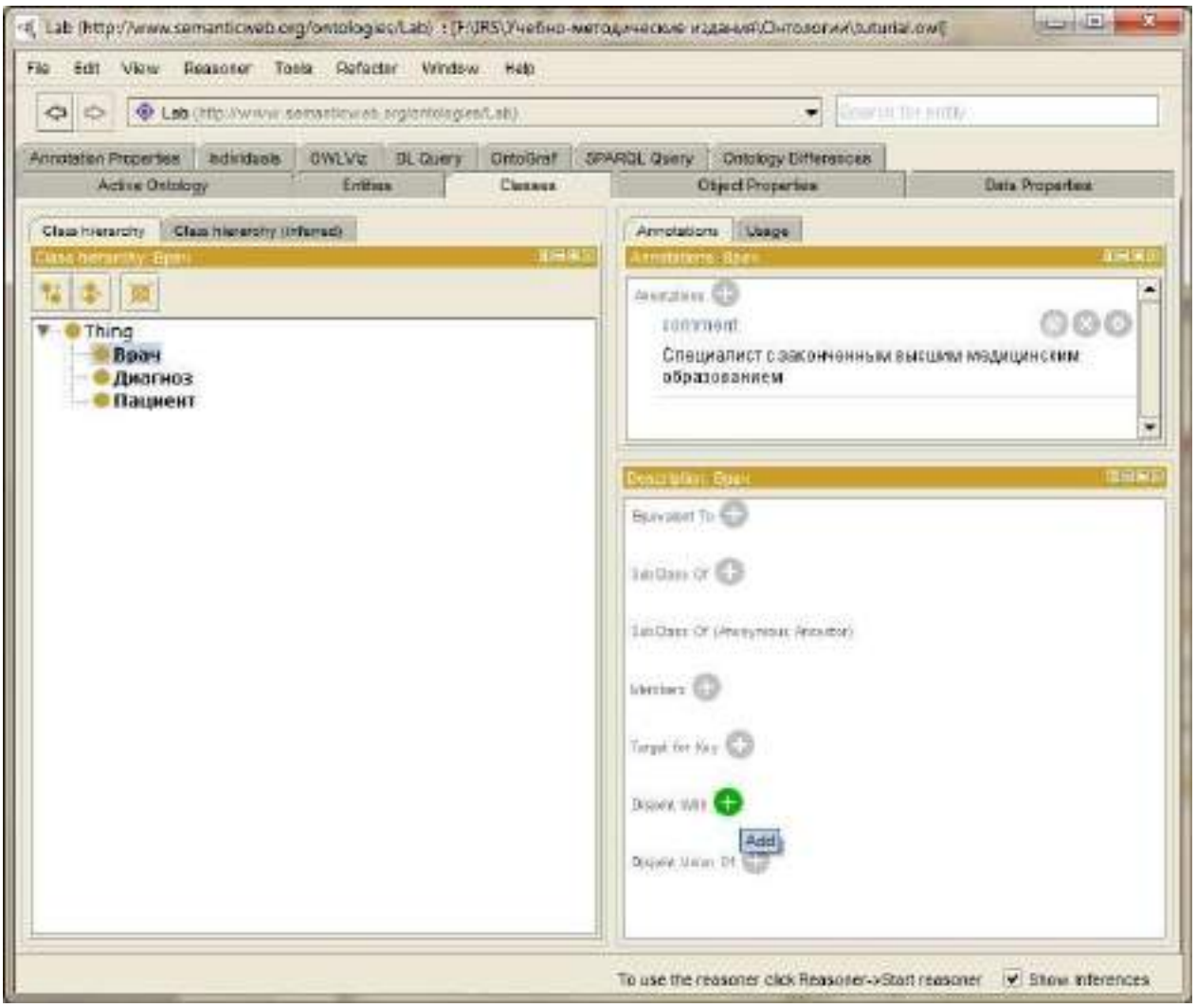


Рисунок 14. Объявление классов непересекающимися

В появившемся окне выберите класс Диагноз (рис. 5.15). Нажмитекнопку «ОК».

Рисунок 14. Объявление класса Диагноз **непересекающимся с классом** Врач

В результате выбранные классы онтологии станут непересекающимися. Аналогично сделайте классы **Пациент** и **Диагноз** непересекающимися (рис. 5.15).

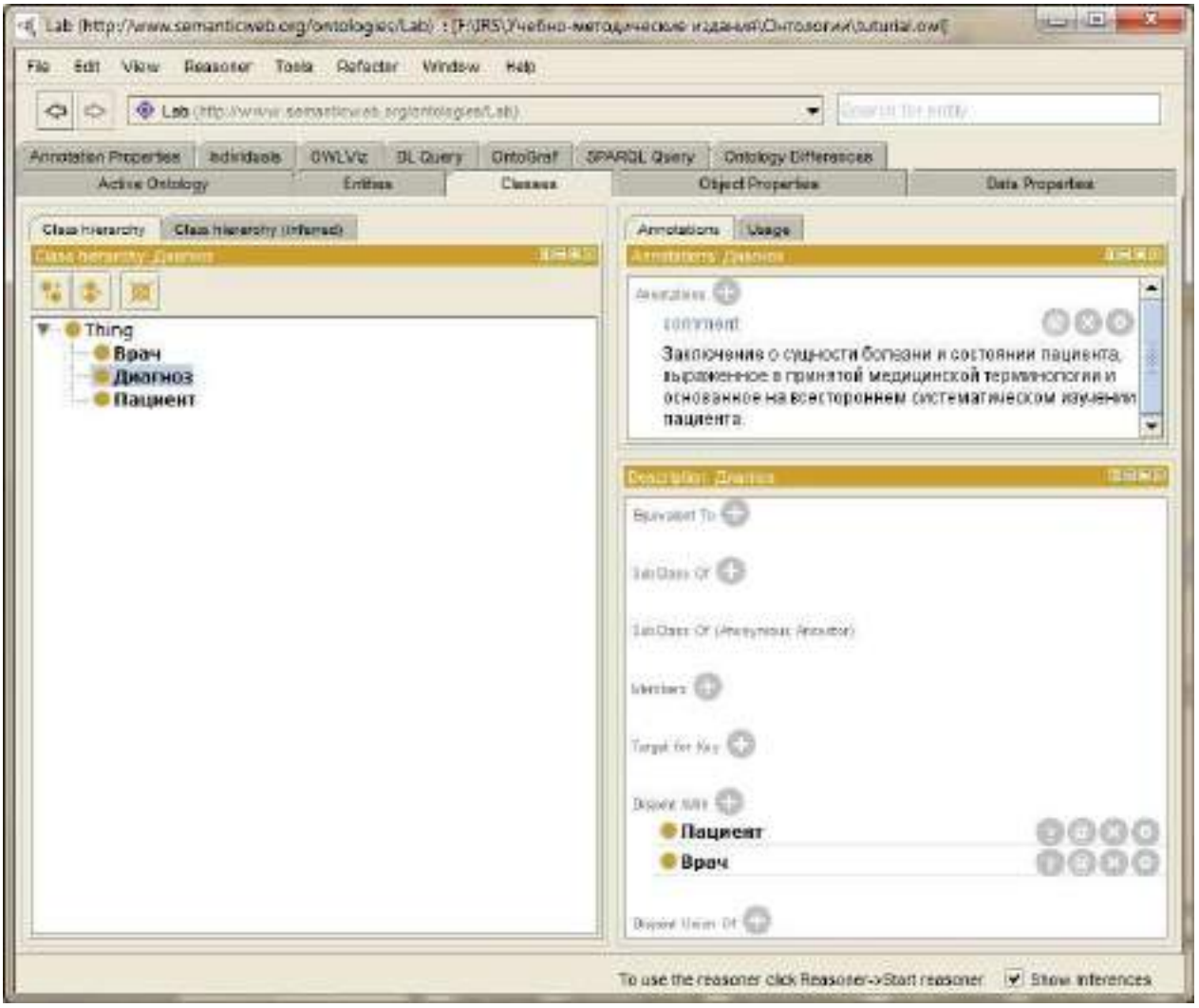


Рис. 5.15 Объявление класса Диагноз **непересекающимся с классом** Пациент

В диалоговом окне на рис. 5.15 можно выбрать несколько классов, непересекающиеся с данным. Для одновременного выбора нескольких классов нажмите клавишу «Ctrl». В результате выбранные классы онтологии станут попарно непересекающимися.

5.6. Использование мастера создания иерархии классов

Врач осуществляет прием – производимые по определенному плану действия врача при возникновении у пациента потребности в медицинской помощи, представляющие собой сложную или комплексную медицинскую услугу, дающие возможность составить представление о состоянии организма пациента, результатом которых является профилактика, диагностика или лечение определенного заболевания, синдрома.

Создайте класс Прием и напишите к нему аннотацию. Для внесения видов приема в онтологию воспользуемся инструментом «Create Class hierarchy» («Создать иерархию классов»). Для этого выберите пункт меню «Tools» ⇒ «Create Class hierarchy». Появится окно (рис. 5.16), в котором следует указать корневой класс создаваемой иерархии. Выберите класс Прием и нажмите кнопку «Continue».

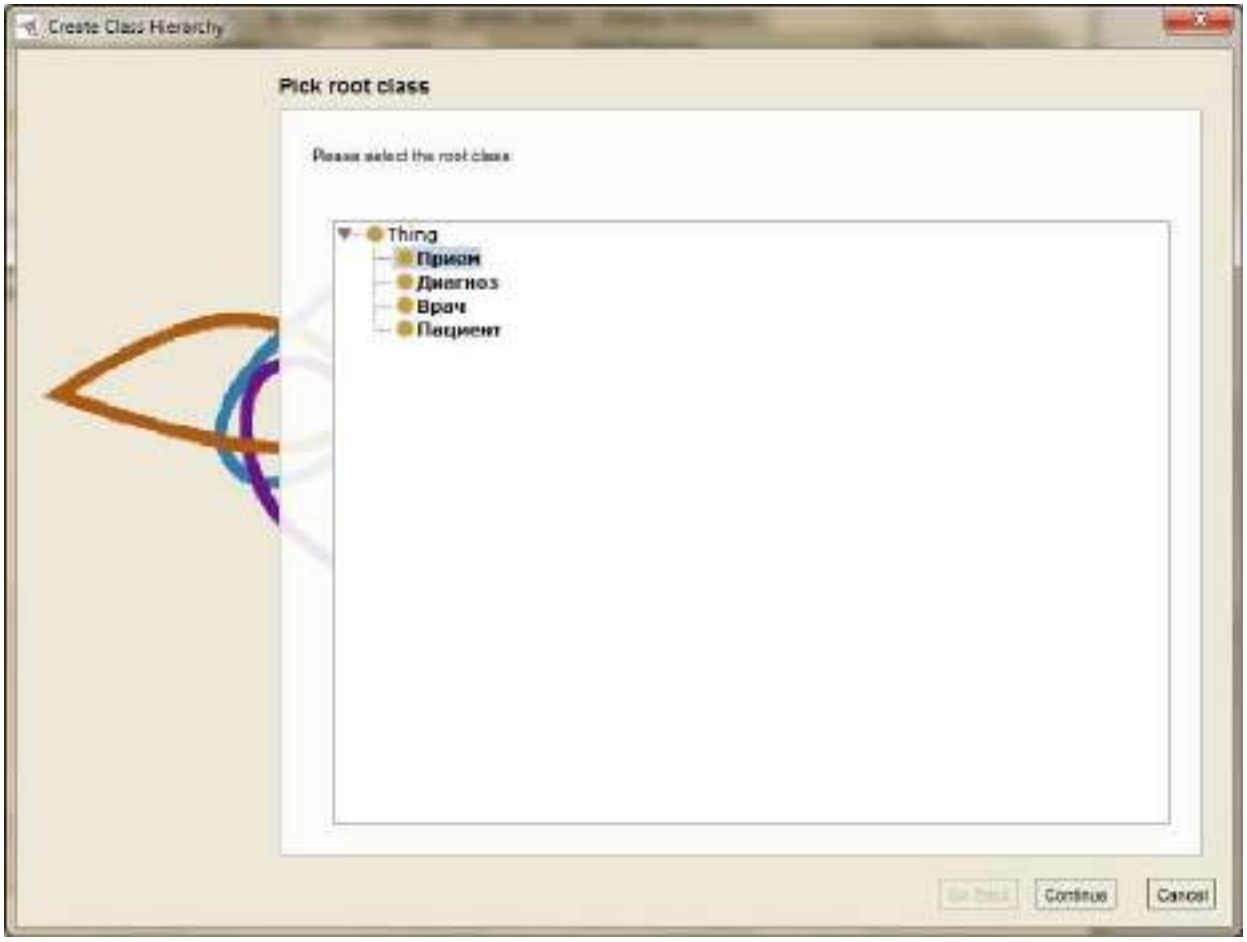


Рис. 5.16. Создание иерархии классов с помощью мастера

Откроется окно создания иерархии классов (рис. 5.17). Поля «Prefix» и «Suffix» используются для автоматического добавления приставки или окончания к именам создаваемых классов. Мастер «Create Class hierarchy» позволяет не только добавить классы к онтологии, но и установить их иерархию с помощью клавиши «Tab». Создайте иерархию классов, как показано на рис. 5.17. Нажмите кнопку «Continue».

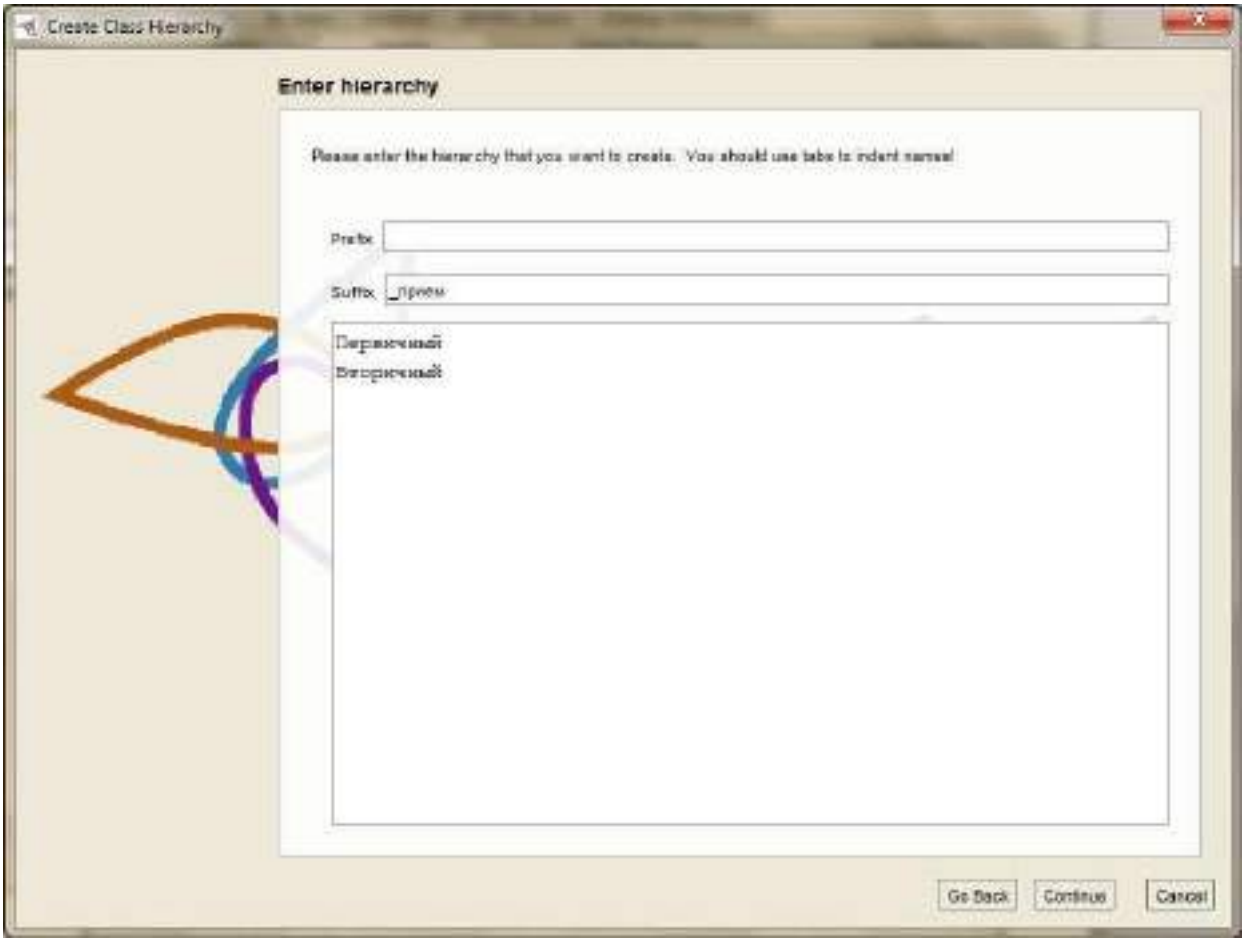


Рис. 5.17 Создание иерархии классов: ввод наименований подклассов

В открывшемся окне (рис. 5.18) убедитесь, что поле «Make sibling classes disjoint» («Сделать одноуровневые классы непересекающимися») отмечено галочкой. Нажмите кнопку «Finish».

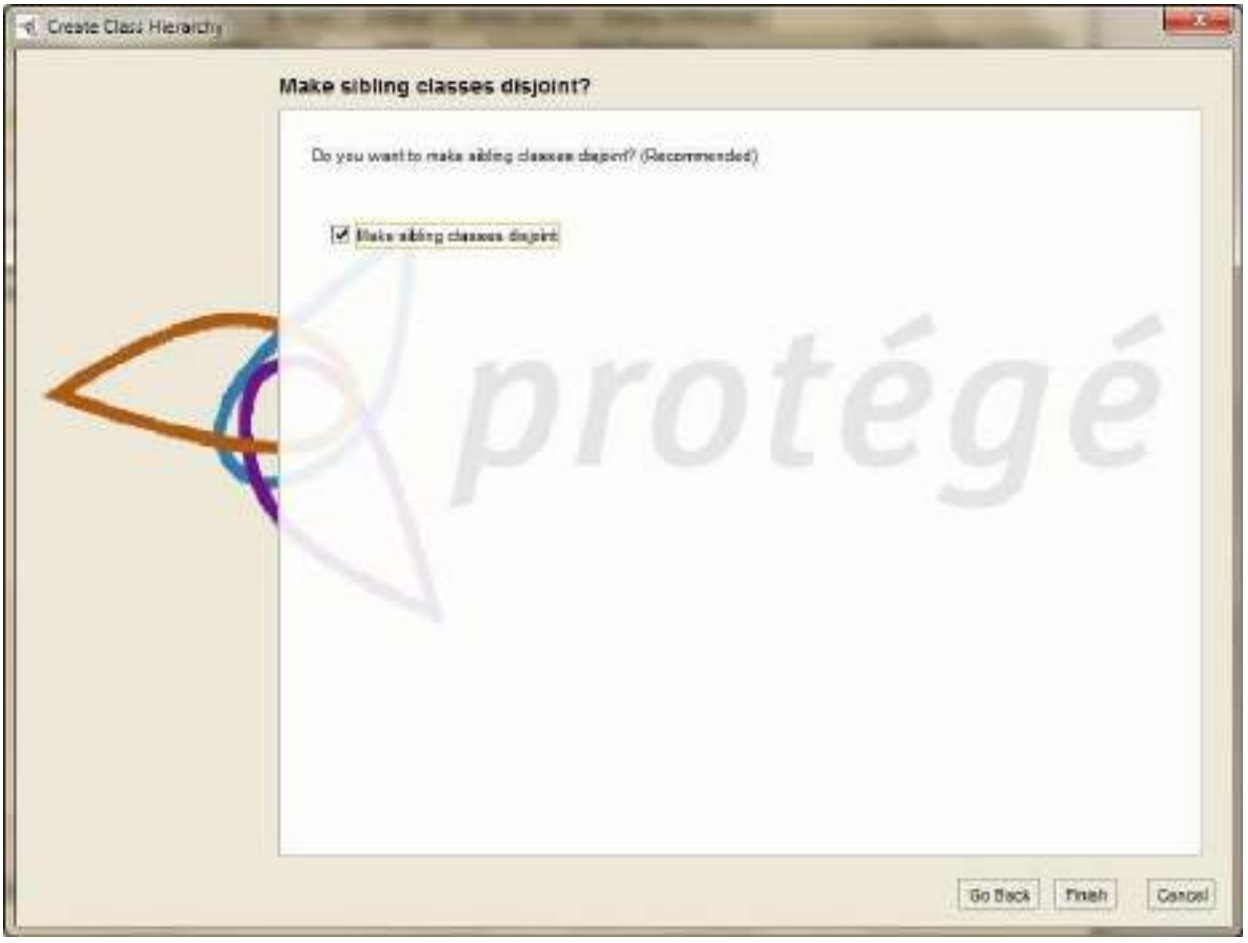


Рис. 5.18 Объявление классов непересекающимися

В результате в онтологию добавлены подклассы класса Прием (рис. 5.19). Обратите внимание, что при выделении подкласса в иерархии классов в поле «Sub Class Of» области «Description» отображается название класса, к которому принадлежит данный подкласс.

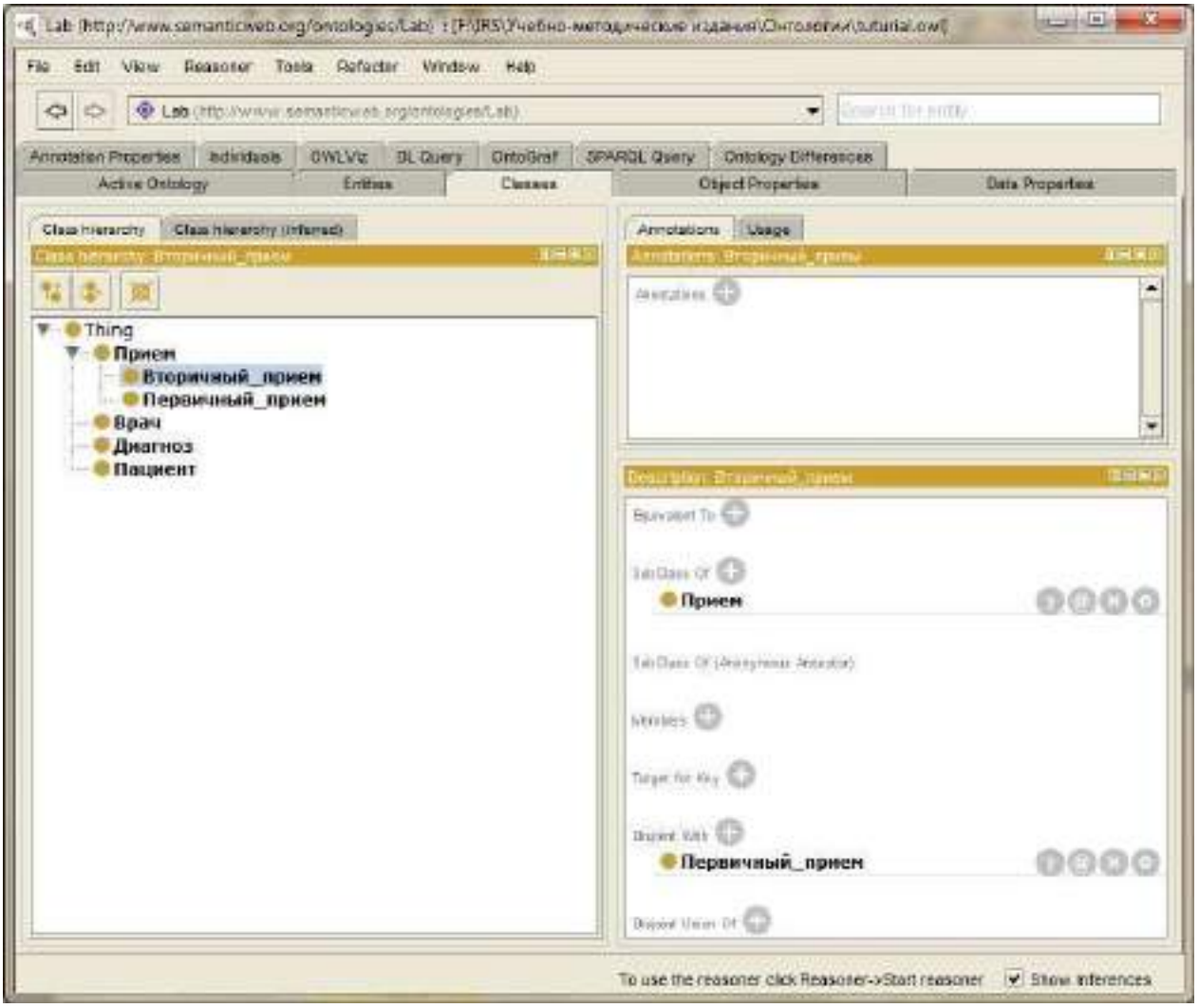


Рис. 5.19. Измененная иерерхия классов

Создайте комментарии к подклассам класса **Прием**.

5.7. Свойства онтологии

Свойства онтологии представляют собой бинарные отношения. Существует два основных типа свойств: свойства объектов и свойства данных.

Свойства объектов отражают отношения между двумя классами онтологии. Формально свойство объектов – это бинарное отношение $R_{об} \subseteq K_1 \times K_2$, где K_1, K_2 – классы онтологии. На рис. 5.20 изображен пример свойства объектов.

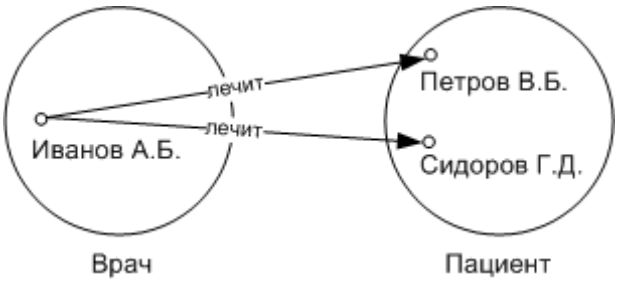


Рис. 5.20. Свойство объектов, связывающее класс Врач **и класс** Пациент

Свойства данных связывают классы с диапазоном значений, которые может принимать экземпляры классов указанного типа. Формально свойство данных – это бинарное отношение $R_{\text{тд}} \subseteq K \times D$, где K – класс онтологии, D множество значений определенного типа данных. На рис. 5.21 изображен пример свойства объектов.

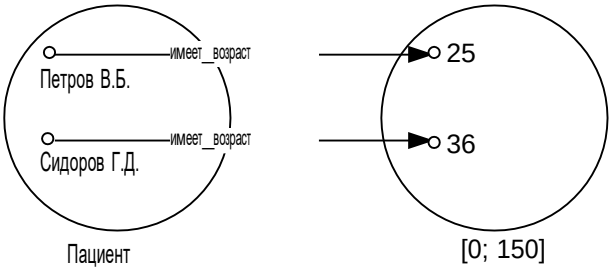


Рис. 5.21. Свойство данных, связывающее класс Пациент **и множество [0; 150]**

Свойства имеют *домен* (область определения отношения, множество всех первых компонент пар из R) и *диапазон* (область значения отношения, множество всех вторых компонент пар из R). Домены и диапазоны выступают в качестве аксиом при выводе и используются для проверки корректности онтологии.

Свойство может иметь соответствующее обратное свойство, а также обладать свойствами транзитивности, симметричности, асимметричности, рефлексивности, антирефлексивности (иррефлексивности), являться функцией.

Хотя не уществует строгого наименования для свойств, рекомендуется имена свойств начинать с маленькой буквы, не допускать пробелов. Также рекомендуется начинать имя свойства с префикса «имеет» или «является», например, имеет_симптомы**.** Также можно формировать имя свойства из наименований связываемых этим свойством множеств. Например, врач_пациент – имя отношения, связывающего экземпляры класса Врач с элементами класса Пациент.

5.8. Создание свойства объектов

Переключитесь на вкладку «Свойства объектов» («Object Properties») (рис. 5.22)

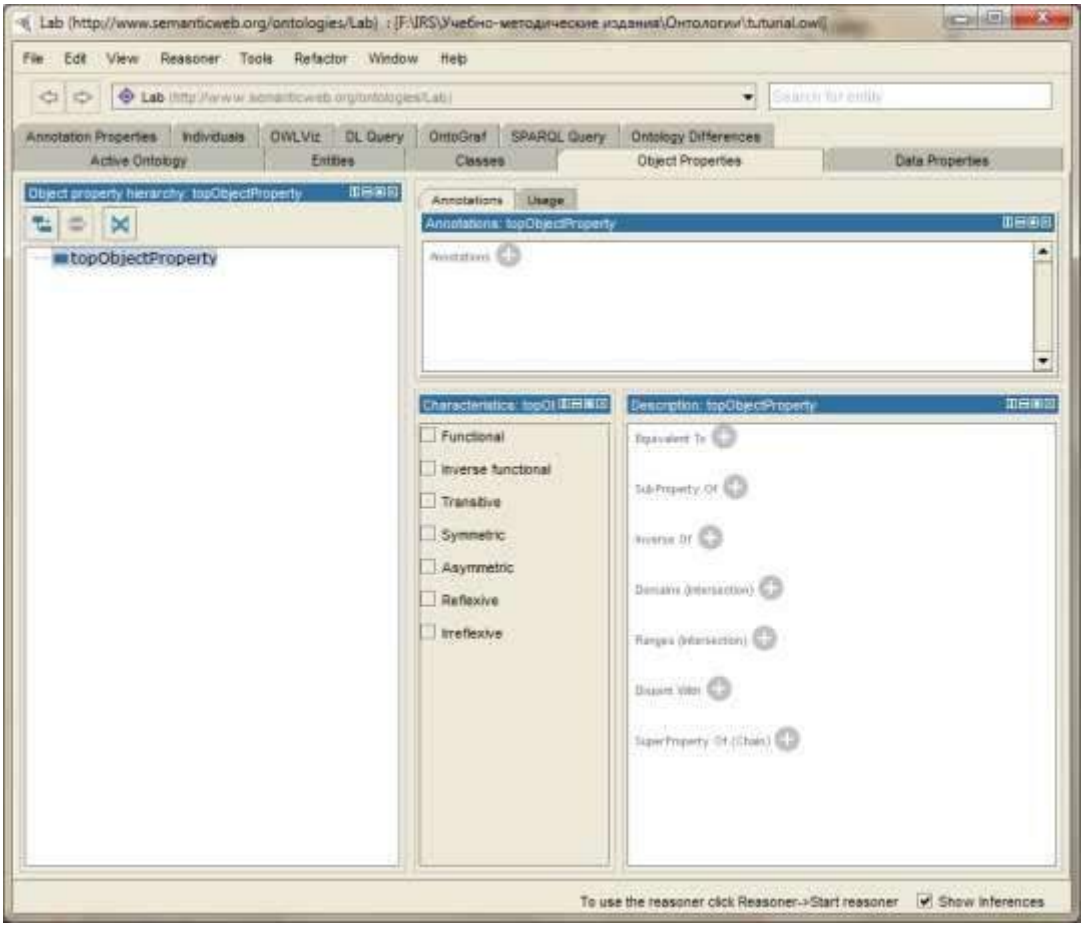


Рис. 5.22. Вкладка «Свойства объектов» («Object Properties»)

Выделите универсальное свойство **topObjectProperty** и нажмите кнопку  – **«**Add sub property**»** («Создать подсвойство») (рис. 5.23)

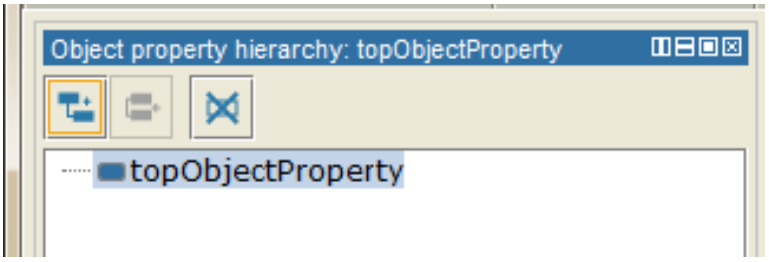


Рис. 5.23 Кнопки для добавления и удаления свойств объектов

Введите имя свойства в диалоговом окне, как показано на рис. 5.24, нажмите кнопку «OK».

Рис. 5.24. Задание имени свойства

Укажите домен и диапазон отношения (рис. 5.25).

- домен: Прием
- диапазон: Врач

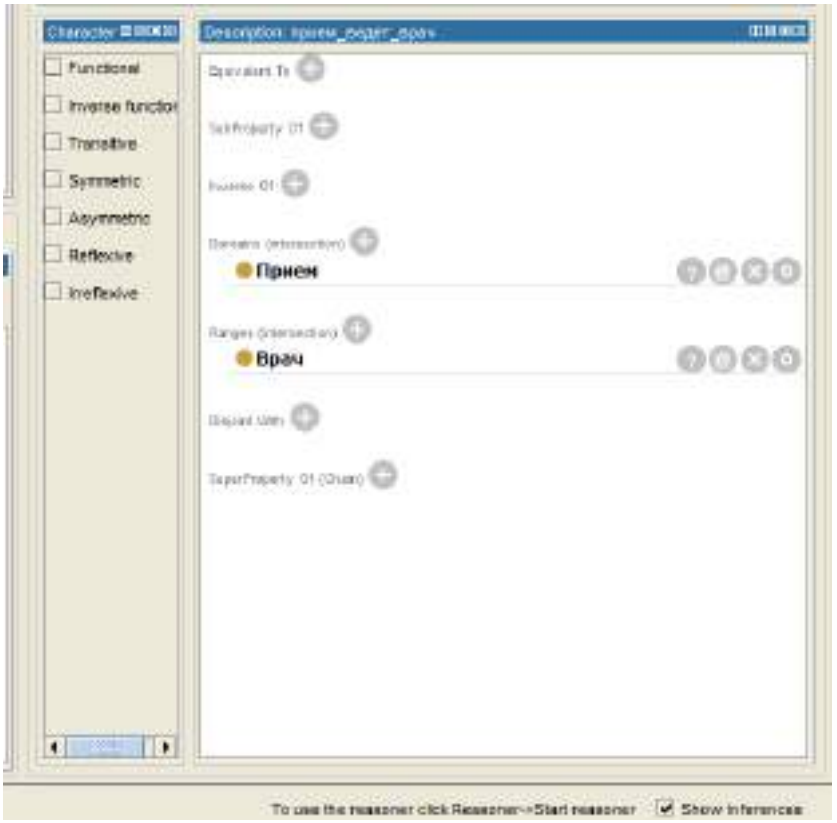


Рис. 5.25. Указание домена и диапазона свойства

Создайте свойство, обратное данному. Для этого на вкладке «Свойства объектов» («Object Properties») создайте новое свойство с именем врач_ведет_прием**.** Затем нажмите значок **(+)** рядом с заголовком «Inverse Of» (рис. 5.26)

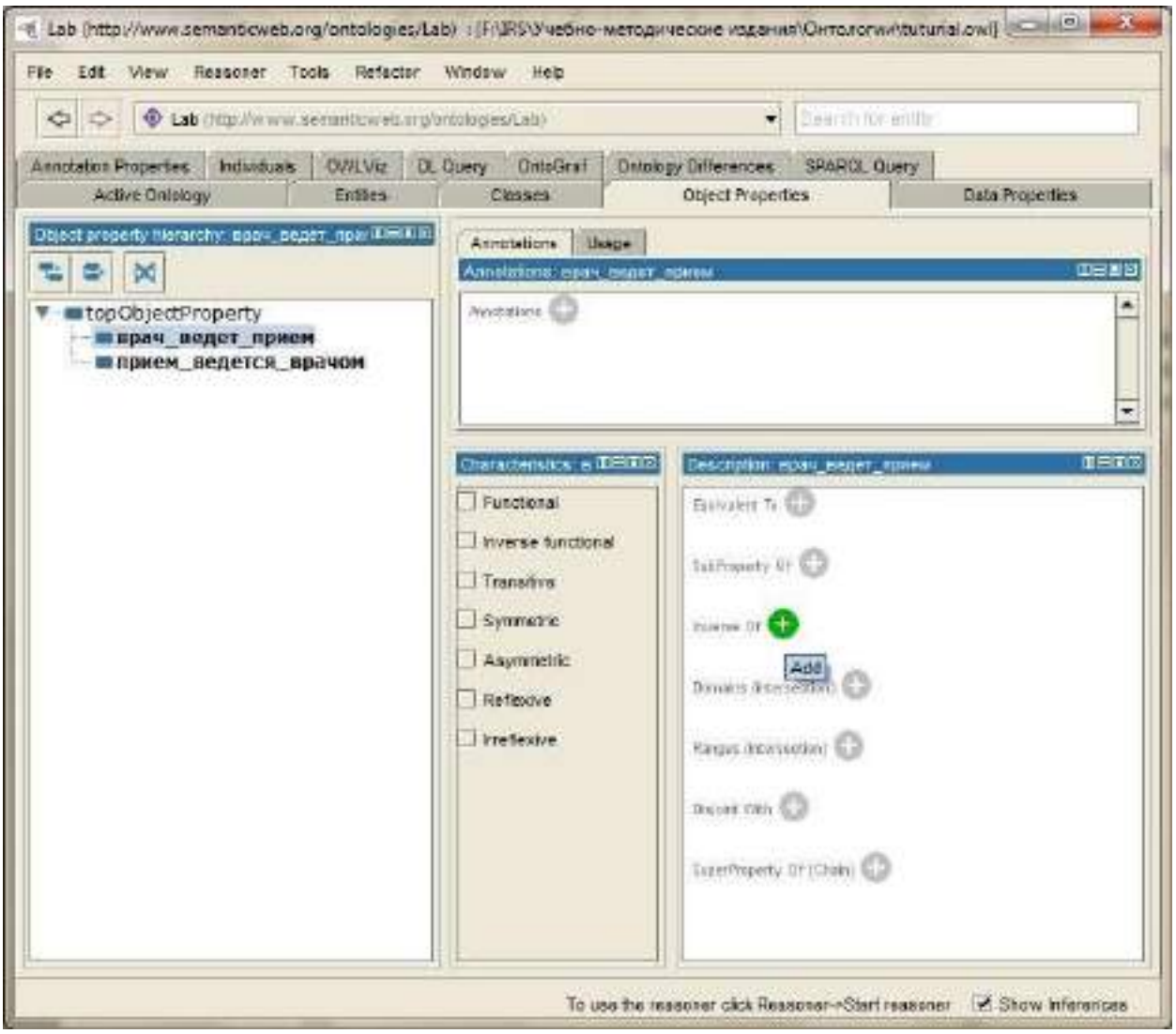


Рис. 5.26. Создание обратного свойства

На экране появится диалоговое окно, в котором в дереве свойств могут быть выбраны нужные свойства. Выберите элемент прием_ведется_врачом и нажмите «ОК». Свойство прием_ведется_врачом теперь должно отображаться как обратное.

В Protégé можно создавать композицию свойств и благодаря этому получать новые отношения. Рассмотрим на примере.

Создайте свойства пациент_пришел_на_прием и пациент_принимается_врачом. Выберите свойство пациент_принимается_врачом в иерархии свойств. Далее на панели «Description» в секции «Super Property Of (Chain)» нажмите кнопку (+). Откроется диалоговое окно текстового редактора, в котором наберите

цепочку «пациент_принимается_врачом **o** прием_ведется_врачом». В этом выражении знак «**o**» (английская маленькая буква «о») обозначает операцию композиции двух отношений. В этом редакторе также можно использовать технику автозаполнения.

Рис. 5.27. Создание композиции свойств

Композиция отношений позволяет в более естественной и краткой форме описывать свойства объектов с помощью логических формул.

5.9. Виды ограничений

В Protégé можно формулировать утверждения о классах и их экземплярах. Семантика онтологии задается путем интерпретации ограничений – высказываний о ее классах и свойствах. Ограничение может быть задано в форме необходимых условий (в секции Sub Class Of) и в форме необходимых и достаточных условий (в секции Equivalent To). В случае необходимых условий описание говорит, что если экземпляр принадлежит классу, то он обладает указанными свойствами. Класс, который удовлетворяет только необходимым ограничениям, известен как *примитивный класс*. В случае необходимых и достаточных условий описание, кроме вышеуказанного смысла, несет еще информацию, что если экземпляр обладает заданными свойствами, то он принадлежит описываемому классу.

Ограничения разделяют на три основные категории:

- кванторные ограничения
- ограничения мощности
- ограничения на свойства данных

Квантор – это общее название для логических операций, ограничивающих область истинности какого-либо. В Protege используются квантор всеобщности ∀ и квантор существования ∃.

Ограничение, полученное навешиванием квантора всеобщности, называют *универсальным ограничением*. В Protégé для его обозначения используют ключевое слово only. Ограничение, полученное навешиванием квантора существования, называют *экзистенциальным ограничением*. В Protégé для его обозначения используют ключевое слово some.

Ограничения класса можно просматривать и редактировать, используя панель «Описание класса» («Description») вкладки «Классы».

5.10. Создание кванторных ограничений

Экзистенциальные ограничения являются самым распространенным типом ограничений в OWL. Создадим ограничение, которое определяет подкласс врачей, проводящих операции. Для этого создайте класс Операция и свойство врач_проводит_операцию.

Выберите Врач из иерархии классов на вкладке «Классы». Нажмите «Добавить» – значок **(+)** рядом с заголовком «Sub Class Off» в панели «Описание класса». Это позволит открыть диалоговое окно редактора ограничений, в котором можно создать ограничение с помощью имеющихся классов и свойств. В открывшемся окне перейдите на вкладку «Object restriction creator». Для создания ограничения нужно сделать следующее (рис. 5.28):

- Выбрать указанное свойство из списка свойств,
- Ввести тип ограничения (some) для экзистенциального ограничения;
- Указать требуемый класс как аргумент ограничения.

Рис. 5.28. Окно создания ограничений

Результат представлен на рис. 5.29.

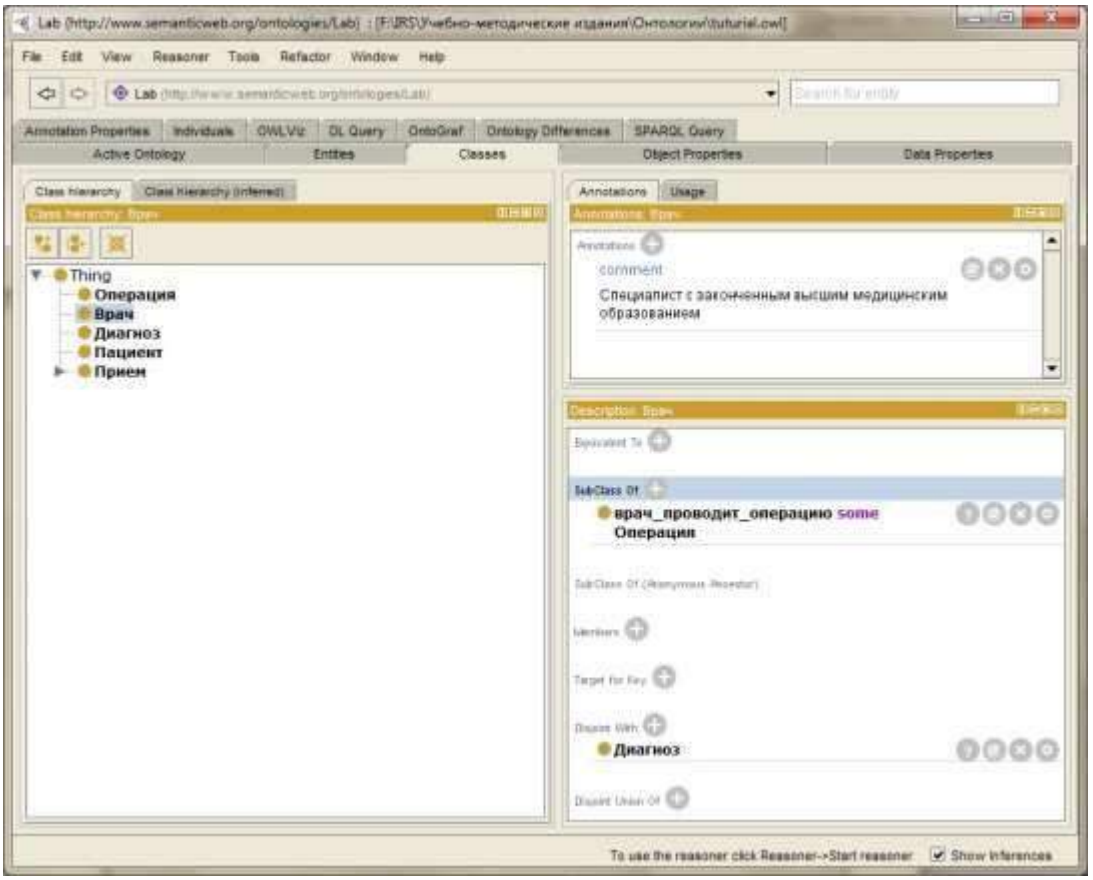


Рис. 5.29. Создание экзистенциального ограничения

Задать ограничение можно также на вкладке «Class expression editor» (рис. 5.30).

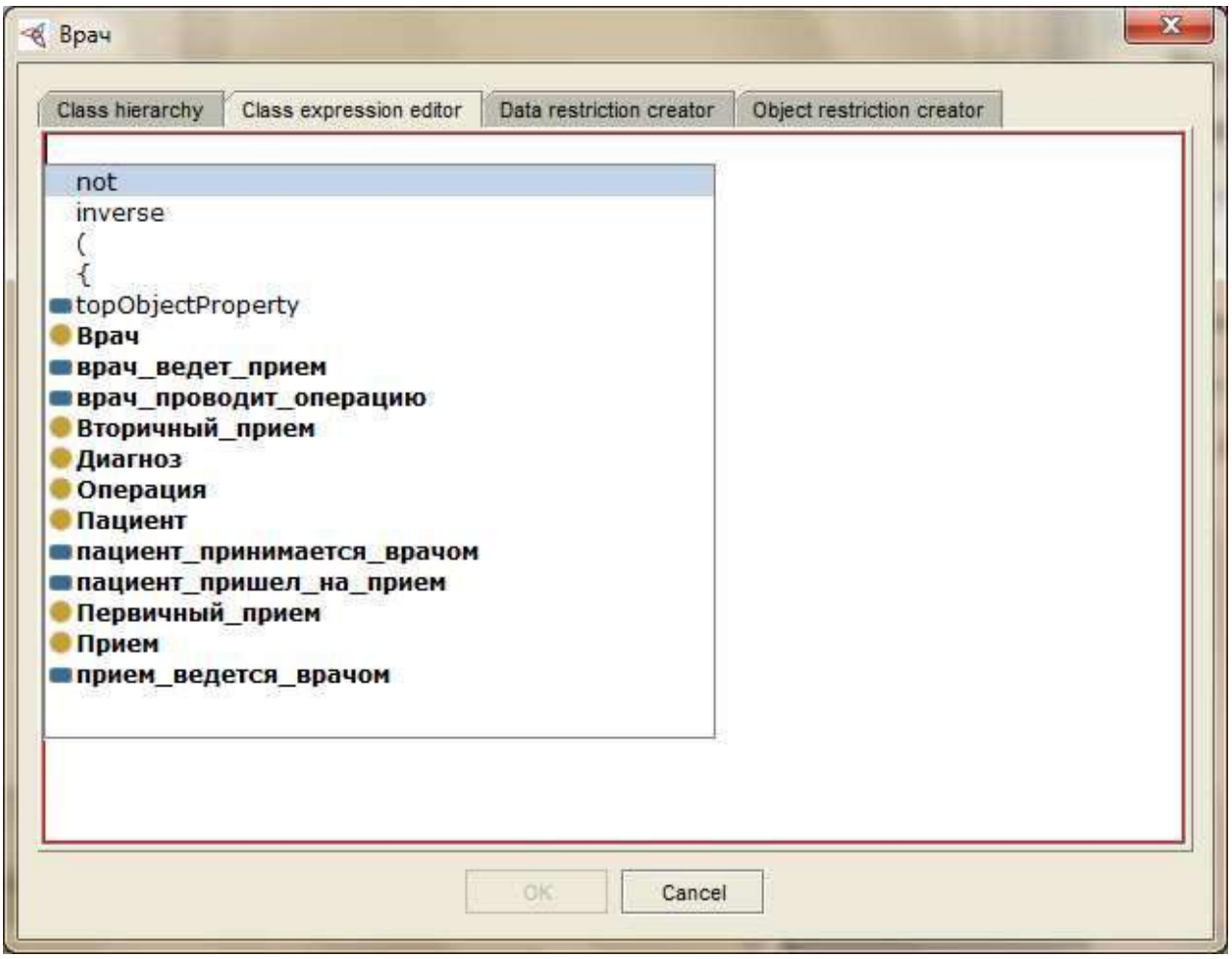


Рис. 5.30. Создание ограничения с помощью построителя выражений

Текстовое окно позволяет создать ограничение с помощью имен классов, свойств и экземпляров. При нажатии на сочетание клавиш «Ctrl-Пробел» активируется построитель выражений. Он предоставляет возможность «автозаполнения» имен классов, свойств и экземпляров

Интерпретация описанного выше ограничения, помещенного в область Sub Class Of, порождает множество, принадлежность которому описывается *необходимым* условием: «Если экземпляр является членом этого класса, то она соответствует указанным ограничениям». При этом мы не можем сказать, что «если сущность удовлетворяет этим условиям, то она должна быть членом этого класса».

Класс, который удовлетворят только необходимым условиям, известен как *примитивный* класс. Класс, который удовлетворяет необходимым и достаточным условиям, называется *определяемым*. Чтобы

построить определяемый класс, следует ограничение поместить в область Equivalent To.

Рассмотрим класс Врач. Что быть допущенным к профессиональной деятельности врача, медицинскому работнику необходимо и достаточно иметь врачебную специальность. Создадим данное ограничение.

- Создайте класс Врачебная_специальность.
- Создайте свойство врач_имеет_специальность
- Нажмите «Добавить» – значок (+) рядом с заголовком «Equivalent To» и задайте соответствующее экзистенциональное ограничение (рис. 5.31)

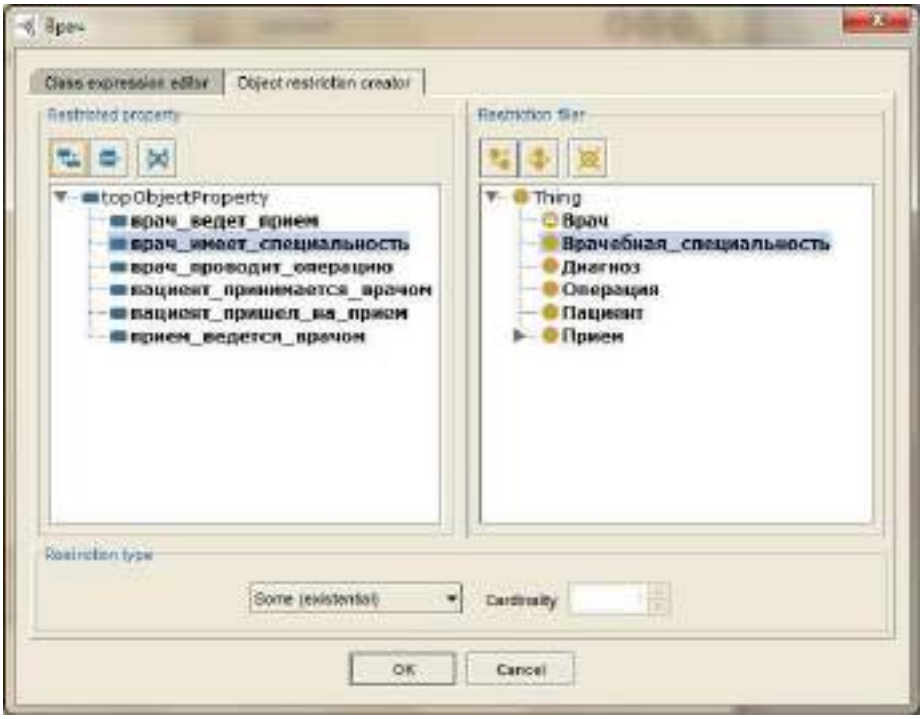


Рис. 5.31. Создание ограничения

Результат представлен на рис. 5.32:

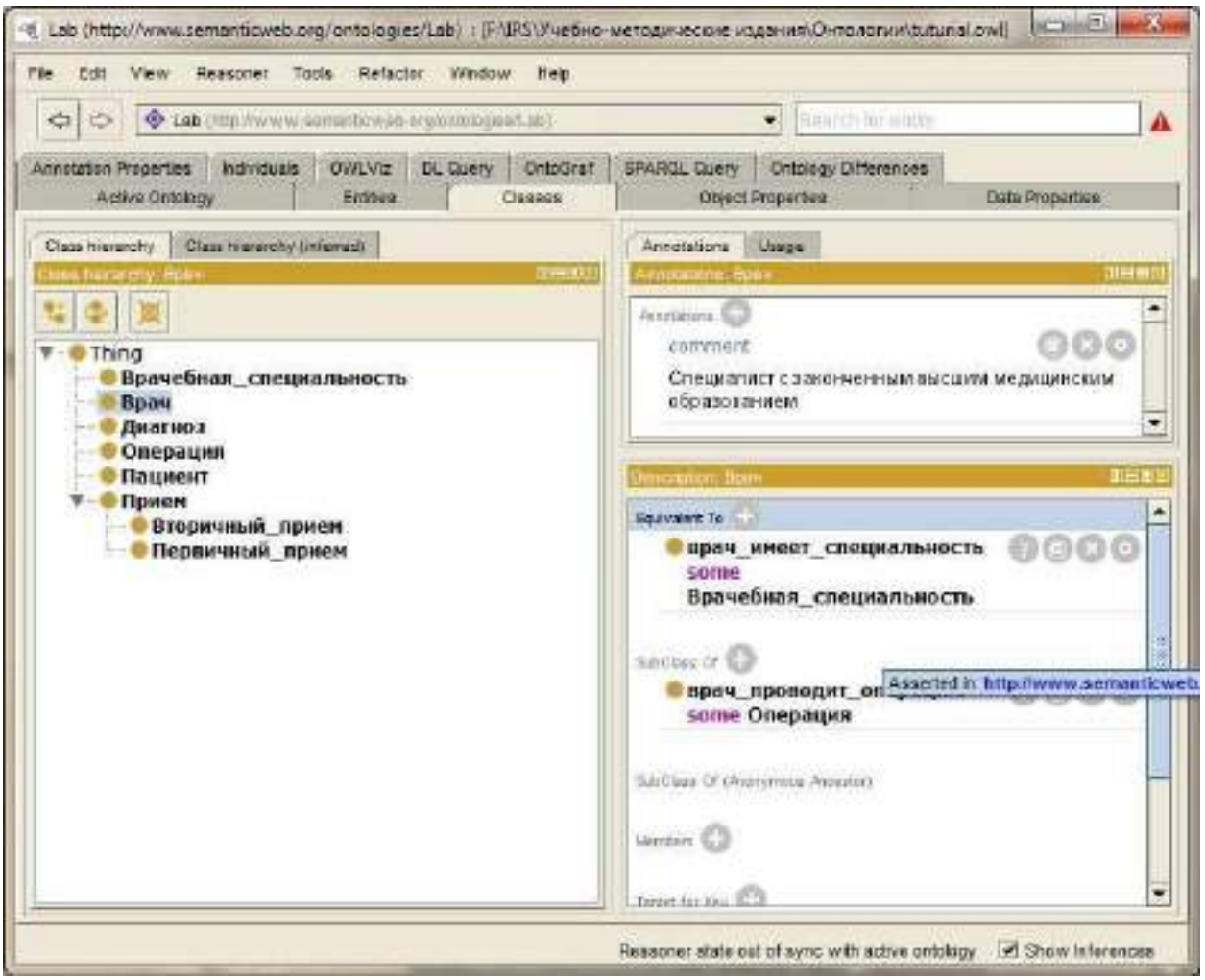


Рис. 5.31. Задание определяемого класса

Универсальные ограничения создаются аналогично. Им соответствует ключевое слово only.

5.11. Использование машины вывода для проверки непротиворечивости иерархии классов

Одной из ключевых особенностей онтологий, описываемых с помощью OWL, является возможность обработки машиной вывода (резонером). Одно из предназначений машины вывода в том, чтобы проверить, действительно ли один класс является подклассом другого класса. Еще одна полезная функции резонера – проверка согласованности ограничений с помощью автоматических рассуждений. При этом выполняется вывод иерархии классов онтологии. класс считается противоречивым, если его ограничениям не удовлетворяет ни

одна сущность. Таким образом, машина вывода классифицирует объекты предметной области и проверяет согласованность онтологии.

Protégé позволяет подключать различные резонеры к редактору. Иерархия классов, которая построена вручную, называется *присоединенной иерархией*. Иерархия классов, которая автоматически вычисляется в процессе рассуждений, называется *выводимой иерархией*.

Чтобы автоматически классифицировать онтологию и проверить ее на непротиворечивость, следует выбрать в главном меню Reasoner⇒ Start reasoner. Если класс будет признан несовместимым, то он будет выделен ярким цветом.

Для того, чтобы продемонстрировать работу машины вывода в обнаружении несоответствий в онтологии, создадим тестовый класс, который определим как подкласс двух непересекающихся классов.

- Выберите класс Врач.
- Создайте подкласс класса Врач с именем Тест.
- Выберите в иерархии классов класс Тест. В секции Sub Class Of нажмите на кнопку «Добавить» (+). В появившемся диалоговом окне выберите класс «Диагноз» и нажмите «OK» (рис. 5.32)

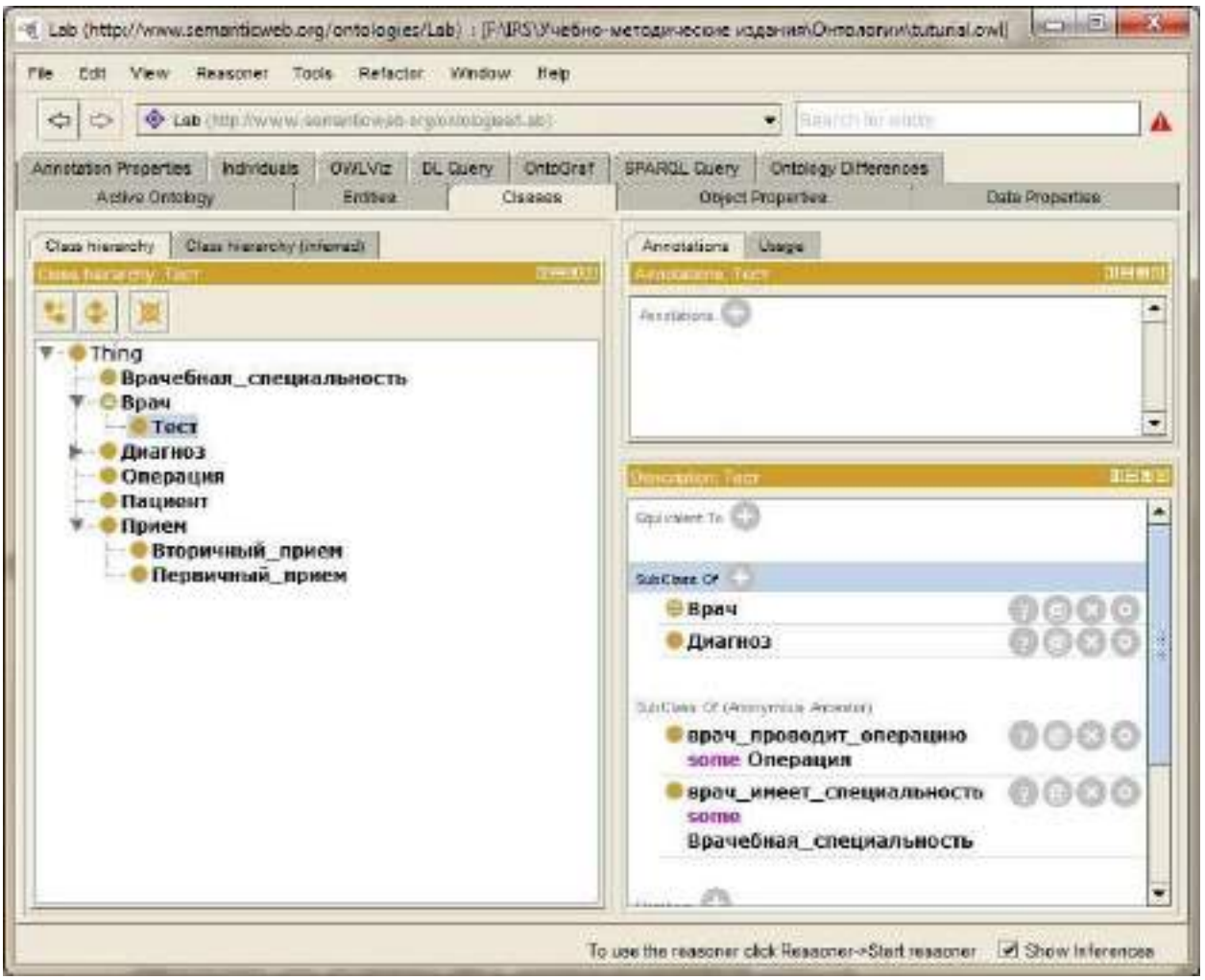


Рис. 5.32. Автоматическая классификация

Таким образом, класс Тест является подклассом двух непересекающихся классом. Это возможно лишь в том случае, если Тест – пустой класс.

Выберите Reasoner⇒Start reasoned. Выводимую иерархию можно увидеть на вкладке «Class hierarchy» (рис. 5.33).

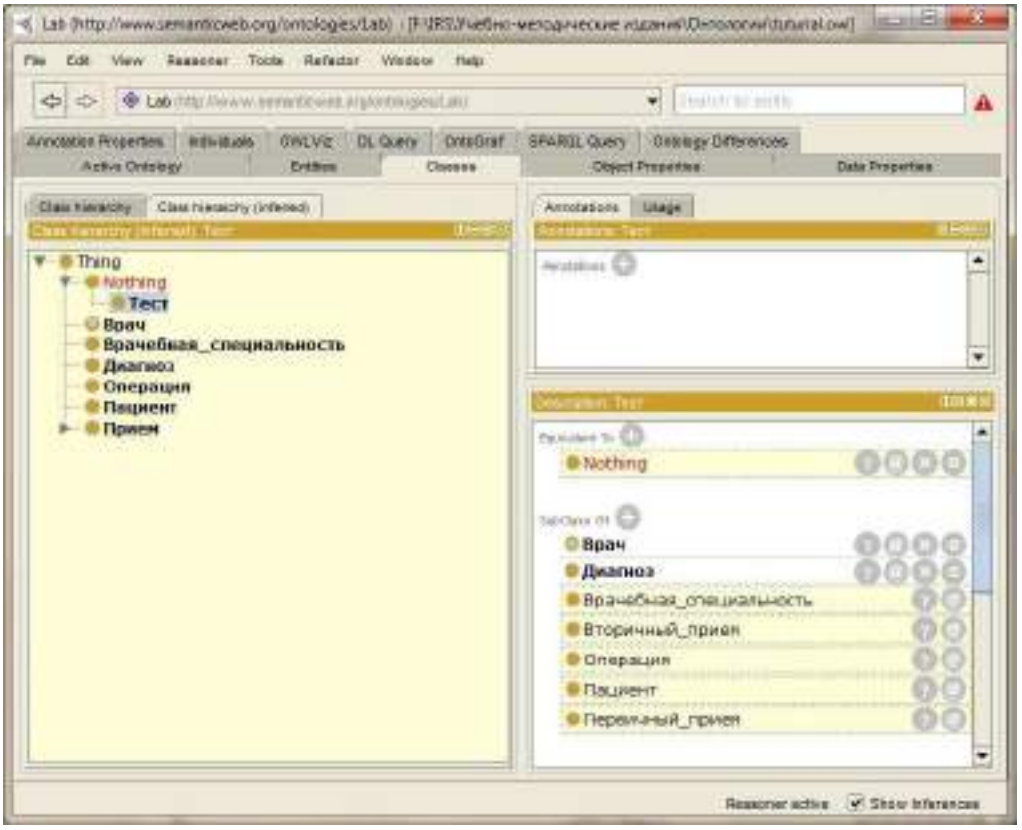


Рис. 5.33. Выводимая иерархия

Как видно, класс Тест признан резонером несоместимым с онтологией (т.е. он не может иметь экземпляров). Вернитесь к первоначальному варианту онтологии. выполните классификацию резонером и убедитесь в непротиворечивости онтологии.

5.12. Описание ограничений мощности

Мощность ограничения используется для описания количества экземпляров заданного отношения, в которых объект может присутствовать. Существуют три варианта мощности ограничений: минимальная мощность ограничения (min), максимальная мощность ограничения (max), точное значение мощности ограничения (exactly).

Для каждого пациента заводится медицинская карта, причем в единственном экземпляре. Опишем данное ограничение в Protégé.

- Создайте класс Медицинская_ карта.
- Создайте свойство пациент_имеет_медкарту.

Выделите класс Пациент и нажмите значок (+) в разделе «Equivalent to». В диалоговом окне создайте ограничение пациент_имеет_медкарту exactly 1 Медицинская_карта (рис. 5.34).

Рис. 5.34. Создание ограничения мощности

Описание класса Пациент представлено на рис. 5.35.

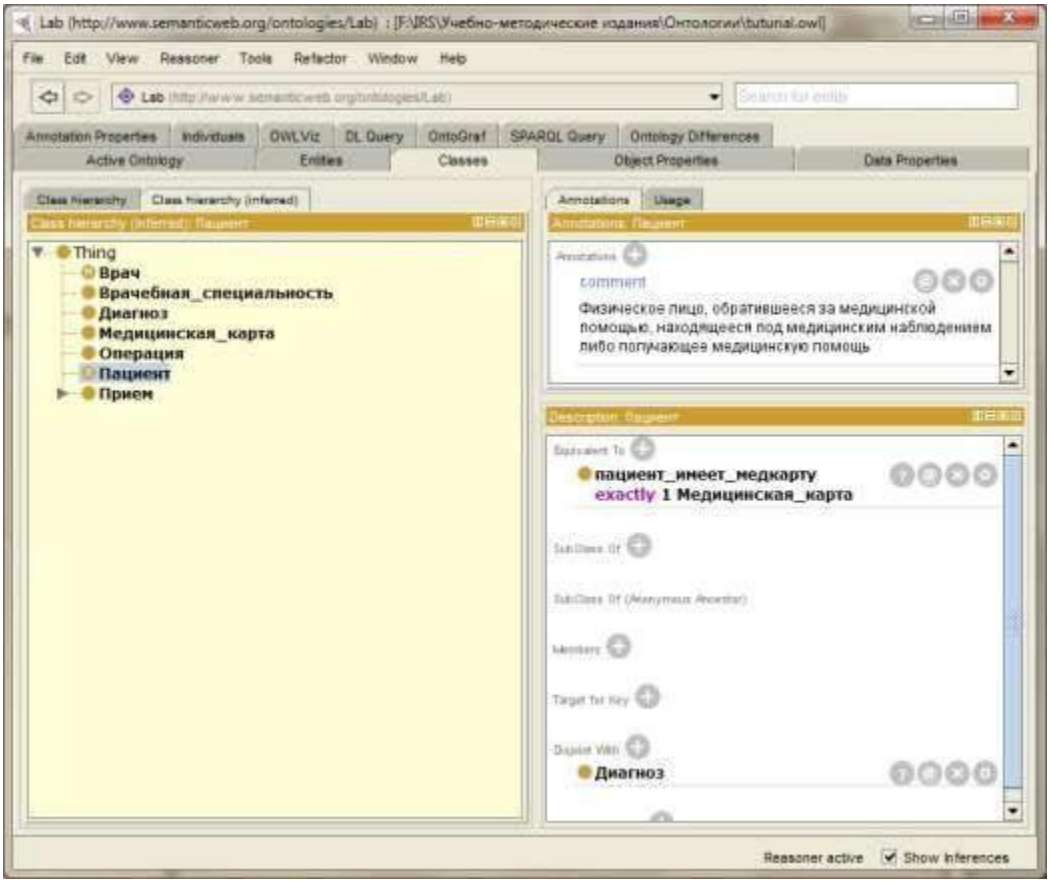


Рис. 5.35. Задание класса Пациент ограничением мощности

5.13. Свойства типа данных

Иерархия классов не позволяет составить полное представление о предметной области. Поэтому после определения классов необходимо описать отличительные свойства, присущие их экземплярам – свойства типа данных, которые описывают связи между характеристиками экземпляров классов и значениями типов данных.

Свойства типа данных могут быть созданы с помощью вкладки **Data Properties**, показанной ниже (рис. 5.36).

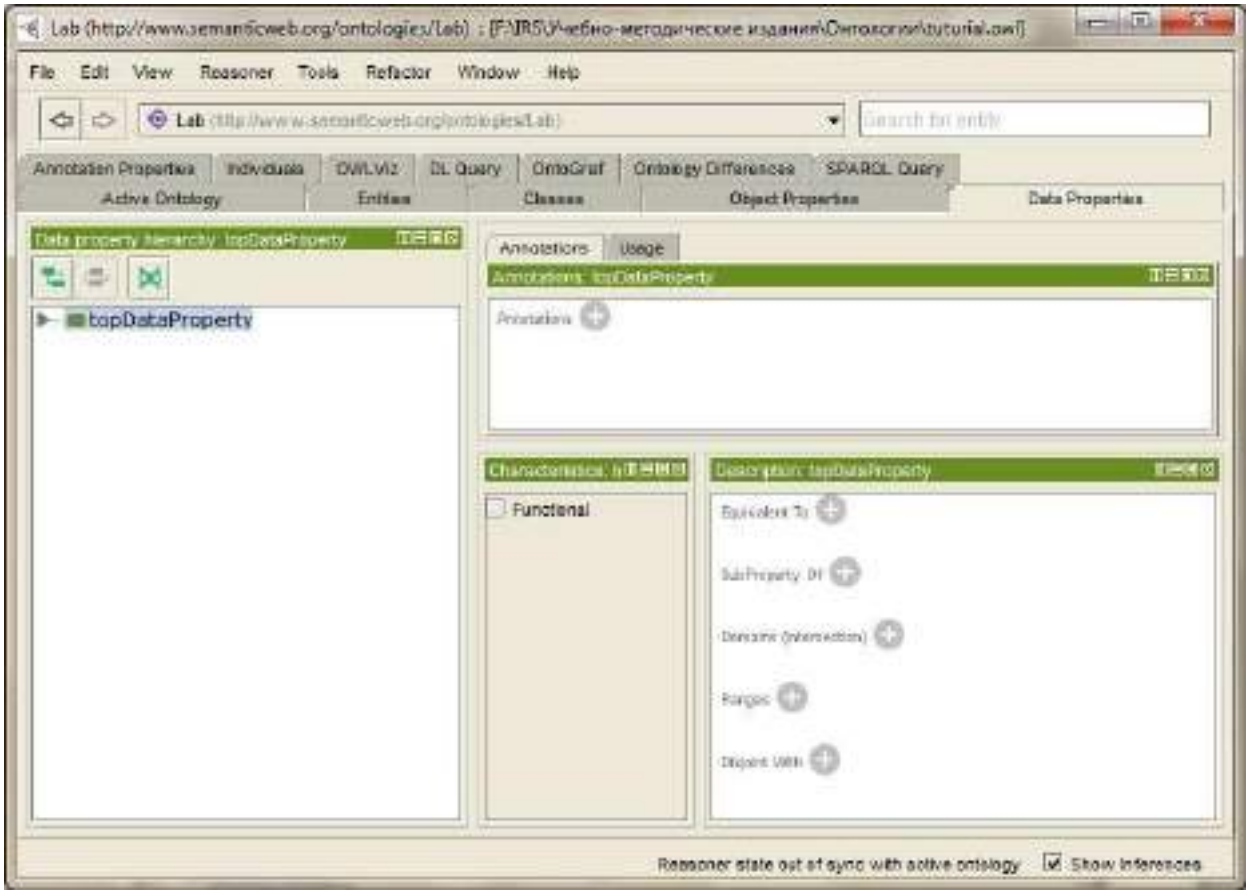


Рис. 5.35. Вкладка «Свойства типа данных» («Data Properties»)

Будем использовать свойства данных для описания возраста пациентов:

- Переключитесь на вкладку «Data Properties».
- Создайте новое свойство данных имеет_возраст (рис. 5.36)

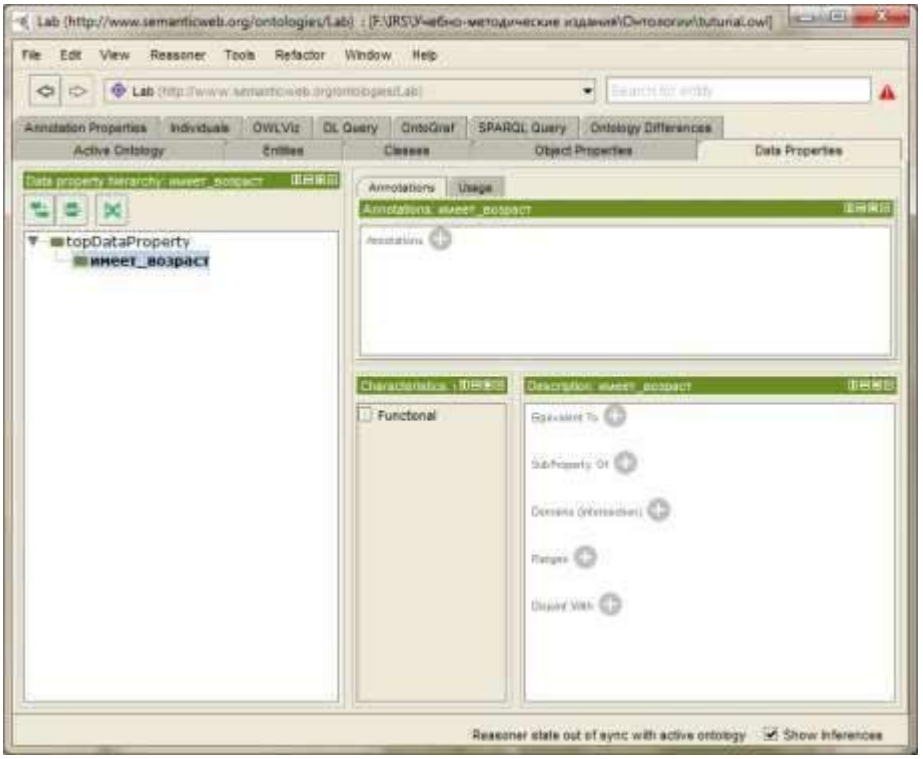


Рис. 5.35. Создание свойства типа данных имеет_возраст

Создадим экзистенциальное ограничение, устанавливающее, что все экземпляры класса Пациент обладают свойством имеет_возраст (рис. 5.36):

- На вкладке «Классы» выберите класс Пациент.
- Нажмите кнопку «Добавить» (+) в секции «Sub Class Of».
- В открывшемся диалоговом окне выберите вкладку «Data restriction creator» («Создать ограничение на свойство данных»).
- В правой части выберите свойство имеет_возраст.
- Установите тип ограничения some.
- В правой части укажите тип данных integer.

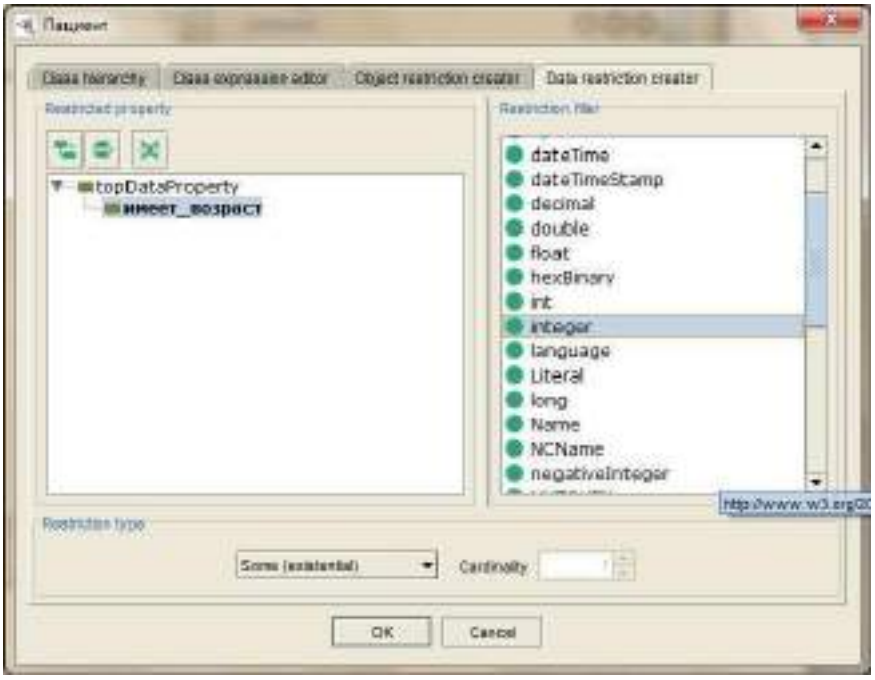


Рис. 5.35. Создание ограничения

Нажмите «ОК». Созданное ограничение появилось в секции «Sub Class Of» (рис. 5.36).

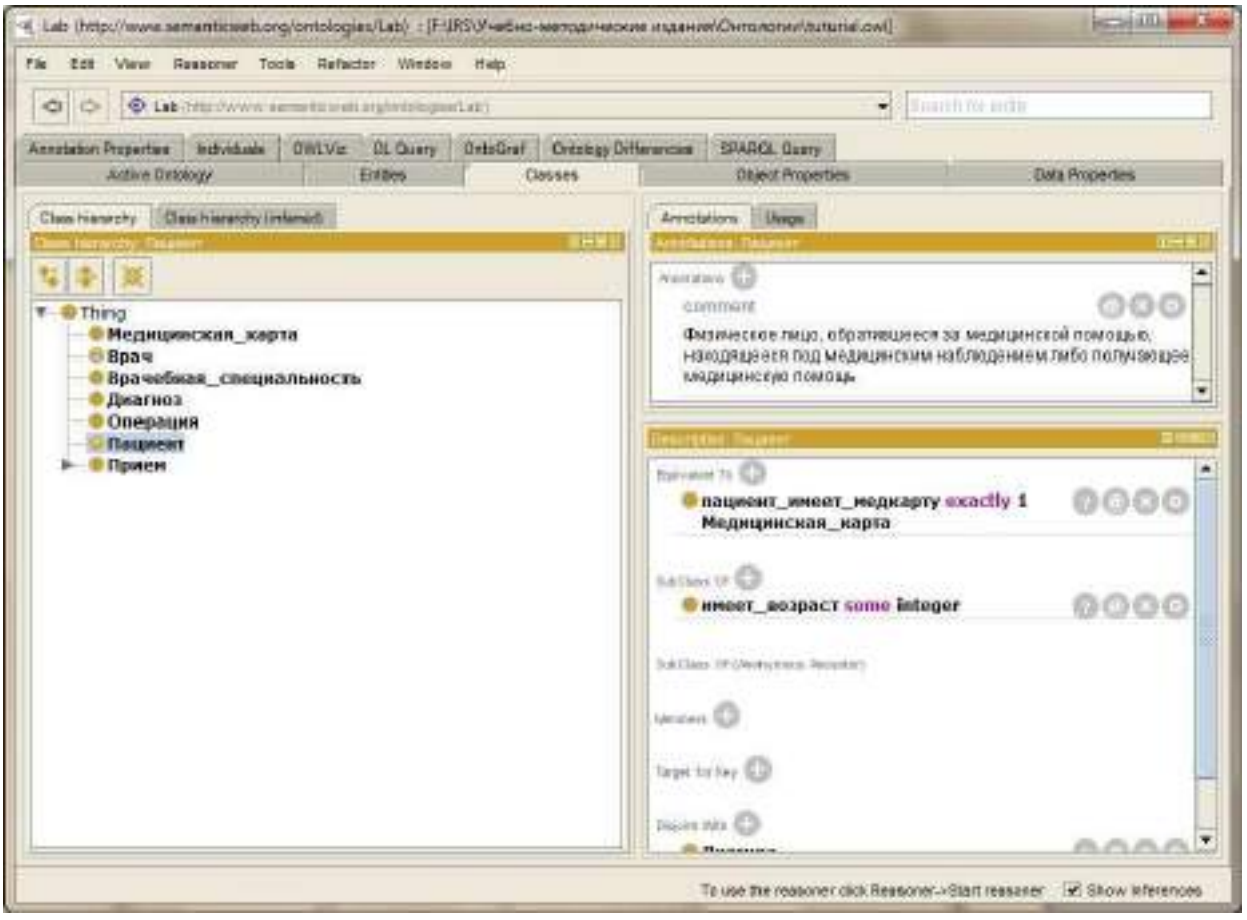


Рис. 5.35. Описание класса Пациент **экзистенциальным ограничением**

После создания свойства типа данных необходимо описать экземпляров класса, обладающих этим войством. Экземпляры класса имеют изначально при создании только имя, а потом при описании у них появляется набор свойств, который может быть абсолютно индивидуальным. Задание экземпляров классов и приписывание им набора свойств происходит на вкладке «Individuals»:

- На вкладке «Individuals» нажмите кнопку  – «Добавить экземпляр» и введите его имя – Петров_В.Д.
- В панели «Description» («Описание») укажите тип – Пациент.
- В панели «Property assertions» («Определение связей») выберите секцию «Data property assertions» и нажмите кнопку «Добавить». В появившемся диалоговом окне укажите в качестве свойства данных имеет_возраст, в качестве типа данных – integer и в текстовом поле введите значение возраста пациента 42 (рис. 5.36).

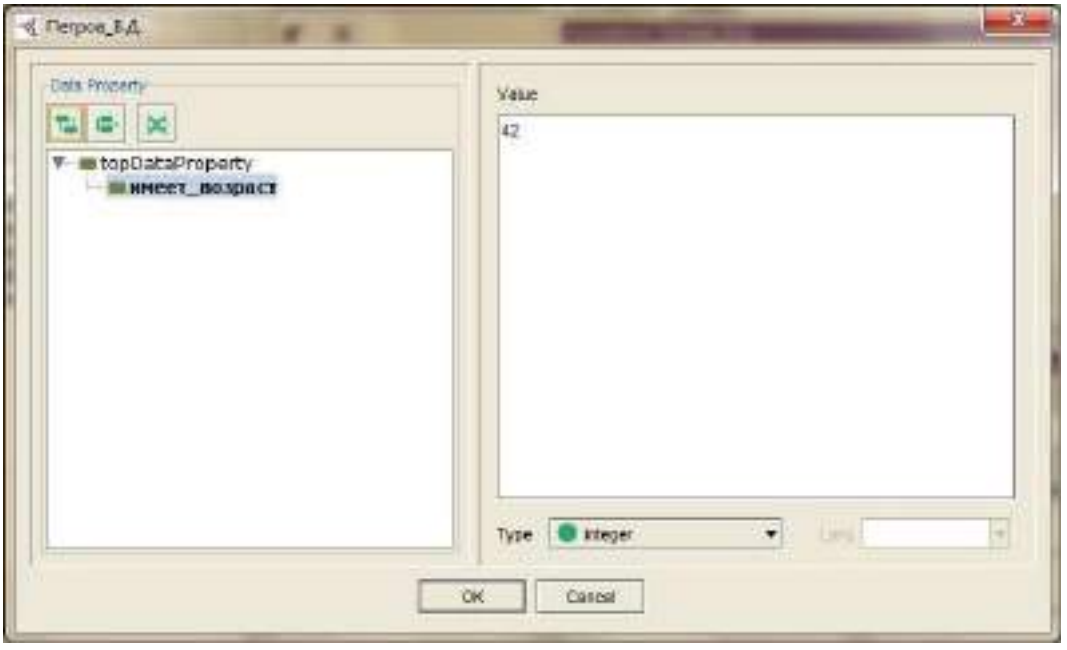


Рис. 5.36. Задание экземпляра класса

Создайте еще несколько экземпляров класса Пациент: Иванов М.А. – 20 лет, Сидоров К.Л. – 65 лет.

5.14. Описание ограничений на свойства типа данных

Введем несколько возрастных интервалов для классификации пациентов по возрастным группам. Используя типизированные данные, можно задавать ограничения вида <=, >=, <, >, = на их значения. Например:

Молодой_пациент имеет_возраст some integer [< 25] Пожилой_пациент имеет_возраст some integer [>=60] Пациент_средних_лет имеет_возраст some integer [>=25, <60]. Создадим первое ограничение. Для этого на вкладке «Classes» создайте подкласс Молодой_пациент класса Пациент. На панели «Description» в секции «Sub Class Of» нажмите кнопку «Добавить» (+). В редакторе выражений наберите имеет_возраст some integer [< 25] и нажмите «OK». Выделите созданное ограничение и выберите Edit –> Convert to defined class для конвертации примитивного класса в определяемый. Описание класса Молодой_пациент будет иметь вид, представленный на рис. 5.37.

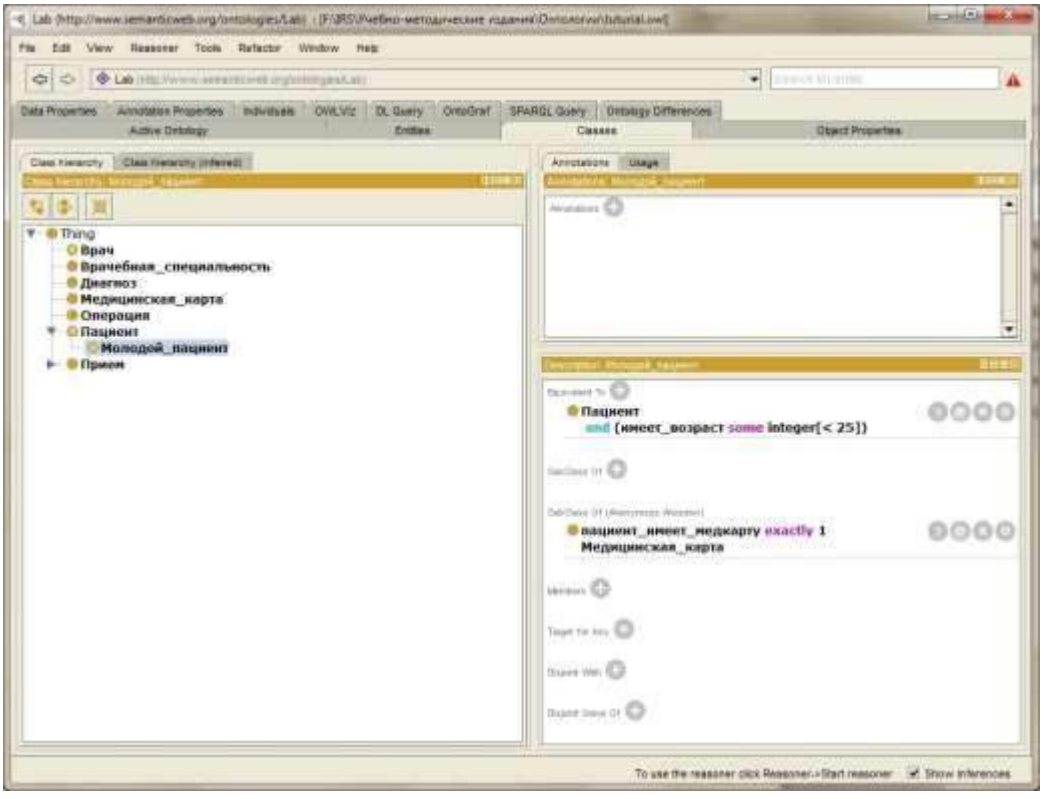


Рис. 5.37. Создание ограничения на свойство типа данных

Аналогично создайте подкласс Пожилой_пациент и Пациент_средних_лет.

Классифицируйте пациентов по возрасту с помощью резонера (рис. 5.38):

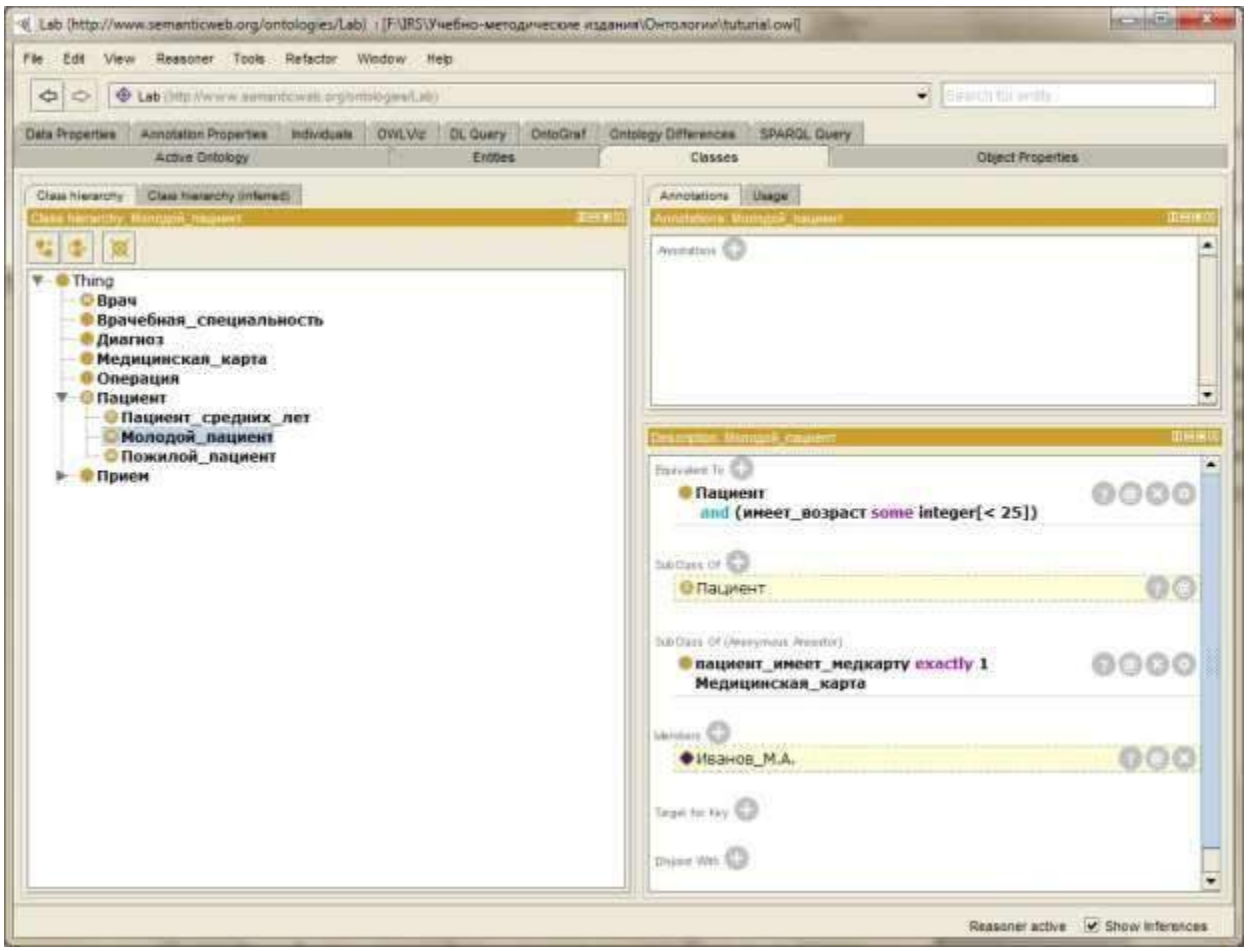


Рис. 5.38. Классификация пациентов

Обратите внимание на секцию «Members». Она должна включать экземпляры, имеющие возраст в границах указанного диапазона

5.15. Создание DL-QUERY запросов

Для создания запросов необходимо перейти в закладку DL-Query, а также на вкладке «Reasoner» выбрать «Start reasoner» (рис. 5.39).

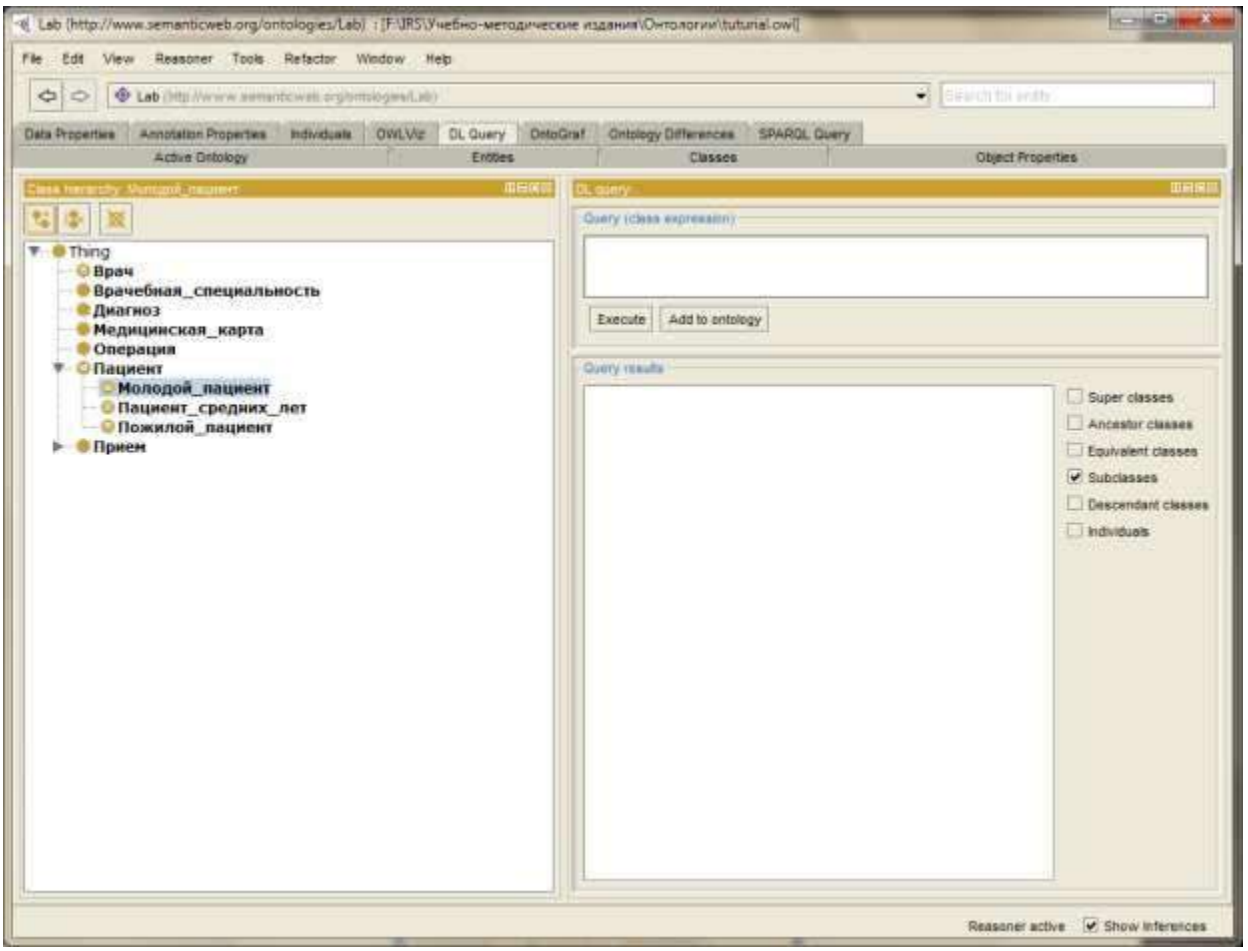


Рис. 5.39. Вкладка создания запросов («DL-Query»)

Базовые запросы позволяют вывести для выбранного класса его суперкласс, предков, эквивалентные классы, подклассы, потомков, экземпляры. Чтобы создать необходимый запрос, нужно выбрать класс, в котором будет производиться поиск, свойство, по которому будет производится поиск, а также указать признак. Под признаком может пониматься как строчка, так и условия is, is not, contains, begins with и так далее. Созданный запрос можно сохранить в онтологии, нажав кнопку «Add to ontology».

К примеру, мы хотим вывести все экземпляры класса «Пациент». Для этого в запросное окне необходимо перетащить название класса и выбрать на панели результатов «Individuals» (рис. 5.40).

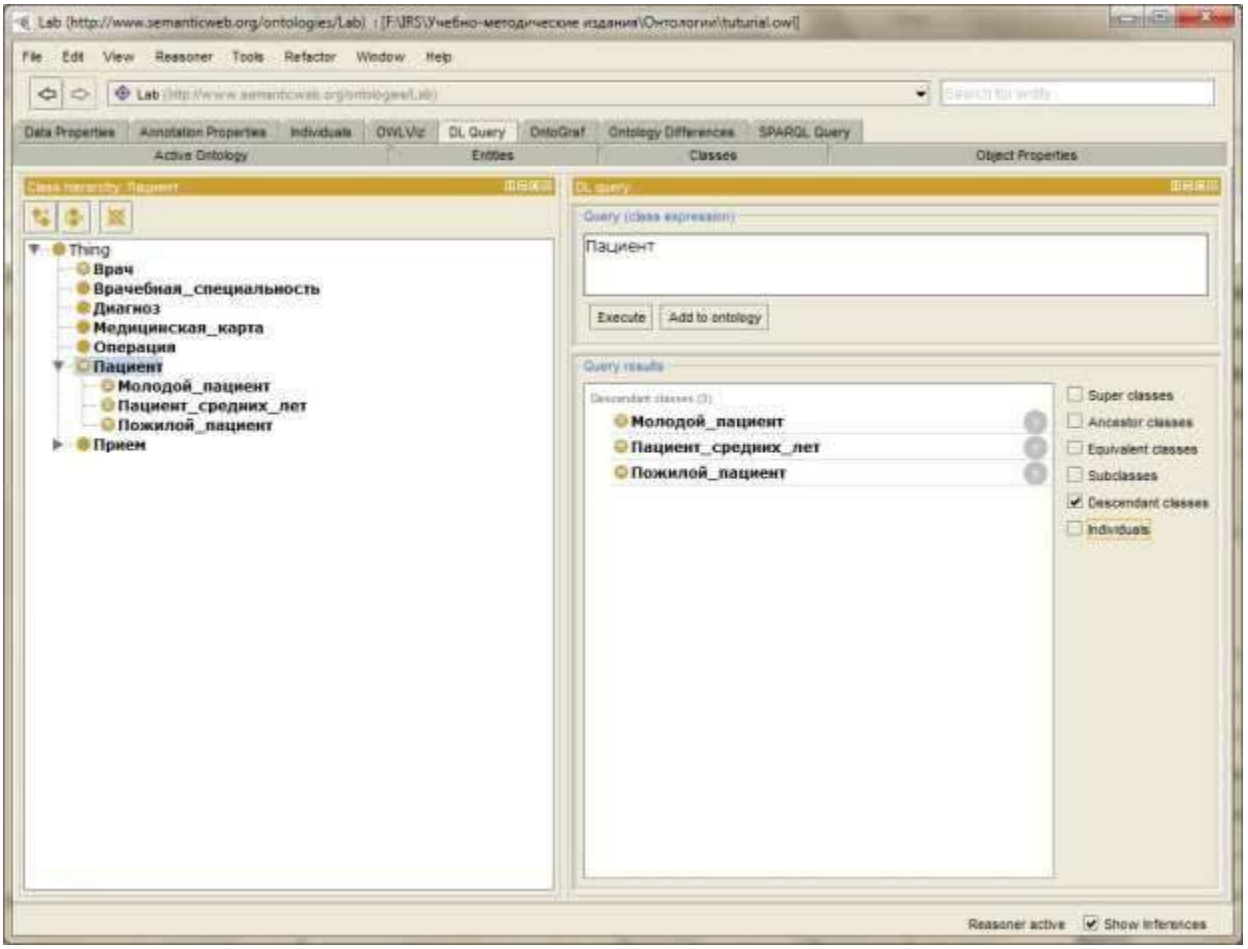


Рис. 5.40. Результат запроса

Аналогично можно создать запросы к онтологии, позволяющие вывести для выбранного класса его суперкласс, предков, эквивалентные классы, подклассы, потомков, экземпляры. Также при построении запросов можно пользоваться построителем выражений.

ЛИТЕРАТУРА

1. Antoniou G., Frank van Harmelen. Web Ontology Language: OWL. (in Handbook on Ontologies). – Springer Verlag, 2004. – P. 67 – 92.
2. Baader F. (editor), et al. The Description Logic Handbook. – Cambridge University Press, 2003. – 574 p.
3. Berners-Lee T., Hendler J., Lassila O. The Semantic Web // Scientific American. – May 2001.
4. Despres C., Chauvel D. The Present and the Promise of Knowledge Man- agement. – Butterworth-Heinemann, 2000. – 352 p.
5. Handschuh S., Staab S. (eds.). Annotation for the Semantic Web. Volume 96 Frontiers in Artificial Intelligence and Applications. – IOS Press, 2003. – 240 p.
6. Maier R. Knowledge management systems: Information and communication technologies for knowledge management. – Berlin Heidelberg: Springer Verlag, 2004. – 635 p.
7. Антониоу Г., Грос П., Хармелен ван Ф., Хоекстра Р. Семантический веб. – М. : ДМК Пресс, 2016. — 240 с.
8. Букович У., Уильямс Р. Управление знаниями: руководство к действию (Wendi R. Bukowitz, Ruth L. Williams The Knowledge Management Field- book). – М.: ИНФРА-М, 2002. – 504 с.
9. Вебер А.В., Данилов А.Д., Шифрин С.И. Knowledge-технологии в кон- салтинге и управлении предприятием. – М.: Наука и техника, 2002. – 176 с.
10. Гаврилова Т.А., Хорошевский В.М. Базы знаний интеллектуальных систем. – Спб: Питер, 2000. – 384 с.
11. Люггер Д.Ф. Искусственный интеллект: стратегии и методы решения сложных проблем. 4-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2003. – 864 с.

12. Маннинг К. Д., Рагхаван П., Шютце Х. Введение в информационный поиск. – М. : Вильямс, 2011. – 528 стр.
13. Осипов Г.С. Методы искусственного интеллекта. – М.: ФИЗМАТЛИТ, 2011. – 296 с.
14. Палагин А.В., Крывый С.Л., Петренко Н.Г. Онтологические методы и средства обработки предметных знаний: монография /. – Луганск: изд-во ВНУ им. В. Даля, 2012. – 324 с.
15. Тузовский А.Ф., Чириков С.В., Ямпольский В.З. Системы управления знаниями (методы и технологии) / Под общ. ред. В.З. Ямпольского. – Томск: Изд-во НТЛ, 2005. – 260 с.
16. Цуканова Н.И. Онтологическая модель представления и организации знаний. Учебное пособие для вузов. – М.: Горячая линия–Телеком, 2016. – 276 с.

Оглавление

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ
ОСНОВЫ ГЛУБОКОГО ОБУЧЕНИЯ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

Одно правильно реализованное задание - 1 балл.

Лабораторная работа 1

Алгоритм обучения нейрона по правилу Хебба (файл lab1_perceptron.py строки 50-56) реализован с фиксированным количеством итераций.

Принимать решение об остановке алгоритма обучения можно опираясь на теоремы о “сходимости” и “зацикливании” перцептрона (лекция 1 слайд 17).

ЗАДАНИЕ: реализовать правила остановки алгоритма обучения перцептрона при его зацикливании или сходимости.

Лабораторная работа 2

Алгоритм обучения нейронной сети (файл lab2_mlp_batchgradient.py строки 82-85) реализован так, что функционал ошибки сети вычисляется по всей обучающей выборке (аналогично классическому градиентному спуску) .

ЗАДАНИЕ 1: реализовать алгоритм стохастического обучения (аналогично стохастическому градиентному спуску).

ЗАДАНИЕ 2: реализовать нейронную сеть, осуществляющую классификацию цветков Ириса на 3 класса по всем 4 имеющимся признакам.

Лабораторная работа 3

Класс многослойного перцептрона class MLPptorch (описан в файле neural.py) имеет только один скрытый слой.

ЗАДАНИЕ 1: создать класс, имеющий количество скрытых слоев >2.

ЗАДАНИЕ 2: в созданном классе изменить функцию активации слоев на ReLU и оценить изменение скорости обучения по сравнению с функцией активации Sigmoid.

Лабораторная работа 4

ЗАДАНИЕ 1: создать сверточную нейронную сеть, решающую задачу классификации животных на классы СОБАКА, КОШКА или ДИКОЕ по их изображению. Набор данных для решения задачи содержится в архиве https://drive.google.com/file/d/1GwGgM_qqmV2_PbMkMhFYP--5x59S-fyg/view?usp=sharing

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ ОСНОВЫ КОМПЬЮТЕРНОЙ ГРАФИКИ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

Создание изображений средствами графических редакторов и графической библиотеки
Задания и программы лабораторных работ по дисциплине «Основы компьютерной графики» для студентов направления 09.04.01 – Информатика и вычислительная техника

Приведены задания и программы выполнения лабораторных работ с применением популярных средств компьютерной графики: векторного редактора Inkscape, растрового редактора Gimp, графической библиотеки OpenGL. Показаны примеры результатов выполнения работ. Программы выполнения некоторых работ предусматривают дополнительные задания, выполняемые студентами самостоятельно, по всем работам приведены контрольные вопросы.

Составители: **********

СОДЕРЖАНИЕ

Лабораторная работа 1
Создание изображения в векторном редакторе

1.1. Цель работы и лабораторное задание

Целью работы является практическое освоение тексто-графических возможностей редактора Inkscape, их использование при проектировании графического логотипа, Создание тексто-графического объекта со встроенным логотипом (визитной карточки, буклета, рекламы).

Необходимо, используя изученные инструменты, создать собственный логотип, отражающий направленность занятий и интересов исполнителя или предметную область выбранного исполнителем документа (по согласованию с преподавателем). Изображение может быть выполнено в палитре серого или быть цветным. Далее логотип используется при создании тексто-графического документа. В нем используются текстовые фрагменты, различные заливки и фильтры (по выбору исполнителя). Возможно добавление в документ еще одного готового изображения, кроме логотипа.

1.2. Программа работы

Создать эскиз логотипа, обсудить с преподавателем. Используя изученные инструменты рисования редактора Inkscape, нарисовать спроектированное изображение. Готовый логотип предъявить преподавателю для контроля и запомнить на диске в формате **svg**.

Следующий этап работы: с помощью панели «Файл», вкладки «Создать» выбрать формат тексто-графического документа. В случае визитной карточки выбирается размер 85-54мм или 90-50мм, при необходимости размеры могут быть изменены. При конструировании документа на рабочий стол помещается один документ, доступный для редактирования. В каталоге шрифтов выбрать наиболее подходящий для оформления шрифт, с помощью инструмента «Вставка текста» создать нужные записи и расположить ее на заготовке документа. Вставить логотип и дополнительное готовое изображение.

Изображение документа сохранить в формате **svg** и представить преподавателю на экране для проверки. Под рисунком необходимо написать номер группы и фамилию автора. Примеры внешнего вида визитных карточек с логотипами представлены на рис.1.

Рис.1. Примеры оформления визитных карточек

1.3. Контрольные вопросы и дополнительное задание к лабораторной работе 1

1. Перечислите инструменты, которые использованы при проектировании логотипа.
2. Опишите на примере визитки операции над текстом и объектами.
3. Опишите последовательность действий, необходимых для подготовки вывода на печать страницы текстового документа с иллюстрацией.
4. Расскажите об использовании панели «Заливка и обводка».

После сдачи практической части работы и ответов на вопросы студент получает у преподавателя дополнительное задание. В качестве дополнительного задания выполняется самостоятельная работа по созданию логотипа по предложенной преподавателем тематике. В отчете по лабораторной работе описать выбор инструментов и эффектов, а также последовательность операций, примененных для изготовления документа.

Лабораторная работа 2
Создание изображения в растровом редакторе

2.1. Цель работы и лабораторное задание

Целью работы является изучение инструментов графического редактора GIMP и получение практических навыков создания многослойных изображений средствами редактора. В лабораторное задание входит создание многослойных изображений, которые включают фрагменты, выделенные из других графических объектов. Для выполнения работы необходимо изучить инструменты выделения, приемы работы со слоями многослойного изображения, знать способы копирования.

2.2. Программа работы

После загрузки графического редактора следует изучить набор его инструментов и их настройки. Для освоения работы с инструментами выделения необходимо загрузить изображение, выбранное базовым слоем-фоном, и, поочередно выделяя фрагменты других изображений (изображения выбираются студентом самостоятельно) всеми возможными инструментами используемого редактора, перенести их на базовый слой-фон. Для каждого фрагмента автоматически создается свой слой (нужно уметь менять порядок слоев). Каждому фрагменту нужно присвоить свой номер, который указывает на используемый инструмент. На полученное изображение наложить текстовый слой с номером лабораторной работы и фамилией автора.

Изображение сохранить в формате используемого растрового редактора **XCF** и в одном из стандартных растровых форматов (jpeg, png…), а также представить на экране монитора для проверки преподавателем. После сдачи практической части работы и оформления отчета студенту предстоит ответить на дополнительные вопросы.

Пример многослойного изображения приведен на рис. 2. На нем цифрами помечены использованные инструменты (1 – прямоугольное выделение, 2 – округлое выделение, 3 – лассо и т.д.).

Рис.2. Пример многослойного изображения

2.3. Контрольные вопросы и дополнительное задание к лабораторной работе 2

1. Расскажите о настройках инструментов выделения редактора GIMP.
2. Какие инструменты выделения (перечислить) есть в редакторе GIMP?
3. Как выделение превратить в слой, в новое изображение?
4. Как поменять порядок видимости слоев?

После сдачи практической части работы и ответов на вопросы студент получает у преподавателя дополнительное задание. В качестве дополнительного задания выполняется самостоятельная работа по созданию тематического коллажа. Тема коллажа предлагается студентом и утверждается преподавателем. Пример выполнения приведен на рис.3.

В отчете по работе нужно привести фоновое изображение, выделяемые фрагменты, итоговое многослойное изображение, а также описать цель применения инструментов, их названия и результат их применения. В отчет следует вставить скриншот окна редактора, на котором видны названия созданных слоев.

Рис.3. Пример коллажа

Лабораторная работа 3

Отображение геометрических форм средствами графической библиотеки

3.1 Цель работы и лабораторное задание

Цель работы – изучить на практике отображение геометрических фигур средствами открытой графической библиотеки OpenGL. Лабораторная работа состоит из выполнения двух заданий. По первому заданию студенты формируют фигуры из плоских геометрических примитивов. По второму заданию рисуются поверхности второго порядка (квадрики) и многогранники (полиэдры). Размеры графических объектов задаются в пределах нормализованного объема видимости.

Задание 1

Изучаемые команды

Команды инициализации: auxInitWindowPosition, auxInitWindowSize, glutInitWindowPosition, glutInitWindowSize.

Команды настройки модельно-видовых и проективных матриц: glMatrixMode, glLoadIdentity, glOrtho.

Команды настройки цветов фона и изображения: glClearColor, glClear, glColor.

Команды рисования геометрических примитивов: glVertex, glBegin – glEnd.

Команды задания стиля рисования: glShadeModel, glLineWidth, glPointSize, glEnable(GL_LINE_SMOOTH), glEnable(GL_POINT_SMOOTH).

Варианты заданий приведены в таблице.

N варианта	Фигура	Геометрический примитив
1	Треугольник, вписанный в прямоугольник	Треугольник (GL_TRIANGLES), связанные отрезки (GL_LINE_STRIP)
2	Смежные пятиугольник и треугольник	Треугольник (GL_TRIANGLES), связанные треугольники (GL_TRIANGLE_STRIP)
3	«Крыша дома» – параллелограмм, сопряженный с треугольником	Треугольник (GL_TRIANGLES), четырехугольник (GL_QUADS)
4	«Флажок» – вектор и невыпуклый пятиугольник	Отрезок (GL_LINES), полигон (GL_POLYGON)
5	«Кораблик» – трапеция и вытянутый прямоугольник	Четырехугольник (GL_QUADS), отрезок (GL_LINES)
6	«Молоток» – трапеция и вытянутый прямоугольник	Полигон (GL_POLYGON), четырехугольник (GL_QUADS)
7	«Окно» – прямоугольник и многоугольник	Замкнутый контур (GL_LINE_LOOP), полигон (GL_POLYGON)

8	«Лопата» – выпуклый пятиугольник и вытянутый прямоугольник	Полигон (GL_POLYGON), отрезок (GL_LINES)
9	«Дупель один-один» – два смежных квадрата с точками в центрах	Четырехугольник (GL_QUADS), точка (GL_POINTS),
10	Пятиугольник с двумя диагоналями	Треугольник (GL_TRIANGLES), отрезок (GL_LINES)
11	«Веселый Роджер» – вытянутая по вертикали трапеция и ниже – косой крест из двух прямоугольников	Четырехугольник (GL_QUADS), полигон (GL_POLYGON)
12	«Рюмка» – равнобокая трапеция на 6-угольной ножке	Полигон (GL_POLYGON), замкнутый контур (GL_LINE_LOOP),
13	«Часы» – квадрат, два узких прямоугольника разной длины и крупная точка в центре	Связанные отрезки (GL_LINE_STRIP), четырехугольник (GL_QUADS), точка (GL_POINTS)
14	«Наполовину пустая бутылка» – прямоугольник и замкнутая ломаная линия из 8 сегментов	Связанные треугольники (GL_TRIANGLE_STRIP), связанные отрезки (GL_LINE_STRIP)
15	«Квадратная гайка» – правильный квадрат с отверстием (многоугольником) внутри	Полигон (GL_POLYGON), четырехугольник (GL_QUADS)

Задание 2

Изучаемые команды

Вспомогательные команды для отображения квадрик: gluNewQuadric, gluDeleteQuadric, glScale.

Команды рисования фигур: gluCylinder, gluDisk, gluPartialDisk, gluSphere, glut(Wire|Solid)Sphere, glut(Wire|Solid)Cone, glut(Wire|Solid)Cube, glut(Wire|Solid)Tetrahedron (в скобках через черту – варианты написания).

Варианты заданий приведены в таблице.

N варианта	Фигура	Геометрический примитив
1	Эллиптический цилиндр	Цилиндр
2	Эллиптический диск с отверстием	Диск (gluDisk)
3	Сектор эллиптического диска с отверстием	Диск (gluPartialDisk)
4	Эллипсоид	Сфера
4	Конус	Конус или цилиндр
6	Усеченный конус	Цилиндр

7	Куб	Куб или цилиндр
8	4-гранная пирамида	Цилиндр или тетраэдр
9	6-гранная усеченная пирамида	Цилиндр
10	Прямоугольный параллелепипед	Цилиндр или куб
11	Вытянутая 4-угольная рамка	Диск
12	Вытянутый угол («клюшка»)	Диск (gluPartialDisk)
13	«Блин» – эллипсоид, одна из осей которого намного меньше двух других	Сфера
14	«Шпиль» – сильно вытянутая 4-гранная пирамида	Цилиндр или тетраэдр
15	«Граненая» сфера с 48 гранями	Сфера

3.2. Программы выполнения заданий

Программа выполнения первого задания

1. Спроектировать изображение, выбрав координаты вершин, которые ограничивают примитивы. Координаты *x* и *y* вершин не должны превышать ±1, координата *z* берется равной нулю. Выбрать размеры окна вывода. Определить цвет фона (не черный) и цвет изображения: разный для каждого примитива.
2. В программу-заготовку добавить команды задания окна и рисования изображения. Использовать формат команды задания вершин glVertex3f. Получить изображение заданных фигур на экране. Предъявить результаты преподавателю.
3. Изменяя координаты *x* и *y* вершин в диапазоне (-4,+4), определить границы объема видимости по соответствующим координатам, а также направления координатных осей и точку их начала.
4. Задать координату *z* одной из вершин равной +0.4, затем -0.4. Объяснить наблюдаемый эффект. Задать координату *z* одной из вершин равной +4, затем -4. Объяснить наблюдаемый эффект.
5. Ответить на контрольные вопросы.

Программа выполнения второго задания

1. Спроектировать изображение, выбрав для заданных примитивов значения геометрических параметров. Линейные размеры и радиусы фигур выбрать из диапазона (0,1). Соотношение (ширина/высота/глубина) задавать с помощью команды масштабирования. Выбрать цвета фона и фигуры.
2. Подготовить программу отображения заданной геометрической фигуры в окне. Фигуры будут выглядеть объемными, если прибегнуть к моделированию их освещения. Для этого перед рисованием необходимо разрешить их освещение командой

glEnable(GL_LIGHTING). Для улучшения восприятия фигуры следует задать ракурс ее представления, применив команды glRotatef(30,1,0,0); glRotatef(30,0,1,0);

3. В программу-заготовку добавить команды рисования окна вывода и геометрической фигуры. Получить изображение.

4. Изменяя параметры команд рисования, добиться соответствия изображения заданию. Выяснить зависимость между значениями всех параметров и внешним видом фигуры. Представить результаты преподавателю.

5. Ответить на контрольные вопросы.

3.3. Контрольные вопросы к заданиям лабораторной работы 3

Контрольные вопросы к первому заданию

1. Для чего применяются команды glMatrixMode, glLoadIdentity, glOrtho?
2. В каком диапазоне могут быть заданы координаты вершин примитивов командой glVertex?
3. Какими задаются по умолчанию цвет пера и цвет фона? Как их можно изменить?
4. Какие размеры имеет область видимости? Как ее можно изменить?
4. Сколько раз нужно применить команду glVertex при рисовании замкнутого шестиугольника с помощью примитива GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP?
6. Если по очереди задать координату z разных вершин объекта равной +2 или -2, произойдет ограничение (отсечение) примитива, при этом линия отсечения будет иметь разный вид. Нужно объяснить вид линии отсечения.
7. Примитивы GL_QUADS, GL_POLYGON, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN можно окрасить со смешиванием цветов и без смешивания, т.е. чистыми цветами. Как это сделать? Как сделать, чтобы каждый треугольник этих примитивов имел свой цвет?

Контрольные вопросы ко второму заданию

1. Поясните назначение параметров команды gluCylinder, gluDisk, gluPartialDisk, gluSphere. В каком диапазоне могут меняться значения параметров? На что это влияет?
2. Чем различаются примитивы WireSphere и SolidSphere расширения glut?
3. Среди геометрических примитивов библиотеки OpenGL нет пятигранной пирамиды. Можно ли ее получить, используя другие примитивы?
4. Что происходит в программе при выполнении команды gluNewQuadric?
5. Как будет нарисован цилиндр (gluCylinder) если перед его рисованием применить команду glPolygonMode?

По работе 3 выполняется отчет. В нем описывается ход выполнения двух заданий работы по созданию геометрических фигур из заданных примитивов. Нужно описать закон размещения фигур в системе координат окна вывода и выбор координат вершин геометрических примитивов. В отчет поместить скриншот полученного изображения и результаты выполнения п.4 программы выполнения первого задания (можно в палитре серого).

Лабораторная работа 4

Отображение динамической пространственной сцены

4.1. Цель работы и лабораторное задание

Целью работы является практическое изучение средств графической библиотеки OpenGL, необходимых для выполнения модельно-видовых преобразований, проектирование и отображение пространственной сцены, а также самостоятельное освоение и применение средств придания объектам пространственной сцены динамики поворота, перемещения и масштабирования в соответствии с заданием преподавателя.

Изучаемые команды

Команды задания геометрических преобразований объектов: glTranslate, glRotate, glScale.

Команды проецирования: glOrtho, glFrustum, gluPerspective.

Команды работы со стеком: glPushMatrix, glPopMatrix.

Команды создания и вызова дисплейного списка: glNewList, glEndList, glCallList.

Команды работы с буфером глубины: glEnable(GL_DEPTH_TEST), glClear(GL_DEPTH_BUFFER_BIT).

Варианты заданий по размещению объектов в сцене приведены в таблице.

N варианта	Расположение объектов в сцене	Поворот объектов
1	По диагонали объема видимости, начиная с ближнего левого нижнего угла	*AX*=30º, *AУ*=-20º
2	По диагонали объема видимости, начиная с ближнего левого верхнего угла	*AX*=30º, *AУ*=20º
3	По диагонали объема видимости, начиная с ближнего правого нижнего угла	*AX*=-30º, *AУ*=-20º
4	По диагонали объема видимости, начиная с ближнего правого верхнего угла	*AX*=-30º, *AУ*=20º
4	По диагонали объема видимости, начиная с дальнего левого нижнего угла	*AX*=40º, *AУ*=-14º
6	По диагонали объема видимости, начиная с дальнего левого верхнего угла	*AX*=30º, *AУ*=-20º
7	По диагонали объема видимости, начиная с дальнего правого нижнего угла	*AX*=30º, *AУ*=-40º
8	По диагонали объема видимости, начиная с дальнего правого верхнего угла	*AX*=-14º, *AУ*=-30º

9	От середины нижнего ребра ближней грани объема видимости до середины верхнего ребра дальней грани	*АХ*=30º, *АУ*=40º
10	От середины верхнего ребра ближней грани объема видимости до середины нижнего ребра дальней грани	*АХ*=-30º, *АУ*=30º
11	От середины левого ребра ближней грани объема видимости до середины правого ребра дальней грани	*АХ*=30º, *АУ*=-34º
12	От середины правого ребра ближней грани объема видимости до середины левого ребра дальней грани	*АХ*=40º, *АУ*=-10º
13	От ближнего левого нижнего угла объема видимости до середины верхнего ребра дальней грани	*АХ*=-24º, *АУ*=-40º
14	От ближнего правого нижнего угла объема видимости до середины верхнего ребра дальней грани	*АХ*=-20º, *АУ*=-40º
15	От ближнего левого верхнего угла объема видимости до середины нижнего ребра дальней грани	*АХ*=24º, *АУ*=40º

4.2. Программа работы

1. Число объектов в сцене – не менее трех. В качестве объектов сцены использовать геометрические фигуры из заданий 1,2 лабораторной работы 3. Оформить рисование объектов в виде дисплейных списков. Для получения требуемого числа объектов в сцене объекты рисуются несколько раз. Спроектировать сцену, выбрав необходимые команды и их параметры. Для повышения наглядности сцены размеры объектов задать с помощью операции масштабирования. Чтобы упростить размещение объектов в сцене, использовать стек модельно-видовых матриц. Получить изображение сцены. Надлежащим выбором параметров команд сдвига добиться размещения объектов сцены в соответствии с заданием. Предъявить результат преподавателю.

2. В соответствии с заданием разработать законы движения или трансформации объектов, составить математическое или логическое описание их динамики. Выбрать команды графической библиотеки, обеспечивающие реализацию заданных видов динамики, разработать последовательность их применения. Составить программу на выбранном языке программирования, предусмотрев программные средства реализации динамики. Проанализировать вид сцены с включенным и выключенным буфером глубины. Динамическую сцену предъявить преподавателю для контроля.

4.3. Контрольные вопросы к лабораторной работе 4

1. Поясните суть параметров команд glTranslate, glRotate. Каковы диапазоны значений этих параметров? На какой угол повернется объект на экране, если к нему применить команду glRotatef(7200.0,0.0,0.0,4.0)?

2. Как используются в программе параметры команды glScale? Какое преобразование будет выполнено командой glScalef(-1.0,-1.0,1.0), glScalef(2.0,0.0,1.0)?

3. Для чего применяется дисплейный список? Как он описывается, как вызывается?

4. Для чего применяется стек модельно-видовых матриц? Как им пользоваться?

5. С помощью каких средств программирования и графической библиотеки можно реализовать вращение объекта вокруг центра, положение которого не совпадает ни с одной вершиной объекта?

6. Чем различается реализация кругового движения объекта, при котором ось объекта остается параллельной самой себе, и кругового движения, при котором ось объекта все время проходит через центр кругового движения?

7. Как реализовать попеременное увеличение и уменьшение размеров объекта вдоль одной координаты объектной системы координат?

8. Как можно задать криволинейную траекторию движения объекта в окне вывода?

9. Почему при повороте 2D-объекта на 90 градусов вокруг вертикальной оси он пропадает из вида? Проверьте, будет ли в этом случае пропадать из вида контурный примитив GL_LINE_LOOP.

По лабораторной работе 4 оформляется отчет. В нем следует описать ход создания динамической сцены, выбор параметров команд, задающих геометрические преобразования. Особое внимание нужно уделить алгоритмам задания динамических преобразований в режиме реального времени. В отчете поместить и описать скриншоты нескольких фаз динамики изображения, можно в палитре серого.

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ ПЛАНИРОВАНИЕ И ОРГАНИЗАЦИЯ НАУЧНЫХ ИССЛЕДОВАНИЙ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

ОСНОВЫ НАУЧНЫХ ИССЛЕДОВАНИЙ

Конспект лекций

ОГЛАВЛЕНИЕ

Модуль I (темы 1-4)

Тема 1. Наука и ее роль в развитии общества

Основные подходы к определению понятий «наука», «научное знание». Отличительные признаки науки. Награды и премии за научные достижения. Наука как система. Процесс развития науки. Цель и задачи науки. Субъект и объект науки. Классификация наук. Характерные особенности современной науки.

Тема 2. Научное исследование и его этапы

Определение научного исследования. Цели и задачи научных исследований, их классификация по различным основаниям. Основные требования, предъявляемые к научному исследованию. Формы и методы научного исследования. Теоретический уровень исследования и его основные элементы. Эмпирический уровень исследования и его особенности.

Этапы научно-исследовательской работы. Правильная организация научно-исследовательской работы.

Тема 3. Методологические основы научного знания

Понятие методологии научного знания. Уровни методологии. Метод, способ и методика. Общенаучная и философская методология: сущность, общие принципы. Классификация общенаучных методов познания. Общелогические, теоретические и эмпирические методы исследования.

Тема 4. Выбор направления научно-исследовательской работы. Планирование научно-исследовательской работы

Формулирование темы научного исследования. Критерии, предъявляемые к теме научного исследования. Постановка проблемы исследования, ее этапы. Определение цели и задач исследования. Планирование научного исследования. Рабочая программа и ее структура. Субъект и объект научного исследования. Интерпретация основных понятий. План и его виды. Анализ теоретико-экспериментальных исследований. Формулирование выводов.

Модуль II (темы 5-9)

Тема 5. Научная информация: поиск, накопление, обработка

Определение понятий «информация» и «научная информация». Свойства информации. Основные требования, предъявляемые к научной информации. Источники научной информации и их классификация по различным основаниям. Информационные потоки. Работа с источниками информации. Универсальная десятичная классификация. Особенности работы с книгой. Ведение записей.

Тема 6. Патентные исследования. Техническое и интеллектуальное творчество и его правовая охрана. Патент и порядок его получения. Изобретение, полезные модели, промышленные образцы: определения, условия патентоспособности, правовая охрана. Особенности патентных исследований. Последовательность работы при проведении патентных исследований. Интеллектуальная собственность и ее защита.

Тема 7. Внедрение научных исследований и их эффективность

Процесс внедрения НИР и его этапы. Эффективность научных исследований. Основные виды эффективности научных исследований. Экономический эффект от внедрения научно- исследовательских разработок. Оценка эффективности исследований.

Тема 8. Общие требования к научно-исследовательской работе

Структура научно-исследовательской работы. Способы написания текста. Язык и стиль экономической речи. Оформление таблиц, графиков, формул, ссылок.

Тема 9. Основные требования к написанию, оформлению и защите научных работ студентов

Подготовка рефератов и докладов. Подготовка и защита курсовых, дипломных работ. Рецензирование.

Модуль 1

Тема 1. Наука и ее роль в развитии общества

1.1. Понятие науки

Понятие «наука» имеет несколько основных значений.

Во-первых, под наукой (греч. episteme, лат. scientia) мы понимаем сферу человеческой деятельности, направленную на выработку и теоретическую схематизацию объективных знаний о действительности.

Во втором значении наука выступает как результат этой деятельности – система полученных научных знаний.

В-третьих, термин «наука» употребляется для обозначения отдельных отраслей научного знания.

В-четвертых, науку можно рассматривать как отрасль культуры, которая существовала не во все времена и не у всех народов. В ходе исторического развития наука превратилась в производительную силу общества и важнейший социальный институт.

Непосредственные цели науки – это получение знаний об окружающем мире, предсказание процессов и явлений действительности на основе открываемых ею законов. В широком смысле ее цель – теоретическое отражение действительности. Наука создана для непосредственного выявления существенных сторон всех явлений природы, общества и мышления.

К основным задачам науки можно отнести:

1) открытие законов движения природы, общества, мышления и познания;
2) сбор, анализ, обобщение фактов;
3) систематизация полученных знаний;
4) объяснение сущности явлений и процессов;
5) прогнозирование событий, явлений и процессов;
6) установление направлений и форм практического использования полученных знаний.

Не всякое знание можно рассматривать как научное. Нельзя признать научными те знания, которые получает человек лишь на основе простого наблюдения. Эти знания играют в жизни людей важную роль, но они не раскрывают сущности явлений, взаимосвязи между ними, которая позволила бы объяснить, почему данное явление протекает так или иначе, и предсказать дальнейшее его развитие.

Правильность научного знания определяется не только логикой, но, прежде всего обязательной проверкой его на практике. Научные знания принципиально отличаются от слепой веры, от беспрекословного признания истинным того или иного положения, без какого-либо логического его обоснования и практической проверки. Раскрывая закономерные связи действительности, наука выражает их в абстрактных понятиях и схемах, строго соответствующих этой действительности.

Будучи неотъемлемой, от практического способа освоения мира, наука как производство знания представляет собой весьма специфическую форму деятельности, отличную как от деятельности в сфере материального производства, так и от других видов духовной деятельности. Если в материальном производстве знания используются лишь в качестве идеальных средств, то в науке их получение образует главную и непосредственную цель, независимо от того, в каком виде воплощается эта цель - в виде ли теоретического описания, схемы технологического процесса, сводки экспериментальных данных или формулы какого-либо препарата. В отличие от видов деятельности, результат которых зачастую известен заранее или задан до начала деятельности, научная деятельность правомерно называется таковой лишь постольку, поскольку она даёт приращение нового знания, т.е. её результат принципиально нетрадиционен. Именно поэтому наука выступает как сила, постоянно революционизирующая другие виды деятельности.

От эстетического (художественного) способа освоения действительности, носителем которого является искусство, науку отличает стремление к обезличенному, максимально обобщённому объективному знанию, в то время как в искусстве результаты художественного познания неотделимы от индивидуально-неповторимого личностного элемента. Часто искусство характеризуют как "мышление в образах", а науку - как "мышление в понятиях", имея целью подчеркнуть, что первое развивает преимущественно чувственно-образную сторону творческой способности человека, а наука - в основном интеллектуально-понятийную. Однако эти различия не означают непроходимой грани между наукой и искусством, которые объединяет творчески-познавательное отношение к действительности. С одной стороны, в построениях науки, в частности в конструкции теории, в математической формуле, в схеме эксперимента или его идее, существенную роль нередко играет эстетический элемент, что специально отмечали многие учёные. С другой стороны, произведения искусства несут, помимо эстетической, и познавательную нагрузку.

Сложный характер имеет взаимосвязь между наукой и философией как специфическими формами общественного сознания. Философия всегда в той или иной мере выполняет по отношению к науке функции методологии познания и мировоззренческой интерпретации его результатов. Философию объединяет с наукой также стремление к построению знания в теоретической форме, к логической доказательности своих выводов. Высшего воплощения это стремление достигает в диалектическом материализме - философии, которая сознательно и открыто связывает себя с наукой, с научным методом, делая предметом своего изучения наиболее общие законы развития природы, общества и мышления и, опираясь при этом на результаты науки.

Развитию науки свойствен кумулятивный характер: на каждом историческом этапе она суммирует в концентрированном виде свои прошлые достижения, и каждый результат науки входит неотъемлемой частью в её

общий фонд, не перечёркиваясь последующими успехами познания, а лишь уточняясь и перерабатываясь.

Преемственность науки приводит к единой линии её поступательного развития и необратимому его характеру. Она обеспечивает также функционирование науки как особого вида "социальной памяти" человечества, теоретически кристаллизующей прошлый опыт познания действительности и овладения её законами.

Процесс развития науки находит своё выражение не только в возрастании суммы накапливаемых положительных знаний. Он затрагивает также всю структуру науки. На каждом историческом этапе научное познание использует определённую совокупность познавательных форм - фундаментальных категорий и понятий, методов, принципов и схем объяснения, т.е. всего того, что объединяют понятием стиля мышления. Например, для античного стиля мышления характерно было наблюдение как основной способ получения знания; наука нового времени опирается на эксперимент и на господство аналитического подхода, направляющего мышление к поиску простейших, далее не разложимых первоэлементов исследуемой реальности. Современная наука характеризуется стремлением к целостному и многостороннему охвату изучаемых объектов.

Каждая конкретная структура научного мышления после своего утверждения открывает путь к экстенсивному развитию познания, к его распространению на новые сферы реальности. Однако накопление нового материала, не поддающегося объяснению на основе существующих схем, заставляет искать новые, интенсивные пути развития науки, что приводит время от времени к научным революциям, т.е. радикальной смене основных компонентов содержательной структуры науки,к выдвижению новых принципов познания, категорий и методов науки. Чередование экстенсивных и революционных периодов развития, характерное как для науки в целом, так и для отдельных её отраслей, рано или поздно находит своё выражение также и в соответствующих изменениях форм организации науки.

Всю историю науки пронизывает сложное диалектическое сочетание процессов дифференциации и интеграции; освоение всё новых областей реальности и углубление познания приводят к дифференциации науки, к дроблению её на всё более специализированные области знания; вместе с тем потребность в синтезе знания постоянно находит выражение в тенденции к интеграции науки. Первоначально новые отрасли науки формировались по предметному признаку - сообразно с вовлечением в процесс познания новых областей и сторон действительности. Для современной науки становится всё более характерным переход от предметной к проблемной ориентации, когда новые области знания возникают в связи с выдвижением определённой крупной теоретической или практической проблемы. Так возникло значительное количество стыковых (пограничных) наука типа биофизики и т.п. Их появление продолжает в новых формах процесс дифференциации науки, но вместе с тем даёт и новую основу для интеграции прежде разобщенных научных дисциплин.

Важные интегрирующие функции по отношению к отдельным отраслям науки выполняют философия, которая обобщает научную картину мира, а также отдельные научные дисциплины типа математики, логики, кибернетики, вооружающие науку системой единых методов.

Науку можно рассматривать **как систему,** состоящую: **из теории; методологии, методики и техники исследований; практики внедрения полученных результатов.** Если науку рассматривать с точки зрения взаимодействия субъекта и объекта познания, то она включает в себя следующие элементы: объект – то, что изучает конкретная наука. Например, объектом теории финансов являются основные закономерности возникновения и развития финансов, их сущность, назначение и функционирование; субъект – конкретный научный работник, специалист, исследователь, научная организация; научная деятельность субъектов, применяющих определенные приемы, методы для обнаружения законов действительности.

Развитие науки идет от сбора фактов, их изучения и систематизации, обобщения и раскрытия отдельных закономерностей к связанной, логически стройной системе научных знаний, которая позволяет объяснить уже известные факты и предсказать новые.

Путь познания определяется от живого созерцания к абстрактному мышлению и от последнего к практике. Процесс познания включает накопление фактов. Без систематизации и обобщения, без логического осмысления фактов не может существовать ни одна наука. Но хотя факты — это необходимый материал для ученого, сами по себе они еще не наука. Факты становятся составной частью научных знаний, когда они выступают в систематизированном, обобщенном виде. Факты систематизируют и обобщают с помощью простейших абстракций — понятий (определений), являющихся важными структурными элементами науки. Наиболее широкие понятия называют категориями. Это самые общие абстракции. К категориям относятся философские понятия о форме и содержании явлений, в экономической теории — это товар, стоимость и т. д.

Важная форма знаний — принципы (постулаты), аксиомы. Под принципом понимают исходные положения какой-либо отрасли науки. Они являются начальной формой систематизации знаний (аксиомы евклидовой геометрии, постулат Бора в квантовой механике и т. д.).

Важнейшим составным звеном в системе научных знаний являются научные законы, отражающие наиболее существенные, устойчивые, повторяющиеся объективные внутренние связи в природе, обществе и мышлении. Обычно законы выступают в форме определенного соотношения понятий, категорий.

Наиболее высокой формой обобщения и систематизации знаний является теория. Под теорией понимают учение об обобщенном опыте (практике), формулирующее научные принципы и методы, которые позволяют обобщить и познать существующие процессы и явления, проанализировать действие на

них разных факторов и предложить рекомендации по использованию их в практической деятельности людей.

За выдающиеся научные достижения учёным присуждаются научные премии и медали.

- Нобелевская премия — самая престижная и знаменитая научная премия, присуждается в ряде номинаций. На неё существует пародия в виде Шнобелевской премии.
- Премия и медаль Филдса — за успехи в области математики.
- Премия Рольфа Неванлинны — за крупные достижения в математических аспектах информатики.
- Премия Карла Фридриха Гаусса — за выдающийся вклад в математику посредством открытий в других науках.
- Премия Крафурда — награда вручается по следующим направлениям: Астрономия и Математика, Биологические науки и Науки о Земле.
- Премия Абеля — за вклад в математику.
- Премия Шао Ифу — за вклад в астрономию, математику и медицину или науки о жизни.
- Премия Тьюринга — самая престижная премия в информатике, вручаемая Ассоциацией вычислительной техники.

Премия учреждена Ассоциацией вычислительной техники в честь выдающегося английского учёного Алана Тьюринга, получившего первые глубокие результаты относительно вычислимости задолго до появления первых электронных вычислительных машин.

Премия ежегодно вручается одному или нескольким специалистам в области информатики и вычислительной техники, чей вклад в этой области оказал сильное и продолжительное влияние на компьютерное сообщество[1]. Премия может быть присуждена одному человеку не более одного раза. В сфере информационных технологий премия Тьюринга имеет статус, аналогичный Нобелевской премии в академических науках. Впервые Премия Тьюринга была присуждена в 1966 году Алану Перлису за развитие технологии создания компиляторов.

В настоящее время премия спонсируется корпорациями Intel и Google и составляет 250 000 долларов США

- Премия Декарта — за выдающиеся достижения в науке и технике.
- Большая золотая медаль имени М. В. Ломоносова — высшая награда Российской академии наук.
- Золотая медаль имени Д. И. Менделеева — научная награда Российской академии наук за выдающиеся научные работы в области химической науки и технологии.

1.2. Классификация наук

Проблема классификации наук – это проблема связи между науками и вместе с тем проблема структуры всего научного знания. Если проследить эволюцию классификаций наук, можно выделить следующие основные тенденции:

1. От дифференциации наук к их интеграции. Принято считать, что дифференциация наук (то есть, возникновение отдельных отраслей научного знания) началась в эпоху Возрождения, а поворот к интеграции - в середине 19 века.
2. От координации наук к их субординации. То есть, от внешнего соположения наук к внутреннему их соподчинению.
3. От субъективности к объективности в обосновании связи наук.
4. От изолированности наук к междисциплинарности.
5. От однолинейности к разветвленности в изображении классификации наук.

Одним из принципов классификации наук является различение наук по объекту (предмету), методу и практическому применению.

Научные дисциплины, образующие в своей совокупности систему наук в целом, весьма условно можно подразделить на 4 большие группы (подсистемы) - естественные, технические, общественные и гуманитарные, различающиеся по своим предметам и методам. Резкой грани между этими подсистемами нет - ряд научных дисциплин занимает промежуточное положение. Так, например, на стыке технических и общественных наука находится техническая эстетика, между естественными и техническими наука - бионика, между естественными и общественными наука - экономическая география. Каждая из указанных подсистем, в свою очередь, образует систему разнообразным способом координированных и субординированных предметными и методическими связями отдельных наук, что делает проблему их детальной классификации крайне сложной и полностью не решенной до сегодняшнего дня.

Наряду с традиционными исследованиями, проводимыми в рамках какой-либо одной отрасли науки, проблемный характер ориентации современной наука вызвал к жизни широкое развёртывание междисциплинарных и комплексных исследований, проводимых средствами нескольких различных научных дисциплин, конкретное сочетание которых определяется характером соответствующей проблемы. Примером этого является исследование проблем охраны природы, находящееся на перекрёстке технических наук, биологии, наук о Земле, медицины, экономики, математики и др. Такого рода проблемы, возникающие в связи с решением крупных хозяйств, и социальных задач, типичны для современной науки.

По своей направленности, по непосредственному отношению к практике отдельные науки принято подразделять на фундаментальные и прикладные. Задачей фундаментальных наук является познание законов, управляющих поведением и взаимодействием базисных структур природы, общества и мышления. Эти законы и структуры изучаются в «чистом виде», как таковые, безотносительно к их возможному использованию. Поэтому фундаментальные науки иногда называют «чистыми». Непосредственная цель прикладных наук - применение результатов фундаментальных наук для решения не только познавательных, но и социально-практических проблем. Поэтому здесь критерием успеха служит не только достижение истины, но и

мера удовлетворения социального заказа. На стыке прикладных наук и практики развивается особая область исследований - разработки, переводящие результаты прикладных наука в форму технологических процессов, конструкций, промышленных материалов и т.п.

Прикладные науки могут развиваться с преобладанием как теоретической, так и практической проблематики. Например, в современной физике фундаментальную роль играют электродинамика и квантовая механика, приложение которых к познанию конкретных предметных областей образует различные отрасли теоретической прикладной физики – физику металлов, физику полупроводников и т.п. Дальнейшее приложение их результатов к практике порождает разнообразные практические прикладные науки - металловедение, полупроводниковую технологию и т.п., прямую связь которых с производством осуществляют соответствующие конкретные разработки. Все технические науки являются прикладными.

Как правило, фундаментальные науки опережают в своём развитии прикладные, создавая для них теоретический задел. В современной науке на долю прикладных приходится до 80-90% всех исследований и ассигнований. Одна из насущных проблем современной организации науки - установление прочных, планомерных взаимосвязей и сокращение сроков движения в рамках цикла «фундаментальные исследования - прикладные исследования - разработки – внедрение».

В приказе от 12 сентября 2013 г. N 1061 «Об утверждении перечней специальностей и направлений подготовки высшего образования (в ред. Приказа Минобрнауки России от 29.01.2014 N 63) выделены:
математические и естественные науки;
инженерное дело, технологии и технические науки;
здравоохранение и медицинские науки;
сельское хозяйство и сельскохозяйственные науки;
науки об обществе;
образование и педагогические науки;
гуманитарные науки;
искусство и культура.

В Номенклатуре специальностей научных работников (в редакции приказов Минобрнауки РФ от 11.08.2009 № 294, от 10.01.2012 № 5) выделены следующие отрасли науки: физико-математические, химические, биологические, технические, сельскохозяйственные, гуманитарные, филологические, философские, социально-экономические и общественные, педагогические, социологические, юридические, медицинские, медико-биологические, фармацевтические и науки о земле. Каждая из названных групп наук подвергнута дальнейшему дроблению. В статистических сборниках обычно выделяют следующие секторы науки: академический, отраслевой, вузовский.

Тема 2. Научное исследование и его этапы

Формой существования и развития науки является научное исследование. В Федеральном законе РФ от 23 августа 1996 г. «О науке и государственной научно-технической политике» (ред. от 02.11.2013 г.) научная (научно-исследовательская) деятельность определена как деятельность, направленная на получение и применение новых знаний. Цель научного исследования — определение конкретного объекта и всестороннее, достоверное изучение его структуры, характеристик, связей на основе разработанных в науке принципов и методов познания, а также получение полезных для деятельности человека результатов, внедрение в производство с дальнейшим эффектом. Объектом научного исследования являются материальная или идеальная системы, а предметом – структура системы, взаимодействие ее элементов, различные свойства, закономерности развития.

Результаты научных исследований оцениваются тем выше, чем выше научность сделанных выводов и обобщений, чем достовернее они и эффективнее. Они должны создавать основу для новых научных разработок.

Одним из важнейших требований, предъявляемых к научному исследованию, является научное обобщение, которое позволит установить зависимость и связь между изучаемыми явлениями и процессами и сделать научные выводы. Чем глубже выводы, тем выше научный уровень исследования.

Научные исследования классифицируются по различным основаниям.

фундаментальные научные исследования - экспериментальная или теоретическая деятельность, направленная на получение новых знаний об основных закономерностях строения, функционирования и развития человека, общества, окружающей среды (в ред. Федерального закона от 30.12.2008 N 309-ФЗ);

прикладные научные исследования - исследования, направленные преимущественно на применение новых знаний для достижения практических целей и решения конкретных задач;

поисковые научные исследования - исследования, направленные на получение новых знаний в целях их последующего практического применения (ориентированные научные исследования) и (или) на применение новых знаний (прикладные научные исследования) и проводимые путем выполнения научно-исследовательских работ (абзац введен Федеральным законом от 02.11.2013 N 291-ФЗ).

Источником финансирования научных исследований могут быть: собственные средства организации (предприятия); средства учредителей; средства министерств (бюджетные средства); средства государственных научных фондов (РНФ, РФФИ, РГНФ, фонда содействия развитию малых форм предприятий в научно-технической сфере); средства субъектов РФ, местных бюджетов; средства хоздоговоров; средства зарубежных контрактов и грантов; средства ФЦП; средства из других источников (спонсоров, некоммерческих организаций и т.п.). Ученый может проводить нефинансируемые (инициативные) исследования.

Постановлением Правительства Российской Федерации от 15 июля 2009 г. № 602 (с изм. от 21 июня 2014 г. № 573) утвержден перечень российских организаций, получаемые налогоплательщиками гранты (безвозмездная помощь) которых, предоставленные для поддержки науки, образования, культуры и искусства в Российской Федерации, не подлежат налогообложению.

По длительности научные исследования можно разделить на долгосрочные, краткосрочные и экспресс-исследования.

В науке можно выделить эмпирический и теоретический уровни исследования и организации знания. Теоретический уровень научного знания предполагает наличие особых абстрактных объектов (конструктов) и связывающих их теоретических законов, создаваемых с целью идеализированного описания и объяснения эмпирических ситуаций, т.е. с целью познания сущности явлений. Цель их — расширить знания общества и помочь более глубоко понять законы природы. Такие разработки используют в основном для дальнейшего развития новых теоретических исследований, которые могут быть долгосрочными, бюджетными и др.

Элементами эмпирического знания являются факты, получаемые с помощью наблюдений и экспериментов и констатирующие качественные и количественные характеристики объектов и явлений. Устойчивая повторяемость и связи между эмпирическими характеристиками выражаются с помощью эмпирических законов, часто имеющих вероятностный характер.

Итак, **теоретический уровень** исследования характеризуется **преобладанием логических методов познания**. На этом уровне полученные факты исследуются, обрабатываются с помощью логических понятий, умозаключений, законов и других форм мышления. Здесь исследуемые объекты мысленно анализируются, обобщаются, постигаются их сущность, внутренние связи, законы развития. На этом уровне познание с помощью органов чувств (эмпирия) может присутствовать, но оно является подчиненным.

Структурными компонентами теоретического познания являются проблема, гипотеза и теория.

Под проблемой понимают сложную теоретическую или практическую задачу, способы решения которой неизвестны или известны не полностью. Гипотеза – научное предположение, выдвигаемое для объяснения каких-либо явлений или гипотеза - это требующее проверки и доказывания предположение о причине, которая вызывает определенное следствие, о структуре исследуемых объектов и характере внутренних и внешних связей структурных элементов. Гипотеза является научной лишь в том случае, если она подтверждается фактами и она может существовать лишь до тех пор, пока не противоречит достоверным фактам опыта, в противном случае она становится просто фикцией. Гипотеза верифицируется соответствующими фактами опыта, в особенности экспериментом, получая характер истины. Таким образом, научная гипотеза должна отвечать следующим требованиям:1) релевантности, т.е. относимости к фактам, на которые она

опирается; 2) проверяемости опытным путем (исключение составляют непроверяемые гипотезы); 3) совместимости с существующим научным знанием; 4) обладания объяснительной силой, т.е. из гипотезы должно выводиться некоторое количество подтверждающих ее фактов, следствий. Большей объяснительной силой будет обладать та гипотеза, из которой выводится наибольшее количество фактов; 5) простоты, т.е. она не должна содержать никаких произвольных допущений, субъективистских наслоений.

Факты опыта какой-либо ограниченной научной области **вместе с** осуществленными, строго **доказанными гипотезами образуют теорию**. Теория представляет собой целостную систему достоверных знаний. Она является наиболее высокой формой обобщения и систематизации знаний. Теория - это учение об обобщенном опыте (практике), формулирующее научные принципы и методы, которые позволяют обобщить и познать существующие процессы и явления, проанализировать действие на них разных факторов и предложить рекомендации по использованию их в практической деятельности людей. Теория не только описывает совокупность фактов, но и объясняет их, т.е. выявляет происхождение и развитие явлений и процессов, их внутренние и внешние связи, причинные и иные зависимости. Все содержащиеся в теории положения и выводы обоснованы, доказаны. Структуру теории образуют понятия, суждения, законы, научные положения, учения, идеи и другие элементы.

Понятие – это мысль, отражающая существенные и необходимые признаки определенного множества предметов или явлений.

Категория – общее, фундаментальное понятие, отражающее наиболее существенные свойства и отношения предметов и явлений. Категории бывают философскими, общенаучными и относящимися к отдельной отрасли науки. Примеры категорий в экономических науках: цена, финансы, кредит.

Научный термин – это слово или сочетание слов, обозначающее понятие, применяемое в науке. Совокупность понятий (терминов), которые используются в определенной науке, образует ее понятийный аппарат.

Суждение – это мысль, в которой утверждается или отрицается что-либо.

Принцип – это сходные положения какой-либо отрасли науки. Они являются начальной формой систематизации знаний (аксиомы евклидовой геометрии, постулат Бора в квантовой механике и т. д.).

Аксиома – это положение, которое является исходным, недоказуемым, и из которого по установленным правилам выводятся другие положения. Логическими аксиомами являются, например, закон тождества, закон противоречия, закон исключения третьего.

Закон – положение, выражающее всеобщий ход вещей в какой-либо области; высказывание относительно того, каким образом что-либо является необходимым или происходит с необходимостью. Законы объективны и выражают наиболее существенные, устойчивые, причинно обусловленные связи и отношения между явлениями и процессами. Законы

могут быть классифицированы по различным основаниям. Так, по основным сферам реальности можно выделить законы природы, общества, мышления и познания; по объему действия – всеобщие, общие и частные. Научный закон – это знание, формулируемое людьми в понятиях, которое, однако, имеет свое основание в природе, объективном мире.

Положение – научное утверждение, сформулированная мысль.

Учение - совокупность теоретических положений о какой-либо области явлений действительности. Например,

Идея – это: 1) новое интуитивное объяснение события или явления; 2) определяющее стержневое положение в теории.

Концепция – это система теоретических взглядов, объединенных научной идеей (научными идеями); основная мысль.

Эмпирический уровень исследования характеризуется преобладанием чувственного познания (изучения внешнего мира посредством органов чувств). На этом уровне формы теоретического познания присутствуют, но имеют подчиненное значение. Взаимодействие эмпирического и теоретического уровней исследования заключается в том, что:

1) совокупность фактов составляет практическую основу теории или гипотезы;
2) факты могут подтверждать теорию или опровергать ее;
3) научный факт всегда пронизан теорией, поскольку он не может быть сформулирован без системы понятий, истолкован без теоретических представлений;
4) эмпирическое исследование в современной науке предопределяется, направляется теорией.

Формирование теоретического уровня науки приводит к качественному изменению эмпирического уровня. Если до формирования теории эмпирический материал, послуживший её предпосылкой, получался на базе обыденного опыта и естественного языка, то с выходом на теоретический уровень он "видится" сквозь призму смысла теоретических концепций, которые начинают направлять постановку экспериментов и наблюдений - основных методов эмпирического исследования.

Структуру эмпирического уровня исследования составляют факты, эмпирические обобщения и законы (зависимости).

Понятие «факт» употребляется в нескольких значениях:

1) объективное событие, результат, относящийся к объективной реальности (факт действительности) либо к сфере сознания и познания (факт сознания);

2) знание о каком-либо событии, явлении, достоверность которого доказана (истина);

3) предложение, фиксирующее знание, полученное в ходе наблюдений и экспериментов.

Эмпирическое обобщение – это система определенных научных фактов, на основании которой можно сделать определенные выводы или выявить недочеты и ошибки.

Эмпирические законы отражают регулярность в явлениях, устойчивость в отношениях между наблюдаемыми явлениями. Эти законы теоретическим знанием не являются. В отличие от теоретических законов, которые раскрывают существенные связи действительности, эмпирические законы отражают более поверхностный уровень зависимостей.

Для успеха научного исследования его необходимо правильно организовать, спланировать и выполнять в определенной последовательности (процедура исследования). Эти планы и последовательность действий зависят от вида, объекта и целей научного исследования. Так, если оно проводится на технические темы, то вначале разрабатывается основной предплановый документ – технико-экономическое обоснование, а затем осуществляются теоретические и экспериментальные исследования, составляется научно-технический отчет и результаты работы внедряются в производство. Применительно к работам студентов на экономические темы можно наметить следующие последовательные этапы их выполнения:

1) подготовительный;
2) проведение теоретических и эмпирических исследований;
3) работа над рукописью и её оформление;
4) внедрение результатов научного исследования.

Представляется необходимым сначала дать общую характеристику каждому этапу научно-исследовательской работы, а затем более подробно рассмотреть те из них, которые имеют важное значение для выполнения научных исследований студентами.

Подготовительный этап включает: выбор темы; обоснование необходимости проведения исследования по ней; определение гипотез, целей и задач исследования; разработку плана или программы научного исследования; подготовку средств исследования (инструментария). Вначале формулируется тема научного исследования и обосновываются причины её разработки. Путем предварительного ознакомления с литературой и материалами ранее проведенных исследований выясняется, в какой мере вопросы темы изучены и каковы полученные результаты. Особое внимание следует уделить вопросам, на которые ответов вообще нет либо они недостаточны. Составляется список нормативных актов, отечественной и зарубежной литературы, картотека опубликованной судебной практики. Разрабатывается методика исследования. Подготавливаются средства НИР в виде анкет, вопросников, бланков интервью, программ наблюдения и др. Для проверки их годности могут проводиться пилотажные исследования. Исследовательский этап состоит из систематического изучения литературы по теме, статистических сведений и архивных материалов; проведения теоретических и эмпирических исследований, в том числе сбора обработки, обобщения и анализа полученных данных; объяснения новых научных фактов, аргументирования и формулирования положений, выводов и практических рекомендаций и предложений. Третий этап включает: определение композиции (построения, внутренней структуры) работы; уточнение заглавия, названий глав и параграфов; подготовку черновой

рукописи и её редактирование; оформление текста, в том числе списка использованной литературы и приложений. Четвертый этап состоит из внедрения результатов исследования в практику и авторского сопровождения внедряемых разработок. Научные исследования не всегда завершаются этим этапом, но иногда научные работы студентов (например, дипломные работы) рекомендуются для внедрения в практическую деятельность правоохранительных органов и в учебный процесс.

Тема 3. Методологические основы научного знания

3.1. Методология научных исследований

Методология в широком смысле слова представляет собой систему принципов и способов организации и построения теоретической и практической деятельности, а также – учение об этой системе. Существует другое определение методологии как «учения о методе научного познания и преобразования мира». Методология науки дает характеристику компонентов научного исследования, его объекта, предмета, задач, совокупности средств, необходимых для решения задач исследования, а также формирует представление о последовательности действий исследователя в процессе решения задачи. В современной литературе методология – это прежде всего объект, предмет, совокупность средств, необходимых для решения задач исследования; методология также формирует представление о последовательности действий исследователя в процессе решения задачи. Методологическое знание может выступать либо в описательной форме, либо в нормативной, т.е. в форме прямых предписаний и указаний к деятельности. В таком виде методология прямо направлена на реализацию деятельности.

Различают 4 уровня методологии:

1. Философская методология – общие принципы познания.
2. Общенаучная методология (содержательные общенаучные концепции, воздействующие на достаточно большое число научных дисциплин – системный подход, кибернетический подход и др.).
3. Конкретно-научная методология (совокупность методов, принципов исследования и процедур, применяемых в той или иной научной дисциплине)
4. Методология данного конкретного исследования – методика и техника исследования, набор процедур, обеспечивающих получение эмпирического материала, его первичную обработку.

Методологические принципы - кратко сформулированные теоретические положения, обобщающие достижения науки в определенной области и служащие основанием для дальнейших исследований

Примечание.

Например, методологическими (общенаучными) принципами психолого-педагогического исследования являются:

– принцип объективности;

– принцип системного изучения;
– генетический принцип;
– принцип сущностного анализа.

Основополагающим принципом любого научного исследования является методологический принцип объективности. Он выражается во всестороннем учете факторов и условий, в которых возникают и развиваются явления. Суть состоит в том, чтобы выделить – и оценить все возможные варианты решения, выявить все точки зрения на исследуемый вопрос.

Многообразие сторон, элементов, отношений, внутренних и внешних факторов функционирования и развития социально-педагогического процесса определяет принцип системного изучения. Системный подход основан на положении о том, что специфика сложного объекта (системы) не исчерпывается особенностями составляющих ее элементов, а связана, прежде всего, с характером взаимодействия между всеми ее элементами. На первый план поэтому выдвигается задача познания характера и механизма этих связей и отношений.

Для психолого-педагогических исследований важно соблюдение генетического принципа, сущностью которого является рассмотрение изучаемого факта или явления на основе анализа условий его происхождения и последующего развития. При организации исследования следует учитывать то, что сами психические явления непрерывно изменяются, что влечет за собой необходимость целенаправленной организации образовательного процесса.

Еще одним принципом является принцип сущностного анализа. Соблюдение этого принципа связано с раскрытием законов существования и функционирования явлений, условий и факторов их развития, возможностей целенаправленного их изменения. Этот принцип предполагает движение исследовательской мысли от описания к объяснению, а от него – к прогнозированию развития педагогических явлений ипроцессов. Необходимо учитывать непрерывные изменения, развитие исследуемых элементов и педагогической системы в целом. Функции многих элементов в процессе развития существенно меняются, и эффективное педагогическое средство уже на следующем, ближайшем этапе может оказаться не только малоэффективным, но и вредным.

Метод или по-другому путь исследования представляет собой способ достижения определенной цели, совокупность приемов и операций практического или теоретического освоения действительности. В области науки метод есть путь познания, который исследователь прокладывает к своему предмету. Таким образом, метод научного исследования – это способ познания объективной действительности. К методам эмпирического уровня относят наблюдение, описание, сравнение, счет, измерение, анкетный опрос, собеседование, тестирование, эксперимент, моделирование и т.д. К методам теоретического уровня причисляют аксиоматический, гипотетический (гипотетико-дедуктивный), формализацию, абстрагирование, общелогические методы (анализ, синтез, индукцию, дедукцию, аналогию) и другие.

Способ – это действие или система действий, применяемые при исполнении какой-либо работы, при осуществлении чего-либо.

Методику можно определить как совокупность способов и приемов познания. Любое научное исследование осуществляется определенными приемами и способами, по определенным правилам.

3.2. Общенаучная и философская методология: сущность, общие принципы

Среди философских методов наиболее известными являются диалектический и метафизический. Эти методы могут быть связаны с различными философскими системами. Для Гегеля диалектика есть

«использование в науке закономерности, заключенной в природе мышления, и в то же время сама эта закономерность». Диалектика - движение, которое лежит в основе всего. Для марксистского диалектического материализма диалектика есть прежде всего внутренняя закономерность экономического развития и – поскольку от последнего зависит все остальное - закономерность всего происходящего вообще. При изучении предметов и явлений диалектика рекомендует исходить из следующих принципов:

1. Рассматривать изучаемые объекты в свете диалектических законов: а) единства и борьбы противоположностей; б) перехода количественных изменений в качественные; в) отрицания отрицания;

2. Описывать, объяснять и прогнозировать изучаемые явления и процессы, опираясь на философские категории: общего, особенного и единичного; содержания и формы; сущности и явления; возможности и действительности; необходимого и случайного; причины и следствия.

3. Относиться к объекту исследования как к объективной реальности.

4. Рассматривать исследуемые предметы и явления: а) всесторонне; б) во всеобщей связи и взаимозависимости; в) в непрерывном изменении, развитии; г) конкретно-исторически.

5. Проверять полученные знания на практике.

Метафизика рассматривает вещи и явления изолированно, отдельно, независимо друг от друга. Метафизическая мысль устремлена к простому, единому и целостному.

Все общенаучные методы для анализа целесообразно распределить на три группы: общелогические, теоретические и эмпирические.

Общелогическими методами являются анализ, синтез, индукция, дедукция, аналогия.

Анализ – метод исследования, с помощью которого изучаемое явление или процесс мысленно расчленяются на составные элементы с целью изучения каждого в отдельности. Разновидностями анализа являются классификация и периодизация.

Синтез – метод исследования, предполагающий мысленное соединение составных частей или элементов изучаемого объекта, его изучение как единого целого. Методы анализа и синтеза взаимоувязаны, их одинаково используют в научных исследованиях.

Индукция – это движение мысли (познания) от фактов, отдельных случаев к общему положению. Индукция приводит к всеобщим понятиям и законам, которые могут быть положены в основу дедукции.

Дедукция – это выведение единичного, частного из какого-либо общего положения; движение мысли (познания) от общих утверждений к утверждениям об отдельных предметах или явлениях. Посредством дедуктивных умозаключений «выводят» определенную мысль из других мыслей.

Аналогия – это способ получения знаний о предметах и явлениях на основании того, что они имеют сходство с другими; рассуждение, в котором

из сходства изучаемых объектов в некоторых признаках делается заключение об их сходстве и в других признаках.

К методам теоретического уровня причисляют аксиоматический, гипотетический, формализацию, абстрагирование, ранжирование, обобщение, восхождение от абстрактного к конкретному, исторический, метод системного анализа.

В научных исследованиях широко применяется способ абстрагирования, т. е. отвлечение от второстепенных фактов с целью сосредоточиться на важнейших особенностях изучаемого явления. Например, при исследовании работы какого-либо механизма анализируют расчетную схему, которая отображает основные, существенные свойства механизма. Иногда при анализе явлений и процессов возникает потребность рассмотреть большое количество фактов (признаков). Здесь важно уметь выделить главное. В этом случае может быть применен способ ранжирования, с помощью которого исключают все второстепенное, не влияющее существенно на рассматриваемое явление.

Аксиоматический метод заключается в том, что некоторые утверждения (аксиомы, постулаты) принимаются без доказательств и затем по определенным логическим правилам из них выводятся остальные знания. В ряде случаев используют способ формализации. Сущность его состоит в том, что основные положения процессов и явлений представляют в виде формул и специальной символики. Путем операций с формулами искусственных языков можно получать новые формулы, доказывать истинность какого-либо положения. Формализация является основой для алгоритмизации и программирования, без которых не может обойтись компьютеризация знания и процесса исследования. Применение символов и других знакомых систем позволяет установить закономерности между изучаемыми фактами.

Гипотетический метод основан на разработке гипотезы, научного предположения, содержащего элементы новизны и оригинальности. Гипотеза должна полнее и лучше объяснить явления и процессы, подтверждаться экспериментально и соответствовать общим законам диалектики и естествознания. Этот метод исследования является основным и наиболее распространенным в прикладных науках.

Обобщение – установление общих свойств и отношений предметов и явлений; определение общего понятия, в котором отражены существенные, основные признаки предметов или явлений данного класса. Вместе с тем обобщение может выражаться в выделении не существенных, а любых признаков предмета или явления. Этот метод научного исследования опирается на философские категории общего, особенного и единичного.

Исторический метод позволяет исследовать возникновение, формирование и развитие процессов и событий в хронологической последовательности с целью выявить внутренние и внешние связи, закономерности и противоречия. Данный метод исследования используется преимущественно в общественных и, главным образом, в исторических

науках. В прикладных же науках он применяется, например, при изучении развития и формирования тех или иных отраслей науки и техники.

Восхождение от абстрактного к конкретному как метод научного познания состоит в том, что исследователь вначале находит главную связь изучаемого предмета (явления), затем, прослеживая, как она видоизменяется в различных условиях, открывает новые связи и таким путем отображает во всей полноте его сущность.

К методам эмпирического уровня относятся: наблюдение, описание, счет, измерение, сравнение, эксперимент, моделирование.

Первичным в познании физической и экономической сущности процессов выступают наблюдения. Наблюдение – это способ познания, основанный на непосредственном восприятии свойств предметов и явлений при помощи органов чувств. Каждое наблюдение может зафиксировать лишь некоторые факторы. Для того чтобы наиболее полно понять процесс, необходимо иметь большое количество наблюдений. Как метод научного исследования наблюдение применяется, например, для сбора социологической информации в области экономики. В зависимости от положения исследователя по отношению к объекту изучения различают простое и включенное наблюдение. Первое состоит в наблюдении со стороны, когда исследователь – постороннее по отношению к объекту лицо, не являющееся участником деятельности наблюдаемых. Второе характеризуется тем, что исследователь открыто или инкогнито включается в группу, её деятельность в качестве участника. Если наблюдение проводилось в естественной обстановке, то его называют полевым, а если условия окружающей среды, ситуация были специально созданы исследователем, то оно будет считаться лабораторным. Результаты наблюдения могут фиксироваться в протоколах, дневниках, карточках, на кинопленках и другими способами.

Наиболее важной составной частью научных исследований являются эксперименты. Это один из основных способов получить новые научные знания. От обычного, обыденного, пассивного наблюдения эксперимент отличается активным воздействием исследователя на изучаемое явление. Основной целью эксперимента является проверка теоретических положений (подтверждение рабочей гипотезы), а также более широкое и глубокое изучение темы научного исследования. Эксперимент должен быть проведен по возможности в кратчайший срок с минимальными затратами при самом высоком качестве полученных результатов. Различают эксперименты естественные и искусственные. Естественные эксперименты характерны при изучении социальных явлений (социальный эксперимент) в обстановке, например, производства, быта и т. п. Искусственные эксперименты широко применяются во многих естественнонаучных исследованиях. В этом случае изучают явления, изолированные до требуемой степени, чтобы оценить их в количественном и качественном отношениях.

Экспериментальные исследования бывают лабораторные и производственные. Лабораторные опыты проводят с применением типовых

приборов, специальных моделирующих установок, стендов, оборудования и т. д. Эти исследования позволяют наиболее полно и доброкачественно, с требуемой повторяемостью изучить влияние одних характеристик при варьировании других. Лабораторные опыты в случае достаточно полного научного обоснования эксперимента (математическое планирование) позволяют получить хорошую научную информацию с минимальными затратами. Однако такие эксперименты не всегда полностью моделируют реальный ход изучаемого процесса, поэтому возникает потребность в проведении производственного эксперимента. Производственные экспериментальные исследования имеют целью изучить процесс в реальных условиях с учетом воздействия различных случайных факторов производственной среды.

Описание – это фиксация признаков исследуемого объекта, которые устанавливаются, например, путем наблюдения, измерения или эксперимента. Описание бывает:

1) непосредственным, когда исследователь непосредственно воспринимает и указывает признаки объекта;

2) опосредованным, когда исследователь отмечает признаки объекта, которые воспринимались другими лицами.

Счет (количественный метод) - это определение количественных соотношений объектов исследования или параметров, характеризующих их свойства. Так, экономическая статистика изучает количественную сторону экономически значимых явлений и процессов, т.е. их величину, степень распространенности, соотношение отдельных составных частей, изменение во времени и пространстве.

Сравнение – это сопоставление признаков, присущих двум или нескольким объектам, установление различия между ними или нахождение в них общего. В научном исследовании этот метод применяется, например, для сравнения экономических систем, институтов различных государств.

Выделить главное и затем глубоко исследовать процессы или явления с помощью обширной, но не систематизированной информации затруднительно. Поэтому такую информацию стремятся "сгустить" в некоторое абстрактное понятие — "модель". Под моделью понимают искусственную систему, отображающую основные свойства изучаемого объекта - оригинала. Модель — это изображение в удобной форме многочисленной информации об изучаемом объекте. Она находится в определенном соответствии с последним, может заменить его при исследовании и позволяет получить информацию о нем. Метод моделирования — изучение явлений с помощью моделей — один из основных в современных исследованиях. Различают физическое и математическое моделирование. При физическом моделировании физика явлений в объекте и модели и их математические зависимости одинаковы. При математическом моделировании физика явлений может быть различной, а математические зависимости одинаковыми. Математическое моделирование приобретает особую ценность, когда возникает

необходимость изучить очень сложные процессы. При построении модели свойства и сам объект обычно упрощают, обобщают. Чем ближе модель к оригиналу, тем удачнее она описывает объект, тем эффективнее теоретическое исследование и тем ближе полученные результаты к принятой гипотезе исследования. Модели могут быть физические, математические, натурные. Физические модели позволяют наглядно представлять протекающие в натуре процессы. С помощью физических моделей можно изучать влияние отдельных параметров на течение физических процессов. Математические модели позволяют количественно исследовать явления, трудно поддающиеся изучению на физических моделях. Натурные модели представляют собой масштабно изменяемые объекты, позволяющие наиболее полно исследовать процессы, протекающие в натурных условиях. Стандартных рекомендаций по выбору и построению моделей не существует. Модель должна отображать существенные явления процесса. Мелкие факторы, излишняя детализация, второстепенные явления и т. п. лишь усложняют модель, затрудняют теоретические исследования, делают их громоздкими, нецеленаправленными. Поэтому модель должна быть оптимальной по своей сложности, желательно наглядной, но главное — достаточно адекватной, т. е. описывать закономерности изучаемого явления с требуемой точностью. Для построения наилучшей модели необходимо иметь глубокие и всесторонние знания не только по теме и смежным наукам, но и хорошо знать практические аспекты исследуемой задачи.

Тема 4. Выбор направления и планирование научно-исследовательской работы. Анализ теоретико-экспериментальных исследований и формулирование выводов

5.1 Формулирование темы научного исследования

Подготовительным этапом научно-исследовательской работы является выбор темы научного исследования. Тема научно-исследовательской работы может быть отнесена к определенному научному направлению или к научной проблеме.

Под научным направлением понимают сферу научных исследований научного коллектива, посвященных решению каких-либо крупных, фундаментальных теоретических и экспериментальных задач в определенной отрасли науки. Например, научные исследования, выполняемые экономистами, охватываются общим направлением «экономика» (экономические науки). Внутри его можно выделить конкретные направления, основой которых являются специальные экономические науки: экономическая теория, экономический анализ, налогообложение, мировая экономика, экономика организаций. Структурными единицами направления являются комплексные проблемы, проблемы, темы и вопросы. Комплексная проблема включает в себя несколько проблем.

Научная проблема – это совокупность сложных теоретических или практических задач; совокупность тем научно-исследовательской работы. Проблема охватывает значительную область исследования и имеет перспективное значение. Проблема может быть отраслевой, межотраслевой, глобальной. Проблема состоит из ряда тем. Тема — это научная задача, охватывающая определенную область научного исследования. Она базируется на многочисленных исследовательских вопросах. Под научными вопросами понимают более мелкие научные задачи, относящиеся к конкретной области научного исследования. Результаты решения этих задач имеют не только теоретическое, но, главным образом, и практическое значение, поскольку можно сравнительно точно установить ожидаемый экономический эффект. Темы могут быть теоретическими, практическими и смешанными. Теоретические темы разрабатываются преимущественно с использованием литературных источников. Практические темы разрабатываются на основе изучения, обобщения и анализа фактов. Смешанные темы сочетают в себе теоретический и практический аспекты исследования. При разработке темы или вопроса выдвигается конкретная задача в исследовании — разработать новую конструкцию, прогрессивную технологию, новую методику и т. д. Выбору тем предшествует тщательное ознакомление с отечественными и зарубежными источниками данной и смежной специальности. Постановка (выбор) проблем или тем является трудной, ответственной задачей, включает в себя ряд этапов.

Первый этап — формулирование проблем. На основе анализа противоречий исследуемого направления формулируют основной вопрос — проблему — и определяют в общих чертах ожидаемый результат.

Второй этап включает в себя разработку структуры проблемы. Выделяют темы, подтемы, вопросы. Композиция этих компонентов должна составлять древо проблемы (или комплексной проблемы). По каждой теме выявляют ориентировочную область исследования.

На третьем этапе устанавливают актуальность проблемы, т. е. ценность ее на данном этапе для науки и техники. Для этого по каждой теме выставляют несколько возражений и на основе анализа, методом исследовательского приближения, исключают возражения в пользу реальности данной темы. После такой "чистки" окончательно составляют структуру проблемы и обозначают условным кодом темы, подтемы, вопросы. При выборе важно уметь отличать псевдопроблемы от научных проблем. Псевдопроблемы (ложные, мнимые), какую бы не имели внешнюю форму, в основе своей имеют антинаучный характер.

После обоснования проблемы и установления ее структуры научный работник (или коллектив), как правило, самостоятельно приступает к выбору темы научного исследования. По мнению некоторых ученых, выбрать тему зачастую более сложно, чем провести само исследование. К теме предъявляют ряд требований. Тема должна быть актуальной, т. е. важной, требующей разрешения в настоящее время. Это требование одно из основных. Критерия для установления степени актуальности пока нет. Так,

при сравнении двух тем теоретических исследований степень актуальности может оценить крупный ученый данной отрасли или научный коллектив. При оценке актуальности прикладных научных разработок ошибки не возникают, если более актуальной окажется та тема, которая обеспечит большой экономический эффект. Тема должна решать новую научную задачу. Это значит, что тема в такой постановке никогда не разрабатывалась и в настоящее время не разрабатывается, т. е. дублирование исключается. Дублирование возможно только в том случае, когда по заданию руководящих организаций одинаковые темы разрабатывают два конкурирующих коллектива в целях разрешения важнейших государственных проблем в кратчайшие сроки. Таким образом, оправданное дублирование тем (разработок) иногда может быть одним из требований. Тема должна быть экономически эффективной и должна иметь значимость. Любая тема прикладных исследований должна давать экономический эффект в народном хозяйстве. Это одно из важнейших требований. На стадии выбора темы исследования ожидаемый экономический эффект может быть определен, как правило, ориентировочно. Иногда экономический эффект на начальной стадии установить вообще нельзя. В таких случаях для ориентировочной оценки эффективности можно использовать аналоги (близкие по названию и разработке темы). При разработке теоретических исследований требование экономичности может уступать требованию значимости. Значимость, как главный критерий темы, имеет место при разработке исследований, определяющих престиж отечественной науки или составляющих фундамент для прикладных исследований, или направленных на совершенствование общественных и производственных отношений и др.

4.2. Планирование научной работы

Планирование научно-исследовательской работы имеет важное значение для ее рациональной организации. Научно-исследовательские организации и образовательные учреждения разрабатывают планы работы на год на основе целевых комплексных программ, долгосрочных научных и научно-технических программ, хозяйственных договоров и заявок на исследования, представленных заказчиками. Научная работа кафедр учебных заведений организуется и проводится в соответствии с планами работы на учебный год. Профессора, преподаватели и аспиранты выполняют научно-исследовательские работы по индивидуальным планам. Планируется и научно-исследовательская работа студентов. Планы работы учебных заведений и кафедр могут содержать соответствующий раздело НИРСе. По планам работают студенческие научные кружки и проблемные группы. В научно-исследовательских и образовательных учреждениях по темам научно-исследовательских работ составляются рабочие программы и планы-графики их выполнения. При подготовке монографий, учебников, учебных пособий и лекций разрабатываются планы-проспекты этих работ.

Рабочая программа – это изложение общей концепции исследования в соответствии с его целями и гипотезами. Она состоит, как правило, из двух

разделов: методологического и процедурного. Методологический раздел включает: 1) формулировку проблемы или темы; 2) определение объекта и предмета исследования; 3) определение цели и постановку задач исследования; 4) интерпретацию основных понятий; 5) формулировку рабочих гипотез.

Формулировка проблемы (темы) – это определение задачи, которая требует решения. Проблемы бывают социальные и научные. Под социальной проблемой понимают противоречие в развитии общественной системы или отдельных ее элементов. Научная (гносеологическая) проблема – это противоречие между знаниями о потребностях общества и незнанием путей и средств их удовлетворения. Такие проблемы решаются путем создания теории, выработки практических рекомендаций.

Определение объекта и предмета исследования является важным методологическим этапом научной научно-исследовательской работы.

Объект исследования – это то социальное явление (процесс), которое содержит противоречие и порождает проблемную ситуацию.

Предмет исследования – это те наиболее значимые с точки зрения практики и теории свойства, стороны, особенности объекта, которые подлежат изучению. Например, если тема научной работы посвящена формированию механизма кредитно-денежного регулирования, то объектом исследования являются процессы трансформации системы кредитно-денежного регулирования в определенных условиях, а предметом – механизм кредитно-денежного регулирования экономики.

Цель исследования – это общая его направленность на конечный результат.

Задачи исследования – это то, что требует решения в процессе исследования; вопросы, на которые должен быть получен ответ.

Интерпретация основных понятий – это истолкование, разъяснение значения основных понятий. Существуют теоретическая и эмпирическая интерпретация понятий. Теоретическое истолкование представляет собой логический анализ существенных свойств и отношений интерпретируемых понятий путем раскрытия их связей с другими понятиями.

Эмпирическая интерпретация – это определение эмпирических значений основных теоретических понятий, перевод их на язык наблюдаемых фактов. Эмпирически интерпретировать понятие – это значит найти такой показатель (индикатор, референт), который отражал бы определенный важный признак содержания понятия и который можно было бы измерить.

Формулировка гипотез. Гипотеза как научное предположение, выдвигаемое для объяснения каких-либо фактов, явлений и процессов, является важным инструментом успешного решения исследовательских задач. Программа исследования может быть ориентирована на одну или несколько гипотез.

Конкретное научное исследование осуществляется по принципиальному плану, который строится в зависимости от количества информации об

объекте исследования. Планы бывают разведывательные, аналитические (описательные) и экспериментальные. Разведывательный план применяется, если об объекте и предмете исследования нет ясных представлений и трудно выдвинуть рабочую гипотезу. Цель составления такого плана – уточнение темы (проблемы) и формулировка гипотезы. Обычно он применяется, когда по теме отсутствует литература или ее очень мало. Описательный план используется тогда, когда можно выделить объект и предмет исследования и сформулировать описательную гипотезу. Цель плана – проверить эту гипотезу, описать факты, характеризующие объект исследования. Экспериментальный план включает проведение социального (правового) эксперимента. Он применяется тогда, когда сформулированы научная проблема и объяснительная гипотеза. Цель плана – определение причинно следственных связей в исследуемом объекте. В процедурной части программы обосновывается выбор методов исследования, показывается связь данных методов с целями, задачами и гипотезами исследования.

4.3. Анализ теоретико-экспериментальных исследований и формулирование выводов

Основой совместного анализа теоретических и экспериментальных исследований является сопоставление выдвинутой рабочей гипотезы с опытными данными наблюдений.

Теоретические и экспериментальные данные сравнивают методом сопоставления соответствующих графиков. Критериями сопоставления могут быть минимальные, средние и максимальные отклонения экспериментальных результатов от данных, установленных расчетом на основе теоретических зависимостей. Возможно также вычисление среднеквадратического отклонения и дисперсии. Однако наиболее достоверными следует считать критерии адекватности (соответствия) теоретических зависимостей экспериментальным.

В результате теоретико-экспериментального анализа могут возникнуть три случая:

1) установлено полное или достаточно хорошее совпадение рабочей гипотезы, теоретических предпосылок с результатами опыта. При этом дополнительно группируют полученный материал исследований таким образом, чтобы из него вытекали основные положения разработанной ранее рабочей гипотезы, в результате чего последняя превращается в доказанное теоретическое положение, в теорию;

2) экспериментальные данные лишь частично подтверждают положение рабочей гипотезы и в той или иной ее части противоречат ей. В этом случае рабочую гипотезу изменяют и перерабатывают так, чтобы она наиболее полно соответствовала результатам эксперимента. Чаще всего производят дополнительные корректировочные эксперименты с целью подтвердить изменения рабочей гипотезы, после чего она также превращается в теорию;

3) рабочая гипотеза не подтверждается экспериментом. Тогда ее критически анализируют и полностью пересматривают. Затем проводят новые экспериментальные исследования с учетом новой рабочей гипотезы.

Отрицательные результаты научной работы, как правило, не являются бросовыми, они во многих случаях помогают выработать правильные представления об объектах, явлениях и процессах.

После выполненного анализа принимают окончательное решение, которое формулируют как заключение, выводы или предложения. Эта часть работы требует высокой квалификации, поскольку необходимо кратко, четко, научно выделить то новое и существенное, что является результатом исследования, дать ему исчерпывающую оценку и определить пути дальнейших исследований. Обычно по одной теме не рекомендуется составлять много выводов (не более 5—10). Если же помимо основных выводов, отвечающих поставленной цели исследования, можно сделать еще и другие, то их формулируют отдельно, чтобы не затемнить конкретного ответа на основную задачу темы.

Тема 5. Научная информация: поиск, накопление и обработка

5.1 Научная информации и ее источники

Умственный труд в любой его форме всегда связан с поиском информации. Тот факт, что этот поиск становится сейчас все сложнее и сложнее, в доказательствах не нуждается. Усложняется сама система поиска, постепенно она превращается в специальную отрасль знаний. Знания и навыки в этой области становятся все более обязательными для любого специалиста. Понятие подготовленности в этом отношении складывается из следующих основных элементов:

1) четкого представления об общей системе информационных ресурсов и тех возможностях, которые дает использование информационных источников своей области;
2) знания всех возможных источников информации по своей специальности;
3) умения выбрать наиболее рациональную схему поиска в соответствии с его задачами и условиями;
4) наличия навыков в использовании вспомогательных библиографических и информационных материалов.

Характерной чертой развития современной науки является бурный поток новых научных данных, получаемых в результате исследований. Ежегодно в мире издается более 500 тысяч книг по различным вопросам. Еще больше издается журналов. Но, несмотря на это, огромное количество научной информации остается неопубликованной. Информация имеет свойство "стареть". Это объясняется появлением новой печатной и неопубликованной информации или снижением потребности в данной информации. По зарубежным данным интенсивность падения ценности

информации ("старения") ориентировочно составляет 10% в день для газет, 10% в месяц для журналов и 10% в год для книг. Таким образом, отыскать новое, передовое, научное в решении данной темы — сложная задача не только для одного научного работника, но и для большого коллектива.

Недостаточное использование мировой информации приводит к дублированию исследований. Количество повторно получаемых данных достигает в различных областях научно-технического творчества 60 и даже 80%. А это потери, которые в США, например, оцениваются многими миллиардами долларов ежегодно.

Что же следует понимать под термином «информация»? Приведем несколько определений информации:

1) сообщение, осведомление о положении дел, сведения о чём-либо, передаваемые людьми;
2) уменьшаемая, снимаемая неопределённость в результате получения сообщений;
3) сообщение, неразрывно связанное с управлением, сигналы в единстве синтаксических, семантических и прагматических характеристик;
4) передача, отражение разнообразия в любых объектах и процессах (неживой и живой природы).

Научная информация – это получаемая в процессе познания логическая информация, которая адекватно отображает закономерности объективного мира и используется в общественно-исторической практике. Из определения вытекает, что научной можно считать только ту информацию, которая удовлетворяет нескольким серьезным требованиям.

Во-первых, научная информация получается человеком в процессе познания, и, следовательно, неразрывно связана с его практической, производственной деятельностью, поскольку последняя является основой познания.

Во-вторых, научная информация – это логическая информация, которая образуется путем обработки информации, поставляемой человеку органами чувств, при помощи абстрактно-логического мышления. Например, совокупность данных о температуре в различных точках нашей страны, не будет еще научной информацией. Информация будет научной в том случае, когда между данными будет установлена связь.

При этом надо учитывать и третье условие отнесения той или иной информации к научной. Она должна адекватно отображать объективный мир. Однако выполнения этих условий не достаточно.

Чтобы информация считалась научной, она должна удовлетворять еще одному, четвертому условию: она должна непременно использоваться в общественно-исторической практике. Именно поэтому к научной информации не могут быть отнесены научно-фантастические литературные произведения. Не может считаться научной адекватная и логически обработанная информация, полученная кем-то в результате многолетних наблюдений за погодой только с той целью, чтобы выбрать себе наиболее

подходящее время для отпуска. Этот пример показывает, что не всякое использование информации делает ее научной.

Под «источником научной информации» понимается документ, содержащий какое-то сообщение, а отнюдь не библиотека или информационный орган, откуда он получен. Это часто путают. Документальные источники содержат в себе основной объем сведений, используемых в научной, преподавательской и практической деятельности, и поэтому в этом разделе речь идет именно о них. К документам относят различного рода издания, являющиеся основным источником научной информации. Издание – это документ, предназначенный для распространения содержащейся в нем информации, прошедший редакционно-издательскую обработку, полученный печатанием или тиснением, полиграфически самостоятельно оформленный, имеющий выходные сведения.

Документы создают огромные информационные потоки, темпы которых ежегодно возрастают. Различают восходящий и нисходящий потоки информации. Восходящий — это поток информации от пользователей в регистрирующие органы. Исполнитель научной работы (НИИ, вузы и др.) после утверждения плана работ обязан в месячный срок представить информационную карту в соответствующие вышестоящие институты. К восходящему потоку относят также статьи, направленные в различные журналы. Нисходящий — это поток информации в виде библиографических обзорных реферативных и других данных, который направляется в низовые организации по их запросам.

Все документальные источники научной информации делятся на первичные и вторичные. Первичные документы содержат исходную информацию, непосредственные результаты научных исследований (монографии, сборники научных трудов, авторефераты диссертаций и т.д.), а вторичные документы являются результатом аналитической и логической переработки первичных документов (справочные, информационные, библиографические и другие тому подобные издания). Рассмотрим, в первую очередь, те издания, из которых может быть почерпнута необходимая для научно-исследовательской работы информация. Это научные, учебные, справочные и информационные издания.

Научные издания. Под научным понимают издание, содержащее результаты теоретических и/или экспериментальных исследований, а также научно подготовленные к публикации памятники культуры и исторические документы. Научные издания можно разделить на следующие виды: монография, автореферат, диссертации, препринт, сборник научных трудов, материалы научной конференции, тезисы докладов научной конференции, научно-популярное издание.

Монография - научное или научно-популярное книжное издание, содержащее полное и всестороннее исследование одной проблемы или темы; принадлежащее одному или нескольким авторам.

Автореферат диссертации – научное издание в виде брошюры, содержащее составленный автором реферат проведенного им исследования, предоставляемого на соискание ученой степени.

Препринт - научное издание, содержащее материалы предварительного характера, опубликованные до выхода в свет издания, в котором они могут быть помещены.

Сборник научных трудов - сборник, содержащий исследовательские материалы научных учреждений, учебных заведений или обществ.

Тезисы докладов научной конференции - научный непериодический сборник, содержащий опубликованные до начала конференции материалы предварительного характера: аннотации, рефераты докладов и/или сообщений.

Материалы научной конференции – научный непериодический сборник, содержащий итоги научной конференции (программы, доклады, рекомендации, решения).

Научно-популярное издание - издание, содержащее сведения о теоретических или экспериментальных исследованиях в области науки, культуры и техники; изложенные в форме, доступной читателю-неспециалисту.

Учебные издания. Учебное издание – это издание, содержащее систематизированные сведения научного или прикладного характера, изложенные в форме, удобной для изучения и преподавания, и рассчитанное на учащихся разного возраста и ступени обучения. К учебным изданиям относятся: учебник, учебное пособие, учебное наглядное пособие, учебно-методическое пособие, хрестоматия и т.д.

Учебник - учебное издание, содержащее систематическое изложение учебной дисциплины, ее раздела или части, соответствующее учебной программе и официально утвержденное в качестве учебника.

Учебно-методическое пособие - учебное издание, содержащее материалы по методике преподавания учебной дисциплины или по методике воспитания.

Учебное пособие – это учебное издание, дополняющее или частично заменяющее учебник и официально утвержденное в качестве учебного пособия.

Хрестоматия - учебное пособие, содержащее литературно-художественные, исторические и иные произведения или отрывки из них, составляющие объект изучения учебной дисциплины.

Учебное наглядное пособие - учебное изоиздание, содержащее материалы в помощь изучению, преподаванию или воспитанию.

Справочно-информационные издания. Справочным называют издание, содержащее краткие сведения научного или прикладного характера, расположенные в порядке, удобном для их быстрого отыскания, не предназначенное для сплошного чтения.

Информационное издание - издание, содержащее систематизированные сведения об опубликованных, непубликуемых или

неопубликованных документах или результат анализа и обобщения сведений, представленных в первоисточниках.

Информационные издания выпускаются организациями, осуществляющими научно-информационную деятельность.

Информационные издания могут быть библиографическими, реферативными, обзорными.

Библиографическое издание - библиографическое пособие, выпущенное в виде отдельного документа. По многим экономическим наукам публикуются тематические библиографические справочники.

Реферативное издание – это информационное издание, содержащее упорядоченную совокупность библиографических записей, включающих рефераты.

Издания могут быть непериодическими, периодическими и продолжающимися.

Непериодические издания – это издания, выходящие однократно и не имеющие продолжения. К ним относятся: книги, брошюры, листовки и т.д.

Книга - книжное издание объемом свыше 48 страниц. Брошюра - книжное издание объемом более 4-х, но не более 48 страниц. Листовка – в издательском деле - листовое издание объемом до четырех страниц.

Периодическое издание - сериальное издание, выходящее, через определенные промежутки времени, постоянным для каждого года числом номеров (выпусков) и не повторяющимися по содержанию, однотипно оформленными нумерованными или датированными выпусками, имеющими одинаковое заглавие. К периодическим печатным изданиям – по законодательству РФ относят: газеты, журналы, альманах, бюллетени, иное издание, имеющее постоянное название, текущий номер и выходящее в свет не реже одного раза в год.

Газета – это периодическое газетное издание, выходящее через краткие промежутки времени, содержащее официальные материалы, оперативную информацию и статьи по актуальным общественно-политическим, научным, производственным и другим вопросам, а также литературные произведения и рекламу. Обычно газета издается в виде больших листов (полос).

Журнал - периодическое журнальное издание, содержащее статьи или рефераты по различным общественно-политическим, научным, производственным и другим вопросам, литературно-художественные произведения; имеющее постоянную рубрикацию, официально утвержденное в качестве журнального издания. Журнал может иметь приложения. В России выпускается множество экономических журналов, таких как «Экономист», «Вопросы экономики», «Российский экономический журнал», «Экономические науки» и др.

Альманах - сборник, содержащий литературно-художественные и/или научно-популярные произведения, объединенные по определенному признаку.

Бюллетень - периодическое или продолжающееся издание, выпускаемое оперативно, содержащее краткие официальные материалы по вопросам, входящим в круг ведения выпускающей его организации. Обычно периодические бюллетени имеют постоянную рубрикацию. Примерами таких изданий могут служить: биржевой бюллетень (периодический орган биржи, в котором публикуются курсы ценных бумаг, биржевые цены товаров, сведения о заключенных сделках), бюллетень курсов иностранной валюты (издаваемый Центральным Банком РФ официальный документ, содержащий сведения о курсе иностранных валют по отношению к рублю), бюллетень Комиссии по ценным бумагам и биржам (в США – периодическая публикация, касающаяся интерпретации правил и практики бухгалтерского учета, которых придерживается Комиссия по ценным бумагам и биржам), бюллетень по вопросам исследований в области бухгалтерского учета (издание Совета по принципам бухгалтерского учета).

5.2 Работа с источниками информации

Приступая к поиску необходимых сведений, следует четко представлять, где их можно найти и какие возможности в этом отношении имеют те организации, которые существуют для этой цели, — библиотеки и органы научной информации.

Библиотеки. В первую очередь это библиотеки научные и специальные, т. е. предназначенные для обслуживания ученых, преподавателей и специалистов различного профиля. По своим возможностям они не равны, но, тем не менее, формы обслуживания читателей у них в основном одни и те же:

- справочно-библиографическое;
- читальный зал;
- абонемент;
- межбиблиотечный обмен;
- заочный абонемент;
- изготовление фото- и ксерокопий;
- микрофильмирование.

Для справочно-библиографического обслуживания каждая библиотека имеет специальный отдел (бюро), в котором в дополнение к системе каталогов и картотек собраны все имеющиеся в библиотеке справочные издания, позволяющие ответить на вопросы, связанные с подбором литературы по определенной теме, уточнением фамилий авторов, названия произведения и т. д. Задачей библиографических отделов является также обучение читателей правилам пользования библиотечными каталогами и библиографическими указателями. Научная и специальная литература издается, как правило, сравнительно ограниченными тиражами. Поэтому в большинстве научных и специальных библиотек основной формой обслуживания является не абонемент, а читальный зал.

Пользуясь им и абонементом, каждый обязан помнить, что в больших книгохранилищах, имеющих сотни тысяч томов, подбор книг — сложный и

трудоемкий процесс. Он значительно облегчается и ускоряется, если в заявке точно указаны все данные книги и ее шифр, особенно важен шифр, показывающий место ее хранения.

Для ускорения подбора литературы в большинстве библиотек практикуется система открытого доступа к полкам, при этом экономится время, появляется возможность ознакомиться с широким кругом литературы по интересующему вопросу. Во многих библиотеках отдельные материалы находятся в виде микрофильмов или микроафиш, для чтения их используется специальная аппаратура.

Межбиблиотечный абонемент (МБА) представляет собой территориально-отраслевую систему взаимного использования фондов всех научных и специальных библиотек страны. Зная о существовании той или иной книги, но не найдя ее в доступной библиотеке, можно заказать ее по МБА. Присланные на определенный срок книги выдаются для работы в читальном зале.

Многие научные и специальные библиотеки практикуют и такую форму обслуживания, как заочный абонемент. Иногородние читатели зачисляются на него по заполнению гарантийного обязательства, заверенного руководителем учреждения. По заявкам требуемые книги высылаются по почте.

Все большее развитие получает изготовление фото- и ксерокопий материалов из книг, журналов, газет и их микрофильмов. Это дает огромную экономию времени и возможность иметь нужные для работы источники в их подлинном виде. В тех крупных библиотеках, где это налажено, заказы на все виды копирования могут быть сделаны при непосредственном обращении или по почте.

Органы научно-технической информации. Исходя из задач развития науки и практики, в соответствии с социально-экономической структурой нашего общества создана единая государственная система научно технической информации (ГСНТИ), включающая в себя сеть специальных учреждений, предназначенных для ее сбора, обобщения и распространения. Предназначена она для обслуживания как коллективных потребителей информации — предприятий, научно-исследовательских и проектно конструкторских организаций, — так и индивидуальных.

В основу информационной деятельности в нашей стране положен принцип централизованной обработки научных документов, позволяющий с наименьшими затратами достигнуть полного охвата мировых источников информации и наиболее квалифицированно их обобщить и систематизировать. В результате этой обработки подготавливаются различные формы информационных изданий.

Реферативные журналы (РЖ) — основное информационное издание, содержащее преимущественно рефераты, иногда аннотации и библиографические описания литературы, представляющей наибольший интерес для науки и практики.

Бюллетени сигнальной информации (БСИ) — включают в себя библиографические описания литературы, выходящей по определенным отраслям знаний. Основная их задача — оперативное информирование обо всех научных и технических новинках.

Экспресс-информация (ЭИ) — информационные издания, содержащие расширенные рефераты статей, описаний изобретений и других публикаций, позволяющих не обращаться к первоисточнику.

Аналитические обзоры (АО) — информационные издания, дающие представление о состоянии и тенденциях развития определенной области (раздела, проблемы) науки или техники.

Реферативные обзоры (РО) — в целом преследуют ту же цель, что и аналитические, но в отличие от них носят более описательный характер, без оценки содержащихся в обзоре сведений.

• Печатные библиографические карточки — содержат полное библиографическое описание источника информации.

• Аннотированные печатные библиографические карточки.

• Рефераты на картах (в том числе на перфокартах).

• Фактографическая информация на картах.

• Копии оглавлений текущих (иностранных) журналов, позволяющих составить представление о содержании номера.

Большая часть этих изданий распространяется по индивидуальной подписке. Просмотрев информационные материалы, каждый специалист может заказать ксеро-, фото- и микрофотокопии заинтересовавших его публикаций.

Непосредственную помощь специалистам в поиске информации оказывают отделы (бюро) научной информации в научно-исследовательских и проектных институтах и на предприятиях. Работа каждого из них строится с учетом информационных потребностей учреждения в целом и отдельных категорий специалистов. В соответствии с ними формируется справочно-информационний фонд (СИФ), состоящий из массива информационных документов и справочно-поискового аппарата, включающего в себя, помимо традиционных указателей и каталогов, различные картотеки: отчетов о выполненных научных исследованиях, проектной документации, авторских свидетельств и патентов, стандартов и нормалей, выпускаемых изделий, материалов, комплектующих деталей, узлов и аппаратуры, переводов, микрофильмов и т. д. Помимо справочных, во многих отделах научно-технической информации практикуется создание фактографических картотек, содержащих в себе не только указание, где можно найти те или иные материалы, но и сами эти материалы: схемы, описания, нормативы и т. д.

Каталоги и картотеки. Каталоги и картотеки — это принадлежность любой библиотеки и справочно-информационных фондов бюро научной информации. Под каталогом понимается перечень документальных

источников информации, имеющихся в фонде данной библиотеки или бюро НТИ. Картотека — перечень всех материалов, выявленных по какой-то определенной тематике. Их, как правило, несколько, и речь обычно идет не просто о каталогах и картотеках, а о системе каталогов и картотек, где они взаимосвязаны и взаимно дополняют друг друга. Создается, по крайней мере, два вида каталогов, один из которых алфавитный, а другой, группирующий литературу по содержанию, — систематический, или предметный. Чтобы правильно пользоваться каталогами, совершенно необходимо знать общие принципы их построения. Кроме того, надо постараться разобраться в их системе в той библиотеке, в которой предстоит работать. В общем, составленные по единой схеме, все они тем не менее имеют свои особенности.

Алфавитный каталог. Ведущее место в системе каталогов занимает алфавитный. По нему можно установить, какие произведения того или иного автора имеются в библиотеке, и наличие в ней определенной книги, автор или название которой известны. Карточки алфавитного каталога расставлены по первому слову библиографического описания книги: фамилии автора или названию книги, не имеющей автора. Если первые слова совпадают, карточки расставляются по второму слову, при совпадении вторых слов — по третьему и т. д. В тех случаях, когда первое совпадающее слово относится к разным типам книжного описания, на первое место ставятся описания под индивидуальным автором, затем — под коллективным, а после этого под заглавием. Карточки авторов-однофамильцев расставляются по алфавиту их инициалов. При этом сначала идут карточки без инициалов, затем с одним или двумя инициалами, а потом с именем и отчеством. По определенной схеме идет расстановка различных произведений одного автора: на первом месте — описания полного собрания сочинений, после них — собрания сочинений, затем сочинения, избранные произведения, избранные сочинения и уже после них отдельные произведения по алфавиту названий.

На разделителях алфавитного каталога указываются буквы алфавита, фамилии наиболее известных авторов и наименования учреждений.

Систематический каталог. Карточки здесь сгруппированы в логическом порядке по отдельным отраслям знаний. С его помощью можно выяснить, по каким отраслям знаний и какие именно произведения имеются в библиотеке, подобрать нужную литературу, а также установить автора и название книги, если известно ее содержание.

Последовательность расположения карточек систематического каталога всегда соответствует определенной библиографической классификации. В стране используются две такие классификации: Универсальная десятичная классификация (УДК); Библиотечно-библиографическая классификация (ББК). Для того чтобы осмысленно пользоваться систематическими каталогами, нужно иметь представление о принципах построения этих классификаций.

Универсальная десятичная классификация (УДК). В основу этой международной классификации положен десятичный принцип, в

соответствии с которым вся совокупность знаний и направлений деятельности условно разделена в таблицах УДК на десять отделов, каждый из которых подразделяется на десять подотделов, те в свою очередь на десять подразделений и т. д. При этом каждое понятие получает свой цифровой индекс. Теоретически такое деление можно производить бесконечно, образовывая индексы для более узких вопросов. Индексы, составленные по основным таблицам УДК, называются простыми. Для удобства произношения каждые три цифры в них, считая слева, отделяются от последующих точкой (например, 533.76). Помимо основных таблиц в УДК имеется еще некоторое количество «Таблиц определителей», содержащих понятия, необходимые для индексирования произведений по их дополнительным признакам. Каждый из этих признаков, выраженный соответствующей цифрой, имеет свой особый символ для его выделения в общем ряду. Универсальная десятичная система служит основой для библиографических и реферативных изданий по естественным наукам и технике для организации систематических каталогов научно-технических библиотек. Не предусматривается ее применение в каталогах универсальных библиотек и библиотек гуманитарного профиля.

Организация систематического каталога. Принятая в данном каталоге классификационная система отражается с помощью карточек-разделителей, на выступах которых пишутся индексы и названия отделов, подотделов и рубрик от общих понятий к частным в порядке детализации того или иного раздела классификации. На поле карточки-разделителя пишется перечень делений, раскрывающих содержание данного индекса. Внутри каждой рубрики карточки могут быть расставлены либо по алфавиту фамилий авторов, либо по году издания книги. В последнем случае обычно применяется обратнохронологическая расстановка, при которой впереди стоят книги, вышедшие в более поздние сроки. Справочный аппарат систематического каталога включает в себя ссылочные, отсылочные и справочные карточки и алфавитно-предметный указатель. Ссылочные карточки указывают на то, где еще находится литература по близкому или смежному вопросу. Обозначаются они словами «см. также» и пишутся на разделителе того индекса, к которому относятся. Отсылочные карточки («см.») указывают, в каком отделе находится литература по искомому вопросу.

Предметный каталог. Задачей этого каталога, так же как и систематического, является группировка литературы по ее содержанию. Однако в отличие от систематического каталога литература по тому или иному вопросу в нем объединена едиными рубриками вне зависимости от того, с каких позиций они изложены. Поэтому в предметном каталоге в одном месте находятся материалы, которые в систематическом каталоге были бы разбросаны по различным ящикам. Рубрикация предметных каталогов производится в соответствии с «рубрикаторами», имеющимися по всем отраслям знаний. Каждый вопрос, выделенный в виде рубрики, в

предметном каталоге получает словесную формулировку, составленную таким образом, чтобы основное понятие определялось первым словом.
Степень детализации рубрик зависит от количества литературы по данному вопросу и ее значимости. Если в пределах рубрики собирается большое количество работ, то для удобства пользования каталогом вводятся новые подрубрики, разбивающие литературу по дополнительным признакам. Рубрики предметного каталога расставлены, как правило, в порядке алфавита первых слов, поэтому в одном алфавитном ряду оказываются предметы, логически между собой не связанные. Вследствие этого в предметном каталоге особое значение приобретает ссылочно-справочный аппарат. Он состоит здесь из тех же элементов, что и справочный аппарат систематического каталога: ссылочных, отсылочных и справочных карточек.

Вспомогательные каталоги и картотеки. Их структура, как документальных, так и фактических, может быть самой различной. Никаких единых требований по поводу того, как они должны быть построены, не существует. Это следует учитывать, приступая к работе с ними.

Библиографические указатели. Рост научной и технической литературы делает очень важной проблему «ключа» к ней. Таким ключом служат библиографические указатели — перечни литературы, составленные по тому или иному принципу. Библиография растет сейчас такими же быстрыми темпами, как и объем печатной продукции. Только в нашей стране ежегодно выпускаются тысячи названий различных библиографий и ряд специальных периодических изданий библиографического характера. Подготовкой различного рода библиографических изданий занимаются многие организации: книжная палата, крупные библиотеки, институты научно-технической информации, многие научные учреждения и учебные заведения. Помимо тех библиографических указателей, которые выпускаются в виде отдельных изданий, библиография в той или иной форме присутствует в большинстве книг и статей. Все это определяет исключительное многообразие библиографических указателей. Они могут быть самыми различными по своим задачам, содержанию и форме. Многообразие библиографических источников делает обязательным для любого специалиста иметь представление о всех их видах, как специальных (отраслевых), так и общих. Здесь приводится характеристика только некоторых основных изданий текущей библиографии. Следить за всем тем, что выходит в стране, позволяет прежде всего комплекс «Летописей», издаваемых Книжной палатой. Сведения о книгах и брошюрах по всем отраслям знаний содержит «Книжная летопись». В основном ее выпуске, выходящем еженедельно, приводятся данные о научной, научно-популярной, производственной и художественной литературе, а также о продолжающихся изданиях типа «Трудов» и «Ученых записок». В дополнительном выпуске (издается раз в месяц) описываются ведомственные, инструктивно-производственные, нормативные, учебно-методические и информационные издания, книги, вышедшие без цены и бесплатно. Авторефераты диссертаций выходят отдельным выпуском. Наряду со

специальными библиографическими изданиями, основным содержанием которых являются сведения о различных произведениях печати, информацию о литературе дают многие книги и периодические издания. Эта информация составляет их библиографический аппарат, именуемый прикнижной (пристатейной) библиографией. Она рассматривается как составная часть библиографии определенной области или научной дисциплины.

Работа с книгой. Умение работать с книгой — это умение правильно оценить произведение, быстро разобраться в его структуре, взять и зафиксировать в удобной форме все, что в нем оказалось ценным и нужным. Работа с книгой — процесс сложный. Обусловлено это прежде всего тем, что чтение научно-литературных произведений всегда связано с необходимостью усвоения каких-то новых понятий. Сложно это и потому, что практически каждая книга оригинальна по своей композиции и требуются определенные усилия, чтобы понять ход мысли автора. Умением работать с литературой обладают далеко не все. Наиболее частые ошибки — отсутствие должной целенаправленности в чтении, недостаточное использование справочного аппарата, нерациональная форма записи прочитанного. Все это снижает эффективность умственного труда, приводит к непроизводительным тратам времени.

Техника чтения. Одной из особенностей чтения специальной литературы является то, что оно протекает в определенной последовательности: сначала предварительное ознакомление с книгой и только после этого ее тщательная проработка. Предварительное ознакомление с книгой. Ценность каждого научного произведения колеблется в весьма широких пределах. Далеко не любую книгу следует читать полностью, в ряде случаев могут быть нужны лишь отдельные ее части. Поэтому для экономии времени и с тем, чтобы определить цели и подходы к чтению книги, рекомендуется начинать с предварительного ознакомления с ней в целях общего представления о произведении и его структуре, организации справочно-библиографического аппарата. При этом необходимо принять во внимание все те элементы книги, которые дают возможность оценить ее должным образом. Делать это лучше всего в следующей последовательности: заглавие; автор; издательство (или учреждение, выпустившее книгу); время издания; аннотация; оглавление; авторское или издательское предисловие; справочно-библиографический аппарат (указатели, приложения, перечень сокращений и т. п.). Предварительное ознакомление призвано дать четкий ответ на вопрос о целесообразности дальнейшего чтения книги, в каких отношениях она представляет интерес и какими должны быть способы ее проработки, включая сюда наиболее подходящую для данного случая форму записей.

Чтение книги. Существуют два подхода к чтению научно-литературного произведения: беглый просмотр его содержания и тщательная проработка произведения в целом или отдельных его частей. Беглый просмотр содержания книги необходим в тех случаях, когда предварительное

ознакомление с ней не дает возможности определить, насколько она представляет интерес, и для того, чтобы быть в курсе имеющейся литературы по интересующему вопросу. Бывает и так, что становится ясно — в работе содержатся нужные материалы, и требуется ее полный просмотр, чтобы их найти. Беглый просмотр книги — по существу «поисковое» чтение. Тщательная проработка текста (иногда его называют «сплошным чтением») — это усвоение его в такой степени, в какой необходимо по характеру выполняемой работы. Следует отметить, что прочитать текст — еще не значит усвоить его. Текст надо обязательно понять, расшифровать, осмыслить. Вопрос об усвоении содержания книги часто понимают не совсем правильно. Многие считают, что главное — запомнить содержание прочитанного. Между тем усвоение и запоминание — совершенно разные понятия. Усвоить прочитанное — значит понять все так глубоко и продумать так серьезно, чтобы мысли автора, объединяясь с собственными мыслями, превратились бы в единую систему знаний по данному вопросу. Само собой разумеется, что цель эта тем легче достигается, чем выше уровень подготовки специалиста и чем больше он знаком с тематикой изучаемой литературы. Нужно, однако, помнить и о другом: чтение специальной литературы — это и есть процесс накопления и расширения знаний. Значит, вопрос стоит не о том, какой уровень знаний требуется, чтобы приступить к чтению, а каким образом можно преодолеть те трудности, с которыми приходится сталкиваться в процессе чтения. Рекомендации обычно сводятся к тому, что читать нужно «помедленнее» и «повнимательнее». Сами по себе они бесспорно правильны, но это далеко не основной ключ к тем материалам, в усвоении которых могут встречаться затруднения. Следует попытаться представить возможные причины этих затруднений. Как показывает практика, чаще всего они возникают, если в процессе чтения не всегда удается разобраться в логической структуре материала книги. Это не просто, так как она бывает различной не только в каждой из книг, но может меняться от главы к главе и от страницы к странице в одной и той же работе. Основные мысли любого сочинения можно понять и усвоить лишь в том случае, если в полной мере уяснена схема его построения. Необходимо проследить последовательность хода мыслей автора, логику его доказательств, установить связи между отдельными положениями, выделить то главное, что приводится для их обоснования, отделить основные положения от иллюстраций и примеров. Это уже не просто чтение, а глубокий и детальный анализ текста. И именно при таком подходе становится возможным понять его и по-настоящему усвоить. Проведение такого анализа значительно облегчается, если все это попытаться изобразить на бумаге в текстовой форме, выписывая главные положения, или в форме графической схемы, на которой можно наиболее наглядно представить всю картину логических связей изучаемого явления. Усвоению тех или иных построений автора способствует также система подчеркиваний и выделений в тексте книги и нумерации отдельных положений. В данном случае речь идет о книгах только из личной библиотеки. При работе с однотипными

текстами усвоению способствует использование заранее составленных перечней, содержащих вопросы, которые следует уяснить в процессе чтения. Очень часто «смысловой тупик» обусловлен не структурой текста произведения, а его терминологическими особенностями. В процессе чтения могут попадаться непонятные слова, многие термины используются в различных контекстах неоднозначно, не всегда ясны различного рода сокращения. Все это затрудняет чтение, может приводить к искажению смысла текста. Необходимо приучить себя к обязательному уточнению всех тех терминов и понятий, по поводу которых возникают хоть какие-либо сомнения. Очень важно для этого всегда иметь под рукой необходимые справочники и словари. Часто говорят о необходимости критического и творческого восприятия литературных данных. Думается, что ни то ни другое не может рассматриваться в качестве практических рекомендаций. Это должно прийти само по себе по мере накопления опыта. На определенном профессиональном уровне могут возникнуть те или иные несогласия со взглядами отдельных авторов, появятся аргументированные доводы против каких-то их положений и возможность сопоставления со своими взглядами. Тем более это относится и к творческому подходу. Конечно же, чтение — это стимуляция идей. Внимательное ознакомление с любым текстом должно вызвать какие-то мысли, соображения, даже гипотезы, отвечающие собственным взглядам на вещи. Но все эти вопросы находятся вне того, что касается техники чтения.

Записи при чтении. Чтение научной и специальной литературы, как правило, должно сопровождаться ведением записей. Это непременное условие, а не вопрос вкуса или привычки. Необходимость ведения записей в процессе чтения неотделима от самого существа использования книги в работе, будь то наука или практика. Не случайно всегда говорится о необходимости чтения «с карандашом в руке». Ведение записей способствует лучшему усвоению прочитанного, дает возможность сохранить нужные материалы в удобном для использования виде, помогает закрепить их в памяти, позволяет сократить время на поиск при повторном обращении к данному источнику. Облегчает работу не каждая запись. Нередко можно наблюдать, как выписывание тех или иных данных из книг превращается в совершенно бессмысленное занятие, отнимающее время. Рациональными записи могут быть лишь в том случае, если соблюдены некоторые общие требования к их ведению и правильно выбрана их форма. В качестве первого требования следует повторить то, что уже было сказано в отношении обязательности их ведения. Иногда считают, что записями сопровождается чтение книг, только наиболее важных для работы. Это неверно. Нужно взять за правило вести записи при чтении любой специальной литературы. Ведение записей — обязательный элемент работы над книгой, неотделимый от процесса чтения, и поэтому их нельзя откладывать «на потом». Следует вырабатывать в себе умение читать и вести записи в любых условиях. Особенно важно быть дисциплинированным в отношении немедленной и обязательной записи оригинальных мыслей, появляющихся в процессе

чтения. Надо помнить, что они являются результатом ассоциаций, которые в других условиях не возникнут. Записи должны быть предельно полными. Это, как правило, занимает гораздо меньше времени, чем повторное обращение к книге. Необходимо предвидеть и будущую потребность в материале, имеющемся в книге, и в пределах разумного взять из нее все, что только возможно. Существует ряд практических приемов, направленных на то, чтобы записи в процессе чтения занимали бы как можно меньше времени, и на то, чтобы ими в дальнейшем можно было легко пользоваться. Для этого прежде всего нужно стремиться к лаконизму в изложении и к использованию всякого рода сокращений. Большую экономию времени дает также применение условных знаков-символов (например, математических: равно, больше, меньше и т. д.). Можно здесь вводить и любые свои знаки. Стремление к лаконизму должно, разумеется, иметь определенную меру. Нужно помнить, что всякого рода крючки и закорючки, равно как и «телеграфный язык», становятся со временем столь же трудно читаемыми, как письменность майя. Иногда бывает легче второй раз прочесть книгу, чем разобраться в небрежных записях. Важными требованиями являются также наглядность и обозримость записей и такое их расположение, которое бы помогало уяснить логические связи и иерархию понятий. Сделать это возможно с помощью системы заголовков, подзаголовков и ключевых слов, а также путем расчленения текста за счет абзацных отступов, подчеркиваний, нумерации отдельных понятий и т. д. К общим моментам техники записей относится также вопрос о форме. Выбор здесь идет между так называемой «книжной» формой (использованием материалов в сброшюрованном виде) и «карточной» формой. Подчас можно услышать, что это дело вкуса. В действительности это совсем не так. Несомненные преимущества имеет карточная форма как лучший способ систематизации любых материалов. Практическая рекомендация — вести записи только на одной стороне листа. При этом ускоряется их поиск и систематизация, становится возможным производить любые вставки в текст, использовать записи при работе над докладами и рукописями научно-литературных произведений. В последнем случае целесообразно бывает все записи иметь в двух экземплярах: один остается для хранения, а второй идет на «разрез» для подготовки статей, брошюр, книг и т. д. Постоянный вопрос, встающий в разговоре о записях при чтении, — когда их делать. Единого ответа здесь быть не может: все зависит от вида записей. Насколько различны цели и условия чтения научной, учебной и специальной литературы, настолько могут быть различными и виды тех записей, которыми это чтение сопровождается. Каждый из перечисленных видов записей в значительной степени отличается один от другого и по своему содержанию, и по сложности: одни содержат «сжатую» информацию, в других она дается в развернутом виде — или лишь «ключ» для ее поиска; в одних — те или иные сведения в том самом виде, в котором они были в книге, в других — результат их аналитической переработки и т. д. Далеко не безразлично поэтому, какой вид записи будет использован в каждом конкретном случае. Надо стараться сделать так, чтобы

он в полной мере соответствовал характеру работы с книгой. В отношении каждого отдельного вида записей имеется ряд правил и практических приемов их ведения, направленных на то, чтобы они возможно полнее отвечали своему назначению. Прежде всего, о группе записей, не связанных с необходимостью аналитической переработки текста.

Выписки. По своему характеру они настолько разнообразны, что, казалось бы, между ними ничего не может быть общего. Тем не менее и в отношении их следует сказать об определенных требованиях. Прежде всего — особая тщательность записей. Любая небрежность в выписке данных из книги обычно оборачивается значительными потерями времени на их уточнение или повторный поиск. Иногда пытаются давать рекомендации по поводу того, сколько их надо делать, и предостерегают против большого количества. Выписывают все те данные, которые представляют интерес для работы. Судить о том, сколько их нужно, может только сам специалист, и нелепо придумывать какие-то искусственные ограничения. Исключение составляют лишь текстовые выписки-цитаты. Здесь, действительно, уместно предостеречь от излишнего стремления выписывать все дословно. Часто бывает, что та или иная мысль без всякого ущерба может быть передана своими словами. Дословно выписывать следует лишь то, что обязательно должно быть передано именно в той форме, в какой это было у автора книги. В некоторых случаях бывает целесообразным использование так называемых формализованных выписок. Листы или карточки для выписок должны быть заранее разграфлены, и все данные выписываются на отведенные для них места (строки, графы). Использование таких заранее подготовленных форм ускоряет выборку из книги нужных данных. Имея в перспективе ту или иную форму копирования прочитанного материала — фотографирование, микрофильмирование, ксерокопирование и т. д., следует сразу же по ходу чтения готовить перечень страниц (фрагментов текста), подлежащих копированию. Примером, облегчающим работу с книгой, является использование закладок с надписями. В процессе чтения они позволяют быстро находить нужные разделы — оглавление, всякого рода указатели, перечни сокращений, карты, таблицы и т. д. Кроме того, закладками могут быть обозначены все те места в книге, которые понадобятся в дальнейшем. При чтении научной, учебной и специальной литературы довольно распространена практика всякого рода пометок и выделений в книгах. Делаются они на полях или прямо в тексте, выделяя то главное, на что надо обратить внимание или вернуться еще раз; те или иные непонятные места, положения, с которыми нельзя согласиться; удачные или малоудачные выражения, цитаты, подлежащие выписке или копированию. Систему эту следует всячески рекомендовать, так как использование пометок и выделений позволяет значительно сократить время работы с книгой, облегчая ориентировку в ней и усвоение ее содержания. Какими эти пометки и выделения должны быть по форме, каждый решает сам. Использовать для этого можно различные линии, символы, цифры. Главное, чтобы избранная система была достаточно стройной и стабильной. Выделения в книге могут

касаться не только текста, но и графики. Раскрашивание схем и рисунков, особенно сложных и труднопонимаемых, во многих случаях делает их более наглядными и значительно удобными. В тех случаях, когда в книге нужно выделить какие-то части текста, а пометки в ней делать нельзя, целесообразно пользоваться так называемой «системой чистых листов»: между страницами вкладываются чистые листы бумаги, на которых делаются пометки на уровне интересуемого текста. При необходимости возле этих пометок могут быть краткие пояснения. Листы с пометками нумеруются в соответствии со страницами книги. В дальнейшем, приложив такой лист к тексту, можно сразу же найти нужные места. Результатом проработки книги может быть еще и такой вид записи, как перечень страниц, содержащих материалы по определенным вопросам. В дополнение к номерам страниц в нем целесообразно также указывать, в каких абзацах находятся нужные материалы или расстояние до них от верха или низа страницы в сантиметрах. Вторая группа записей — аналитическая. Простейшими из них являются оценочные записи на библиографических карточках личной картотеки. Этим фиксируется факт, что данная книга была просмотрена или проработана и о ней сложилось определенное мнение в двух-трех словах, из которых станет ясно, следует ли еще раз обращаться к данной книге и что в ней можно найти. Более сложный вид записи — составление плана книги, отражающего ее содержание и структуру. По существу планом любой книги является ее оглавление, но как форма записи при чтении он должен быть несколько подробнее оглавления. Кроме общего плана книги, могут быть еще планы отдельных ее частей, показывающие ход мыслей автора, логику его доказательств и обоснований. Пользуясь планом, можно легко восстановить в памяти содержание любого произведения. Составление плана может рассматриваться также в качестве предварительного этапа работы перед тем, как перейти к более сложным видам записей — тезисам и конспекту.

Тезис — греческое слово, означающее «положение». Таким образом, тезисы — это основные положения книги. Для того чтобы их составить, требуется достаточно полное усвоение содержания произведения, четкое представление о его основной идее и главных положениях, утверждаемых автором. Располагать тезисы следует в логической последовательности, в которой наиболее правильно изложены основные идеи книги. Это не всегда совпадает с последовательностью изложения материала. В самих тезисах, как правило, не должно содержаться фактических данных. Однако иногда бывает целесообразно, выделяя от текста тезисы, дать краткий перечень фактов, которые приводятся автором в обоснование своих положений. В тех случаях, когда в книге наряду с фактическим материалом наличествуют разного рода рассуждения, нужно каким-то образом отделить их друг от друга, чтобы при ознакомлении с каждым из тезисов видеть, обоснован ли он фактами или имеются только общие рассуждения.

Одним из наиболее часто практикуемых видов записей является конспект, т. е. краткое изложение прочитанного. В буквальном смысле слово «конспект» означает «обзор». По существу, его и составлять надо как обзор,

содержащий основные мысли произведения, без подробностей и второстепенных деталей. Слишком подробный конспект — уже не конспект. По своей структуре он чаще всего соответствует плану книги. Помимо обычного текстового конспекта, в ряде случаев целесообразно использовать такой конспект, где все записи вносятся в заранее подготовленные таблицы (формализованный конспект). Это удобно при конспектировании материалов, когда перечень характеристик описываемых предметов или явлений более или менее постоянен.

Табличная форма конспекта может быть применена также при подготовке единого конспекта по нескольким источникам, особенно если есть необходимость сравнения отдельных данных. Разновидностью формализованного конспекта является запись, составленная в форме ответов на заранее подготовленные вопросы, обеспечивающие исчерпывающие характеристики однотипных предметов или явлений. Конспект такого типа также очень удобен, когда предполагается сопоставление тех или иных характеристик. Еще одна форма конспекта — графическая. Суть ее в том, что элементы конспектируемой работы располагаются в таком виде, при котором видна иерархия понятий и взаимосвязь между ними. На первой горизонтали находится формулировка темы, на второй показано, какие основные положения в нее входят. Эти положения имеют свои подразделения и т. д. По каждой работе может быть не один, а несколько графических конспектов, отображающих книгу в целом и отдельные ее части. Ведение графического конспекта — наиболее совершенный способ изображения внутренней структуры книги, а сам этот процесс помогает усвоению ее содержания.

Словарь терминов и понятий. Не случайно относится к группе записей, связанных с необходимостью аналитической переработки текста. Составить для себя такой словарь и дать точное толкование всем специальным терминам и понятиям — дело далеко не механическое. Очень часто оно связано с необходимостью длительного поиска в справочниках и руководствах. Ведение словаря терминов и понятий обычно связывают с процессом обучения чтению профессиональной литературы. Это неверно. При той сложности, которая сейчас характерна для специальной терминологии, при отсутствии единства в ней, при частых изменениях, также при обилии всевозможных сокращений вести подобный словарь совершенно обязательно для специалиста любого уровня подготовки. Он может значительно облегчить работу с источниками информации.

Тема 6. Патентные исследования. Техническое и интеллектуальное творчество и его правовая охрана

6.1 Изобретения, полезные модели, промышленные образцы и их правовая охрана

Как известно, научно-технический прогресс является движущей силой современного общества. Одними из основных составляющих научно-технического прогресса являются такие понятия, как "изобретения", "полезные модели", "промышленные образцы". Все хорошо понимают, что

наличие новых устройств и изделий, защищенных патентами, напрямую связано с экономической прибылью предприятия-патентообладателя, а также иногда сама торговля патентами приносит сверхприбыли.

Таким образом, анализ всей ситуации, сопутствующей появлению и функционированию новшества, способствует созданию новых изобретений и правильному позиционированию уже имеющихся разработок. Согласно Российскому законодательству осуществление государственной политики в сфере правовой охраны изобретений, полезных моделей и промышленных образцов возлагается на федеральный орган исполнительной власти по интеллектуальной собственности.

Под патентом понимают документ, выдаваемый компетентным государственным органом на определенный срок и удостоверяющий авторство и исключительное право на изобретение, наделяющий владельца титулом собственника на изобретение. Патент защищает владельца от внутренних и зарубежных конкурентов и действует на территории той страны, где он выдан. Обычно патент подкрепляется регистрацией товарного знака или промышленного образца.

Рассмотрим правовую охрану изобретения, полезной модели, промышленного образца и условия их патентоспособности. Права на изобретение, полезную модель, промышленный образец охраняются законом и подтверждаются соответственно патентом на изобретение, патентом на полезную модель и патентом на промышленный образец. Патент удостоверяет приоритет, авторство изобретения, полезной модели или промышленного образца и исключительное право на изобретение, полезную модель или промышленный образец. Патент на изобретение действует до истечения двадцати лет с даты подачи заявки в федеральный орган исполнительной власти по интеллектуальной собственности. Патент на полезную модель действует до истечения пяти лет с даты подачи, на промышленный образец - до истечения десяти лет. Патентоспособность – это наличие у технического решения всех критериев изобретения в соответствии с законодательством каждой отдельно взятой страны.

В соответствии с Патентным законом РФ в качестве изобретения охраняется техническое решение в любой области, относящееся к продукту (в частности, устройству, веществу, штамму микроорганизма, культуре клеток растений или животных) или способу (процессу осуществления действий над материальным объектом с помощью материальных средств). Изобретению предоставляется правовая охрана, если оно является новым, имеет изобретательский уровень и промышленно применимо. Изобретение является новым, если оно не известно из уровня техники. Оно имеет изобретательский уровень, если для специалиста явным образом не следует из уровня техники. Уровень техники включает любые сведения, ставшие общедоступными в мире до даты приоритета изобретения. Изобретение является промышленно применимым, если оно может быть использовано в промышленности, сельском хозяйстве, здравоохранении и других отраслях деятельности.

Не считаются изобретениями:

• открытия, а также научные теории и математические методы;

• решения, касающиеся только внешнего вида изделий и направленные на удовлетворение эстетических потребностей;

• правила и методы игр, интеллектуальной или хозяйственной деятельности;

• программы для электронных вычислительных машин;

• решения, заключающиеся только в представлении информации.

Не признаются патентоспособными:

• сорта растений, породы животных;

• топологии интегральных микросхем;

• решения, противоречащие общественным интересам, принципам гуманности и морали.

В качестве полезной модели охраняется техническое решение, относящееся к устройству. Полезная модель признается соответствующей условиям патентоспособности, если она является новой и промышленно применимой. Новизна определяется совокупностью ее существенных признаков, не известных из уровня техники. Полезная модель является промышленно применимой, если она может быть использована в промышленности, сельском хозяйстве, здравоохранении и других отраслях деятельности.

В качестве полезных моделей правовая охрана не предоставляется:

• решениям, касающимся только внешнего вида изделий и направленным на удовлетворение эстетических потребностей;

• топологиям интегральных микросхем;

• решениям, противоречащим общественным интересам, принципам гуманности и морали.

В качестве промышленного образца охраняется художественно-конструкторское решение изделия промышленного или кустарно-ремесленного производства, определяющее его внешний вид. Промышленный образец должен обладать новизной и оригинальностью. Он признается новым, если совокупность его существенных признаков, нашедших отражение на изображениях изделия и приведенных в перечне существенных признаков промышленного образца, не известна из сведений, ставших общедоступными в мире до даты приоритета промышленного образца. Промышленный образец является оригинальным, если его существенные признаки обусловливают творческий характер особенностей изделия.

К существенным признакам промышленного образца относятся признаки, определяющие эстетические и эргономические особенности внешнего вида изделия, в частности форма, конфигурация, орнамент и сочетание цветов.

Не признаются патентоспособными промышленными образцами решения:

• обусловленные исключительно технической функцией изделия;

• объектов архитектуры (кроме малых архитектурных форм), промышленных, гидротехнических и других стационарных сооружений;

• объектов неустойчивой формы из жидких, газообразных, сыпучих или им подобных веществ;

• изделий, противоречащих общественным интересам, принципам гуманности и морали.

Автором изобретения (полезной модели, промышленного образца) является физическое лицо, творческим трудом которого они созданы. Если в создании изобретения, полезной модели или промышленного образца участвовало несколько физических лиц, все они считаются его авторами. Порядок пользования правами, принадлежащими авторам, определяется соглашением между ними. Не признаются авторами физические лица, не внесшие личного творческого вклада в создание объекта промышленной собственности, оказавшие автору (авторам) только техническую, организационную или материальную помощь либо только способствовавшие оформлению прав на него и его использованию. Право авторства является неотчуждаемым личным правом и охраняется бессрочно.

Итак, согласно российскому законодательству патент выдается:

• автору изобретения, полезной модели или промышленного образца;

• работодателю в случаях, предусмотренных Патентным законом РФ.

Патентообладатель - юридическое и (или) физическое лицо которому принадлежит исключительное право на использование охраняемых патентом изобретения.

Право на получение патента на изобретение (полезную модель, промышленный образец), созданные работником в связи с выполнением своих трудовых обязанностей или конкретного задания работодателя (служебное изобретение, служебная полезная модель, служебный промышленный образец), принадлежит работодателю, если договором между ним и работником (автором) не предусмотрено иное. Правительство Российской Федерации вправе устанавливать минимальные ставки вознаграждения за служебные изобретения, служебные полезные модели, служебные промышленные образцы.

Право на получение патента на изобретение, полезную модель или промышленный образец, созданные при выполнении работ по государственному контракту для федеральных государственных нужд или нужд субъекта Российской Федерации, принадлежит исполнителю (подрядчику), если государственным контрактом не установлено, что это право принадлежит Российской Федерации или субъекту Российской Федерации, от имени которых выступает государственный заказчик.

Патентообладателю принадлежит исключительное право на изобретение, полезную модель или промышленный образец. Никто не вправе использовать запатентованные изобретение, полезную модель или

промышленный образец без разрешения патентообладателя, в том числе совершать следующие действия:

• ввоз на территорию Российской Федерации, изготовление, применение, предложение о продаже, продажу, иное введение в гражданский оборот или хранение для этих целей продукта, в котором использованы запатентованные изобретение, полезная модель, или изделия, в котором использован запатентованный промышленный образец;

• совершение действий, указанных в выше в отношении продукта, полученного непосредственно запатентованным способом.

• совершение действий, указанных выше в отношении устройства, при функционировании (эксплуатации) которого в соответствии с его назначением автоматически осуществляется запатентованный способ;

• осуществление способа, в котором используется запатентованное изобретение.

Порядок использования изобретения, полезной модели или промышленного образца в случае, если патент принадлежит нескольким лицам, определяется договором между ними. При отсутствии такого договора каждый из патентообладателей может использовать запатентованные изобретение, полезную модель или промышленный образец по своему усмотрению, но не вправе предоставить лицензию или передать исключительное право (уступить патент) другому лицу без согласия остальных патентообладателей.

Запатентованные изобретение или полезная модель признаются использованными в продукте или способе, если продукт содержит, а в способе использован каждый признак изобретения или полезной модели, приведенный в независимом пункте формулы изобретения или полезной модели, либо признак, эквивалентный ему и ставший известным в качестве такового в данной области техники.

Запатентованный промышленный образец признается использованным в изделии, если такое изделие содержит все существенные признаки промышленного образца, нашедшие отражение на изображениях изделия и приведенные в перечне существенных признаков промышленного образца.

В случае, если при использовании запатентованных изобретения или полезной модели используются также все признаки, приведенные в независимом пункте формулы других запатентованных изобретения или полезной модели, а при использовании запатентованного промышленного образца - все признаки, приведенные в перечне существенных признаков другого запатентованного промышленного образца, другие запатентованные изобретение, полезная модель, промышленный образец также признаются использованными.

Если запатентованные изобретение или промышленный образец не используются либо недостаточно используются патентообладателем и лицами, которым переданы права на них, в течение четырех лет с даты выдачи патента, а запатентованная полезная модель в течение трех лет с даты

выдачи патента, что приводит к недостаточному предложению соответствующих товаров или услуг на товарном рынке или рынке услуг, любое лицо, желающее и готовое использовать запатентованные изобретение, полезную модель или промышленный образец, при отказе патентообладателя от заключения с этим лицом лицензионного договора на условиях, соответствующих установившейся практике, имеет право обратиться в суд с иском к патентообладателю о предоставлении принудительной неисключительной лицензии на использование на территории Российской Федерации таких изобретения, полезной модели или промышленного образца.

Патентообладатель может передать исключительное право на изобретение, полезную модель, промышленный образец (уступить патент) любому физическому или юридическому лицу. Договор о передаче исключительного права (уступке патента) подлежит регистрации в федеральном органе исполнительной власти по интеллектуальной собственности и без такой регистрации считается недействительным.

Патент на изобретение, полезную модель, промышленный образец и право на его получение переходят по наследству. Для получения патента автору изобретения или лицу, обладающему правом на получение патента необходимо подать заявку в федеральный орган исполнительной власти по интеллектуальной собственности.

Заявка на выдачу патента на изобретение (далее - заявка на изобретение) должна относиться к одному изобретению или группе изобретений, связанных между собой настолько, что они образуют единый изобретательский замысел (требование единства изобретения).

Заявка на изобретение должна содержать:

• заявление о выдаче патента с указанием автора (авторов) изобретения и лица (лиц), на имя которого (которых) испрашивается патент, а также их местожительства или местонахождения;

• описание изобретения, раскрывающее его с полнотой, достаточной для осуществления;

• формулу изобретения, выражающую его сущность и полностью основанную на описании;

• чертежи и иные материалы, если они необходимы для понимания сущности изобретения;

• реферат.

К заявке на изобретение прилагается документ, подтверждающий уплату патентной пошлины в установленном размере. Заявка на выдачу патента на полезную модель должна относиться к одной полезной модели или группе полезных моделей, связанных между собой настолько, что они образуют единый творческий замысел.

Заявка на полезную модель должна содержать:

• заявление о выдаче патента с указанием автора (авторов) полезной модели и лица (лиц), на имя которого (которых) испрашивается патент, а также их местожительства или местонахождения;

• описание полезной модели, раскрывающее ее с полнотой, достаточной для осуществления;

• формулу полезной модели, выражающую ее сущность и полностью основанную на описании;

• чертежи, если они необходимы для понимания сущности полезной модели;

• реферат.

К заявке на полезную модель прилагается документ, подтверждающий уплату патентной пошлины в установленном размере, или документ, подтверждающий основания для освобождения от уплаты патентной пошлины, либо уменьшения ее размера, либо отсрочки ее уплаты.

Заявка на выдачу патента на промышленный образец должна относиться к одному промышленному образцу или группе промышленных образцов, связанных между собой настолько, что они образуют единый творческий замысел. Заявка на промышленный образец должна содержать следующую научно-техническую информацию:

• заявление о выдаче патента с указанием автора или авторов промышленного образца и лица или лиц, на имя которых испрашивается патент, а также их местожительства или местонахождения;

• комплект изображений изделия, дающих полное детальное представление о внешнем виде изделия;

• чертеж общего вида изделия, эргономическую схему, конфекционную карту, если они необходимы для раскрытия сущности промышленного образца;

• описание промышленного образца;

• перечень существенных признаков промышленного образца.

К заявке на промышленный образец прилагается документ, подтверждающий уплату патентной пошлины в установленном размере.

По заявке на изобретение, поступившей в федеральный орган исполнительной власти по интеллектуальной собственности, проводится формальная экспертиза, в процессе которой проверяются наличие документов, предусмотренных Патентным законом и экспертиза заявки на изобретение по существу. Она включает в себя: информационный поиск в отношении заявленного изобретения для определения уровня техники и проверку соответствия заявленного изобретения условиям патентоспособности.

За нарушение настоящего Патентного закона РФ наступает гражданско-правовая, административная или уголовная ответственность в соответствии с законодательством Российской Федерации.

6.2. Особенности патентных исследований

Количество запатентованных объектов практически напрямую связано с получаемой прибылью и, в конечном счете, с процветанием предприятия.

Патентные исследования являются тем самым инструментом, с помощью которого менеджеры высшего звена предприятия могут оценивать текущую ситуацию и прогнозировать развитие ситуации вокруг научно-технических новшеств.

Проведение патентных исследований, с одной стороны, позволяет реально оценивать патентоспособность разрабатываемых объектов техники, и с другой стороны, предотвратить нарушение чужих прав, сохранив патентную чистоту объекта.

Патентные исследования проводятся высокопрофессиональными специалистами-патентоведами в тесном взаимодействии с инженерно техническим персоналом фирм-разработчиков объекта техники. Именно патентные исследования являются мощным маркетинговым инструментом, способным в условиях современного рынка периода информационной революции предотвратить повторение уже созданных независимо другими разработчиками новшеств, а также направить творческую активность изобретателей на создание действительно совершенно новых объектов. Особенно это важно для предприятий, работающих на рынках высоких технологий и ориентированных на зарубежные рынки. Немаловажным аспектом является исследование возможности свободного использования изобретений, что важно для таких отраслей промышленности, в развитии которых необходим мощный рывок вперед, и которые пока не в состоянии самостоятельно конкурировать с ведущими мировыми производителями в своей области рынка.

Итак, под патентными исследованиями понимают исследования технического уровня и тенденций развития объектов техники (ОТ), их патентоспособности и патентной чистоты на основе патентной информации и патентно-ассоциируемой литературы.

Патентная чистота - юридическое свойство технического объекта, заключающееся в том, что он может быть свободно использован в определенной стране без опасности нарушения действующих на территории этой страны патентов, принадлежащих третьим лицам.

В перечень работ по патентным исследованиям входят:

1. Исследование технического уровня объектов техники;
2. Анализ научно-технической деятельности ведущих фирм;
3. Анализ тенденций развития данного вида техники;
4. Анализ патентно-лицензионной деятельности ведущих фирм на мировом рынке данного вида техники
5. Технико-экономический анализ технических решений / изобретений, отвечающих задачам разработки;
6. Исследование новизны разработанного объекта техники и его составных частей;
7. Исследования патентной чистоты объекта и его составных частей;
8. Основание целесообразности правовой защиты объекта промышленной собственности.

Все виды работ по патентным исследованиям по содержательной направленности объединяются в 4 группы:

1. Анализ тенденций и перспектив развития техники, исследование мирового и национального научно-технического уровня в соответствующих отраслях техники;

2. Исследование новизны технических решений, заявляемых или не заявляемых в качестве изобретений и промышленных образцов;

3. Исследование патентной чистоты объекта техники;

4. Исследование патентно-лицензионной ситуации при определении целесообразности патентования и продажи лицензий, а так же операций по экспорту.

Патентные исследования позволяют на основе анализа описания изобретений определить требования потребителей к продукции данного вида, выявить фирмы конкуренты и фирмы - потенциальные партнеры. Важную роль играют патентные исследования в рекламе конкурентоспособности продукции формирования стоимостных факторов. Поэтому патентные исследования играют важную роль в процессе разработки и постановки продукции на производство. Результаты патентных исследований оформляются в виде отчета, справки о поиске. Порядок проведения патентных исследований определяет ГОСТ 15.011-82.

Последовательность работы при проведении патентных исследований

Необходимо сформулировать тему поиска. Тема поиска может не совпадать с темой дипломного или курсового проекта, и её необходимо правильно сформулировать. Точная формулировка позволит правильно определить поисковое поле. Поиск начинается с Алфавитно-предметного указателя МПК. Международная патентная классификация, являясь средством для единообразного в международном масштабе классифицирования патентных документов, представляет собой эффективный инструмент для патентных ведомств и других потребителей, осуществляющих поиск патентных документов с целью установления новизны и оценки вклада изобретателя в заявленное техническое решение (включая оценку технической прогрессивности и полезного результата или полезности).

Важным назначением МПК, кроме того, является:

• служить инструментом для упорядоченного хранения патентных документов, что облегчает доступ к содержащейся в них технической и правовой информации;

• быть основой для избирательного распределения информации среди потребителей патентной информации;

• быть основой для определения уровня техники в отдельных областях;

• быть основой для получения статистических данных в области промышленной собственности, что в свою очередь позволит определять уровень развития различных отраслей техники.

Структура индекса МПК

Международная патентная классификация изобретений подразделяет всю совокупность изобретений на 8 разделов, обозначенных буквами латинского алфавита от А до Н, каждый из которых делится на классы (01, 02, 03, и.т.д.), которые в свою очередь разделены на подклассы (согласные буквы латинского алфавита), а те на группы и подгруппы:

Поиск и отбор патентных документов

После определения классификационной рубрики МПК, патентный поиск целесообразно начать с просмотра описаний изобретений, в патентном отделе описания изобретений разложены по соответствующим папкам (перечень папок приведен на страничке отдела). Поиск по описаниям изобретений позволяет определить библиографические данные, описание изобретения в статике и динамике, формулу изобретения. Поиск можно провести по официальному бюллетеню “Изобретения” или “Полезные модели”. Каждый номер бюллетеня содержит систематический и нумерационный указатели, которые значительно сокращают время поиска. Поиск по бюллетеню позволяет определить библиографические данные по изобретению и формулу изобретения. Поиск можно также провести по реферативному журналу “Изобретения стран мира”. В журнале опубликованы патенты, полученные в США, Великобритании, Франции, Германии, Японии, ЕПВ, Реферативный журнал имеет систематический и нумерационный указатель. Поиск по реферативному журналу позволяет определить библиографические данные, реферат изобретения, небольшой чертеж (схему). По окончании поиска необходимо заполнить итоговый документ - “Справка о поиске”.

6.3. Интеллектуальное собственность и ее защита

Интеллектуальная собственность – это собственность на результаты интеллектуальной деятельности, интеллектуальный продукт, входящий в совокупность объектов авторского и изобретательского права.

Особенности изобретательского права мы уже рассмотрели выше, поэтому кратко остановимся на некоторых положениях, касающихся авторского права. Согласно Закону РФ «Об авторских и смежных правах» № 5351-1 авторское право распространяется на:

- произведения, обнародованные либо необнародованные, но находящиеся в какой-либо объективной форме на территории Российской Федерации, независимо от гражданства авторов и их правопреемников;
- произведения, обнародованные либо необнародованные, но находящиеся в какой-либо объективной форме за пределами Российской Федерации, и признается за авторами - гражданами Российской Федерации и их правопреемниками;
- произведения, обнародованные либо необнародованные, но находящиеся в какой-либо объективной форме за пределами Российской Федерации, и признается за авторами (их правопреемниками) - гражданами других государств в соответствии с международными договорами Российской Федерации.

Общие положения

• авторское право распространяется на произведения науки, литературы и искусства, являющиеся результатом творческой деятельности, независимо от назначения и достоинства произведения, а также от способа его выражения;

• авторское право распространяется как на обнародованные произведения, так и на необнародованные произведения, существующие в какой-либо объективной форме:

письменной (рукопись, машинопись, нотная запись и так далее); устной (публичное произнесение, публичное исполнение и так далее); звуко- или видеозаписи (механической, магнитной, цифровой, оптической и так далее);

изображения (рисунок, эскиз, картина, план, чертеж, кино-, теле-, видео- или фотокадр и так далее); объемно-пространственной (скульптура, модель, макет, сооружение и так далее); в других формах;

Авторское право не распространяется на идеи, методы, процессы, системы, способы, концепции, принципы, открытия, факты.

Авторское право на произведение не связано с правом собственности на материальный объект, в котором произведение выражено.

Объекты авторского права

• литературные произведения (включая программы для ЭВМ);

• драматические и музыкально-драматические произведения, сценарные произведения;

• хореографические произведения и пантомимы;

• музыкальные произведения с текстом или без текста;

• аудиовизуальные произведения (кино-, теле- и видеофильмы, слайдфильмы, диафильмы и другие кино- и телепроизведения);

• произведения живописи, скульптуры, графики, дизайна, графические рассказы, комиксы и другие произведения изобразительного искусства;

• произведения декоративно-прикладного и сценографического искусства;

произведения архитектуры, градостроительства и садовопаркового искусства;

• фотографические произведения и произведения, полученные способами, аналогичными фотографии;

• географические, геологические и другие карты, планы, эскизы и пластические произведения, относящиеся к географии, топографии и к другим наукам;

• другие произведения.

Охрана программ для ЭВМ распространяется на все виды программ для ЭВМ (в том числе на операционные системы), которые могут быть

выражены на любом языке и в любой форме, включая исходный текст и объектный код.

К объектам авторского права также относятся:

• производные произведения (переводы, обработки, аннотации, рефераты, резюме,

обзоры, инсценировки, аранжировки и другие переработки произведений науки, литературы и искусства);

• сборники (энциклопедии, антологии, базы данных) и другие составные произведения, представляющие собой по подбору или расположению материалов результат творческого труда.

Не являются объектами авторского права:

• официальные документы (законы, судебные решения, иные тексты законодательного, административного и судебного характера), а также их официальные переводы;

• государственные символы и знаки (флаги, гербы, ордена, денежные знаки и иные государственные символы и знаки);

• произведения народного творчества;

• сообщения о событиях и фактах, имеющие информационный характер.

• Авторское право на произведение науки, литературы и искусства возникает в силу факта его создания. Для возникновения и осуществления авторского права не требуется регистрации произведения, иного специального оформления произведения или соблюдения каких-либо формальностей.

• Обладатель исключительных авторских прав для оповещения о своих правах вправе использовать знак охраны авторского права, который помещается на каждом экземпляре произведения и состоит из трех элементов: латинской буквы "С" в окружности: ©;

• имени (наименования) обладателя исключительных авторских прав;

• года первого опубликования произведения. При отсутствии доказательств иного автором произведения считается лицо, указанное в качестве автора на оригинале или экземпляре произведения.

При опубликовании произведения анонимно или под псевдонимом (за исключением случая, когда псевдоним автора не оставляет сомнения в его личности) издатель, имя или наименование которого обозначено на произведении, при отсутствии доказательств иного считается представителем автора и в этом качестве имеет право защищать права автора и обеспечивать их осуществление. Это положение действует до тех пор, пока автор такого произведения не раскроет свою личность и не заявит о своем авторстве.

Авторское право на произведение, созданное совместным творческим трудом двух или более лиц (соавторство), принадлежит соавторам совместно независимо от того, образует ли такое произведение одно неразрывное целое или состоит из частей, каждая из которых имеет самостоятельное значение.

Часть произведения признается имеющей самостоятельное значение, если она может быть использована независимо от других частей этого

произведения. Каждый из соавторов вправе использовать созданную им часть произведения, имеющую самостоятельное значение, по своему усмотрению, если иное не предусмотрено соглашением между ними.

Право на использование произведения в целом принадлежит соавторам совместно. Взаимоотношения между ними могут определяться соглашением. Если произведение соавторов образует одно неразрывное целое, то ни один из соавторов не вправе запретить использование произведения.

Автору сборника и других составных произведений (составителю) принадлежит авторскоеправо на осуществленные им подбор или расположение материалов, представляющие результаттворческого труда (составительство).

Составитель пользуется авторским правом при условии соблюдения им прав авторов каждого из произведений, включенных в составное произведение. Авторы произведений, включенных в составное произведение, вправе использовать свои произведения независимо от составного произведения, если иное не предусмотрено авторским договором.

Издателю энциклопедий, энциклопедических словарей, периодических и продолжающихся сборников научных трудов, газет, журналов и других периодических изданий принадлежат исключительные права на использование таких изданий. Издатель вправе при любом использовании таких изданий указывать свое наименование либо требовать такого указания.

Авторы произведений, включенных в такие издания, сохраняют исключительные права на использование своих произведений независимо от издания в целом.

За нарушение авторских прав наступает гражданско-правовая, административная или уголовная ответственность в соответствии с законодательством Российской Федерации.

Тема 7. Внедрение научных исследований и их эффективность

7.1. Внедрение завершенных научных исследований в производство

Внедрение завершенных научных исследований в производство — заключительный этап научно-исследовательских работ.

Внедрение — это достижение практического использования прогрессивных идей, изобретений, результатов научных исследований (инноваций). Внедрение инноваций требует перестройки сложившегося производства, переподготовки работников, капитальных затрат и одновременно связано с риском не получить необходимый результат и потерпеть убытки.

Заказчиками на выполнение НИР могут быть технические управления министерств, тресты, управления, предприятия, НИИ. Подрядчиками являются научно-исследовательские организации, выполняющие НИР в

соответствии с подрядным двусторонним договором. Они обязаны сформулировать предложение по внедрения разработок. Предложения должны содержать технические условия, техническое задание, проектную документацию, временную инструкцию, указание и т. д.

Процесс внедрения состоит из двух этапов: опытно-производственного внедрения и серийного внедрения (внедрение достижений науки, новой техники, новой технологии).

Как бы тщательно ни проводились НИР в научно-исследовательских организациях, все же они не могут всесторонне учесть различные, часто случайные факторы, действующие в условиях производства. Поэтому научная разработка на первом этапе внедрения требует опытной проверки в производственных условиях. Предложение о законченных НИР рассматривают на научно-технических советах, а в случаях особо ценных предложений — на коллегиях министерства, и направляют на производство для практического применения.

После опытно-производственного испытания новые материалы, конструкции, технологии, рекомендации, методики внедряют в серийное производство как элементы новой техники. На этом, втором, этапе научно-исследовательские организации не принимают участия во внедрении. Они могут по просьбе внедряющих организаций давать консультации или оказывать незначительную научно-техническую помощь.

После внедрения достижений науки в производство составляют пояснительную записку, к которой прилагают акты внедрения и эксплуатационных испытаний, расчет экономической эффективности, справки о годовом объеме внедрения по включении получаемой экономии в план снижения себестоимости, протокол долевого участия организаций в разработке и внедрении, расчет фонда заработной платы и другие документы.

Внедрение достижений науки и техники финансируют организации, которые его осуществляют.

7.2. Эффективность научных исследований

Под экономической эффективностью научных исследований в целом понимают снижение затрат общественного и живого труда на производство продукции в той отрасли, где внедряют законченные научно-исследовательские работы и опытно-конструкторские разработки (НИР и ОКР).

Основные виды эффективности научных исследований:

1) экономическая эффективность — рост национального дохода, повышение производительности труда, качества продукции, снижение затрат на научные исследования;

2) укрепление обороноспособности страны;

3) социально-экономическая эффективность — ликвидация тяжелого труда, улучшение санитарно-гигиенических условий труда, очистка окружающей среды и т. д;

4) престиж отечественной науки.

Наука является наиболее эффективной сферой капиталовложений. В мировой практике принято считать, что прибыль от капиталовложений в нее составляет 100—200% и намного выше прибыли любых отраслей. По данным зарубежных экономистов, на один доллар затрат на науку прибыль в год составляет 4—7 долларов и больше. С каждым годом наука обходится обществу все дороже. На нее расходуют огромные суммы. Поэтому в экономике науки возникает и вторая проблема — систематическое снижение народнохозяйственных затрат на исследования при возрастающем эффекте от их внедрения. В связи с этим под эффективностью научных исследований понимают также по возможности более экономное проведение НИР.

Хорошо известно, какое большое значение ныне придается вопросам ускоренного развития науки и НТП. Делается это по глубоким стратегическим причинам, которые сводятся к тому объективному факту, что наука и система ее приложений стала реальной производительной силой, наиболее мощным фактором эффективного развития общественного производства.

Есть два кардинально различных пути ведения дел в экономике: экстенсивный путь развития и интенсивный. Путь экстенсивного развития — это расширение заводских площадей, увеличение числа станков и т. д. Интенсивный путь предполагает, чтобы каждый завод с каждого работающего станка, сельскохозяйственное предприятие с каждого гектара посевных площадей получали все больше и больше продукции. Это обеспечивается использованием новых научно-технических возможностей: новых средств труда, новых технологий, новых знаний. К интенсивным факторам относится и рост квалификации людей, и вся совокупность организационных и научно-технических решений, которыми вооружается современное производство.

Сегодня, примерно, каждый рубль, вложенный в науку, в НТП и освоение нововведений (новой техники, новых технологий) в производстве, дает в четыре раза больший эффект, чем тот же рубль, вложенный в экстенсивные факторы. Это очень существенное обстоятельство. Из него вытекает, что и впредь наша хозяйственная политика будет направлена на то, чтобы во всех сферах общественного производства решать проблемы дальнейшего развития преимущественно за счет интенсивных факторов. При этом особая роль отводится науке, а на саму науку распространяется то же самое требование. Сошлемся на характерные цифры. За последние 40—50 лет количество новых знаний увеличилось примерно в два-три раза, в то же время объем информации (публикаций, различной документации) увеличился в восемь-десять раз, а объем средств, отпускаемых на науку, — более чем в 100 раз. Эти цифры заставляют задуматься. Ведь рост ресурсов, затрачиваемых на науку, не является самоцелью. Следовательно, научную политику надо менять, необходимо решительно повысить эффективность работы научных учреждений.

Есть еще одно важное обстоятельство. В данном случае нас интересует не сам по себе прирост новых знаний, а прирост эффекта в

производстве. Мы должны проанализировать: все ли нормально с пропорциями между получением знаний и их применением на производстве. Нужно высокими темпами увеличивать вложения в мероприятия по освоению результатов НТП в производстве.

Существует некоторая теоретическая модель, построенная из соображений наиболее полного использования новых знаний, новых научных данных. В соответствии с этой моделью, если ассигнования в области фундаментальных исследований принять за единицу, то соответствующие показатели составят: по прикладным исследованиям — 4, по разработкам — 16, по освоению нововведений в производство — 250. Эта модель построена академиком В.М. Глушковым исходя из того, что все разумное (из новых идей, сведений, возможностей), полученное в сфере фундаментальных исследований, будет использовано. Для этого будет достаточно наличных мощностей прикладных наук. Затем возможности практического применения будут реализованы в виде новых технологий, новых конструкций и т. п., теми, кто проектирует, ведет разработки. И у них, в свою очередь, будет достаточно мощностей, чтобы все это принять и полностью пустить в дело. Наконец, необходимо иметь достаточно капиталовложений и свободных мощностей, предназначенных для освоения нововведений на производстве, чтобы освоить и реализовать все объективно необходимые нововведения. Если суммарные затраты на фундаментальные и прикладные исследования, а также на опытно-конструкторские разработки принять за единицу, то отношение между вложениями в производство новых знаний и вложениями в освоение этих знаний народным хозяйством составит 1:12. А в действительности такое соотношение 1:7. Это свидетельствует о том, что в народном хозяйстве зачастую нет свободных мощностей, не хватает возможностей для маневра (в США такое соотношение 1:11).

В современной науке каждый четвертый — руководитель. Это действительный факт. Руководителей в науке больше, чем физиков, химиков, математиков и пр., отдельно взятых. Но математиков, физиков, химиков и прочих готовят вузы (и профессиональный уровень их знаний, как правило, очень высок). Руководству же научной деятельностью их не обучали. Этому они учатся сами и самым непродуктивным способом — на своих ошибках. Решение этого вопроса тоже сможет поднять эффективность научных исследований.

Одним из путей повышения эффективности научных исследований является использование так называемых попутных или промежуточных результатов, которые зачастую совсем не используются или используются поздно и недостаточно полно. Например, космические программы. Чем они оправдываются экономически? Конечно, в результате их разработки была улучшена радиосвязь, появилась возможность дальних передач телевизионных программ, повышена точность предсказания погоды, получены большие научные фундаментальные результаты в познании мира и т. д. Все это имеет или будет иметь экономическое значение.

На эффективность исследовательского труда прямо влияет оперативность научных изданий, прежде всего периодических. Анализ сроков нахождения статей в редакциях отечественных журналов показал, что они задерживаются вдвое дольше, чем в аналогичных зарубежных изданиях. Для сокращения этих сроков, по-видимому, целесообразно в нескольких журналах экспериментально проверить новый порядок публикаций: печатать только рефераты статей объемом до 4—5 страниц, а полные тексты издавать методом безнаборной печати в виде оттисков и высылать по запросам заинтересованных лиц и организаций.

Известно, что темпы роста инструментальной вооруженности современной науки должны примерно в 2,5—3 раза превышать темпы роста численности работающих в этой сфере. В целом по стране этот показатель еще недостаточно высок, а в некоторых научных организациях он заметно меньше единицы, что приводит к фактическому снижению КПД интеллектуальных ресурсов науки. Современные научные приборы морально изнашиваются столь быстро, что за 4—5 лет, как правило, безнадежно устаревают. При нынешних темпах НТП абсурдной выглядит так называемая бережная (по нескольку часов в неделю) эксплуатация прибора. Рационально приобретать приборов меньше, но самых совершенных, и загружать их максимально, не боясь износа, а через 2—3 года интенсивной эксплуатации заменять новыми, более современными.

Министерство промышленности, обновляя свою продукцию примерно каждые пять и более лет, лишь 10—13% ее выпускает на уровне мировых показателей. Среди причин этого явления важное место занимает распыленность и слабость научного потенциала соответствующих предприятий, делающие их не подготовленными к восприятию существенно нового, а тем более к разработке его силами своих ученых и инженеров.

В современной науке вопросом вопросов являются кадры. Следует признать, что в целом индустриальный сектор науки еще очень слабо обеспечен высококвалифицированными кадрами исследователей. На каждую сотню центральных заводских лабораторий приходится лишь один кандидат наук. Большинство заводских научных подразделений, по масштабам работ сравнимых с обычными НИИ, имеют в несколько раз меньшее число докторов и кандидатов наук.

Особого внимания заслуживает проблема целевой подготовки кадров для индустриального сектора науки.

Для оценки эффективности исследований применяют разные критерии, характеризующие степень их результативности.

Фундаментальные исследования начинают отдавать капиталовложения лишь спустя значительный период после начала разработки. Результаты их обычно широко применяют в различных отраслях, иногда в тех, где их совсем не ожидали. Поэтому подчас нелегко планировать результаты таких исследований.

Фундаментальные теоретические исследования трудно оценить количественными критериями эффективности. Обычно можно установить

только качественные критерии: возможность широкого применения результатов исследований в различных отраслях народного хозяйства страны; новизна явлений, дающая большой толчок для принципиального развития наиболее актуальных исследований;
существенный вклад в обороноспособность страны; приоритет отечественной науки; отрасль, где могут быть начаты прикладные исследования; широкое международное признание работ; фундаментальные монографии по теме и цитируемость их учеными различных стран.

Эффективность прикладных исследований оценить значительно проще. В этом случае применяют различные количественные критерии.

Об эффективности любых исследований можно судить лишь после их завершения и внедрения, т.е. тогда, когда они начинают давать отдачу для народного хозяйства. Большое значение приобретает фактор времени. Поэтому продолжительность разработки прикладных тем по возможности должна быть короче. Лучшим является такой вариант, когда продолжительность их разработки до трех лет. Для большинства прикладных исследований вероятность получения эффекта в народном хозяйстве в настоящее время превышает 80%.

Как оценить эффективность исследования коллектива (отдела, кафедры, лаборатории и т. д.) и одного научного работника?

Эффективность работы научного работника оценивают различными критериями: публикационным, экономическим, новизной разработок, цитируемостью работ и др.

Публикационным критерием характеризуют общую деятельность — суммарное количество печатных работ, общий объем их в печатных листах, количество монографий, учебников, учебных пособий. Этот критерий не всегда объективно характеризует эффективность научного работника. Могут быть случаи, когда при меньшем количестве печатных работ отдача значительно больше, чем от большего количества мелких печатных работ. Экономическую оценку работы отдельного научного работника применяют редко. Чаще в качестве экономического критерия используют показатель производительности труда научного работника. Критерий новизны НИР — это количество авторских свидетельств и патентов. Критерий цитируемости работ ученого представляет собой число ссылок на его печатные работы. Это второстепенный критерий.

Эффективность работы научно-исследовательской группы или организации оценивают несколькими критериями: среднегодовой выработкой НИР, количеством внедренных тем, экономической эффективностью от внедрения НИР и ОКР, общим экономическим эффектом, количеством полученных авторских свидетельств и патентов, количеством проданных лицензий или валютной выручкой.

Экономический эффект от внедрения — основной показатель эффективности научных исследований — зависит от затрат на внедрение, объема внедрения, сроков освоения новой техники и многих других факторов.

Эффект от внедрения рассчитывают за весь период, начиная от времени разработки темы до получения отдачи. Обычно продолжительность такого периода прикладных исследований составляет несколько лет. Однако в конце его можно получить полный народнохозяйственный эффект.

Уровень новизны прикладных исследований и разработок коллектива характеризуют числом завершенных работ, по которым получены авторские свидетельства и патенты. Данный критерий характеризует абсолютное количество свидетельств и патентов. Более объективными являются относительные показатели, например количество свидетельств и патентов, отнесенных к определенному количеству работников данного коллектива или к числу тем, разрабатываемых коллективом, которые подлежат оформлению свидетельствами и патентами.

Различают три вида экономического эффекта: предварительный, ожидаемый и фактический. Предварительный экономический эффект устанавливается при обосновании темы научного исследования и включении ее в план работ. Рассчитывают его по ориентировочным, укрупненным показателям с учетом прогнозируемого объема внедрения результатов исследований в группу предприятий данной отрасли.

Ожидаемый экономический эффект вычисляют в процессе выполнения НИР. Его условно относят (прогнозируют) к определенному периоду (году) внедрения продукции в производство. Ожидаемая экономия — более точный экономический критерий по сравнению с предварительной экономией, хотя в некоторых случаях она является также ориентировочным показателем, поскольку объем внедрения можно определить лишь ориентировочно. Ожидаемый эффект вычисляют не только на один год, но и на более длительный период (интегральный результат). Ориентировочно такой период составляет до 10 лет от начала внедрения для новых материалов и до 5 лет для конструкций, приборов, технологических процессов. Фактический экономический эффект определяется после внедрения научных разработок в производство, но не ранее, чем через год. Расчет его производят по фактическим затратам на научные исследования и внедрение с учетом конкретных стоимостных показателей данной отрасли (предприятия), где внедрены научные разработки. Фактическая экономия почти всегда несколько ниже ожидаемой: ожидаемую определяют НИИ ориентировочно (иногда с завышением), фактическую — предприятия, на которых осуществляется внедрение. Наиболее достоверным критерием экономической эффективности научных исследований является фактическая экономия от внедрения.

Тема 8. Общие требования к научно-исследовательской работе

Все материалы, полученные в процессе исследования, разрабатывают, систематизируют и оформляют в виде научной работы. Это документ, который содержит исчерпывающие систематизированные сведения о выполненной работе.

Общие требования к научно-исследовательской работе: четкость и логическая последовательность изложения материала; убедительность аргументации; краткость и точность формулировок, исключающих возможность неоднозначного толкования; конкретность изложения результатов работы; обоснованность рекомендаций и предложений.

Общую структуру научно-исследовательской работы можно представить следующим образом:

- титульный лист;
- оглавление;
- введение;
- основная часть;
- заключение;
- список использованных источников;
- приложения.

Титульный лист – это первая страница рукописи, на которой указаны надзаголовочные данные, сведения об авторе, заглавие, подзаголовочные данные, сведения о научном руководителе, место и год выполнения работы.

К надзаголовочным данным относятся: полное наименование учебного заведения, факультета и кафедры, по которой выполнена работа.

В средней части титульного листа пишется заглавие работы.

В подзаголовочных данных указывается вид работы (реферат, курсовая или дипломная работа).

Затем, ближе к правому краю титульного листа, пишутся фамилия, имя и отчество автора. Далее указывается ученая степень, ученое звание, ФИО научного руководителя.

В нижней части титульного листа указываются место и год написания работы.

Оглавление раскрывает содержание работы путем обозначения глав, параграфов и других рубрик научной работы с указанием страниц, с которых они начинаются. Оно должно быть в начале работы. Названия глав и параграфов должно точно повторять соответствующие заголовки в тексте.

Введение работы должно содержать оценку современного состояния решаемой научно-исследовательской проблемы, основание и исходные данные для разработки темы, обоснование необходимости выполнения работы. Во введении должны быть показаны актуальность и новизна темы, связь данной работы с другими НИР.

Обычно объем введения не превышает 5-7% объема основного текста. Основная часть может состоять из нескольких глав, разбитых на параграфы. В них рассматривается действующее законодательство, излагаются теоретические положения, дается анализ различных точек зрения, высказывается и аргументируется свое мнение. В конце каждой главы делаются краткие выводы. Заключение должно содержать выводы по результатам выполненной научной работы и указание по возможности их внедрения. Объем заключения не должен превышать 5-7% объема основного текста. В список литературы включают только те источники, которые были

использованы при написании и упомянуты в тексте или сносках. Список составляется по разделам с учетом требований государственного стандарта. В приложение включаются извлечения из отдельных нормативных актов, копии подлинных документов, выдержки из справок, отчетов, образцы анкет, таблицы, графики и другие вспомогательные материалы, которые загромождают основную часть работы и увеличивают ее объем. При подсчете объема научной работы приложения не учитываются.

Деление текста на составные части с использование заголовков, нумерации и прочих средств называется рубрикацией. Система рубрик включает заголовки частей, разделов, глав и параграфов, которые, как правило, нумеруются. Каждый из названных членов деления текста, в свою очередь, подразделяется на абзацы. Под абзацем понимается отступ вправо в начале первой строки определенной части текста. Понятием «абзац» обозначают также ту часть текста, которая находится между двумя такими отступами. Обычно абзац состоит их нескольких предложений, связанных между собой определенной мыслью. Абзацы одного параграфа или главы должны быть также связаны по смыслу и расположены в логической последовательности. При делении текста на главы и параграфы используются логические правила деления понятий. Под делением понятий понимается мыслительный процесс раскрытия объема понятия посредством выделения в нем видовых понятий. Операция деления должна производиться по определенным правилам:

• деление должно быть соразмерным, т.е. объем всех членов деления должен равняться объему делимого понятия;

• деление должно осуществляться по одному основанию;

• члены деления не должны соотноситься между собой как часть и целое;

• деление должно быть последовательным и непрерывным.

Рубрикация текста обычно связана с нумерацией – числовым (или буквенным) обозначением последовательности расположения его составных частей. Для этого используются римские и арабские цифры, прописные и строчные буквы. Разделы, подразделы, пункты и подпункты следует нумеровать арабскими цифрами и записывать с абзацного отступа. Разделы должны иметь порядковую нумерацию в пределах всего текста. Главы нумеруют римскими цифрами.

Авторы научных работ применяют различные способы написания текста:

• строго последовательный, когда автор переходит к следующему параграфу только после завершения предыдущего;

• целостный, когда пишется вся работа, а затем в нее вносятся исправления и дополнения, шлифуется текст;

• выборочный, когда автор пишет работу в том порядке, в каком ему удобно.

В зависимости от целевого назначения и специфики содержания научной работы используются различные типы изложения материала:

• описательный. Он применяется в тех случаях, когда необходимо дать характеристику исследуемого предмета или явления, описать его развитие, структуру, составляющие элементы и признаки;

• повествовательный. Такой тип изложения характеризуется изложением материала в хронологическом порядке, обрисовкой причинно-следственных связей исследуемых предметов и явлений. Повествовательные тексты обычно начинаются с описания причин и условий, вызвавших то или иное явление;

• объяснительный. Данный тип изложения применяется для объяснения тех или иных установлений, доказывания и опровержения научных положений и выводов.

Особенностью языка научной речи является подчеркнутая логичность. Эта логичность должна проявляться на различных уровнях: всего текста, отдельных частей, абзацев. Она характеризуется последовательным переходом от одной мысли к другой. В качестве средства связи между ними используются: вводные слова и предложения; местоимения, прилагательные и причастия; специальные функционально-синтаксические средства, указывающие на последовательность (прежде всего, затем, во-первых); причинно следственные отношения (следовательно, поэтому) и т.д. Научный язык характеризуется стремлением к объективности изложения материала. Объективность изложения обусловлена спецификой научного познания, направленного на установление истины. Для подтверждения объективности в тексте делается ссылка на то, кем высказана та или иная мысль, в каком источнике содержится использованная информация. Ради объективности в тексте научного произведения личные пристрастия, эмоциональные моменты не отражаются. В рукописи следует избегать штампов, избыточных словосочетаний. Не украшают речь повторения, растянутые фразы, нагромождения. С целью уменьшения объема текста применяется сокращение слов. В настоящее время используются следующие виды сокращений:

• буквенные аббревиатуры, которые составляются из начальных букв каждого слова, входящего в название;

• сложносокращенные слова, составляемые из усеченных слов;

• условные графические сокращения по начальным буквам и частям слова;

Сокращению подлежат различные части речи. При сокращении слов применяют усечение, стяжении или сочетание этих приемов. Вне зависимости от используемого приема при сокращении должно оставаться не менее двух букв, например: ст. – статья, см. – смотри.

Сокращение слов до одной начальной буквы допускается только для общепринятых сокращений и отдельных слов, например: г. – год, р. – рубль.

В качестве иллюстративного материала в курсовых и дипломных работах используются графики, диаграммы и схемы.

Иллюстрации должны быть расположены так, чтобы их было удобно рассматривать без поворота отчета или с поворотом по часовой стрелке. Иллюстрации располагают после первой ссылки на них.

Иллюстрации должны иметь наименование. При необходимости их снабжают поясняющими данными (подрисуночный текст). Наименование иллюстрации помещают над ней, поясняющие данные — под ней.

График – это условное изображение соотношения величин в их динамике при помощи геометрических фигур, линий и точек. График содержит следующие элементы: заголовок; словесные пояснения; оси абсцисс и ординат, шкалу с масштабами, числовые сетки; числовые данные, дополняющие или уточняющие величины нанесенных на график показателей. В зависимости от целей, количественной базы и применяемых геометрических фигур графики могут быть линейными, столбиковыми, полосовыми, секторными. На графике может быть изображена динамика нескольких явлений. Тогда их кривые должны быть хорошо различаемы по цвету или форме. Если для построения графиков используются такие геометрические фигуры, как прямоугольники и круги, то их называют диаграммами. Столбиковые диаграммы строятся в системе прямоугольных координат. Основания столбиков одинаковой ширины помещают на оси абсцисс, а их высота отражает величину явлений. Полосовые диаграммы отличаются от столбиковых тем, что прямоугольники в них расположены не вертикально, а горизонтально (полосками). Секторная диаграмма представляет собой круг, разделенный на секторы, каждый из которых занимает площадь круга, соответствующую величине отражаемого явления.

Схема – это изложение, описание, изображение чего-нибудь в главных чертах. Обычно делается без соблюдения масштаба с помощью условных изображений. Зачастую они вычерчиваются в виде прямоугольников или иных геометрических фигур с простыми связями-линиями.

Таблицы. Цифровой материал, как правило, должен оформляться в виде таблиц. Каждая таблица должна иметь заголовок. Заголовок и слово "Таблица" начинают с прописной буквы. Заголовок не подчеркивают. Заголовки граф таблиц должны начинаться с прописных букв, подзаголовки — со строчных, если они составляют одно предложение с заголовком, и с прописных, если они самостоятельные. Делить заголовки таблицы по диагонали не допускается. Высота строк должна быть не менее 8 мм. Графу "№ п. п." в таблицу включать не следует. Таблицу размещают после первого упоминания о ней в тексте таким образом, чтобы ее можно было читать без поворота работы или с поворотом по часовой стрелке. Таблицу с большим количеством строк допускается переносить на другой лист. При переносе таблицы на другой лист (страницу) заголовок помещают только над ее первой частью. Таблицу с большим количеством граф допускается делить на части и помещать одну часть под другой в пределах одной страницы. Если строки или графы таблицы выходят за формат таблицы, то в первом случае в каждой части таблицы повторяется ее головка, во втором случае — боковик.

Если повторяющийся в графе таблицы текст состоит их одного слова, его допускается заменять кавычками; если из двух или более слов, то при первом повторении его заменяют словами "То же", а далее — кавычками. Ставить кавычки вместо повторяющихся цифр, марок, знаков, математических и химических символов не допускается. Если цифровые или иные данные в какой-либо строке таблицы не приводят, то в ней ставят прочерк.

Формулы. Пояснение значений символов и числовых коэффициентов следует приводить непосредственно под формулой в той же последовательности, в какой они даны в формуле. Значение каждого символа и числового коэффициента следует давать с новой строки. Первую строку объяснения начинают со слова "где" без двоеточия. Уравнения и формулы следует выделять из текста свободными строками. Выше и ниже каждой формулы должно быть оставлено не менее одной свободной строки. Если уравнение не умещается в одну строку, оно должно быть перенесено после знака равенства (=) или после знаков (+), минус (-), умножение (х) и деление (:).

Ссылки в тексте на литературные источники допускается приводить в подстрочном примечании или указывать порядковый номер по списку источников, выделенный двумя косыми чертами.

Ссылки на иллюстрации указывают порядковым номером иллюстрации.

Ссылки на формулы указывают порядковым номером формулы в скобках, например "... в формуле (2.1)".

На все таблицы должны быть ссылки в тексте, при этом слово "Таблица" в тексте пишут полностью, если таблица не имеет номера, и сокращенно — если имеет номер, например: "... в табл. 1.2).

В повторных ссылках на таблицы и иллюстрации следует указывать сокращенно слово "смотри", например: см. табл. 1.3".

Тема 9. Основные требования к написанию, оформлению и защите научных работ студентов

9.1 Особенности подготовки рефератов и докладов

Реферат – научно-исследовательская работа, представляющая собой краткое изложение в письменном виде содержания научных трудов, учебных пособий, научных статей по заданной теме. В реферате студент излагает основные положения, содержащиеся в нескольких источниках, приводит различные точки зрения, обосновывает свое мнение по ним. Реферат состоит из титульного листа, оглавления, введения, основной части и списка использованной литературы. Объем реферата – не менее 5 и не более 15 страниц, отпечатанных через 1,5 интервала. В реферате следует сделать ссылки на использованные источники. Они должны быть оформлены в соответствии с установленным стандартом. Готовый реферат представляется преподавателю для проверки. Оценивая реферат, он учитывает умение студента работать с научной литературой, анализировать различные точки

зрения по спорным вопросам, аргументировать свое мнение, навыки оформления ссылок, списка использованной литературы.

Доклад – это запись устного сообщения на определенную тему. Он предназначен для прочтения на семинарском занятии, научной конференции. Если текст доклада должен быть сдан преподавателю, то он оформляется так же, как текст реферата. В тех случаях, когда сдать текст не требуется, достаточно его подготовить для себя без оформления. Текст доклада может быть написан полностью либо в виде тезисов. В последнем случае в логической последовательности записываются только основные мысли. Студенческие доклады, как правило, состоят из трех частей: вводной, основной и заключительной. В первой части обосновываются актуальность, теоретическая и практическая ценность темы, во второй излагаются основные научные положения, в третьей – выводы.

9.2. Особенности подготовки и защиты курсовых работ

Курсовая работа – это предусмотренная учебным планом письменная работа студента на определенную тему, содержащая элементы научного исследования. Написание курсовой работы помогает студентам углубить и закрепить полученные знания по дисциплине, приобрести навыки самостоятельного проведения научных исследований и обобщения практического материала, оформления результатов творческого труда. Перечень тем курсовых работ определяется кафедрами. Студенту предоставляется право выбора темы. По согласованию с научным руководителем студенту разрешается выполнение работы по теме, которая хотя и не значится в перечне, но имеет прямое отношение к изучаемой дисциплине. Выбранная тема должна быть зарегистрирована на соответствующей кафедре. Научным руководителем студента является, как правило, преподаватель, ведущий занятия в той группе, в которой он учится. С ним необходимо согласовать план работы, список нормативных актов и специальной литературы, метод сбора и обработки практических материалов и сроки предоставления на проверку. В целях упорядочения основных этапов работы полезно составить рабочий план с указание сроков их выполнения. Структура курсовой работы:

- титульный лист;
- оглавление;
- введение;
- основная часть;
- заключение;
- список использованной литературы;
- приложения (факультативно).

Объем курсовой работы должен составлять примерно 20-25 листов машинописного текста (компьютерной распечатки), исполненного на стандартной писчей бумаге формата А4, не считая приложений. При использовании в тексте работы приложений, выводов, предложений, заимствованных из различных источников, ссылки на них обязательны.

Теоретические положения и выводы рекомендуется иллюстрировать материалами опубликованной и неопубликованной практики. При этом необходимо сделать ссылку на источник, откуда они взяты. Выполненная курсовая работа к установленному сроку сдается на кафедру и передается на рецензирование научному руководителю. Отзыв руководителя пишется в произвольной форме, но в нем обязательно следует отметить достоинства работы, ошибки и другие недостатки, соответствие работы установленным требованиям и указать, допускается ли она к защите или не допускается.

Не допускаются к защите работы:

выполненные только на основе учебника, без использования анализа законодательных и практических материалов;

выполненные не самостоятельно, а путем списывания, без ссылок на автора и источник, или являющиеся конспектом учебника;

не раскрывающие содержания темы и имеющие грубые ошибки;

небрежно и неправильно оформленные.

Такие работы возвращаются для устранения недостатков. К повторно выполненной работе студент обязан приложить отзыв руководителя о первоначально выполненной работе, чтобы он мог проверить, устранены ли отмеченные в нем недостатки. Студент защищает работу перед научным руководителем. Если руководитель по объективным причинам не может принять защиту, то заведующий кафедрой может поручить эту работу другому преподавателю.

9.3. Особенности подготовки и защиты дипломных работ

Дипломная работа – это выпускная квалификационная работа, представляющая собой теоретическое или экспериментальное исследование одной из актуальных тем в конкретной области знания, в которой выпускник демонстрирует уровень овладения необходимыми теоретическими знаниями и практическими навыками, позволяющие ему самостоятельно решать профессиональные задачи.

Задачами выпускной дипломной работы являются:

- теоретическое обоснование и раскрытие сущности экономических и финансовых категории, явлений и проблем рыночной экономики по избранной теме;
- развитие навыков самостоятельной работы, полученных за годы учебы, в проведении научного исследования по теме;
- знание и умение применять положения законодательных, нормативных и инструктивных материалов по вопросам, рассматриваемым в дипломной работе;
- умение самостоятельно разрабатывать конкретную научную проблему;
- четкое понимание экономической теории в решении проблем исследуемой темы, включая критическую оценку литературных источников и различных взглядов ученых и практиков, в том числе и зарубежных;

• умение систематизировать и обстоятельно анализировать данные, полученные из статистических сборников, отчетных материалов, периодической и специальной литературы, делать аргументированные выводы и предложения;

• обобщение всего комплекса знаний, полученных за время обучения в Вузе.

При выборе темы дипломной работы нужно учитывать:актуальность темы исследования, практическую значимость, возможность использования в дипломной работе конкретного фактического материала, собранного в период прохождения производственной и преддипломной практик.

Выполнение дипломной работы проходит следующие этапы: выбор темы; изучение литературы; составление плана; определение методов исследования; изучение практики; работа над текстом и оформление.

Далее следует подготовка к защите и защита работы. Дипломная работа по своей структуре состоит из следующих элементов:

• титульного листа;
• оглавления;
• введения;
• основной части;
• заключения;
• списка использованной литературы;
• приложений (если они необходимы).

Объем дипломной работы при техническом исполнении (на печатной машинке, компьютере и др.) не должен превышать 60-70 страниц, (без приложений). Рукописный вариант дипломной работы не допускается. Внутренняя структура может состоять из задания по подготовке дипломной работы, введения, двух - максимум трех глав с 2-3 параграфами каждая, заключения в виде выводов и рекомендаций, библиографии и приложений. Возможна иная структура. Например, без глав и параграфов, а лишь через разделы. Примерное распределение объема дипломной работы по разделам может быть следующим:

Введение (5% общего объема).

Раздел 1 - Теория вопроса (20%).

Раздел 2- Анализ практического материала (экономические расчеты) (50%).

Раздел 3 - Экономическое обоснование выводов и рекомендаций (20%).

Заключение (5%).

Готовая дипломная работа подписывается исполнителем и сдается научному руководителю в срок, установленный задание или планом графиком. После ее прочтения руководитель составляет на нее письменный отзыв. В отзыве следует отразить положительные и отрицательные стороны дипломного проекта по следующей схеме: актуальность, новизна, теоретическая и практическая значимость проведенного исследования; полнота освещения вопросов темы, использования литературы и практического материала; степень самостоятельности автора в раскрытии

темы; обоснованность выводов, логичность аргументации; наличие предложений и рекомендаций, возможность их внедрения в учебных процесс; соответствие оформления работы установленным правилам; неточности, ошибки, спорные предложения, замечания по содержанию и оформлению.

Дипломная работа (переплетена или скреплена зажимами типовой папки) с письменным отзывом руководителя (приложение 4) не менее чем за 10 дней до защиты передается рецензенту для рецензирования.

В качестве рецензентов привлекаются, как правило, научные работники, работающие в соответствующей области, высококвалифицированные практические работники, имеющие высшее образование.

Рецензия (отзыв о научной работе) — это работа, в которой критически оценивают основные положения и результаты рецензируемого исследования. Особое внимание обращают на актуальность его теоретических положений, целесообразность и оригинальность принятых методов исследования, новизну и достоверность полученных результатов, их практическую полезность.

При составлении рецензии обычно придерживаются такой последовательности: обоснование необходимости (актуальность) темы исследования; оценка идейного и научного содержания (основная часть рецензии), языка, стиля; последовательность изложения результатов исследования; оценка иллюстративного материала, объема исследований и рукописи изложения (рекомендации о сокращении или дополнении); общие выводы; итоговая оценка исследования.

Критика рецензента должна быть принципиальной, научно обоснованной, взыскательной, но вместе с тем и доброжелательной, способствующей улучшению исследования.

Дипломная работа защищается студентом перед Государственной комиссией на открытом заседании. Процедура защиты следующая. Для изложения основных результатов исследования автору предоставляется 10-15 минут. В выступлении докладчик (дипломник) не должен озвучивать чужие общеизвестные сведения, положения, определения, а кратко изложить понимание исследуемой проблемы, уделив большее внимание результатам собственного исследования.

В докладе рекомендуется отразить:

- обоснование актуальной темы;
- характеристику объекта исследования;
- основное содержание в по главам;
- обоснование предлагаемых мероприятий;
- выводы и предложения.

Содержание доклада может быть иллюстрировано. Иллюстративный материал должен подтверждать теоретические и практические выводы, представлять наиболее важные цифры, оформленные в табличной, графической или текстовой формах. По окончании доклада члены комиссии

и присутствующие могут задать дипломнику вопросы по теме дипломной работы. Ответы должны быть по существу заданных вопросов, краткими и аргументированными. Затем зачитываются отзыв руководителя и рецензии (замечания и основные выводы из них) и предоставляется слово руководителю и рецензенту, которые сообщают свое мнение о дипломной работе. Решения комиссии об оценке дипломных работ и итогах защиты принимаются на закрытом заседании простым большинством голосов членов комиссии.

Глоссарий

Абстрагирование - отвлечение от второстепенных фактов с целью сосредоточения на
важнейших особенностях изучаемого явления.

Автор изобретения - физическое лицо, творческим трудом которого оно создано

Автор научного открытия - в РФ - физическое лицо, которое путем наблюдения,
изучения, эксперимента или рассуждения самостоятельно сделало научное открытие способом,
обеспечивающим его установление.

Если открытие сделано группой физических лиц, то любая ссылка на автора научного
открытия рассматривается как ссылка на все эти лица.

Автореферат диссертации – научное издание в виде брошюры, содержащее
составленный автором реферат проведенного им исследования, предоставляемого на соискание
ученой степени.

Аксиома – исходное положение, которое не может быть доказано, но в то же время и не
нуждается в доказательстве.

Аналогия – это способ получения знаний о предметах и явлениях на основании того, что
они имеют сходство с другими.

Библиография - информационная инфраструктура, обеспечивающая подготовку,
распространение и использование библиографической информации; перечень различных
информационных документов с указанием определенных данных

Внедрение - распространение нововведений; достижение практического использования
прогрессивных идей, изобретений, результатов научных исследований (инноваций).

Газета - периодическое газетное издание, выходящее через краткие промежутки времени,

содержащее официальные материалы, оперативную информацию и статьи по актуальным
общественно-политическим, научным, производственным и другим вопросам, а также
литературные произведения и рекламу.

Обычно газета издается в виде больших листов (полос).

Гипотеза - научное предположение, выдвигаемое для объяснения некоторого явления и
требующее верификации.

График – условное изображение соотношения величин в их динамике при помощи
геометрических фигур, линий и точек.

Диаграмма – график, построенный с помощью геометрических фигур, таких как
прямоугольник, круг.

Данные - сведения:

- полученные путем измерения, наблюдения, логических или арифметических
  операций;
- представленные в форме, пригодной для постоянного хранения, передачи и
  (автоматизированной) обработки.

Дипломная работа – выпускная квалификационная работа, представляющая собой
теоретическое или экспериментальное исследование одной из актуальных тем в определенной
области.

Доклад – запись устного сообщения на определенную тему, предназначаемая для
прочтения на семинарском занятии, конференции.

Документ - по законодательству РФ - материальный объект с зафиксированной на нем
информацией в виде текста, звукозаписи или изображения, предназначенный для передачи во
времени и пространстве в целях хранения и общественного использования. Документ обязательно
содержит реквизиты, позволяющие однозначно идентифицировать, содержащуюся в нем
информацию.

Журнал - периодическое журнальное издание:

- содержащее статьи или рефераты по различным общественно-политическим,
  научным, производственным и другим вопросам;
- литературно-художественные произведения;
- имеющее постоянную рубрикацию;

• официально утвержденное в качестве журнального издания.

Журнал может иметь приложения.

Задача - координированная и систематизированная серия элементов работы,
используемых для достижения результатов.

Закон – положение, выражающее всеобщий ход вещей в какой-либо области;
высказывание относительно того, каким образом что-либо является необходимым или
происходит с необходимостью.

Идея – это: 1) новое интуитивное объяснение события или явления; 2) определяющее
стержневое положение в теории.

Издание - документ:

- прошедший редакционно-издательскую обработку;
- полученный печатанием или тиснением;
- полиграфически самостоятельно оформленный;
- имеющий выходные сведения;
- предназначенный для распространения содержащейся в нем информации.

Изобретение - новое и обладающее существенными отличиями техническое решение
задачи в любой области экономики, социального развития, культуры, науки, техники, обороны,
дающее положительный эффект. Автор изобретения, получивший авторское свидетельство, имеет
право дать изобретению свое имя или специальное название. Изобретение является одним из
объектов промышленной собственности.

Интеллектуальная собственность - собственность на результаты интеллектуальной
деятельности, интеллектуальный продукт, входящий в совокупность объектов авторского и
изобретательского права.

Информационное издание - издание, содержащее систематизированные сведения об
опубликованных, непубликуемых или неопубликованных документах или результат анализа и
обобщения сведений, представленных в первоисточниках.

Информационные ресурсы - в широком смысле - совокупность данных, организованных
для эффективного получения достоверной информации.

Информационные ресурсы - по законодательству РФ - отдельные документы и
отдельные массивы документов, документы и массивы документов в информационных системах:
библиотеках, архивах, фондах, банках данных, других видах информационных систем.

Источник информации - объект, идентифицирующий происхождение информации; в
теории коммуникации - лицо, от которого исходит сообщение; отправитель сообщения; в теории
перевода - создатель или автор текста оригинала.

Категория – общее, фундаментальное понятие, отражающее наиболее существенные
свойства и отношения предметов и явлений.

Классификация наук - группировка наук на основе определенных принципов.

Конспект - краткое изложение прочитанного.

Концепция – это система теоретических взглядов, объединенных научной идеей
(научными идеями).

Курсовая работа – предусмотренная учебным планом письменная работа студента на
определенную тему, содержащая элементы научного исследования.

Лицензия на изобретение - разрешение, выдаваемое одним лицом (лицензиаром) другому
лицу (лицензиату) на коммерческое использование изобретения, защищенного патентом в
границах строго определенного рынка, в течение определенного срока и за обусловленное
вознаграждение.

Логотип - оригинальное начертание, изображение полного или сокращенного
наименования фирмы или товаров фирмы. Логотип специально разрабатывается фирмой с целью
привлечения внимания к ней и к ее товарам.

Материалы научной конференции – научный непериодический сборник, содержащий
итоги научной конференции (программы, доклады, рекомендации, решения).

Методика – это совокупность способов и приемов познания.

Методология - 1)совокупность методов, применяемых в какой-либо сфере деятельности
(науке, политике и т.д.); 2) учение о научном методе познания.

Моделирование - исследование объектов познания на их моделях. Моделирование
предполагает построение и изучение моделей реально существующих предметов, явлений и
конструируемых объектов:

- для определения или улучшения их характеристик;
- для рационализации способов их построения;
- для управления и прогнозирования.

Монография - научное или научно-популярное книжное издание:

- содержащее полное и всестороннее исследование одной проблемы или темы;
- принадлежащее одному или нескольким авторам.

Научная деятельность - интеллектуальная деятельность, направленная на получение и
применение новых знаний для:

- решения технологических, инженерных, экономических, социальных, гуманитарных и иных проблем;
- обеспечения функционирования науки, техники и производства как единой системы.

Научная информация - логически организованная информация, получаемая в процессе
научного познания и отображающая явления и законы природы, общества и мышления.

Научная проблема – это противоречие между знаниями о потребностях общества и
незнанием путей и средств их удовлетворения.

Научно-популярное издание - издание, содержащее сведения:

- о теоретических и/или экспериментальных исследованиях в области науки, культуры и техники;
- изложенное в форме, доступной читателю-неспециалисту.

Научно-техническая информация - документированная информация, возникающая в
результате научного и технического развития, а также информация, в которой нуждаются
руководители, научные, инженерные и технические работники в процессе своей деятельности,
включая специализированную экономическую и нормативно-правовую информацию.

Научное знание - система знаний о законах природы, общества, мышления. Научное
знание составляет основу научной картины мира и отражает законы его развития.

Научное издание - издание, содержащее результаты теоретических и/или
экспериментальных исследований, а также научно подготовленные к публикации памятники
культуры и исторические документы.

Научное исследование - процесс изучения, эксперимента, концептуализации и проверки
теории, связанный с получением научных знаний. Различают фундаментальные и прикладные
научные исследования.

Научно-технический прогресс - использование передовых достижений науки и техники,
технологии в хозяйстве, в производстве с целью повышения эффективности и качества
производственных процессов, лучшего удовлетворения потребности людей.

Научное открытие - установление явлений, свойств или законов материального мира,
ранее не установленных и доступных проверке.

Научный вопрос - мелкая научная задача, относящаяся к конкретной области научного
исследования.

Научный результат - продукт научной и/или научно-технической деятельности,
содержащий новые знания или решения и зафиксированный на любом информационном носителе.

Научный термин – это слово или сочетание слов, обозначающее понятие, применяемое в
науке.

Общественные науки - совокупность наук, изучающих различные аспекты жизни
человеческого общества.

Объект исследования – это то социальное явление (процесс), которое содержит
противоречие и порождает проблемную ситуацию

Объяснение - этап научного исследования, состоящий:

- в раскрытии необходимых и существенных взаимозависимостей явлений или процессов;
- в построении теории и выявлении закона или совокупности законов, которым подчиняются эти явления или процессы.

Описание - этап научного исследования, состоящий в фиксировании данных эксперимента
или наблюдения посредством определенных систем обозначений, принятых в науке.

Патент - документ:

• выдаваемый компетентным государственным органом на определенный срок;

• удостоверяющий авторство и исключительное право на изобретение; и

наделяющий владельца титулом собственника на изобретение.

Патентная информация - информация, публикуемая патентными организациями. Каждая
публикация содержит:

• список ключевых слов;

• коды;

• сведения о патентном документе, включающие описание изобретения, фамилии

авторов, дату поступления заявки, дату приоритета, сведения о правовом

положении документа.

Патентоспособность - совокупность свойств технического решения, без наличия которых
оно не может быть признано изобретением на основе действующего законодательства.

В РФ патентоспособным признается изобретение, которое:

1) является новым, т.е. неизвестно из уровня техники;

2) имеет изобретательский уровень, т.е. для специалиста явным образом не следует из
уровня техники;

3) промышленно применимо, т.е. может быть использовано в промышленности, сельском
хозяйстве, здравоохранении и других отраслях деятельности.

Первоисточник - источник информации:

• либо являющийся оригинальным документом, содержащим данные исследования;

• либо составленное рукой непосредственного участника описание событий:

дневник, автобиография, письмо, юридический документ, отчет, протокол, деловая

бумага, счет, газета и т.д.

Полезная модель - объект промышленной собственности; конструктивное выполнение
средств производства и предметов потребления, а также их составных частей. Полезной модели
предоставляется правовая охрана, если она является новой и промышленно применимой.

Положение – научное утверждение, сформулированная мысль.

Понятие - мысль, отражающая в обобщенной форме предметы и явления

действительности и существенные связи между ними посредством фиксации общих и
специфических признаков.

Предмет исследования - существенные свойства или отношения объекта исследования,
познание которых важно для решения теоретических или практических проблем. Предмет
исследования определяет границы изучения объекта в конкретном исследовании.

Препринт - научное издание, содержащее материалы предварительного характера,
опубликованные до выхода в свет издания, в котором они могут быть помещены.

Прикладные научные исследования - исследования, направленные преимущественно на
применение новых знаний для достижения практических целей и решения конкретных задач.

Принцип- основное начало, на котором построено что-н. (какая-н. научная система,
теория, политика, устройство и т. п.)

Проблема – неразрешенная задача или вопросы, подготовленные к разрешению.

Процедура исследования - последовательность познавательных и организационных
действий с целью решения исследовательской задачи.

В общем случае научное исследование предполагает:

1. постановку задачи;
2. предварительный анализ имеющейся информации, условий и методов
3. решения задач данного класса;
4. формулировку исходных гипотез;
5. сбор данных;
6. анализ и обобщение полученных результатов;
7. проверку гипотез;
8. формулирование утверждений.

Промышленный образец - графическое описание товара или изделия, отражающее его
внешний вид. Автор промышленного образца после его официальной регистрации получает
патент, дающий исключительное право на производство товара именно в этом внешнем
исполнении.

Публикация - документ, доступный для массового использования.

Рабочая программа – это изложение общей концепции исследования в соответствии с его
целями и гипотезами.

Рецензия — это работа, в которой критически оценивают основные положения и
результаты научного исследования.

Реферат - краткое изложение содержания отдельного документа, его части или
совокупности документов, включающее основные сведения и выводы, а также количественные и
качественные данные об объектах описания.

Рубрикация – деление текста на составные части с использованием заголовков,
нумерации и т.д.

Сборник научных трудов - сборник, содержащий исследовательские материалы научных
учреждений, учебных заведений или обществ.

Способ – это действие или система действий, применяемые при исполнении какой-либо
работы, при осуществлении чего-либо.

Сравнение – это сопоставление признаков, присущих двум или нескольким объектам,
установление различия между ними или нахождение в них общего.

Суждение – это мысль, в которой утверждается или отрицается что-либо.

Схема – изложение, описание, изображение чего-либо в главных чертах; обычно делается
без соблюдения масштаба с помощью условных обозначений.

Счет (количественный метод) - это определение количественных соотношений объектов
исследования или параметров, характеризующих их свойства.

Тезисы докладов научной конференции - научный непериодический сборник,
содержащий опубликованные до начала конференции материалы предварительного характера:
аннотации, рефераты докладов и/или сообщений.

Тема — это научная задача, охватывающая определенную область научного исследования.

Теория - форма достоверных научных знаний:

- представляющая собой множество логически увязанных между собой допущений и суждений;
- дающая целостное представление о закономерностях и существенных характеристиках объектов;
- основывающаяся на окружающей реальности.

Товарный знак - знак:

- имеющий вид рисунка, этикетки, клейма и т.д.
- присвоенный определенному товару или фирме;

• помещаемый на товаре, его упаковке, фирменных бланках, вывесках, рекламных материалах;

• зарегистрированный в соответствующем государственном учреждении;

• защищающий исключительные права продавца на пользование товарным знаком.

Товарные знаки - по законодательству РФ - обозначения, способные отличать товары одних юридических или физических лиц от однородных товаров других юридических или физических лиц.

Учебник - учебное издание, содержащее систематическое изложение учебной дисциплины, ее раздела или части, соответствующее учебной программе и официально утвержденное в качестве учебника.

Учебно-методическое пособие - учебное издание, содержащее материалы по методике преподавания учебной дисциплины или по методике воспитания.

Учебное издание – это издание, содержащее систематизированные сведения научного или прикладного характера, изложенные в форме, удобной для изучения и преподавания, и рассчитанное на учащихся разного возраста и ступени обучения.

Учебное наглядное пособие - учебное изоиздание, содержащее материалы в помощь изучению, преподаванию или воспитанию.

Учебное пособие – это учебное издание, дополняющее или частично заменяющее учебник и официально утвержденное в качестве учебного пособия.

Учение - совокупность теоретических положений о какой-либо области явлений действительности

Факт – действительное, вполне реальное событие, явление; нечто сделанное, совершившееся.

Формализация – представление основных положений процессов и явлений в виде формул и специальной символики.

Фундаментальные научные исследования - экспериментальная или теоретическая деятельность, направленная на получение новых знаний об основных закономерностях строения,

функционирования и развития человека, общества, окружающей природной среды.

Хрестоматия - учебное пособие, содержащее литературно-художественные, исторические
и иные произведения или отрывки из них, составляющие объект изучения учебной дисциплины.

Эксперимент - общенаучный метод получения в контролируемых и управляемых
условиях новых знаний о причинно-следственных отношениях между явлениями и процессами.

Эмпирическое обобщение – это система определенных научных фактов, на основании
которой можно сделать определенные выводы или выявить недочеты и ошибки.

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ ПРОЕКТИРОВАНИЕ КИБЕРФИЗИЧЕСКИХ СИСТЕМ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

Оглавление

Лабораторная работа № 1
Разработка функциональных блоков и приложений

Цель работы:

а) Получить опыт работы с интегрированной средой разработки (ИСР) NxtStudio: как создавать, компилировать, тестировать и модифицировать типы базисных функциональных блоков (ФБ).

б) Научиться использовать типы базисных ФБ в типах составных ФБ.

в) Выполнить распределенное удаленное приложение (FLASHER_TESTR).

Порядок выполнения работы

1. Изучить ИСР NxtStudio.
2. Разработать и протестировать базисный ФБ для вычисления функции X^2-Y^2.
3. Модифицировать базисный ФБ из п.2, добавив в него функцию вычисления суммы X+Y+Z.
4. Разработать составной ФБ для вычисления функции X^2-Y^2.
5. Создание, развертывание и выполнение распределенного приложения FLASHER.

Для выполнения работы необходимо использовать материал из раздела «Основные сведения», находящийся непосредственно ниже.

Основные сведения

Установка программного обеспечения NxtControl

Инсталляция программного обеспечения фирмы *NxtControl* на платформе *Windows* хорошо поддерживается соответствующим инсталляционным пакетом. Перед установкой необходимо иметь свежую версию пакета *Microsoft.NET*. Необходимые инструкции отображаются в мастере установки.

Функционирование ИСР *NxtStudio* интуитивно понятно для пользователей, знакомых с любой современной средой программирования. Особенно данная ИСР структурно похожа на среду программирования *Visual Studio*.

В данном лабораторном практикуме используется система *NxtStudio* версии 2.0, однако все готовые рекомендуемые *NxtStudio*-проекты, которые ориентированы на данную версию, можно адаптировать для и более поздних версий путем указания соответствующих библиотек или, в более сложных случаях, путем перекомпиляции проектов.

Создание базисного ФБ в NxtStudio

В *NxtStudio* нельзя непосредственно создать тип ФБ. Это можно сделать только в рамках некоторого «решения» (*solution*), которое является структурным аналогом «проекта» (*project*) в других средах программирования.

Создадим новое решение с названием *Tutorial* (выбор меню *File->New->Solution*…), как показано на рис. 1а, затем установим курсор на элемент *Basic* в дереве эксплорера (проводника) решений в левой части экрана, нажмем правую кнопку мыши и выберем *New Item* в выпадающем меню (рис. 1б). В появившемся диалоговом окне зададим имя *X2Y2* для типа ФБ.

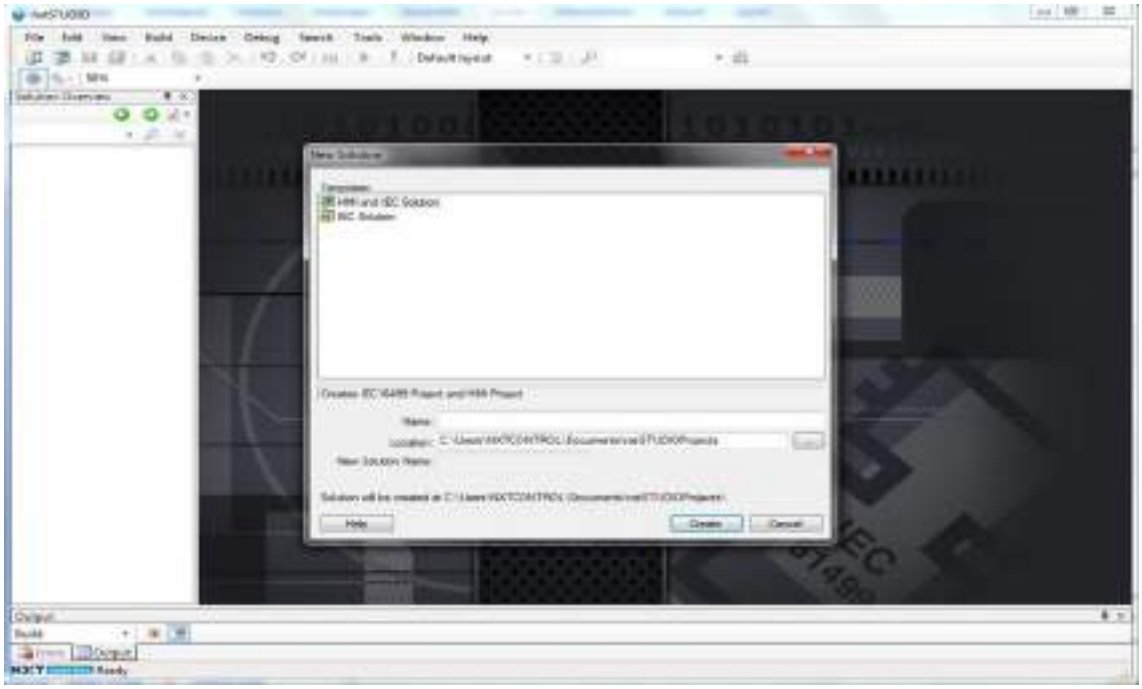


Рис. 1а. Создание решения

Рис. 1б. Создание нового типа базисного ФБ.

После этого на экране можно увидеть тип ФБ, созданный со стандартным интерфейсом, как показано на рис. 2а. Перетащим границу *Панели интерфейсов* вправо и начнем редактировать элементы интерфейса в таблице, как показано на рис. 2б. При этом удаляем событийные входы *INIT*

и *INITO* и переименовываем информационные вход и выход *QI* и *QO* в *X* и *OUT*, соответственно, изменив попутно их тип на *REAL*. Также добавляем еще один вход *Y* типа *REAL*.

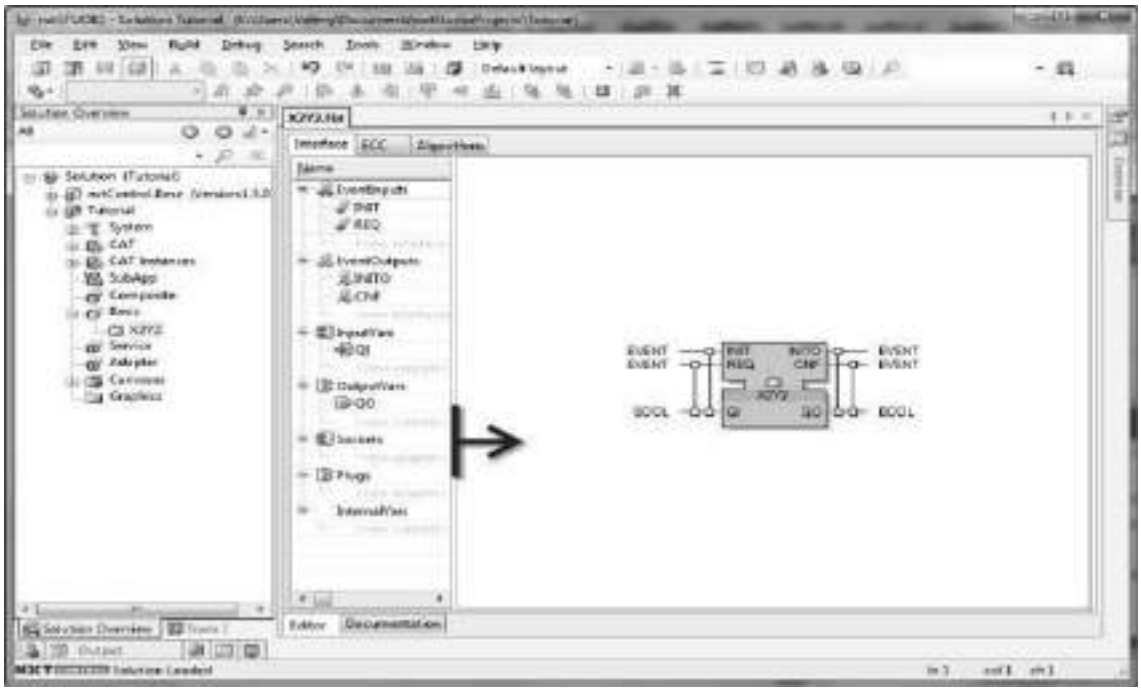


Рис. 2а. Открытие вновь созданного типа ФБ.

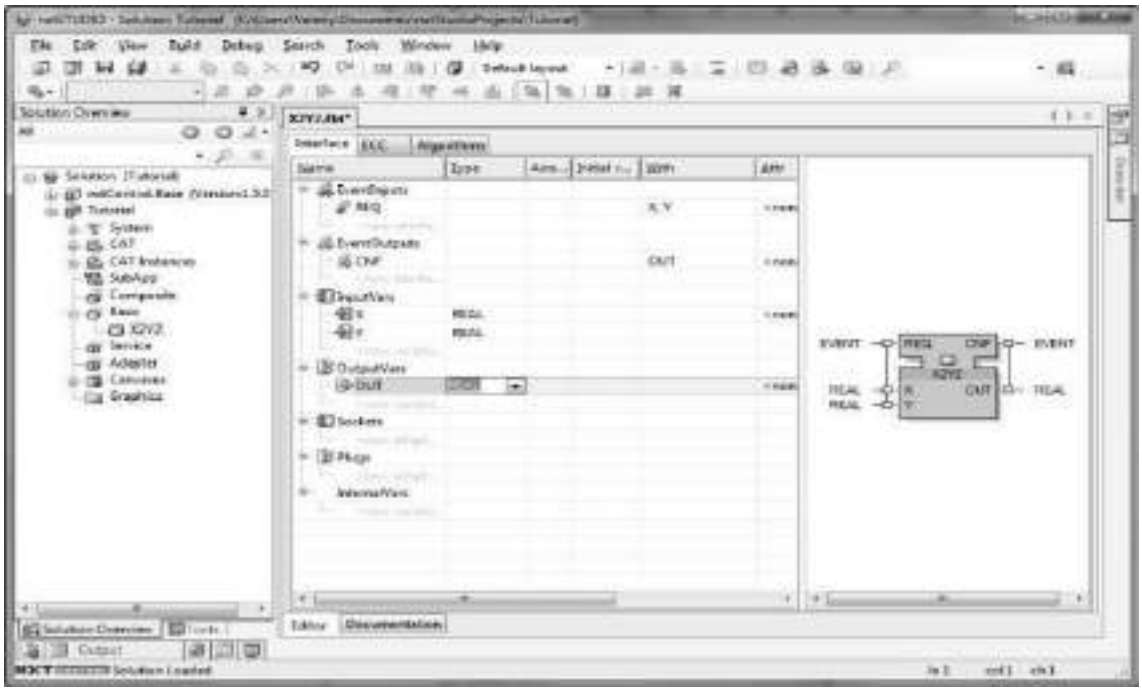


Рис. 2б. Модификация интерфейса созданного типа ФБ.

Чтобы создать связь между событийным входом/выходом и информационным входом/выходом, используем поле *With* в таблице интерфейса.

Теперь переходим на вкладку *ECC* (*Execution Control Chart* - диаграмма управления выполнением) и редактируем *ECC* как показано на рис. 3а. Нужно будет удалить состояние *INIT*, поскольку в этом простом примере нет необходимости в инициализации. Для этого кликаем правой кнопкой мыши по состоянию и выбираем пункт меню *Delete*. Также можно выбрать состояние левым щелчком мыши и нажать клавишу *Del*. После этого удалим алгоритм *INIT* (щелчок правой кнопкой мыши на вкладке *Algorithm* и выбор пункта меню *Delete algorithm*) и отредактируем алгоритм *REQ*, как показано на рис. 3б.

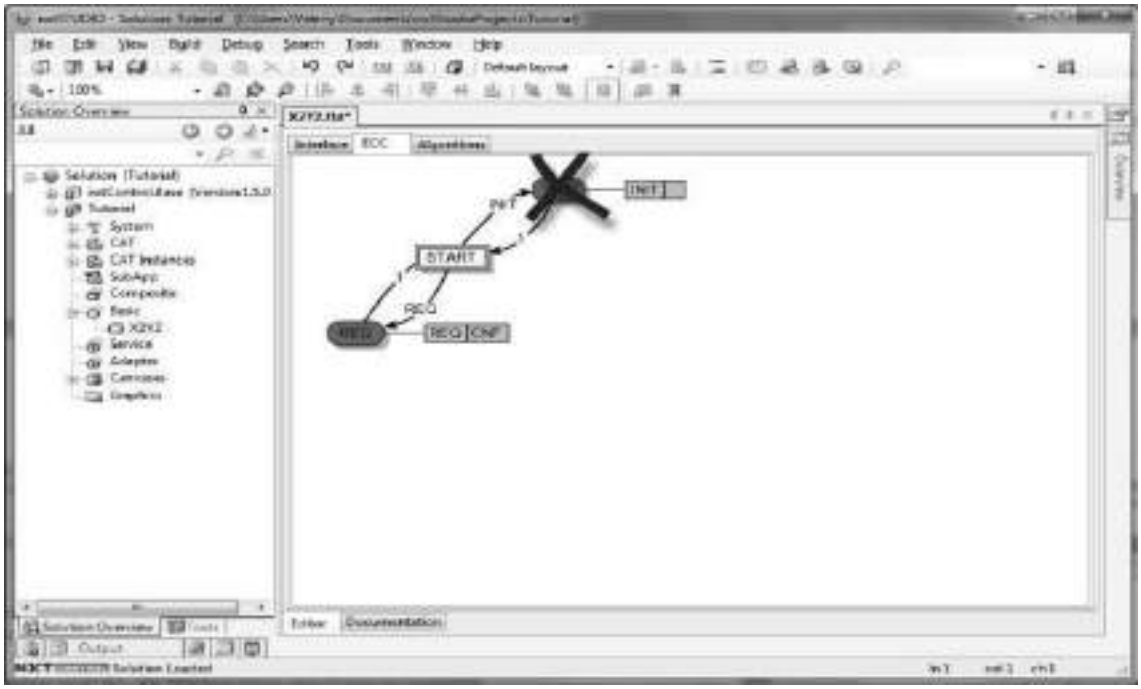


Рис. 3а. Удаление состояния INIT в диаграмме ECC.

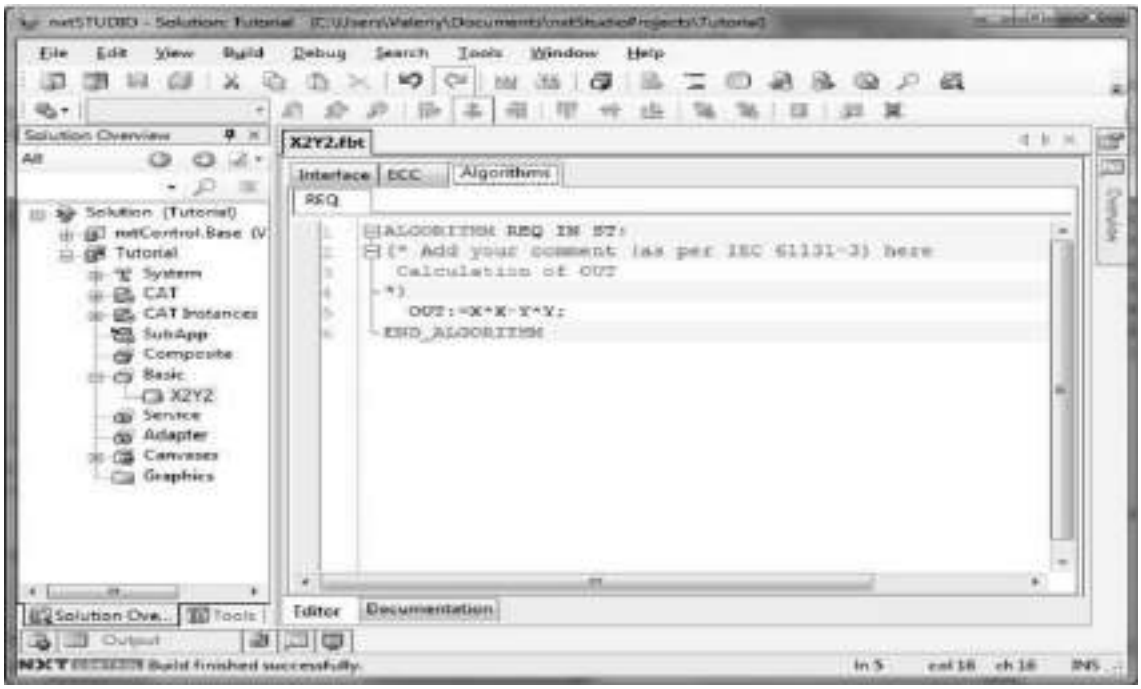


Рис. 3б. Модификация алгоритма REQ.

Теперь ФБ готов и может быть протестирован. Переходим в меню отладки *Debug* и выбираем пункт *Run*. При этом появится отладочное окно (рис. 4).

Для тестирования ФБ необходимо задать определенные значения на информационных входах и инициировать событие *REQ*, кликнув на соответствующей кнопке рядом с ним. Введем параметры *X* и *Y*, нажмем кнопку *REQ* и посмотрим результат: выходное событие *CNF* (в виде вспышки «лампы») и вычисленное значение в поле *OUT*. С левой стороны можно наблюдать текущее состояние *ECC*. Чтобы остановить отладку и вернуться в режим редактирования, нажмите кнопку *Stop*, как показано на рис.4.

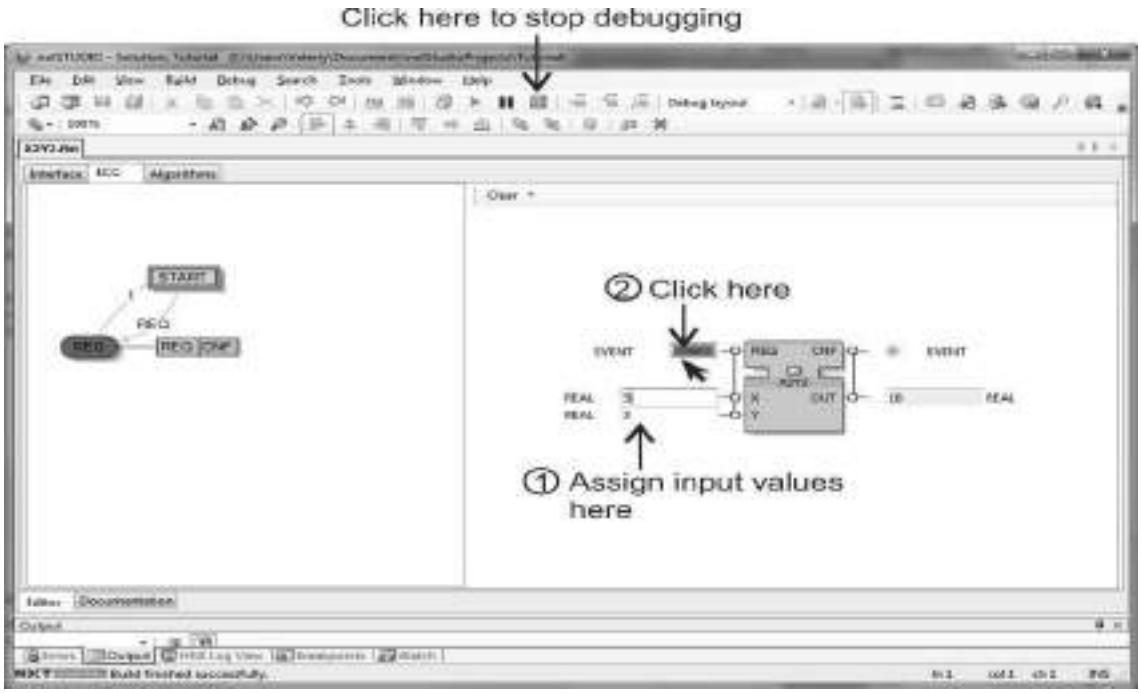


Рис. 4. Окно отладки для функционального блока X2Y2.

Поэкспериментируем с добавлением нескольких входов и выходов в ФБ и с модификацией *ECC*. Например, сделаем следующие изменения: добавим информационный вход *Z* (типа *REAL*), событийный вход *XYZ*, событийный выход *CNF2* и информационный выход *OUT2* (типа *REAL*). Когда блок активируется событием *XYZ*, он должен вычислить *OUT2=X+Y+Z* и вывести сигнал *CNF2*.

Чтобы создать новый тип ФБ на основе старого, необходимо скопировать этот ФБ в *Дерево решений*, а затем вставить его под другим именем, например, *X2Y2_XYZ*. Чтобы скопировать его, перейдите в Дерево решений, щелкните правой кнопкой мыши на старом ФБ и выберите *Copy* из выпадающего меню (рис. 5а). Затем переместите курсор в корень списка базисных ФБ (по имени *Basic*), щелкните правой кнопкой мыши и выберите команду *Paste*. В появившемся диалоговом окне введите новое имя ФБ. Результат будет выглядеть как на рис. 5б.

Замечание: имя в форме ФБ справа может быть обновлено только после сохранения решения (нажатие комбинации клавиш *Ctrl-S* или кнопка

сохранения на панели инструментов) и двойном щелчке мыши на новом ФБ в дереве решений.

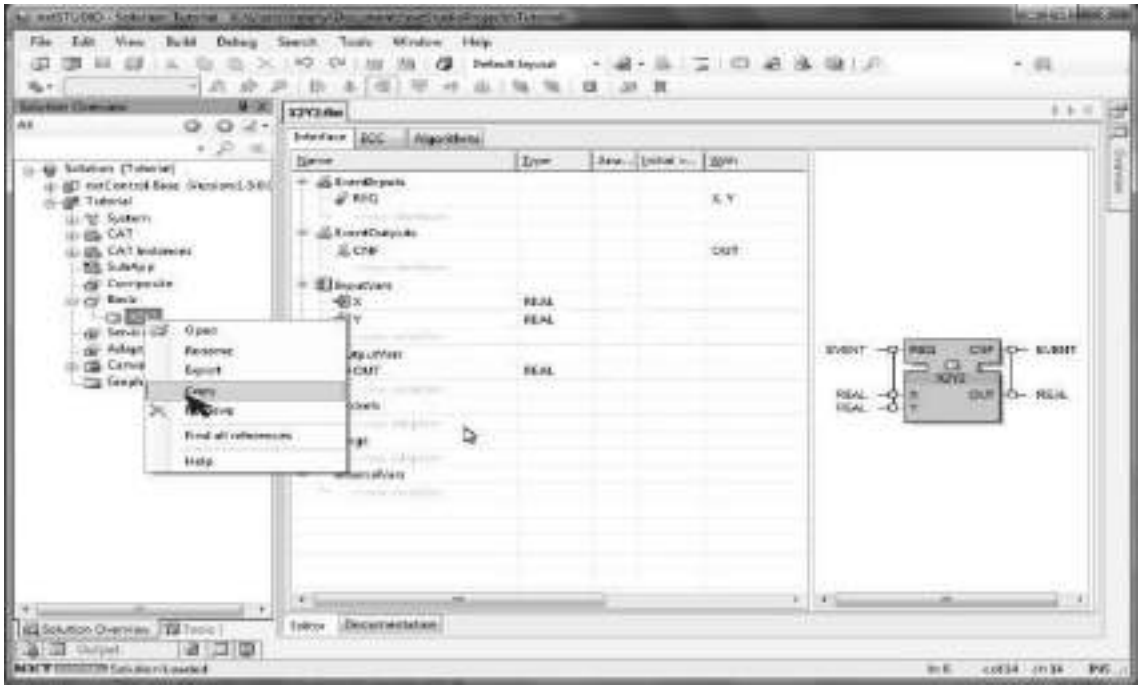


Рис. 5а. Создание нового ФБ на основе старого (копирование)

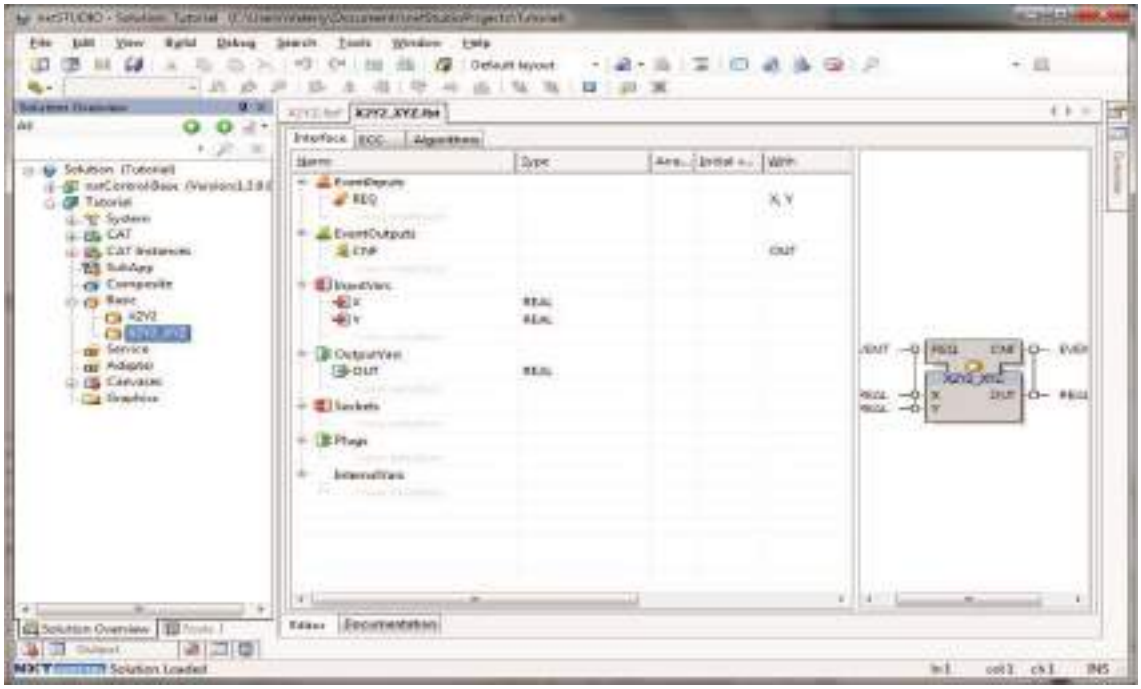


Рис. 5б. Создание нового ФБ на основе старого (добавление и назначение нового имени)

Интерфейс полученного блока показан на рис. 6а. На следующем шаге добавим новый алгоритм, назовем его *XYZ_COMP*. На вкладке *Algorithms* щелкнем правой кнопкой мыши на существующем алгоритме *REQ* и выберем *Add algorithm* (рис. 6б). После этого добавим строку: *OUT2*:=*X*+*Y*+*Z*; в новый алгоритм.

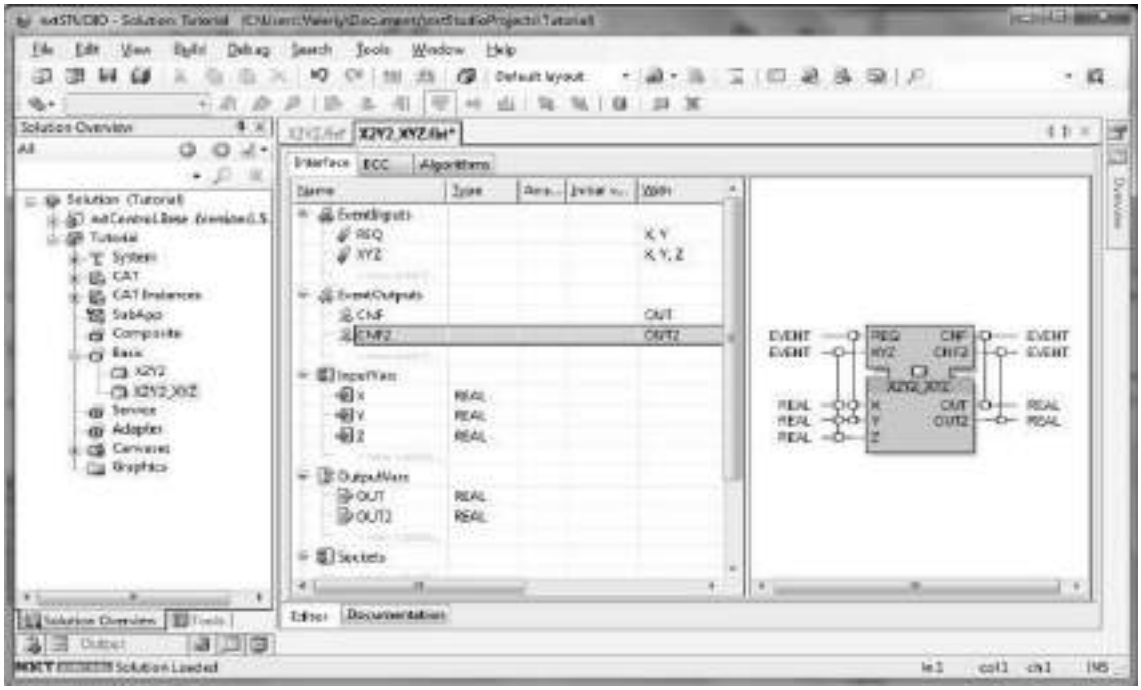


Рис. 6а. Интерфейс нового ФБ

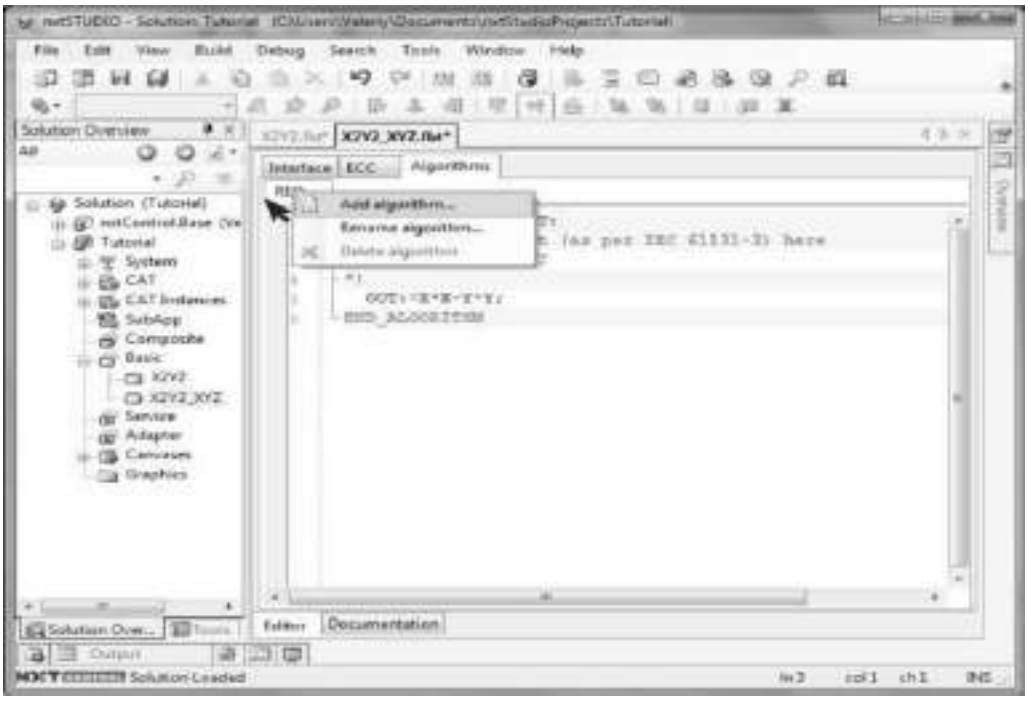


Рис. 6б. Создание нового алгоритма

Следующим шагом является редактирование диаграммы *ECC* путем добавления нового состояния, в котором будет вызываться только что созданный алгоритм. Назовем состояние *XYZ_S*.

Перейдя на панель *ECC*, вызовите всплывающее меню, щелкнув правой кнопкой мыши, затем выберите *New State,* как показано на рис. 7а. Редактирование диаграммы *ECC* интуитивно понятно. Чтобы изменить название состояния, просто нажмите на него. Чтобы нарисовать линию перехода между двумя состояниями, наведите курсор на состояние-источник, найдите положение, где курсор в виде стрелки изменяет свою форму на изображение руки, и, нажав и удерживая левую кнопку мыши, перетащите его в конечное состояние (рис.7б). Далее нажмите на полученную связь, чтобы изменить значение с *XYZ_S* (по умолчанию условие перехода будет равно имени события-приемника) на *XYZ* (это новое событие, которое мы создали на предыдущем шаге, редактируя интерфейс).

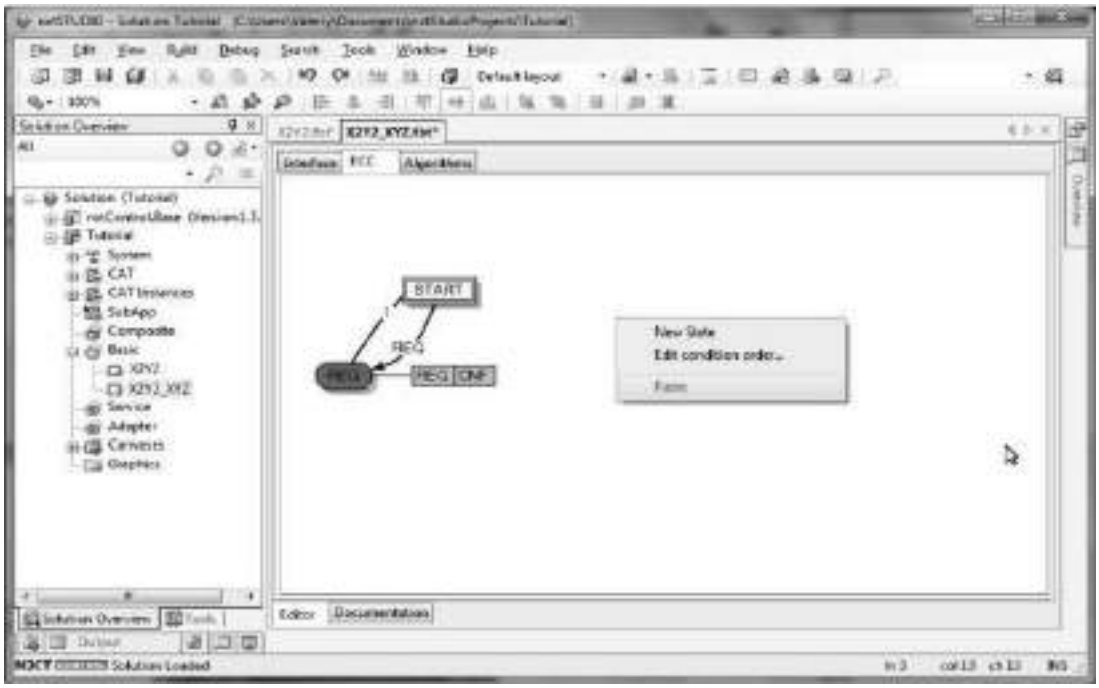


Рис. 7а. Создание нового состояния в ECC

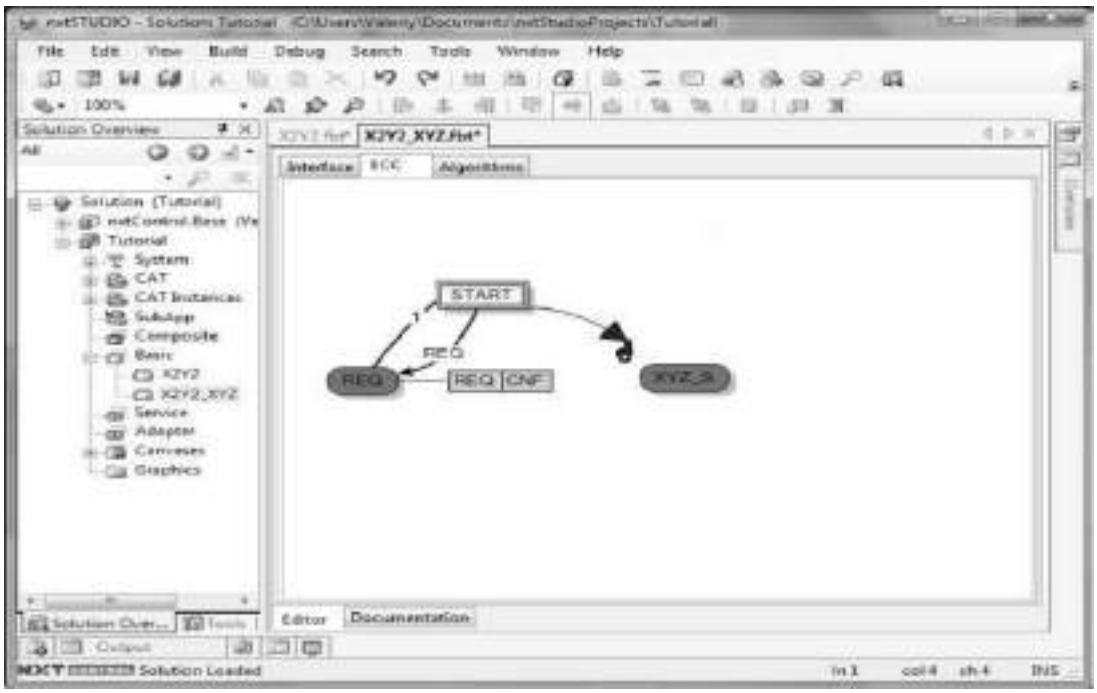


Рис. 7а. Добавление связей

После добавления нового состояния, мы должны создать новую акцию (иначе, *EC*-акцию) для этого состояния для того, чтобы прикрепить к нему алгоритм и определить, какое выходное событие будет выдано. Все эти действия можно легко выполнить, нажимая правой кнопкой мыши на состоянии или части акции. Результат модификации диаграммы *ECC* показан на рис.8.

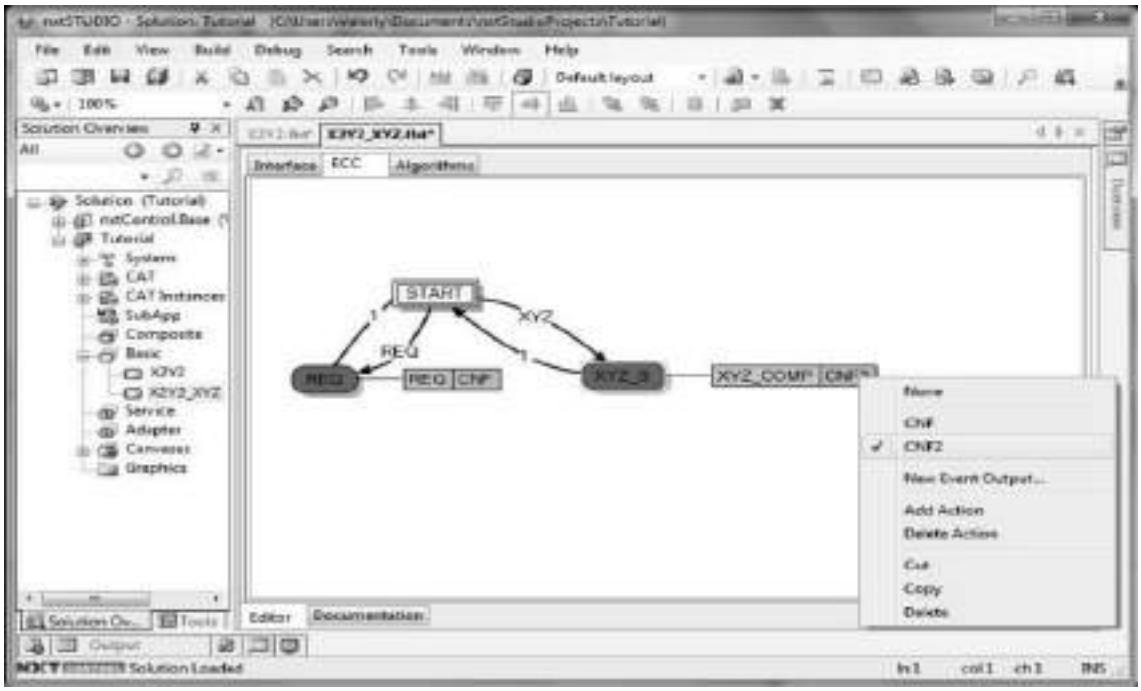


Рис. 8. Модификация диаграммы ECC

Теперь можно протестировать этот новый ФБ, как и раньше, используя опцию *Run* из меню отладки *Debug*.

Создание составного ФБ в NxtStudio

Создадим новый тип составного ФБ с именем *X2Y2_CMP*, как показано на рис. 9. Для этого слева, в дереве эксплорера решений выберем элемент *Composite*, правый клик мыши на нем и далее выбор *New Item*. Затем нужно выполнить те же действия, что и для базисного ФБ, а именно: дать ему имя, определить интерфейс и сохранить.

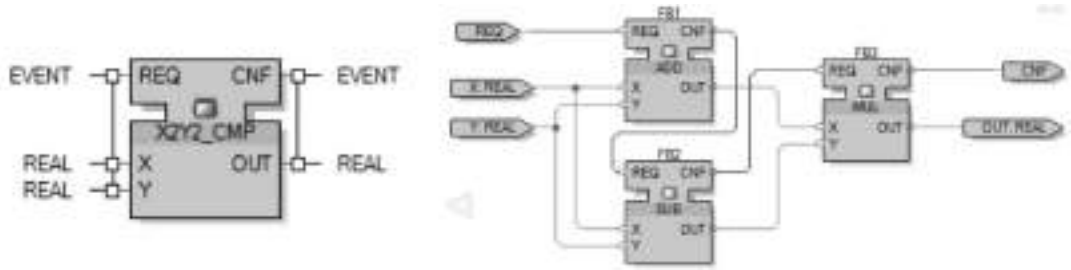


Рис. 9. Интерфейс и содержимое составного ФБ для вычисления функции X^2-Y^2

Чтобы создать тело ФБ, переходим из вкладки *Interface* на вкладку *Composite*, как показано на рис. 10. Для добавления новых экземпляров ФБ, используем всплывающее меню, вызывающееся нажатием правой кнопки мыши.

К сожалению, в текущей версии *NxtStudio* удобная для пользователя среда тестирования базисных ФБ недоступна в отношении составных ФБ. Поэтому составные ФБ могут быть протестированы только в составе приложений, которые будут обсуждаться в следующих разделах.

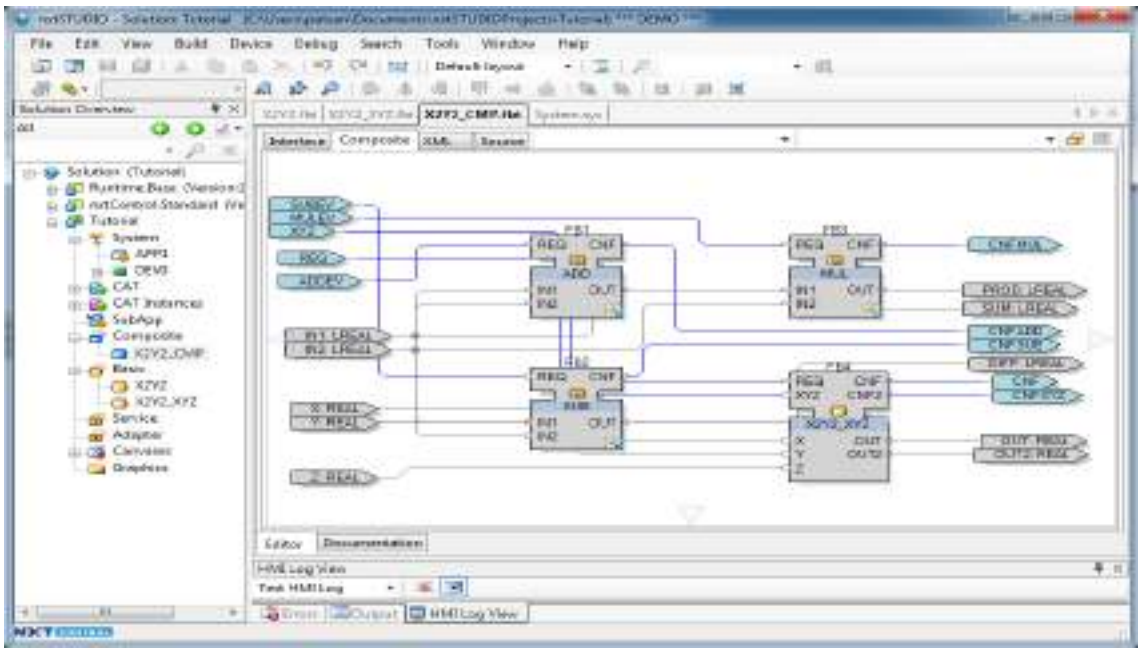


Рис. 10. Добавление компонентных ФБ в сеть составного ФБ

Далее создадим новый составной блок, как показано ниже. Добавим в него следующие компонентные ФБ: ADD, SUB, MUL, а также созданный нами ранее блок X2Y2_XYZ. Сначала создаем интерфейс, как показано на рис. 11, а затем сеть ФБ, как показано на рис. 12. Следует обратить внимание на то, что ФБ типов ADD, SUB и MUL могут быть найдены в библиотеках системы NxtStudio, как показано на рис. 13.

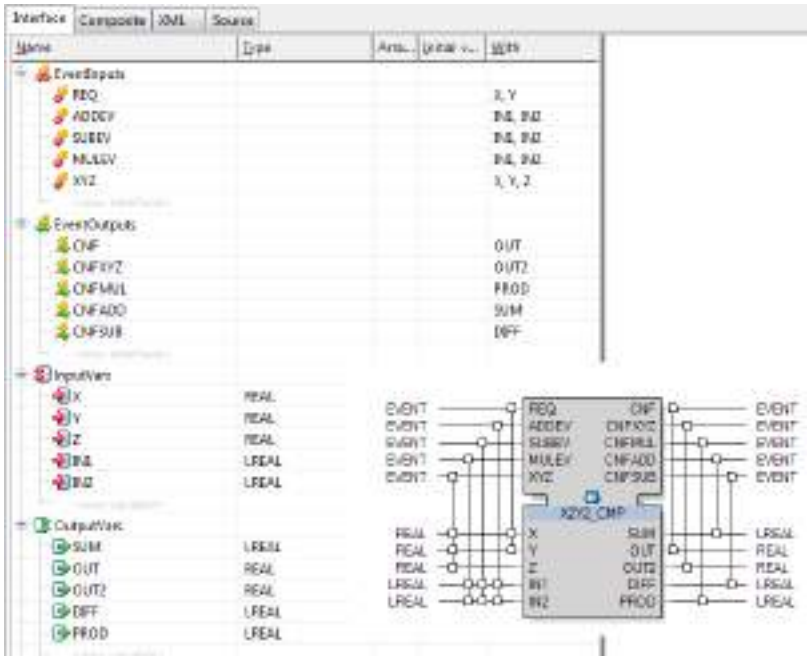


Рис. 11. Интерфейс составного ФБ

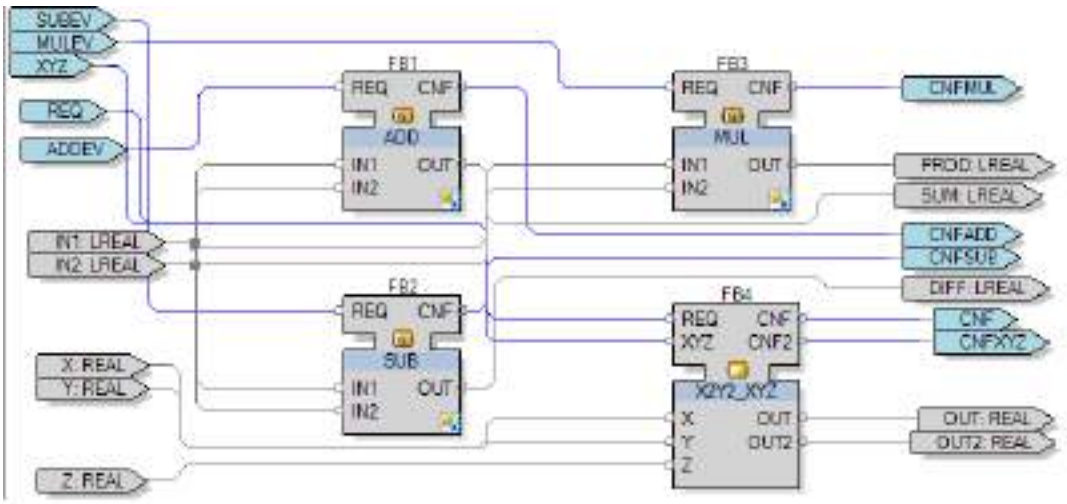


Рис. 12. Сеть составного ФБ

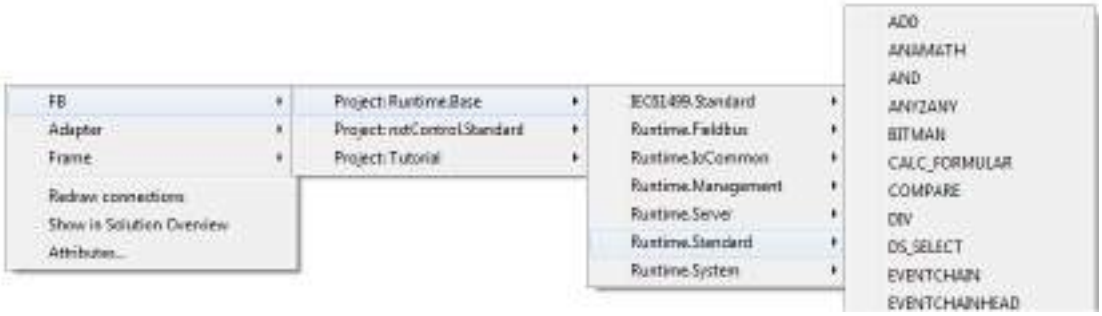


Рис 13. Готовые ФБ в библиотеках системы NxtStudio

Как уже упоминалось ранее, нельзя напрямую протестировать составной блок в отладчике, поэтому необходимо создать функционально-блочное приложение (ФБ-приложение, или просто приложение). Чтобы сделать это, необходимо дважды щелкнуть на элементе *APP1* в разделе *System* дерева эксплорера решений (с левой стороны), как показано на рис. 14. Затем можно просто перетащить составной блок *XYZ_COMP* из эксплорера решений на холст в середину. В дальнейшем, в рамках данного приложения можно создать простой человеко-машинный интерфейс (ЧМИ), содержащий кнопки и текстовые поля для имитации ввода событий и сопутствующих данных, а также «светодиодов» (LEDs) и текстовые поля для визуализации выходных событий и связанных с ними данных.

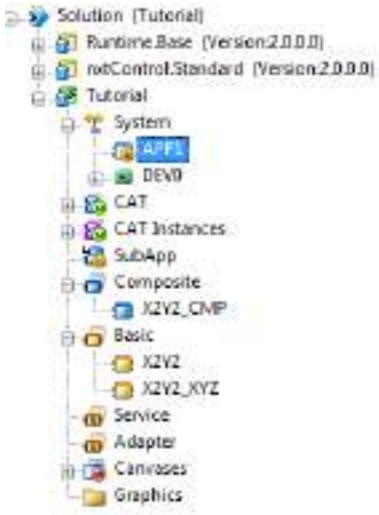


Рис. 14. Создание приложения.

Создание распределенного приложения FLASHER

Далее мы будем использовать существующую систему ФБ под названием *FLASHER*. Идея распределенной реализации *FLASHER* показана на рис. 15. Здесь, микропроцессор устройства ввода параметров считывает все входные события и значения параметров, а затем отправляет их на приемные устройства в сети. Другое устройство (устройство принятия решений) получает эти значения и генерирует значения *on*/*off* сигналов для каждого из светодиодов, которые также публикуются в сети. И, наконец, устройство вывода изображения, которое в данном случае представляет собой блок из четырех светодиодов со встроенным микропроцессором, принимает значения и отображает их.

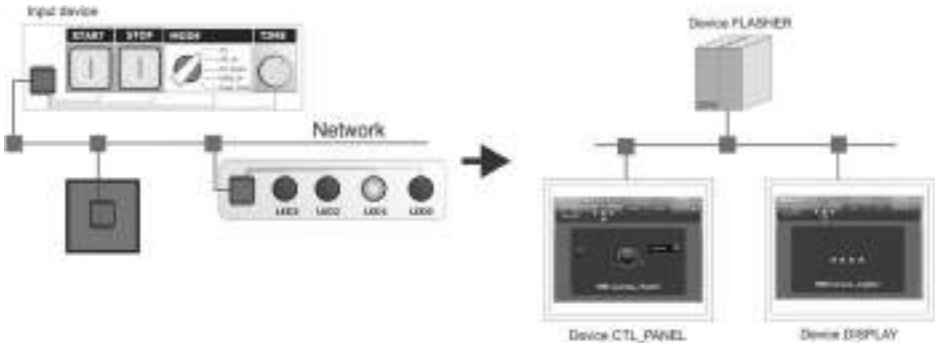


Рис. 15. Система FLASHER, реализованная на трех устройствах.

Рассмотрим детали создания конфигурации распределенной системы. В качестве отправной точки создадим копию централизованной системы FLASHER из материала курса и переименуем ее в FLASHERR (для этого необходимо просто вырезать и вставить папку FLASHER). Откроем решение FLASHERR и изменим архитектуру аппаратных средств, как показано на рис. 16.

Рис. 16. Устройства системы FLASHER_TEST3D: два локальных устройства (CTL_PANEL и DISPLAY) и удаленное устройство FLASHER

Для отображения функциональных блоков приложения на устройства, щелкаем на значок приложения (APP1) в дереве и следуем инструкциям из рис. 17: щелкнуть правой кнопкой мыши по ФБ, выбрать Mapping в выпадающем меню, а затем выбрать ресурс устройства для размещения этого ФБ.

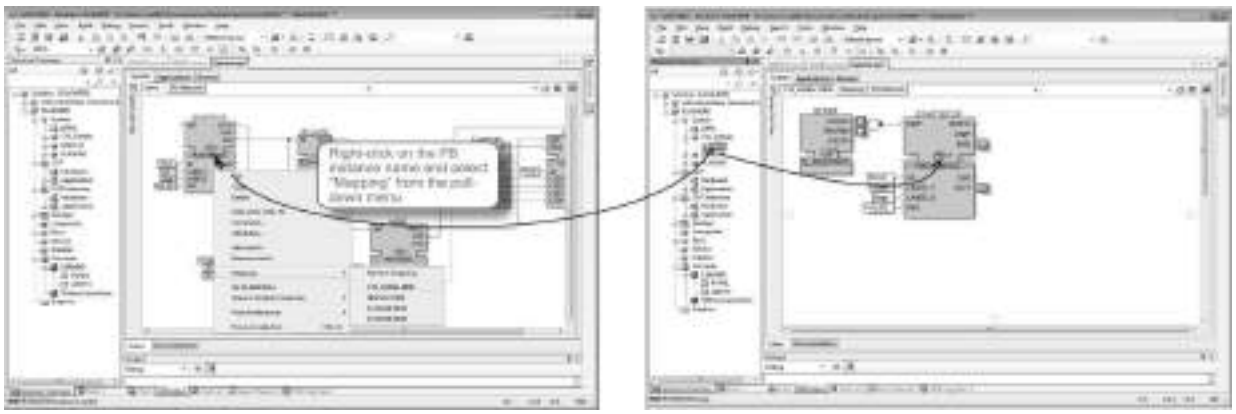


Рис. 17. Отображение функционального блока STARTSTOP на устройство CTL_PANEL

Следует отметить, что *NxtStudio* будет автоматически создавать соединение между двумя ФБ, если они отображаются на один и тот же ресурс, и будет вставлять коммуникационные блоки для ФБ, размещенных на различных устройствах. Это проиллюстрировано на рис.18.

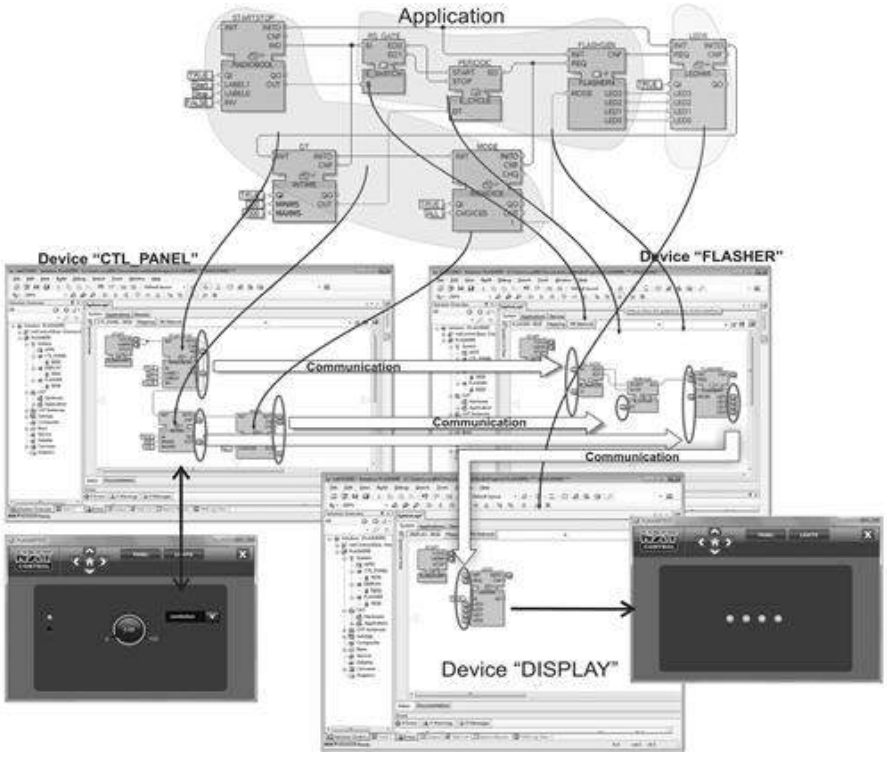


Рис. 18. Автоматическое добавление коммуникационных ФБ в устройства FLASHER и DISPLAY

Полная структура распределенной системы FLASHERR показана на рис. 19. Здесь устройство FLASHER (справа) не производит никаких видимых результатов (нет связей). На самом деле он генерирует выходные значения и посылает их на устройство DISPLAY, учитывая входные параметры, полученные от устройства CTL_PANEL.

Существует два коммуникационных потока: между устройствами CTL_PANEL и FLASHER, а также между устройствами FLASHER и DISPLAY.

Конфигурация системы FLASHERR может быть выполнена в сети из нескольких (1 или 2) подключенных к сети устройств.

В конфигурации системы на рис. 16, значение идентификатора (ID) менеджера устройства FLASHER установлено в *localhost*:*63499*. Эта установка используется, когда удаленное устройство выполняется на локальном хосте *localhost* (то есть, на том же компьютере, на котором выполняется *NxtStudio*). В противном случае, необходимо исправить эту настройку, заменив значение *localhost* на символическое имя или IP-адрес устройства, на котором будет запущена удаленная копия устройства. Таким образом, несколько удаленных устройств могут быть выполнены одновременно на локальном или удаленном компьютере.

Рис. 19 схематично показывает выполнение проекта *FLASHERR* на двух компьютерах, работающих в сети. Следует отметить, что можно купить дополнительную лицензию на выполнение копии устройства *NXT_RMTDEV* на другом компьютере. Вместо *NXT_RMTDEV* может использоваться более простой тип устройства *RMT_DEV*. Реализация этого типа устройства недоступна в *NxtStudio*, однако доступна в ИСР *FBDK*, имеющей среду выполнения *FBRT*, и в ИСР *4DIAC*, основанной на среде выполнения *FORTE*. Однако автоматическое отображение ФБ на устройство *RMT_DEV* возможно только в ограниченной области, так как эти ИСР реализуют только устройства Класса 1 (*Class 1*). При этом программный код для сетевого взаимодействия должен быть включен вручную и тип устройства

(*FLASHER.fbt*) должен быть указан на устройстве (с помощью *FBDK* или *4DIAC*).

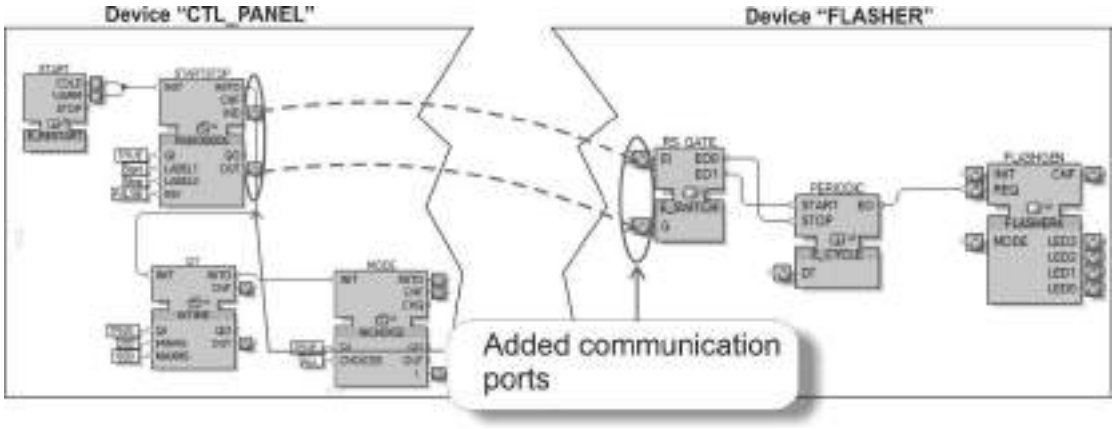


Рис. 19. Распределение приложения FLASHER на два устройства.

3.3. Развертывание и выполнение

Чтобы распределить (развернуть) решение на том же самом компьютере, надо следовать нижеприведенным шагам.

1) На панели *Devices* (Панель устройств) переключите *Active Network Profile* на *Local Test* (рис. 20).

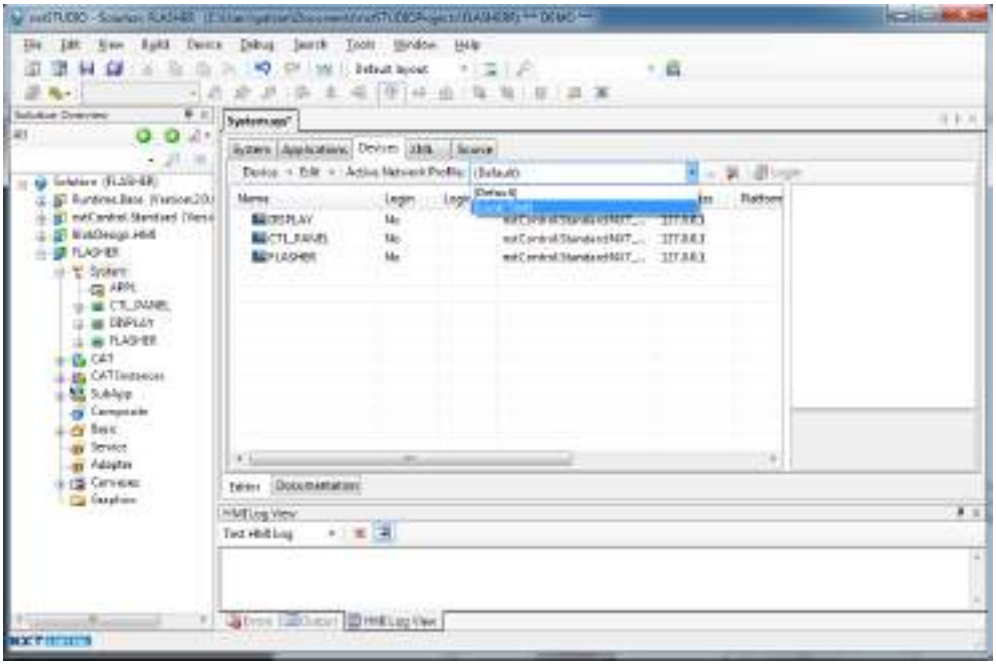


Рис. 20. Переключение на профиль Local Test

2) Установить состояние *Login* в используемых устройствах: *DISPLAY*, *CTL_PANEL* и *FLASHER*. Для этого перейдите на *Панель устройств* и используйте выпадающее меню, как показано на рис. 21. Это требуется для того, чтобы иметь возможность удаленной отладки и просмотра значений переменных процесса.

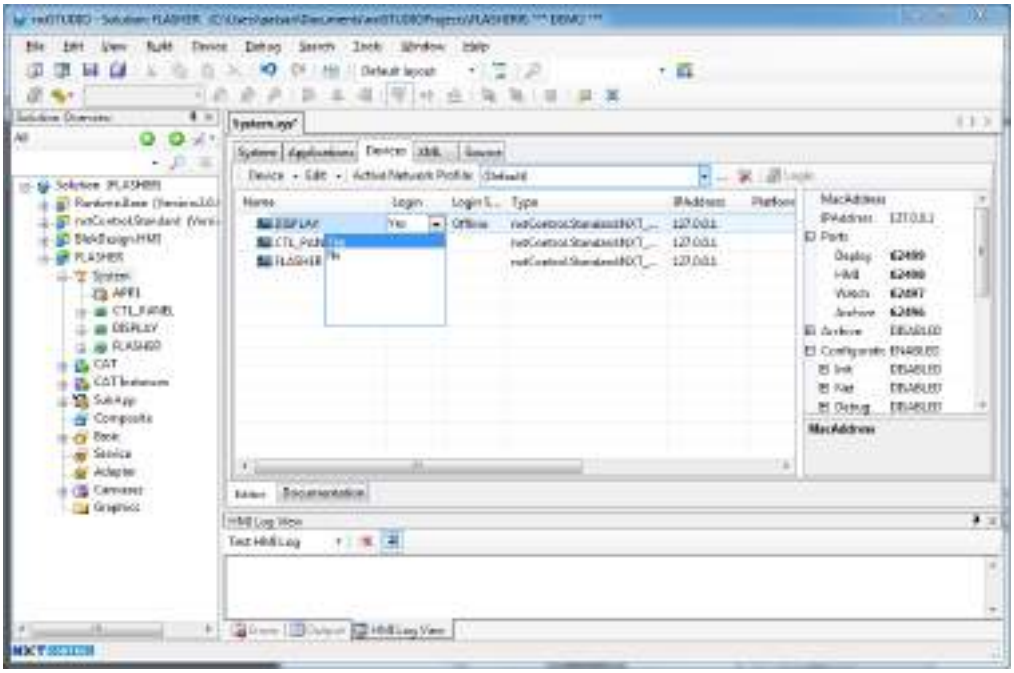


Рис. 21. Разрешение входа на устройство (Login -> Yes).

3) Убедитесь, что все устройства имеют разные номера портов (рис. 22). Обычно номера портов для локальных тестов устанавливаются системой автоматически, например, группа портов 61499, 61498, 61497, 61496 используется для устройства *CTL_PANEL*, группы портов 62499, 62498, 62497, 62496 используется для устройства *DISPLAY* и т.д.

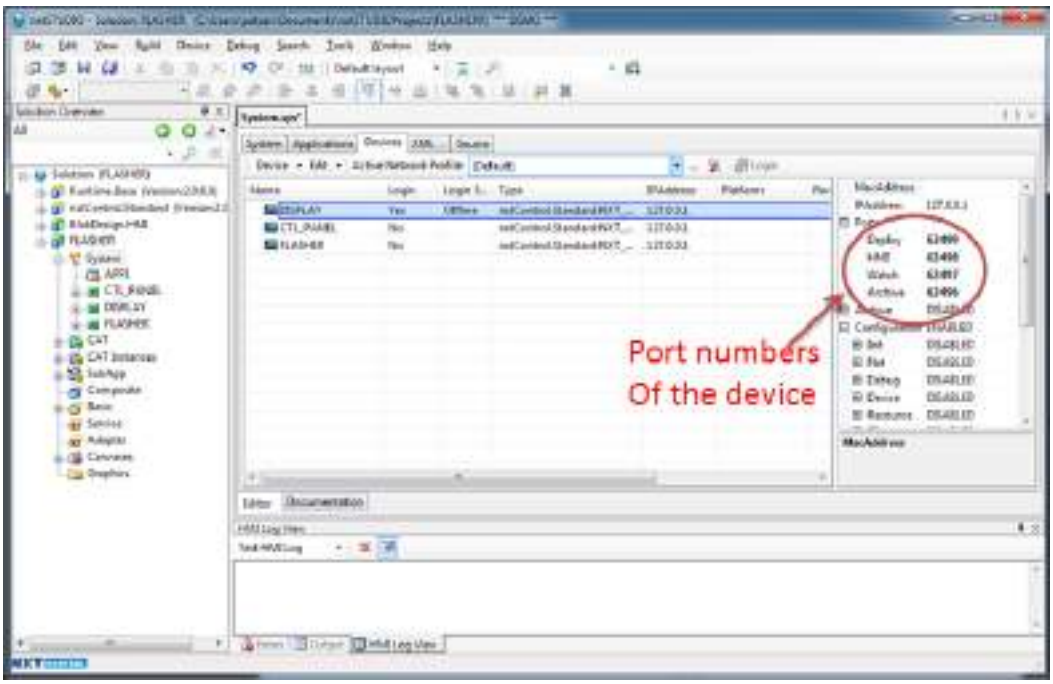


Рис. 22. Установка номеров портов устройств.

4) Выберите все устройства, представляющие интерес, например, *DISPLAY*, *CTL_PANEL* и *FLASHER*, и запустите программные контроллеры *SoftPLC*, выбрав соответствующий пункт в меню, вызываемом нажатием правой кнопки мыши (рис. 23). Это будет проявляться в трех окнах командной строки *Windows* с сообщениями среды выполнения *NxtForte* (рис. 24).

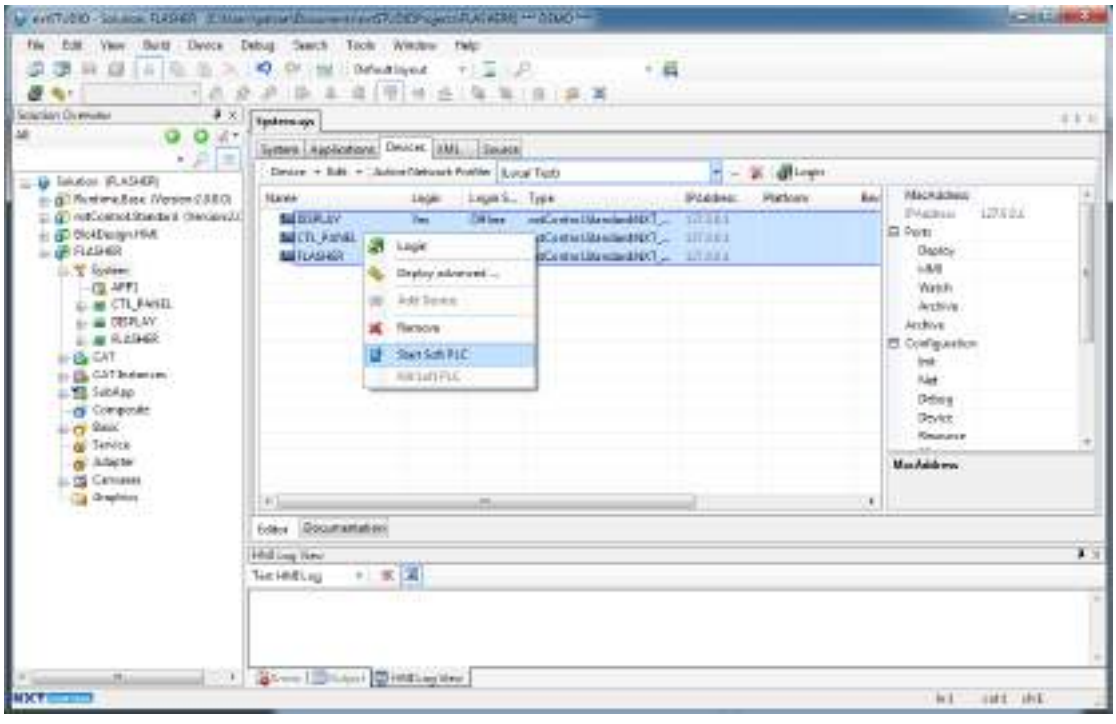


Рис. 23. Запуск программных контроллеров Soft-PLC

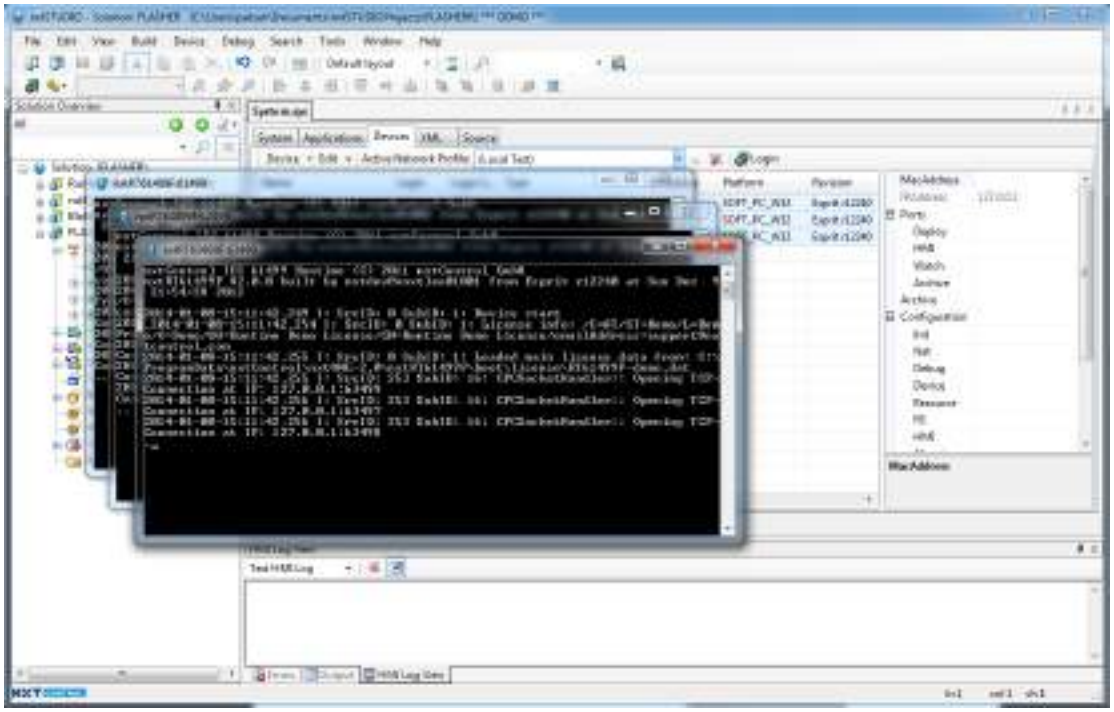


Рис. 24. Программный контроллер Soft-PLC в работе.

5) Разместите ФБ- приложения на запущенных устройствах. Для этого перейдите в диалог *Advanced Deployment* (рис. 25). Щелкните правой кнопкой мыши на каждом устройстве и выберите пункт меню *Delete persistent data*. Когда постоянные данные будут удалены, нажмите кнопку *Deploy*. Загрузка может оказаться неудачной, если постоянные данные не были удалены; иногда удаление должно быть выполнено во второй раз. Важно также удалить из устройства ранее загруженные ФБ-приложения, прежде чем загружать новое. Поэтому желательно выбрать опцию *Clear before deploy* во вкладке *Advanced*. Если эта опция установлена, система *NxtStudio* перед развертыванием будет всегда очищать ранее загруженные приложения.

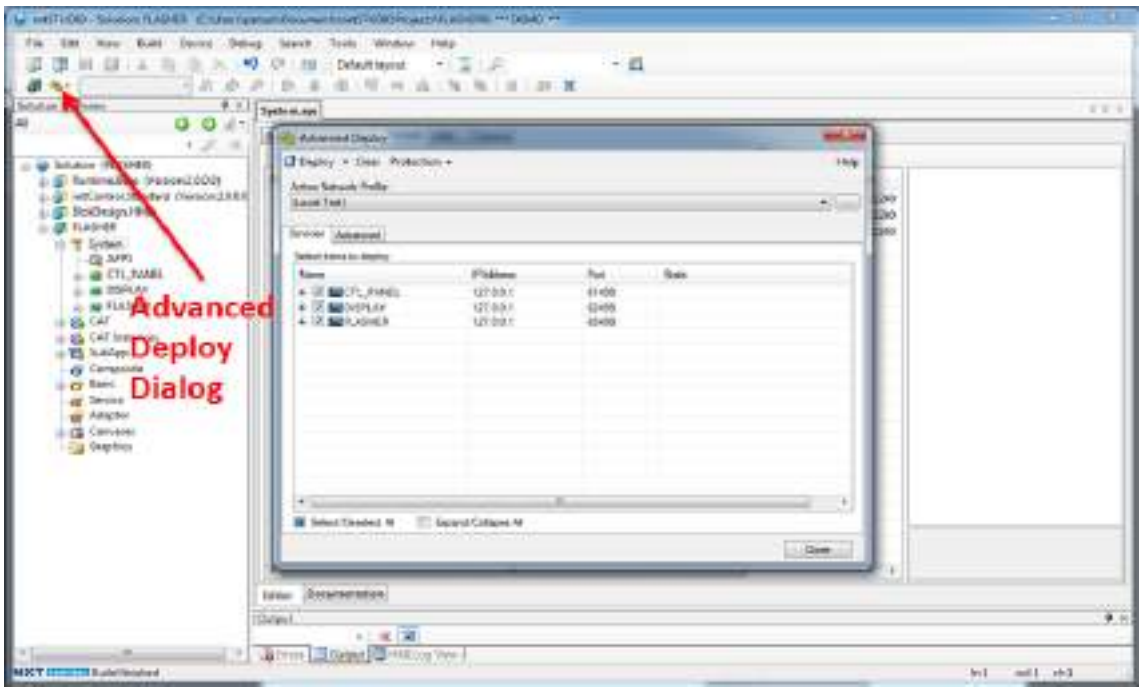


Рис. 25. Размещение систем ФБ на устройствах.

6) Войдите на запущенные устройства , нажав кнопку Login (рис. 26).

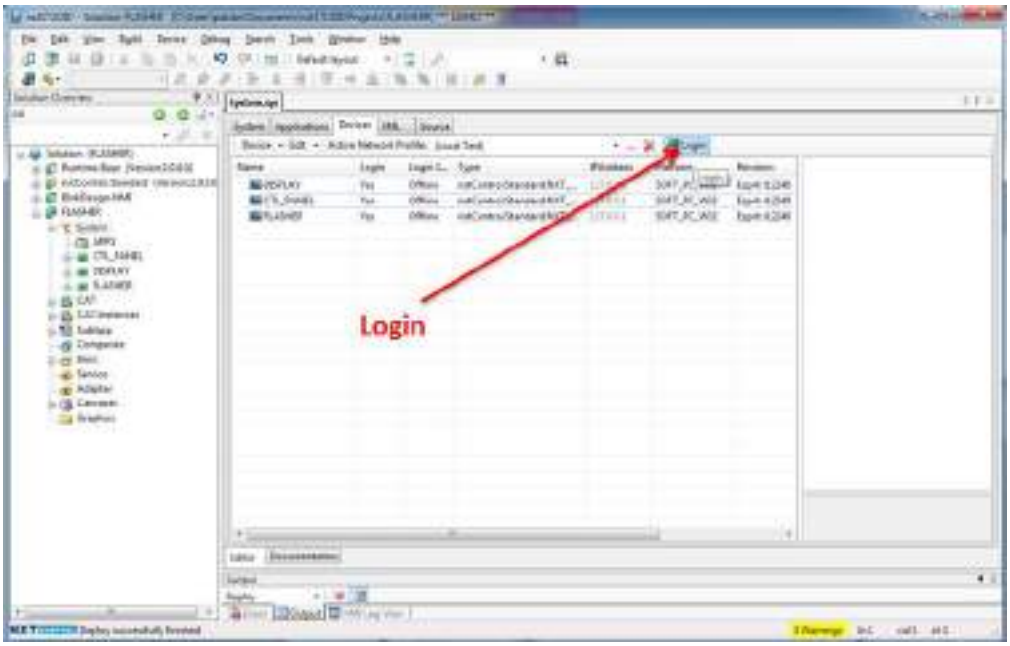


Рис. 26. Вход на стартовавшие устройства.

7) Запустите панель человеко-машинного интерфейса (HMI) (рис.27).

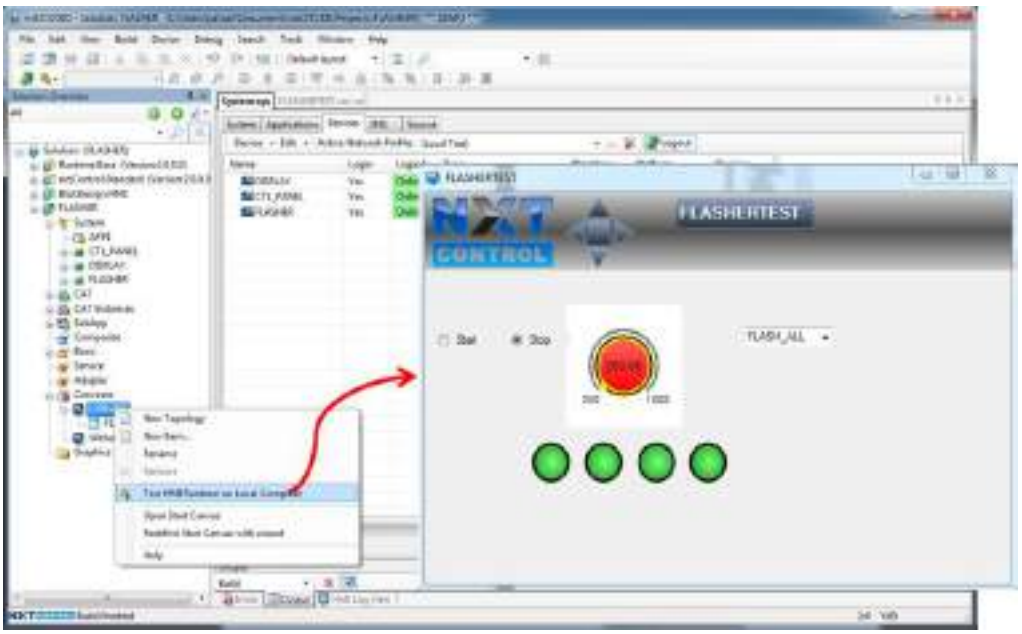


Рис. 27. Запуск панели человеко-машинного интерфейса (HMI).

8) Наблюдайте за переменными, кликая на них правой кнопкой мыши (рис. 28). Попробуйте разные опции в панели HMI и посмотрите, как изменяются переменные в устройствах (рис. 29).

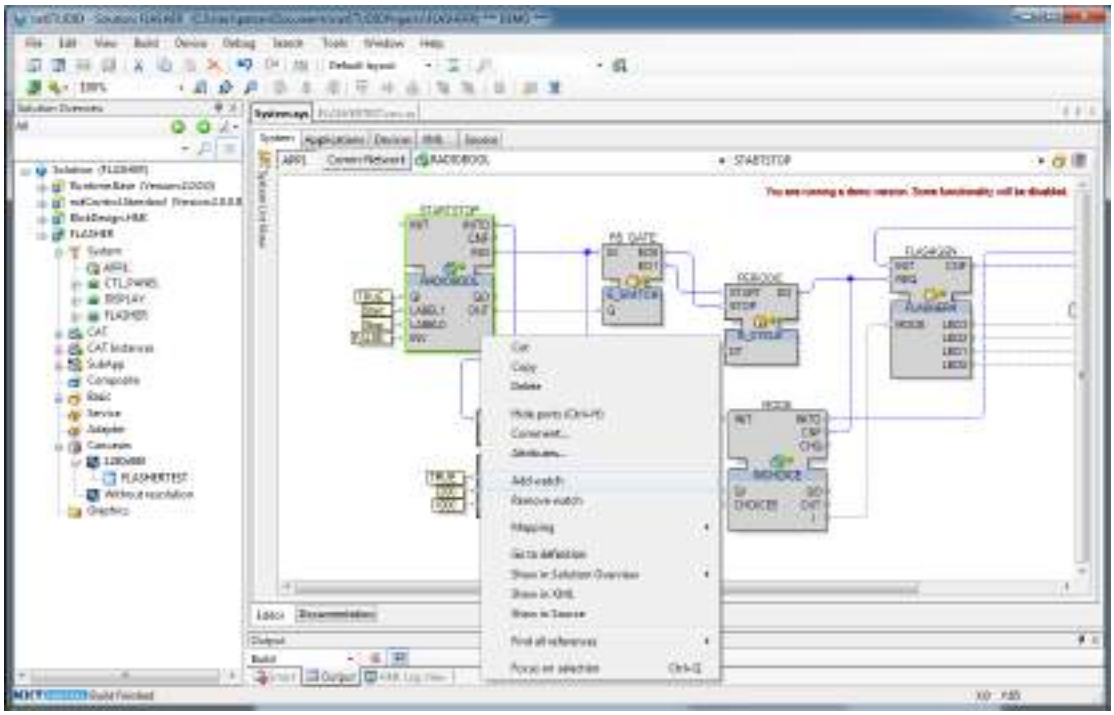


Рис. 28. Установка наблюдаемых переменных.

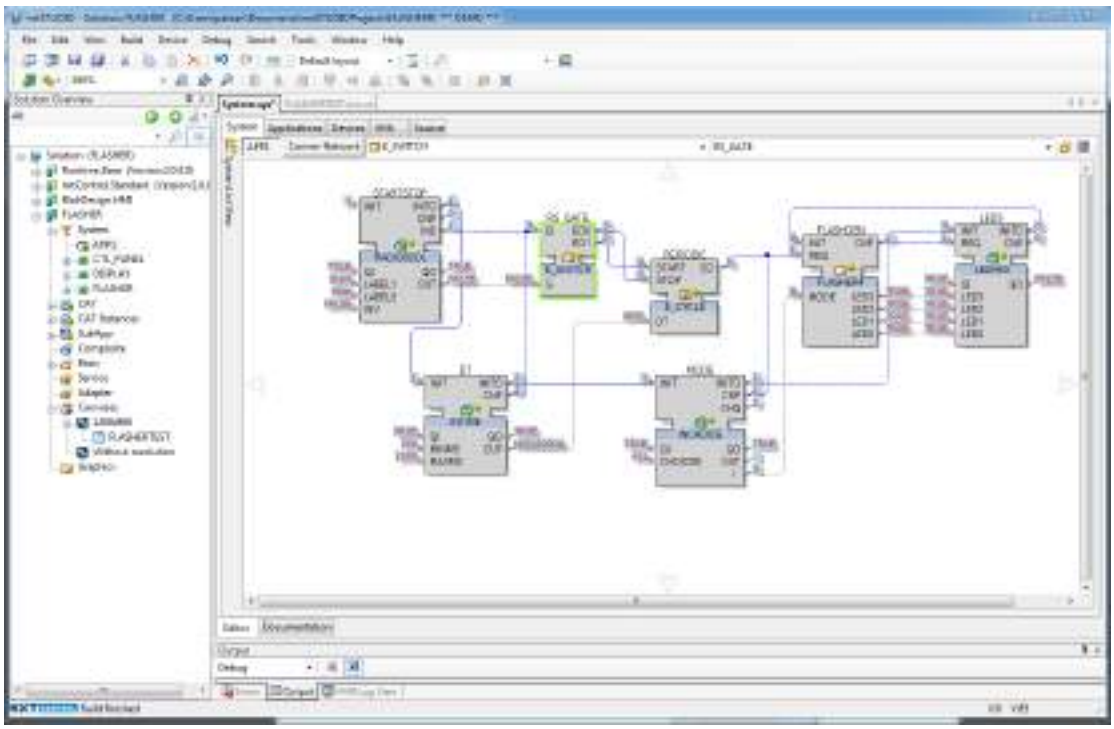


Рис. 29. Наблюдение за значениями переменных.

Лабораторная работа № 2
Разработка CAT-блоков

Цель работы – получить навыки разработки: а) CAT-блоков в среде NxtStudio; б) имитационной модели в виде функционально-блочного приложения, включающего CAT-блоки, с продвинутым человеко-машинным интерфейсом (HMI).

1. Разработка интерфейса «человек-машина» и связанной с ним сети функциональных блоков

В данном разделе работы речь пойдет о создании человеко-машинного интерфейса (HMI) в виде CAT-блока для объекта автоматизации «Вертикальный цилиндр».

1) Откройте проект, названный PnP2.

2) Правой кнопкой мыши щелкните на CAT-> Application и выберите New Item, затем задайте имя VCylinder для создаваемого CAT-приложения (рис. 1).

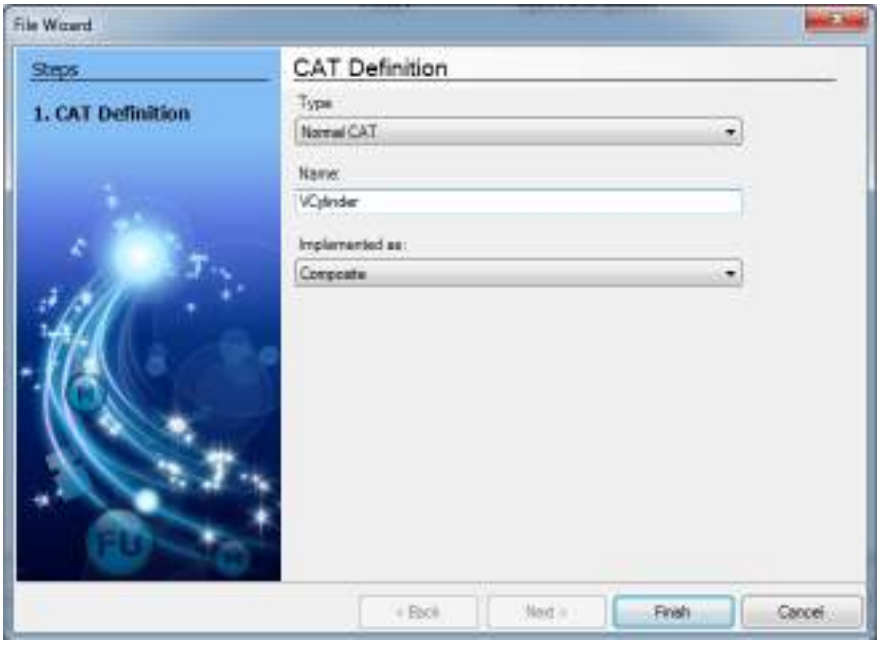


Рис. 1. Добавление нового CAT-приложения

3) Правой кнопкой мыши щелкните на только что созданном элементе CAT, выбирая VCylinder-> IThis:VCylinder_HMI, далее выберите Add HMI Symbol, назовите его HMI и удалите другой объект с именем sDefault (рис. 2).

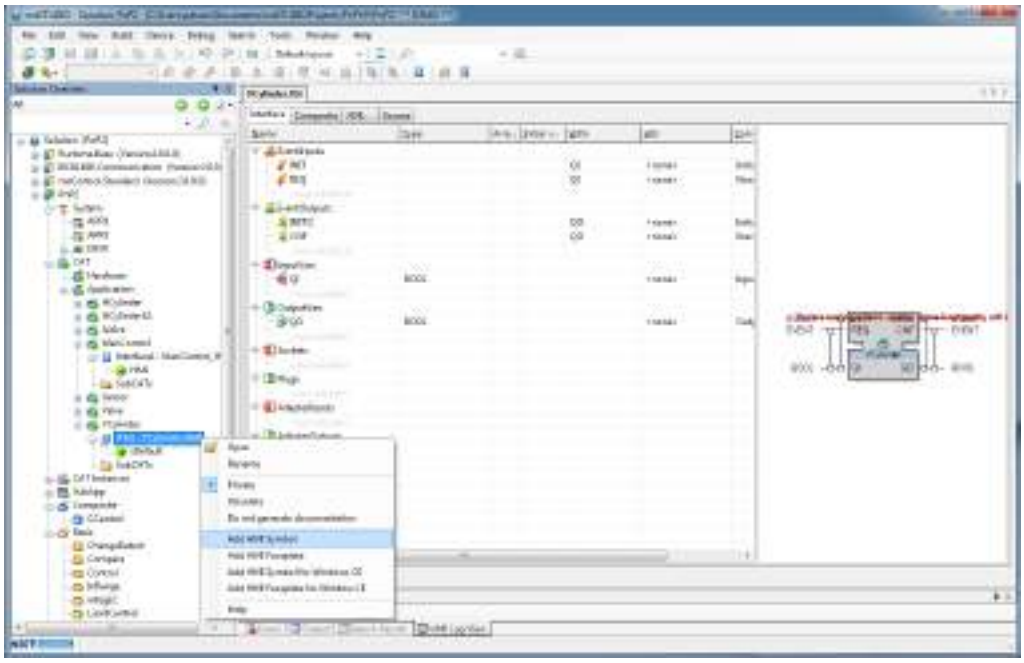


Рис. 2. Добавление символа HMI

4) Измените его интерфейс, как показано ниже на рис. 3.

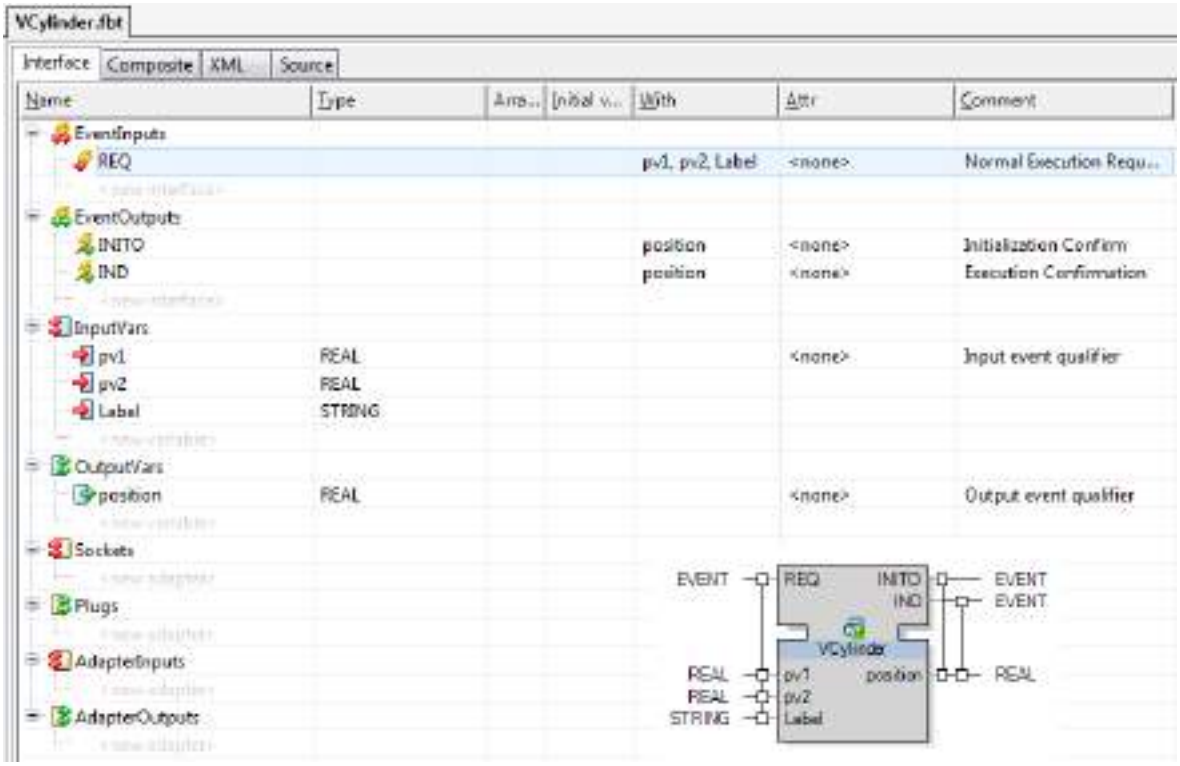


Рис. 3. Интерфейс VCylinder

5) Теперь откройте VCylinder_HMI и измените его интерфейс так, как показано на рисунке ниже.

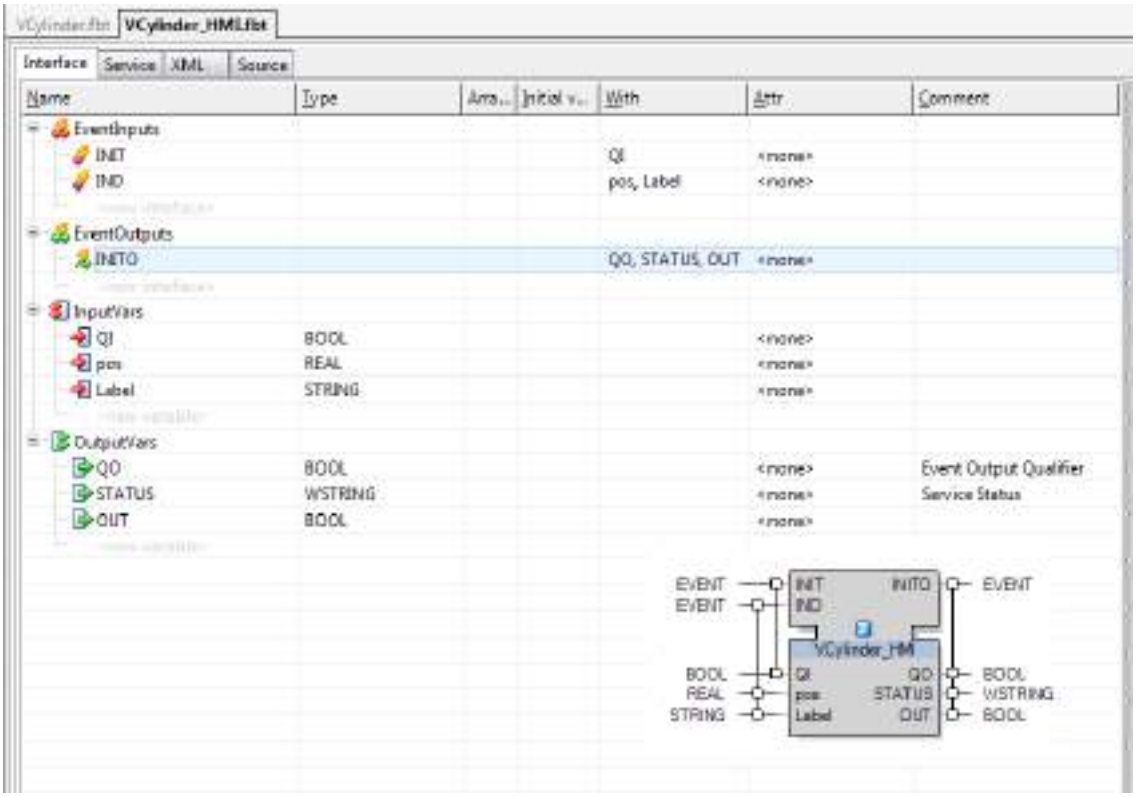


Рис. 4. Интерфейс VCylinder_HMI

6) Вернитесь на вкладку VCylinder, выберите Composite и разработайте составной блок, как показано на рис. 5, причем блоки integEC и ChangeDetect могут быть найдены в пункте Basic для базисных ФБ. VCylinder_HMI – это блок, который вы редактировали на предыдущем шаге. Эти три блока можно перетащить из дерева решений (Solution Overview) в рабочую область редактора (Canvas). Другие два блока: E_RESTART и E_CYCLE можно найти, щелкнув правой кнопкой мыши в рабочей области и выбрав FB-> Project: Runtime Base-> IEC61499.Standard.

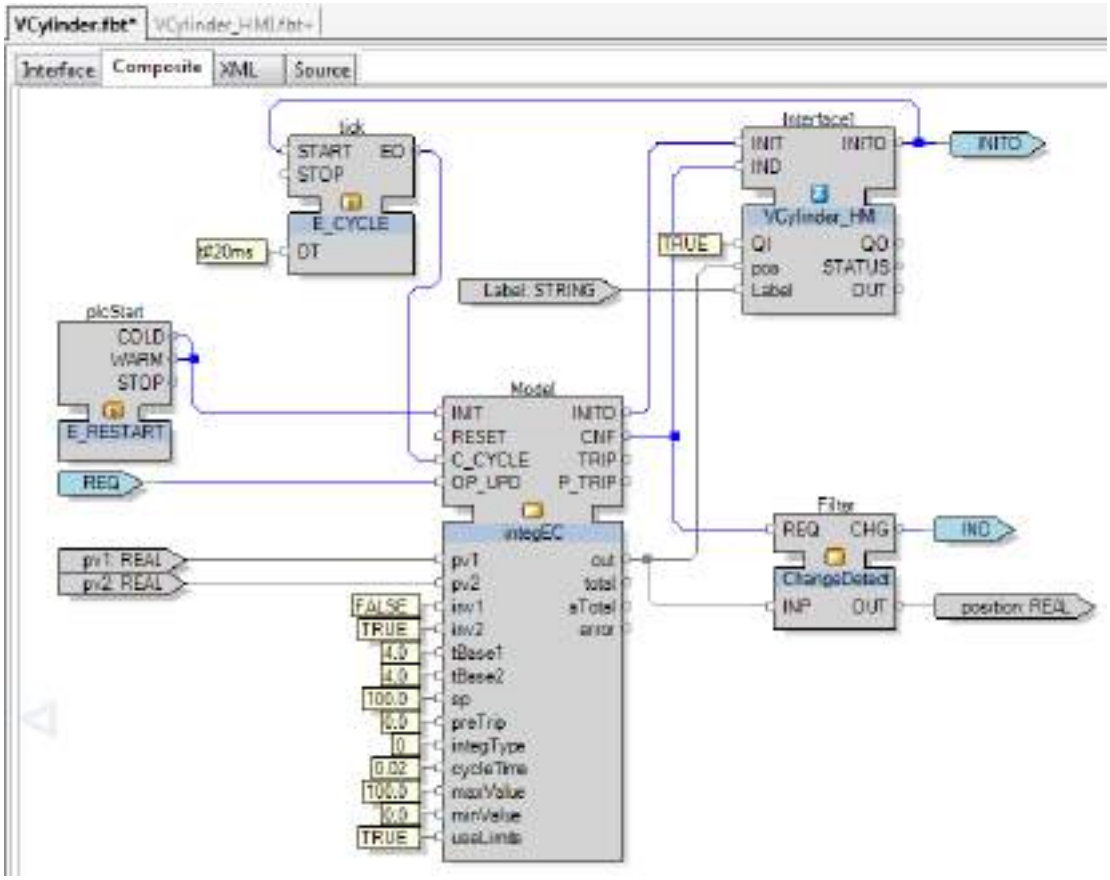


Рис. 5. Проектирование сети ФБ для VCylinder

7) Дважды щелкните на VCylinder-> VCylinder_HMI-> HMI, чтобы открыть рабочую область HMI. Откройте окно свойств, щелкнув на меню View системы и выбрав опцию Properties в нем. Измените размер рабочей области на 161,437 (рис. 6).

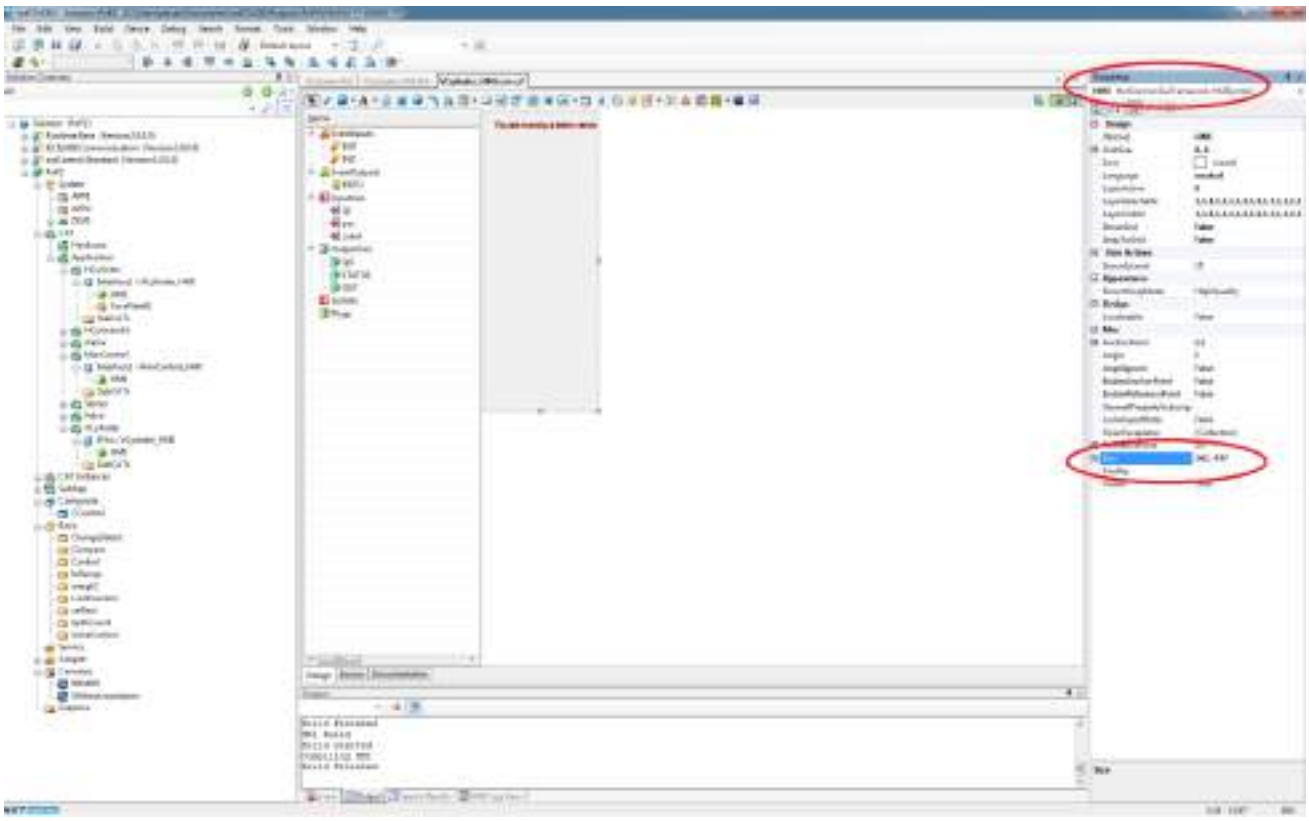


Рис. 6. Изменение размера рабочей области на 161,437

8) Теперь приступим к созданию объекта Цилиндр (Cylinder). Выберите из панели инструментов для конструирования фигур, которая находится между вкладками и рабочей областью, обозначение прямоугольника и нарисуйте произвольный прямоугольник, как показано на рис. 7. Измените свойства ‘Pen’, ‘Location’ и ‘Size’ в соответствии с рис. 7, оставив другие свойства по умолчанию.

Рис. 7. Создание Цилиндра (шаг 1)

9) Придадим фигуре градиентный цвет. Чтобы сделать это, нужно нажать на кнопку выбора, которая находится за пунктом Brush (Кисть) в окне свойств, и выбрать значения, как показано на рис. 8.

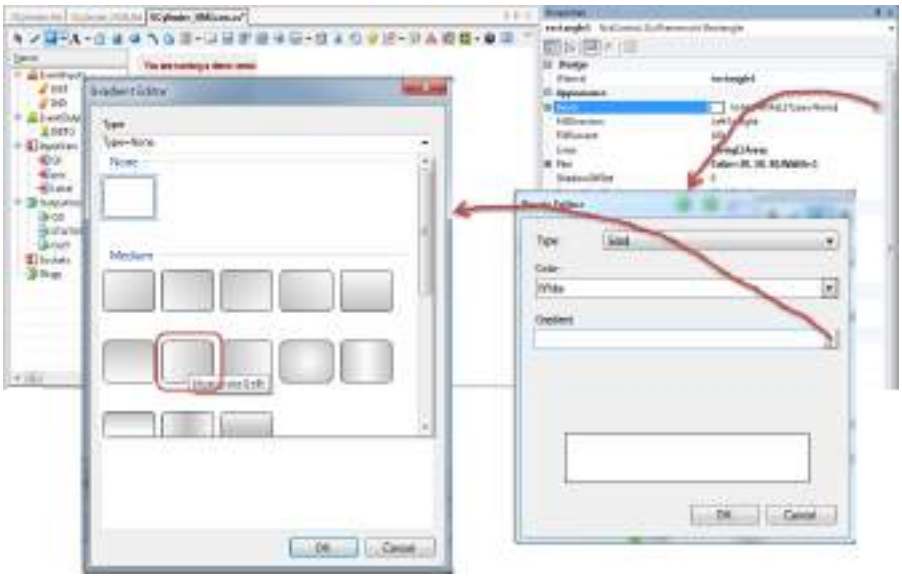


Рис. 8. Создание Цилиндра (шаг 2)

10) Установите флажок Use Color Blend. Используйте три точки установки, установите значения этих точек как показано на рис. 9.

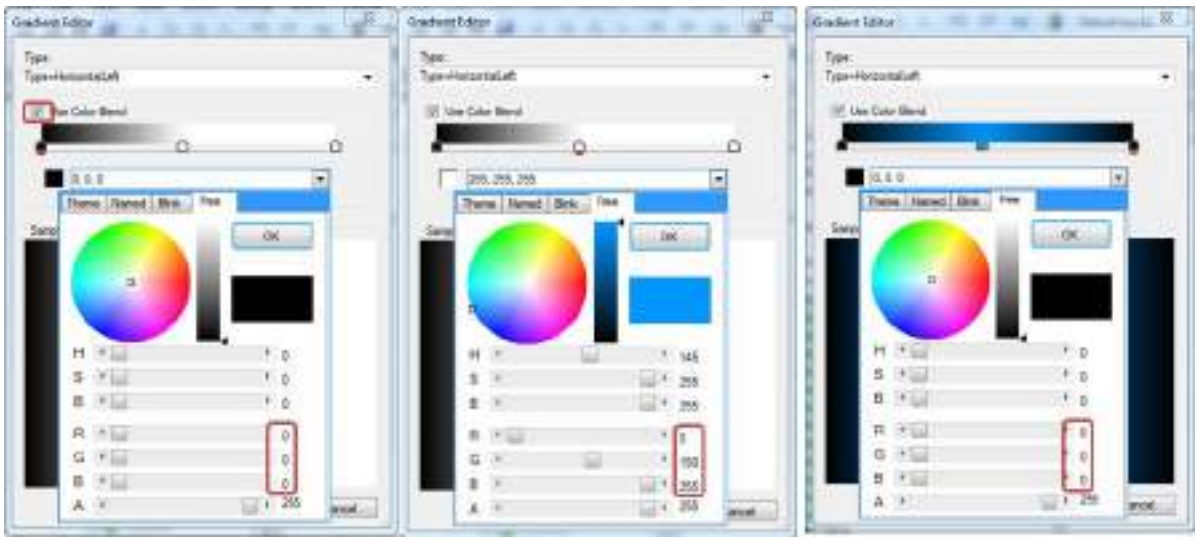


Рис. 9. Создание Цилиндра (шаг 3)

11) Должен получиться прямоугольник, представленный на рис. 10.

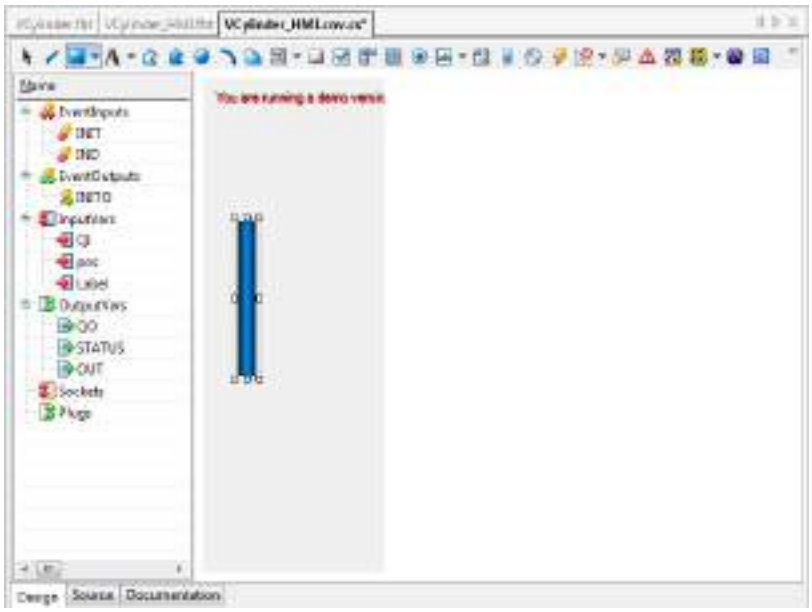


Рис. 10. Создание Цилиндра (шаг 4)

12) Два раза скопируйте и вставьте созданные прямоугольники и измените значения параметров, как показано на рис. 11.

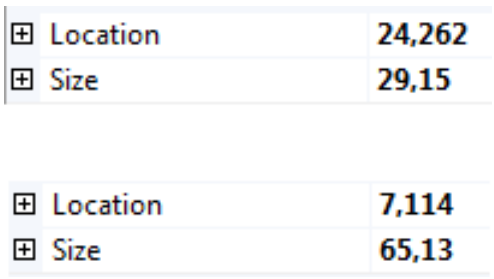


Рис. 11. Создание Цилиндра (шаг 5)

13) Должна получиться фигура, показанная на рис. 12.

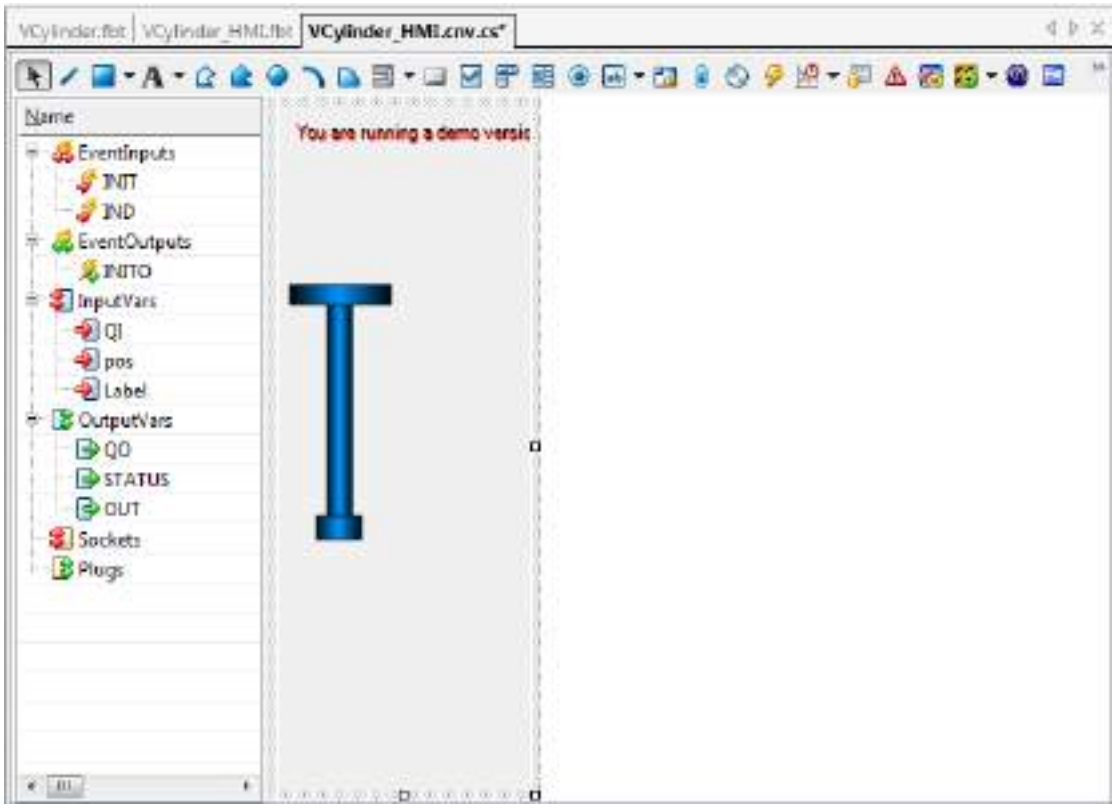


Рис. 12. Создание Цилиндра (шаг 6)

14) Создайте прямоугольник с закругленными краями, как показано на рис. 13.

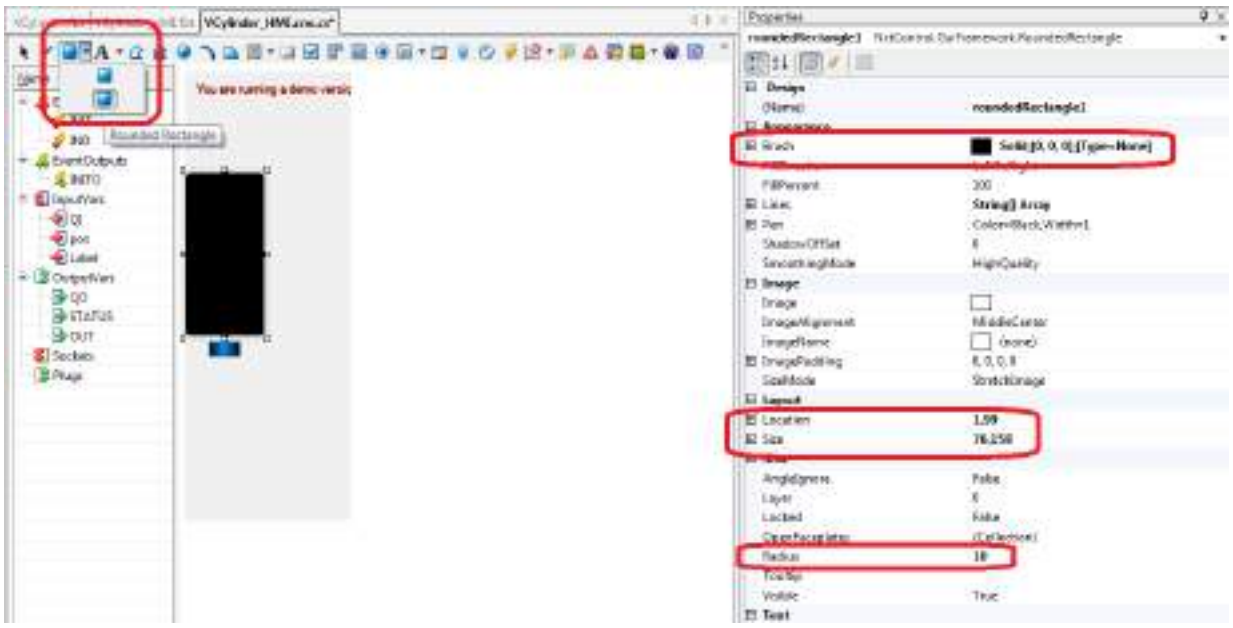


Рис. 13. Создание Цилиндра (шаг 7)

15) Теперь нужно поместить новый прямоугольник на задний план (рис. 14).

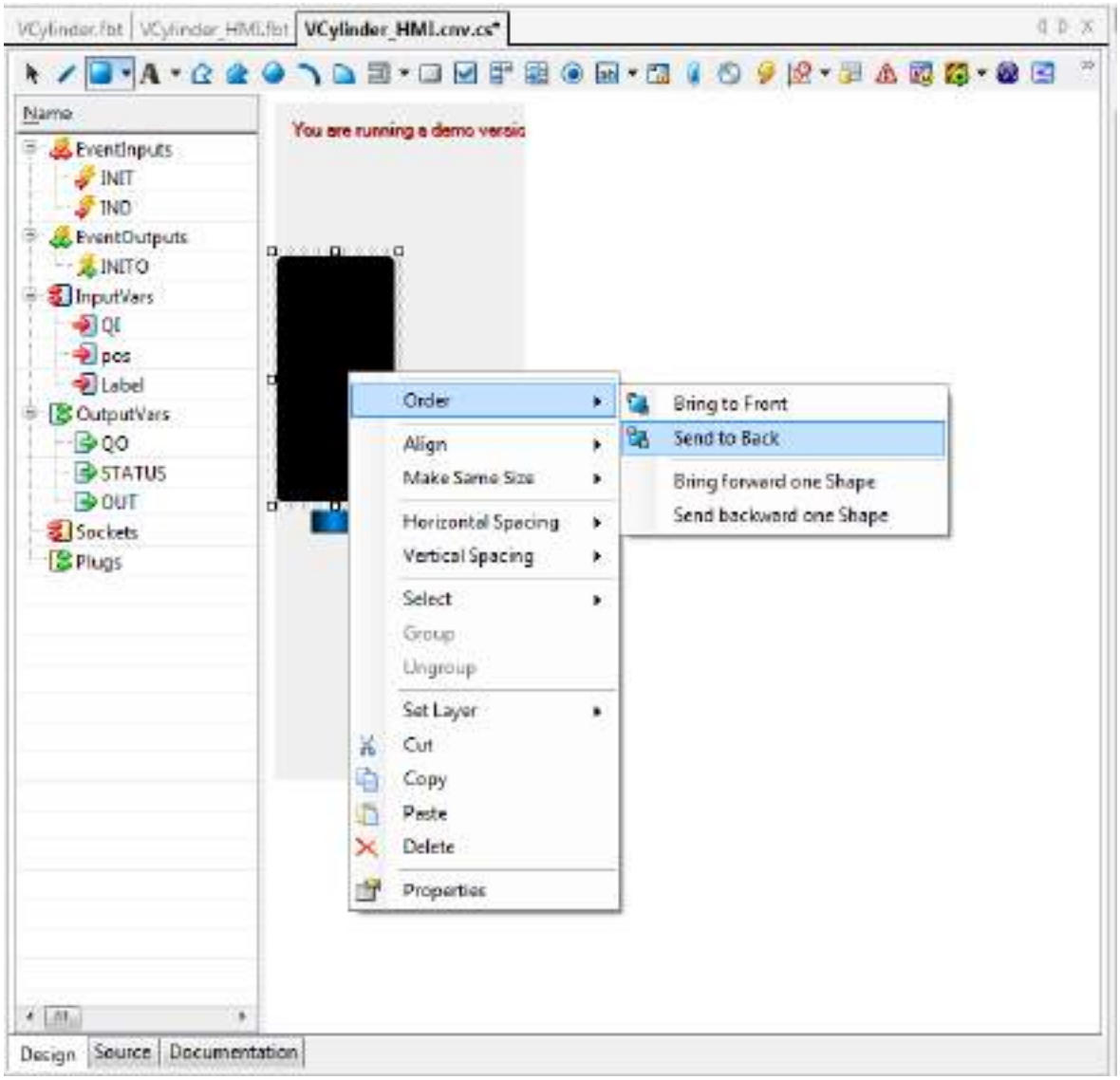


Рис. 14. Создание Цилиндра (шаг 8)

16) С использованием технологии «Скопировать-Вставить» создайте другой прямоугольник и измените его свойства в соответствии с рис. 15.

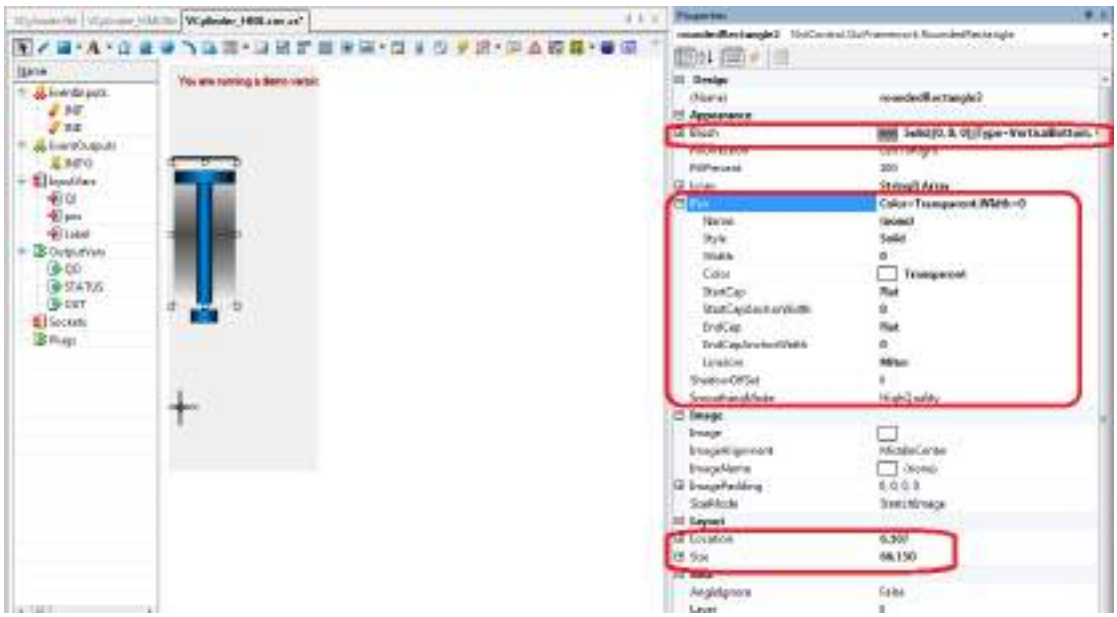


Рис. 15. Создание Цилиндра (шаг 9)

17) Настройте инструмент «Кисть», как показано на рис. 9, но с некоторыми изменениями. Вместо HorizontalLeft (горизонтальный левый) установите VerticalBottom (вертикальный нижний). Будет пять точек установки со значениями:

a. 255, 255, 255
b. 154, 154, 154
c. 78, 78, 78
d. 154, 154, 154
e. 255, 255, 255

Рис. 16. Создание Цилиндра (шаг 10)

18) Нам нужно переместить новый прямоугольник назад на три слоя. Для этого три раза выберем пункт меню Send backward one Shape (рис. 17).

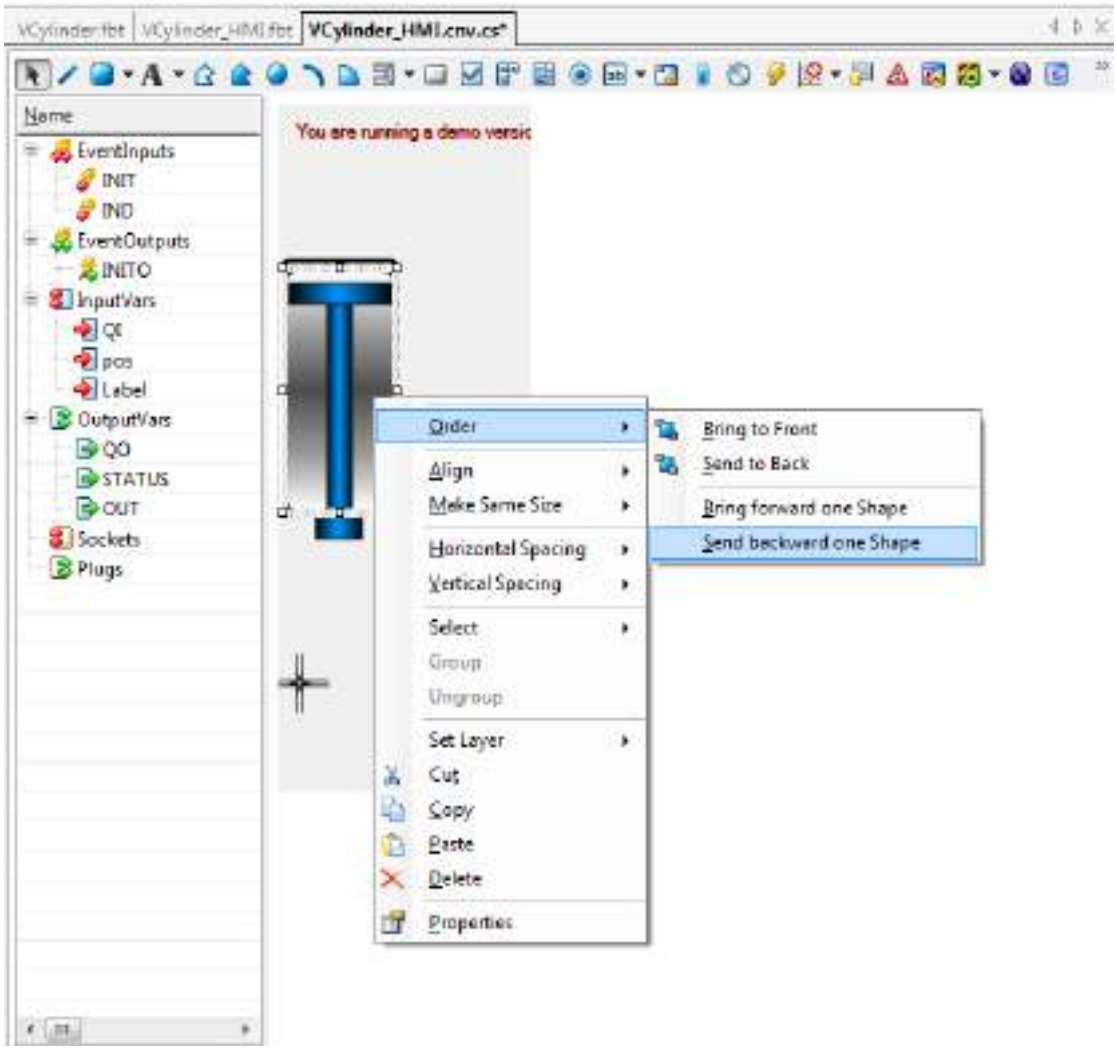


Рис. 17. Создание Цилиндра (шаг 11)

19) Выберем последние прямоугольники с закругленными краями и сгруппируем их. Для этого выделим эти фигуры, щелкнем на них правой кнопкой мыши и выберем пункт Group. Затем поместим группу на задний слой (примерно так же, как показано на рис. 14).

20) Аналогично создадим еще два прямоугольника с закругленными краями, делая и вставляя копию последнего прямоугольника, попутно устанавливая значения свойств. Тип будет HorizontalLeft вместо VerticalBottom, точки установки будут следующими:

a. 255, 255, 255
b. 234, 22, 30
c. 178, 14, 18
d. 234, 22, 30
e. 255, 255, 255

Радиус будет равным ‘7’, а размеры будут равны 66, 9. Положение (Location) для первой фигуры будет 6, 245, а второй - 6, 105.

Цилиндр должен выглядеть так, как показано на рис. 18.

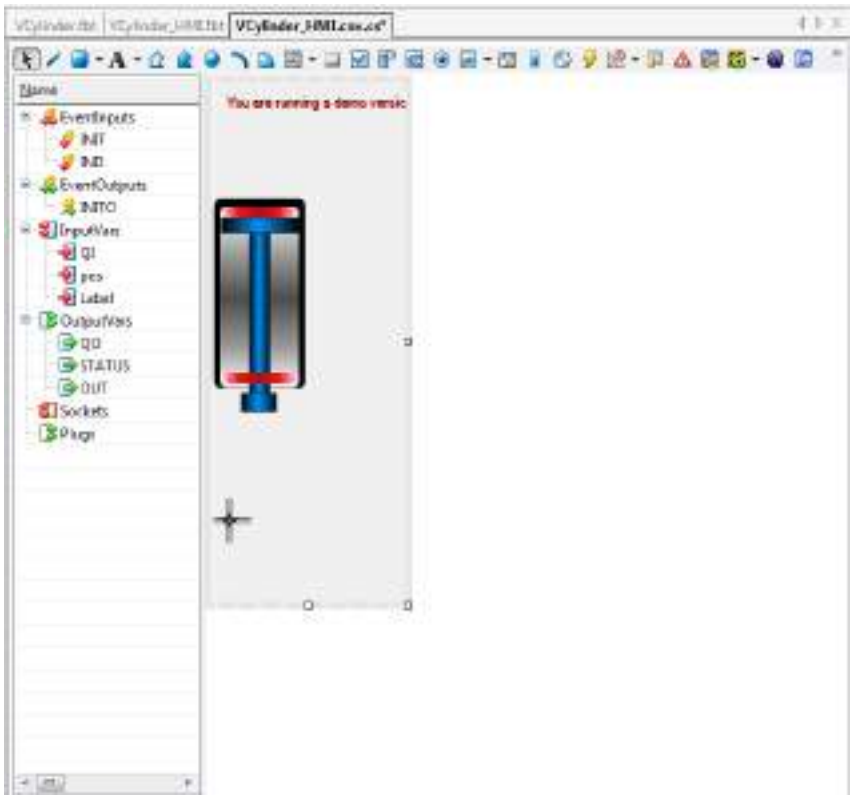


Рис. 18. Создание Цилиндра (шаг 12)

22) Последним элементом для комплексной фигуры «Цилиндр» будет многоугольник (Polygon). Создадим его, как показано на рис. 19. Настроим «Кисть» следующим образом: тип – HorizontalLeft, значения пяти точек установки:

a. 255, 255, 255
b. 114, 114, 114
c. 210, 210, 210
d. 114, 114, 114
e. 255, 255, 255

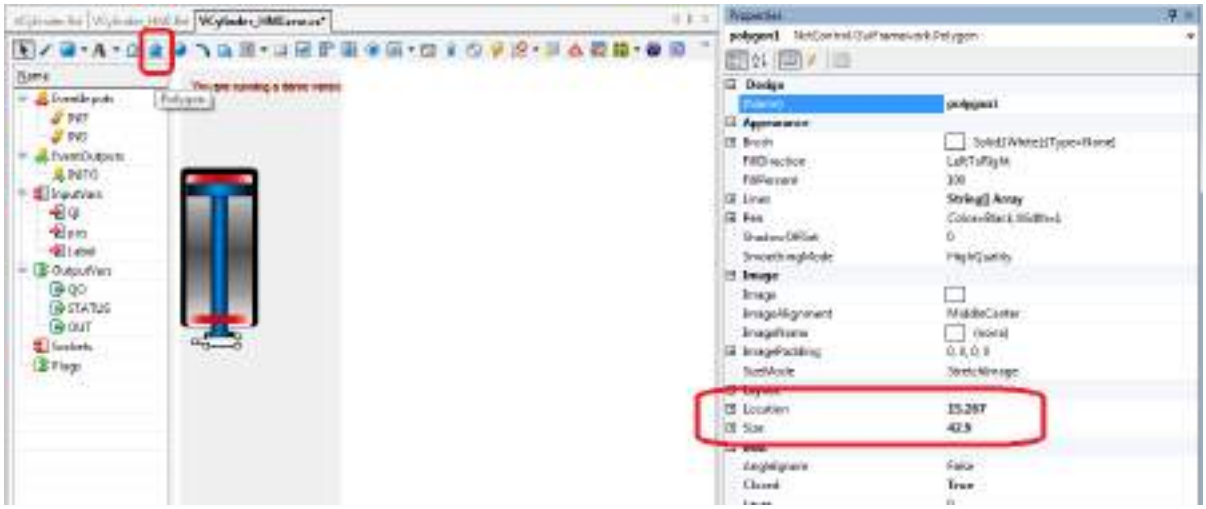


Рис. 19. Создание Цилиндра (шаг 13)

23) Теперь сгруппируем три прямоугольника с синим градиентом, два - с красным и один многоугольник в одну группу. После группирования Цилиндр будет выглядеть как на рис. 20.

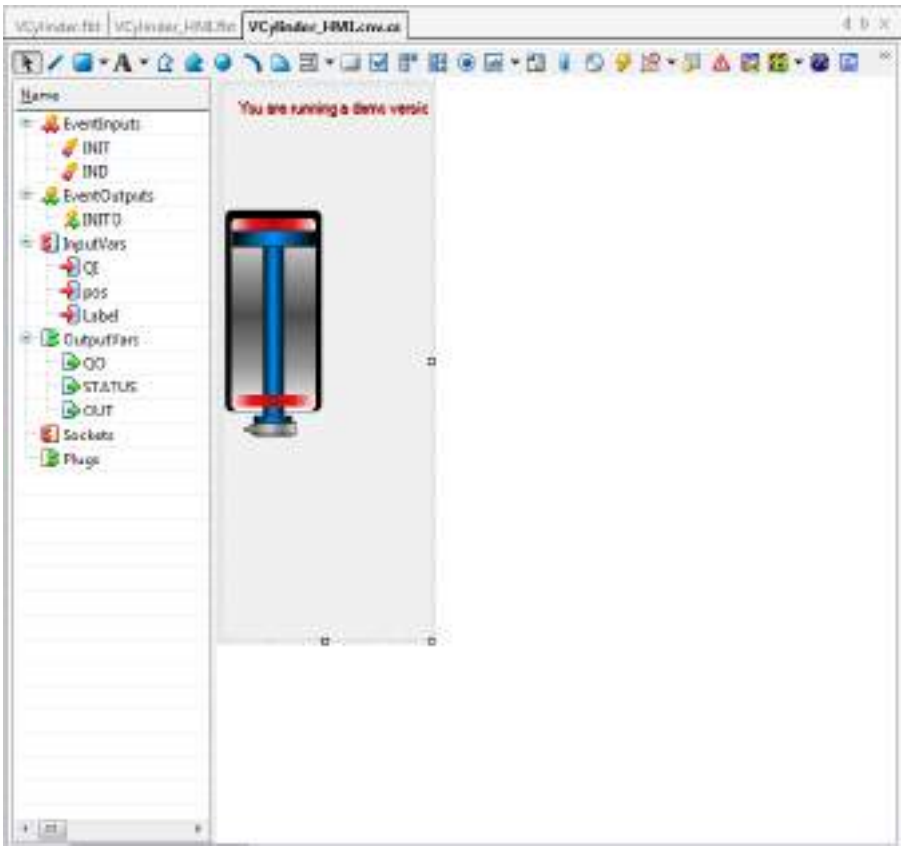


Рис. 20. Создание Цилиндра (шаг 14)

24) Создадим Метку (Label) и назовем ее Name со свойствами, показанными на рис. 21.

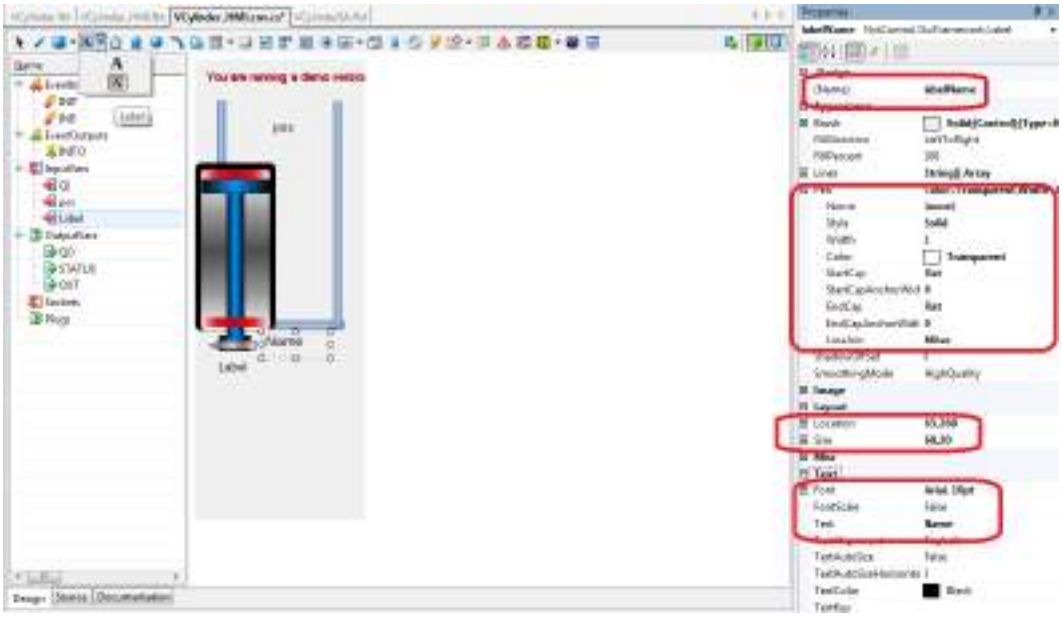


Рис. 21. Создание Цилиндра (шаг 15)

25) Остались последние три элемента для человеко-машинного интерфейса. Необходимо создать Трубу (Pipe) - рис. 22 показывает, как это сделать.

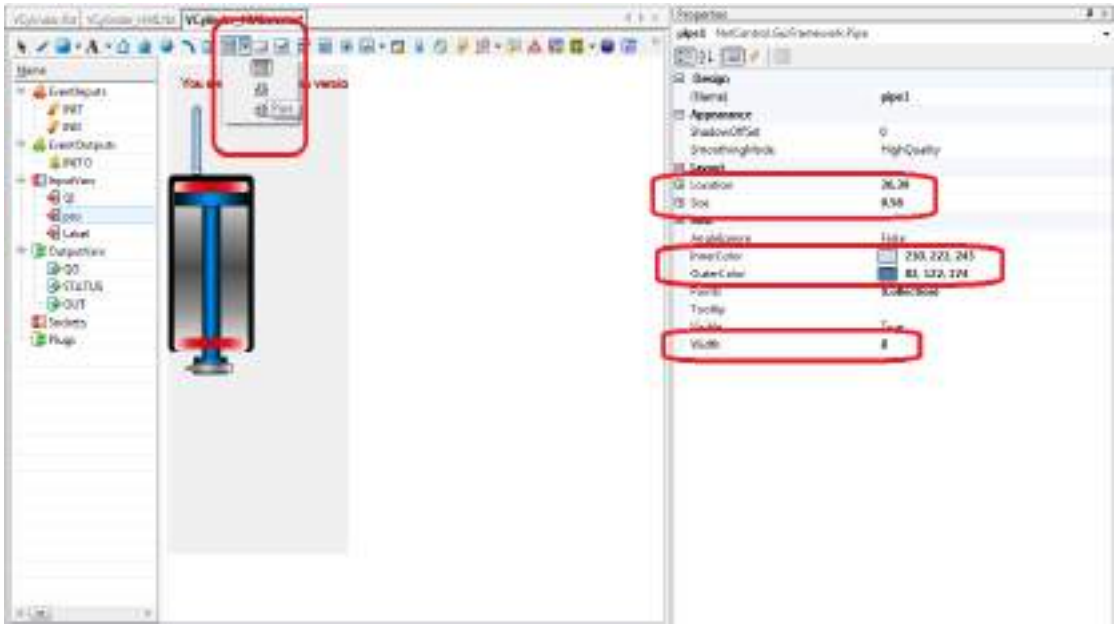


Рис. 22. Создание Цилиндра (шаг 16)

26) Аналогичным образом создадим еще две трубы Pipe1 и Pipe2 со следующими отличиями:

a. Pipe1
    i. Location – 136, 40

ii. Size – 0, 211

b. Pipe2

i. Location – 77, 252

ii. Size – 60, 0

27) Теперь создание человеко-машинного интерфейса завершено, он должен выглядеть так, как показано на рис. 23.

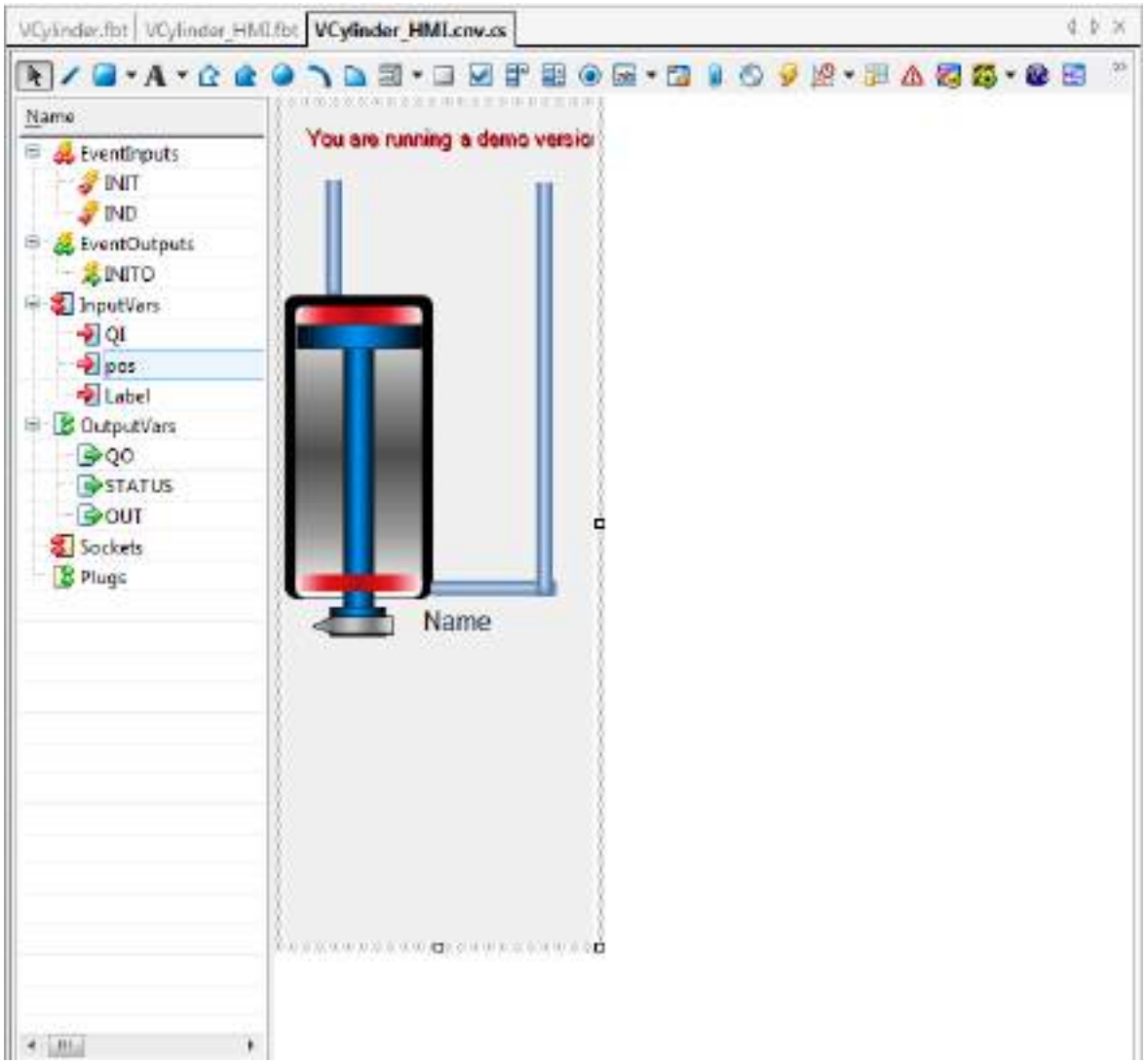


Рис. 23. Создание Цилиндра (шаг 16)

28) Чтобы облегчить программирование (кодирование), которое будет рассмотрено в следующем разделе, переименуем некоторые элементы. Группу, которая была создана в пункте 23, переименуем в grpInnerPart. Верхний красный прямоугольник переименуем в isMin, а нижний - в isMax.

2.2. Добавление логики в CAT HMI

Существуют два входа, на которые должен реагировать HMI: pos и Label. Для того, чтобы добавить функциональность для обработки изменения информации на этих входах, нужно проделать следующее.

Перетащите pos в рабочую область редактора. Когда Вы это сделаете должно возникнуть диалоговое окно, как показано на рис. 24. Выберите опцию Execute.

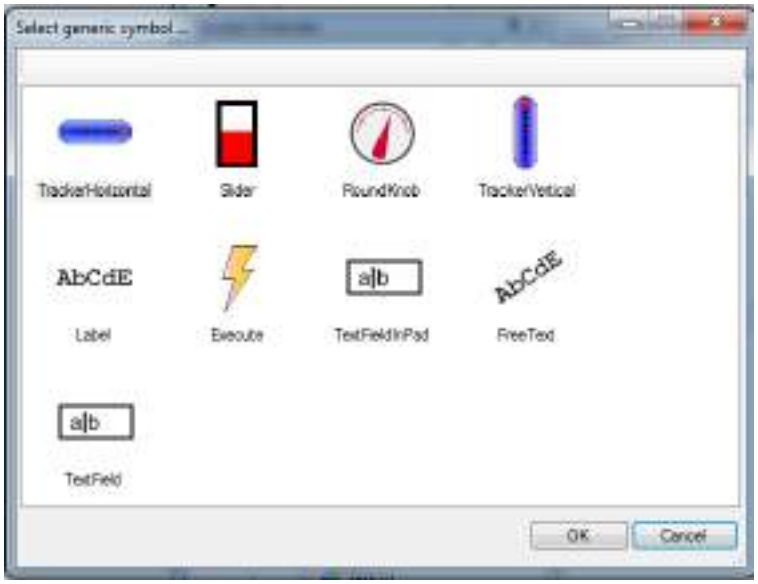


Рис. 24. Логика CAT (шаг 1)

Дважды щелкните мышью на только что добавленном элементе pos. Перед вами будет открыт редактор кода языка C#. Нажмите Yes в возникшем окне. Появится окно, приведенное на рис. 25. Для переключения представлений используйте вкладки Design и Source.

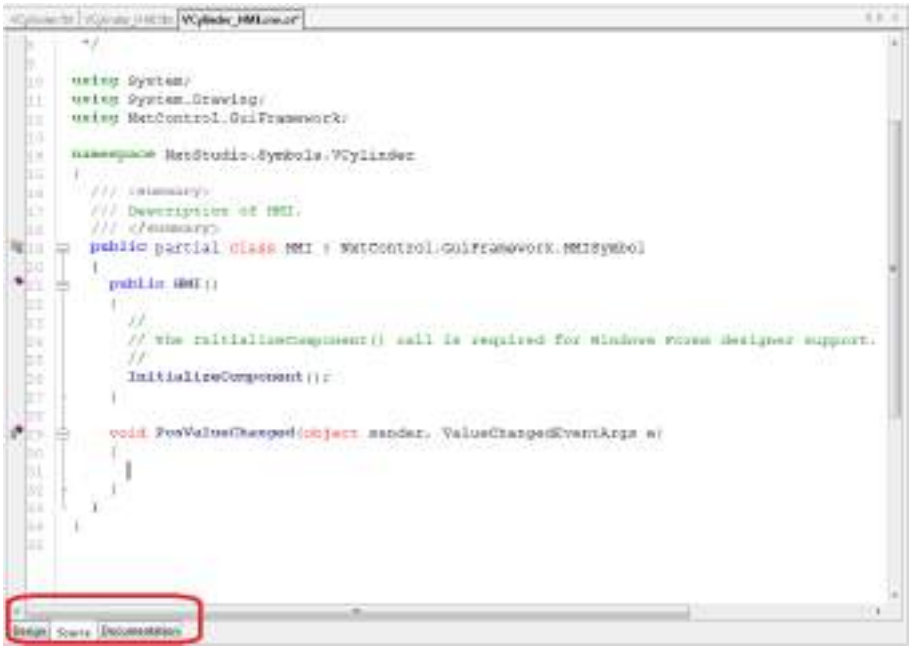


Рис. 25 Логика CAT (шаг 2)

Добавьте следующий код в функцию *PosValueChanged*:

```
NxtControl.Drawing.PointF newPos = grpInnerPart.Location;
          float position = (float)e.Value;
             double relpos = 115.0d/100.0d * position +
115.0d;
             if (relpos > 220.0d) {
                  isMax.Visible = true;
                  isMin.Visible = false;
             } else if (relpos < 125.0d) {
                  isMax.Visible = false;
                  isMin.Visible = true;
             } else {
                  isMax.Visible = false;
                  isMin.Visible = false;
             }
             newPos.Y = relpos;
             grpInnerPart.Location = newPos
```

Выполните похожие действия для входа Label. Код для функции *LabelValueChanged* будет следующим:

```
labelName.Text=(string)Label.Value;
```

2.3. Составной CAT-блок

Внимательно изучите CAT-блок *HCylinderSA* («Горизонтальный цилиндр»). Нам необходимо создать данный блок по аналогии с ранее созданным CAT- блоком «Вертикальный цилиндр». Для этого необходимо выполнить следующие действия.

1) Создайте новый элемент CAT и назовите его VCylinderSA.

2) Раскройте этот элемент и переименуйте sDefault в HMI.

3) Правой кнопкой мыши щелкните на элементе SubCATs и добавьте CAT-блок Vcylinder, созданный ранее, как показано на рис. 26.

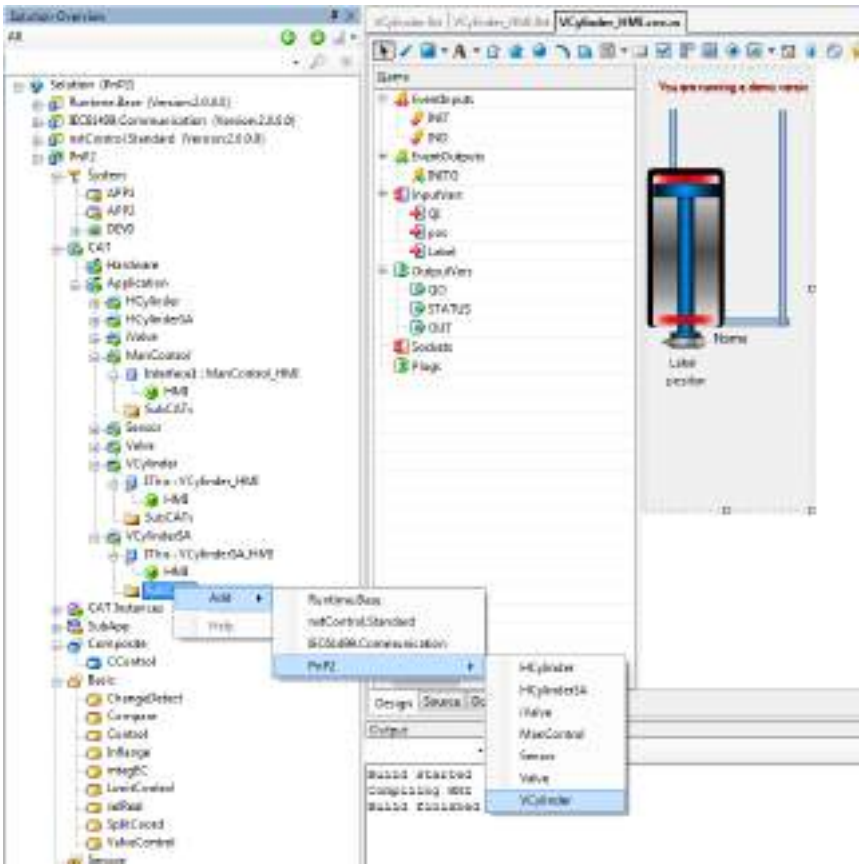


Рис. 26. Создание составного CAT-блока (шаг 1)

4) Аналогично добавьте два экземпляра CAT-блока Valve («Клапан») и два экземпляра CAT-блока Sensor («Датчик»). В итоге объект SubCATs после добавления в него пяти CAT-блоков должен выглядеть, как показано на рис. 27. Следует отметить, что добавленные на рис. 27 объекты были

переименованы. Их имена по умолчанию были VCylinder, Sensor1, Sensor2, Valve1 и Valve2.

Рис. 27. Создание составного CAT-блока (шаг 2)

5) Откройте CAT-блок VCylinderSA и измените его интерфейс так, как показано на рис. 28.

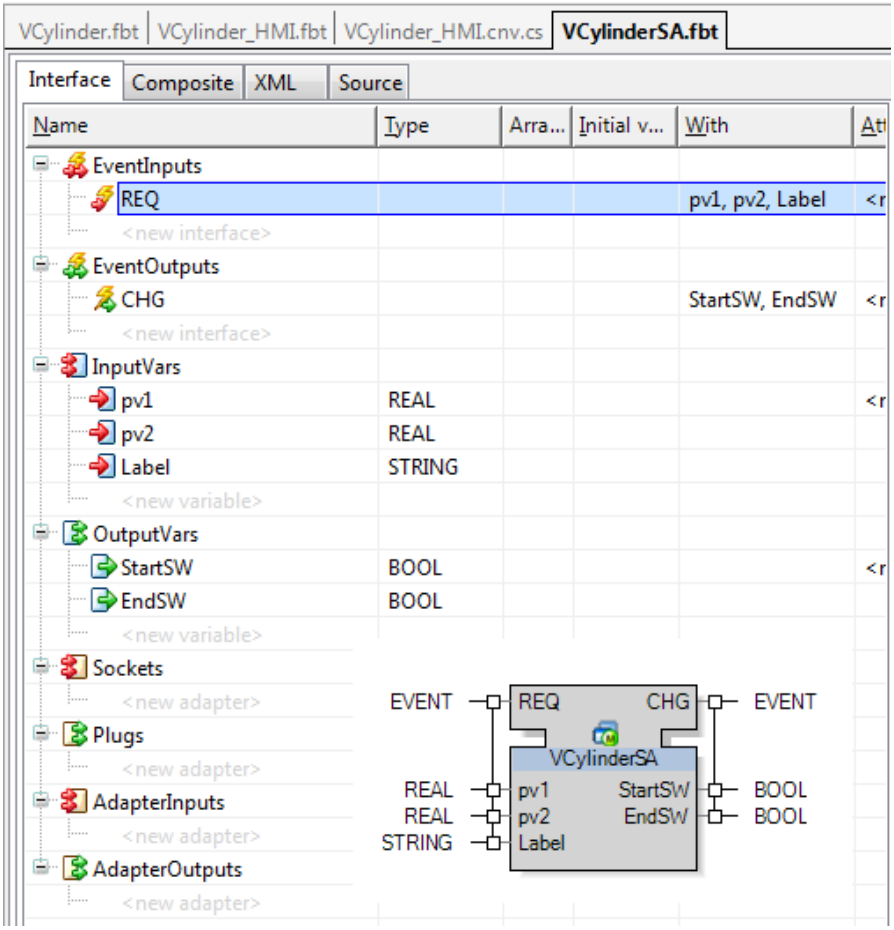


Рис. 28. Создание составного CAT-блока (шаг 3)

6) Переключитесь на вкладку Composite и проведите связи, как показано на рис. 29. Имейте в виду, что блоки нужно перетащить из секции SubCATs.

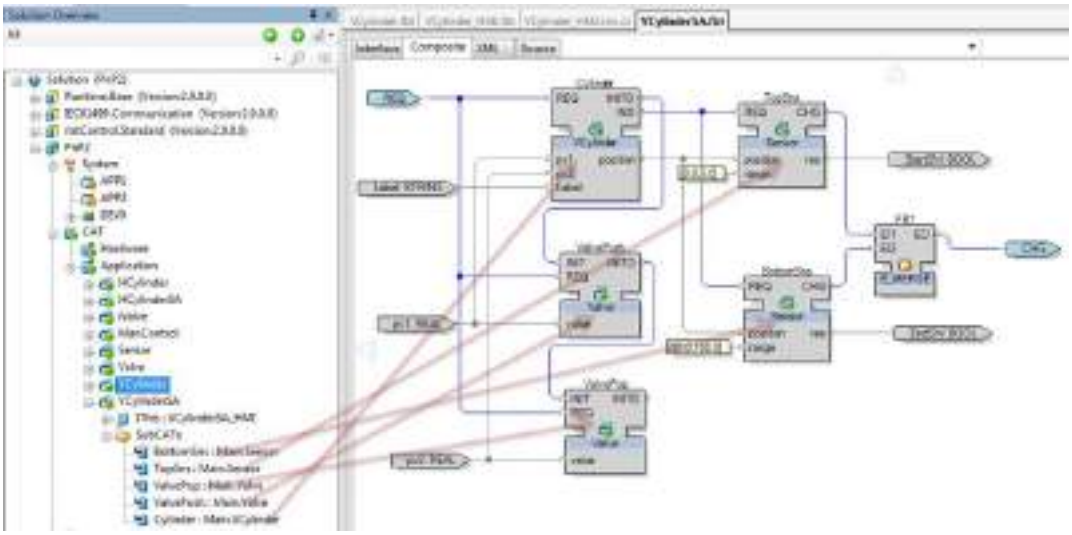


Рис. 29. Создание составного CAT-блока (шаг 4)

Теперь дважды кликните на HMI-части VCylinderSA и, так же как и на предыдущем шаге, перенесите все элементы из секции SubCATS в рабочую область редактора, и используйте следующие параметры:

Блок	Свойства
Actual Canvas (имеющее имя HMI в окне свойств)	Size -> 228, 441
Cylinder	Location-> 0,39
TopSns	Location-> 80,114
BottomSns	Location-> 80,227
ValvePush	Location-> 5,10
ValvePop	Location-> 118,10

8) И, наконец, создайте прямоугольник со следующими свойствами (рис. 30).

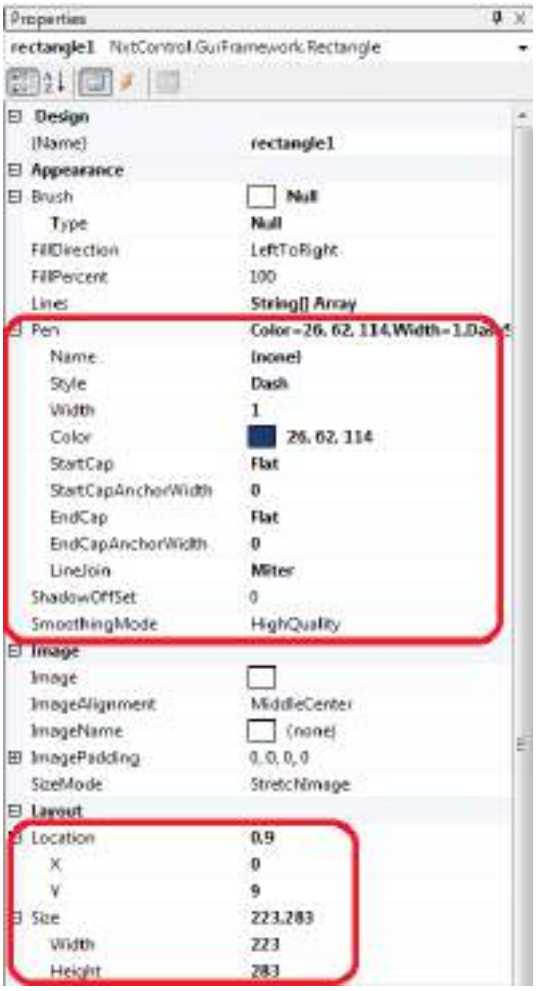


Рис. 30. Создание составного CAT-блока (шаг 5)

9) Окончательный вариант должен выглядеть, как показано на рис. 31.

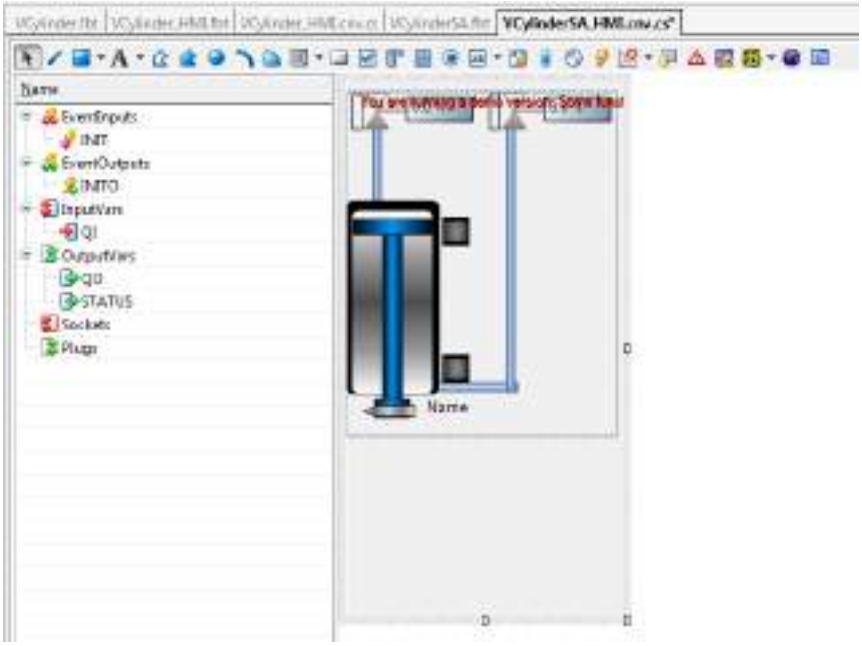


Рис. 31. Создание составного CAT-блока (шаг 6)

2.4. Расширение приложений

Проект шаблона уже имеет несколько блоков в элементе APP1, который находится под элементом System в дереве решений. К нему нужно добавить только что созданный VCylinderSA. Для того, чтобы сделать это, просто перетащите CAT-блок VCylinderSA в систему System, как показано на рис. 32, и создайте соответствующие связи.

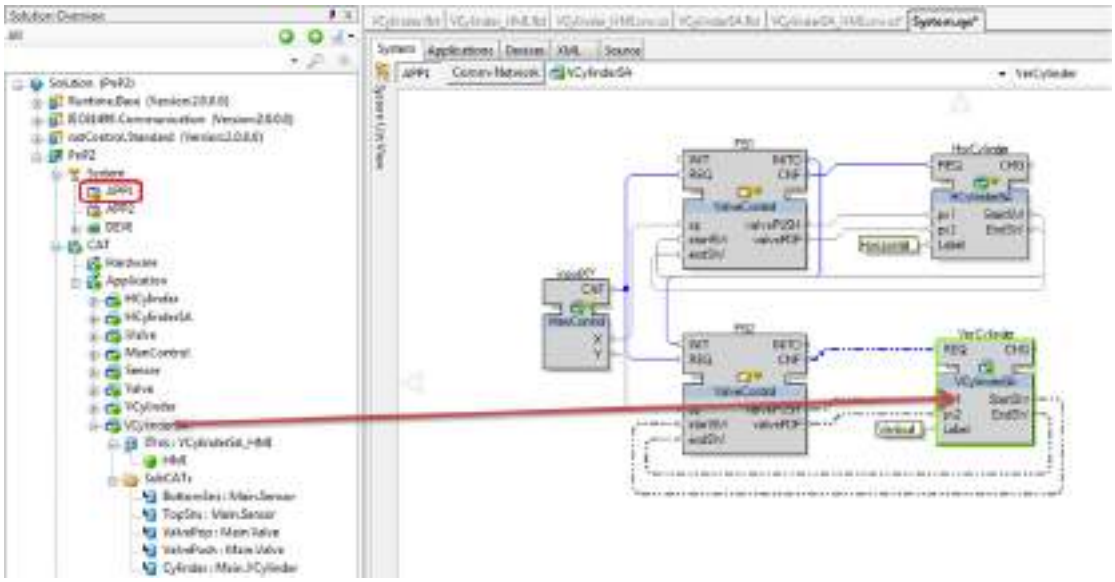


Рис. 32. Добавление CAT-блока VCylinderSA в приложение APP1

Вы можете заметить пунктирные линии. Это связано с тем, что вновь добавленный блок не отображен ни на одно Устройство/Ресурс. Отобразите его на DEV0.RES0 (для этого нажмите правой кнопкой мыши на указанный блок -> Mapping -> DEV0.RES0).

2.5. Расширение рабочей области

Как только мы расширили приложение, сразу возникает необходимость расширить рабочую область. Перетащите созданный экземпляр в рабочую область *MainCanvas* под Canvases -> 800x600.

Обратите внимание, что Вам нужно перенести блок из CAT Instances -> Application, а не из CAT -> Application (рис. 33). Если после перетаскивания ничего не появилось, то сохраните изменения и перезапустите nxtStudio.

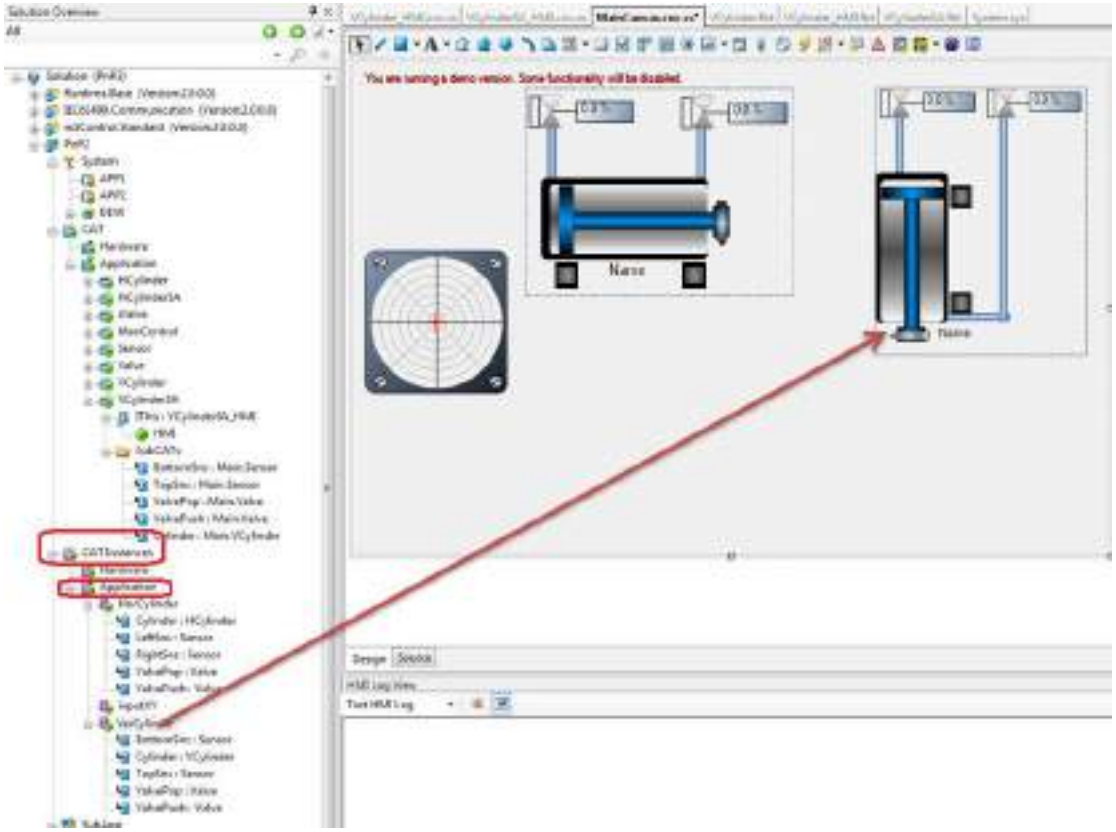


Рис. 33. Расширение рабочей области

Следуя процедуре развертывания из лабораторной работы № 1, запустите приложение, управляйте движением цилиндров с помощью джойстика

2.6. Полезная информация по реализации HMI на основе CAT-блоков с использованием C#

1) Входные события в CAT_HMI

Входная переменная QI по умолчанию связана с событийным входом INIT. В случае, если интерфейс по данным необходимо расширить, новые информационные входы должны быть ассоциированы с событийным входом, отличным от INIT. Можно добавить событийный вход, например, с именем REQ, и ассоциировать новые информационные входы с ним. Если желательно изменить эти информационные входы при инициализации, вход INIT реализующего CAT может быть связан с обоими входами INIT и REQ этого CAT-блока HMI-интерфейса.

2) Выходные события в CAT_HMI

Выходное событие INITO по умолчанию выдается внутри метода public_HMI, который определяется следующим образом:

```
public HMI()
          {
               InitializeComponent();
               //Insert additional initialisation code HERE
          }
```

Этот метод вызывается, когда CAT-блок HMI-интерфейса вызывается событием INIT. Событие INITO выдается после выполнения этого метода. Другой код для инициализации рекомендуется добавлять после вызова InitializeComponent(). В случае, если в CAT-блоке HMI необходимо определить новые информационные выходы, и их значения должны быть определены во время инициализации, они не могут быть ассоциированы с INITO. Следовательно, новые информационные выходы могут быть ассоциированы с другим событийным выходом. Однако, это событие не может быть выдано в данном коде инициализации.

3) Формат метода FireEvent

Если выходное событие (назовем его EO) CAT-блока HMI связано с информационными выходами DO1, DO2, Don, то соответствующий формат метода, генерирующего событие, определяется как `FireEvent_EO(value1, value2, …, value n)`, где параметры-значения value соответствуют типам данных выходов `DO1, …, DOn`. Однако, если событие ассоциировано с выходными переменными по умолчанию QO и STATUS, соответствующие значения в списка параметров FireEvent_EO будут опускаться. Выходные переменные по умолчанию всегда первые в этом списке и не могут быть удалены. Например, если событийный выход IND ассоциирован с тремя информационными выходами QO, STATUS и NEWDATA, то метод, выдающий событие IND и устанавливающий NEWDATA в значение 3, будет выглядеть как `FireEvent_IND(3).`

4) Ограничения

-- Событийный выход INITO CAT-блока HMI не может быть связан с другими выходами этого CAT-блока, отличными от INITO;

-- Событийный выход INITO CAT-блока HMI не может быть ассоциирован с другими информационными выходами, отличными от QO и STATUS.

Лабораторная работа № 3

Изучение коммуникационных блоков

Цель работы

Изучить работу коммуникационных блоков в среде NxtStudio.

Задание

Реализовать в среде NxtStudio приложение, которое бы позволяло передавать сообщения между разными устройствами с использованием коммуникационных блоков.

Ход работы

Для передачи информации между устройствами в среде NxtStudio существует библиотека «IEC61499.Communication», которая входит в комплект с демо-версией программы. В ней реализованы сервисные интерфейсные функциональные блоки следующих типов:

- **Publish** («писатель»), осуществляет рассылку данных всем блокам-читателям, имеющим тот же IP-адрес и порт подключения;
- **Subscribe** («читатель»), принимает данные, рассылаемые соответствующим «писателем»;
- **Client**, может как отправлять данные на блок-сервер, так и получать их от него;
- **Server**, может отправлять данные блоку-клиенту и получать их от него.

Таким образом связка блоков «Publish – Subscribe» предназначена для осуществления широковещательной рассылки между устройствами, а «Client – Server» для соединения «point-to-point».

В данной лабораторной работе для передачи данных между устройствами воспользуемся блоками Publish, Subscribe.

После создания проекта необходимо подключить соответствующую библиотеку функциональных блоков. Для этого нужно кликнуть правой

кнопкой мыши по названию проекта и перейти во вкладку References. Данное окно представлено на рисунке 1. Если необходимая библиотека установлена в среду NxtStudio, то она будет отображаться в верхней таблице, а если библиотека уже привязана к проекту, то в нижней. Для привязки библиотеки необходимо выбрать её в верхней таблице и нажатием кнопки «Add» добавить к проекту.

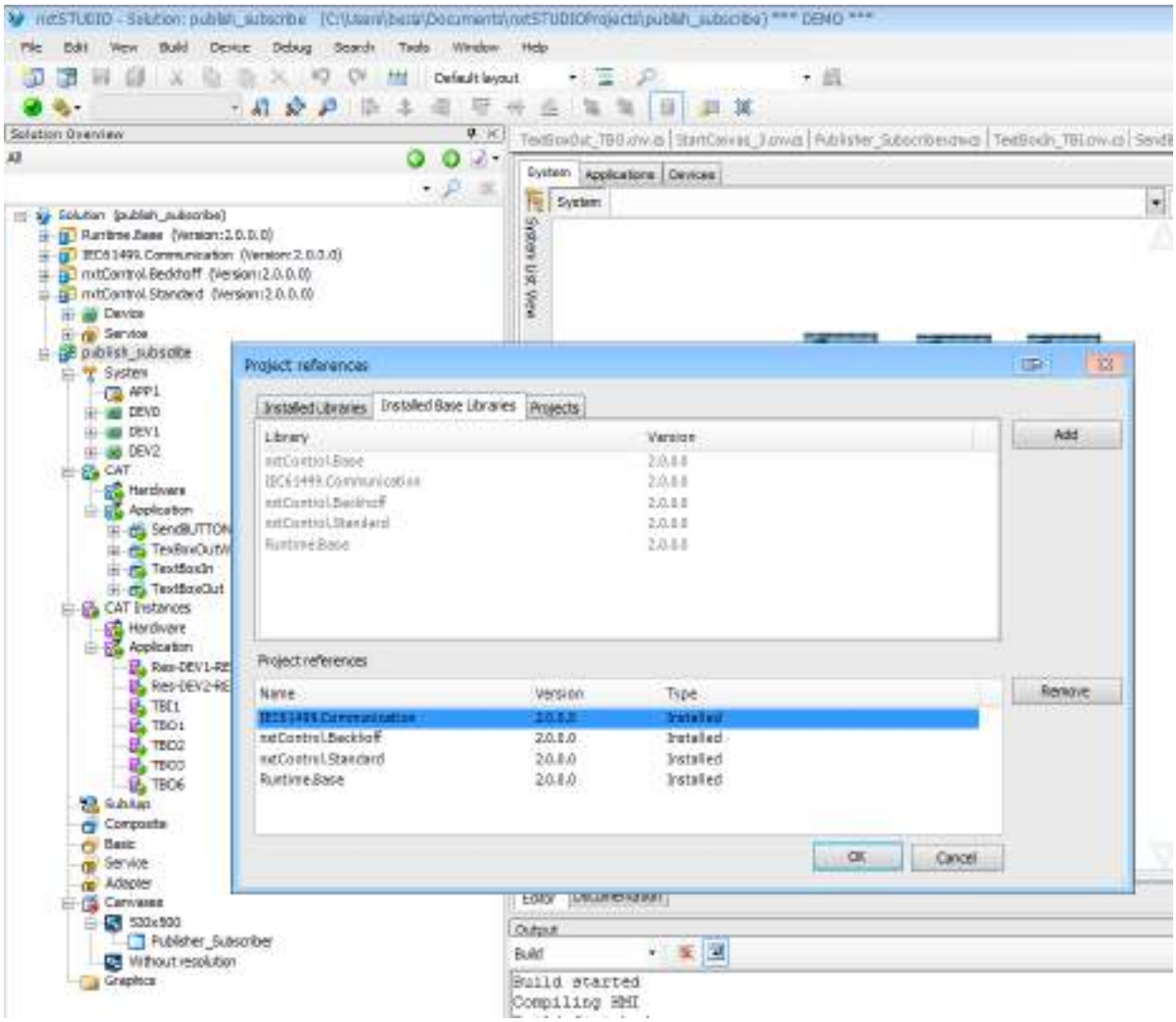


Рисунок 1 – Подключение библиотеки элементов к проекту

После подключения библиотеки в нашем проекте появится вкладка со всеми блоками данной библиотеки. Блоки библиотеки представлены в виде сервисных интерфейсных блоков, т.е. их нельзя редактировать и от нас скрыта реализация их внутренней логики. Пример блока представлен на рисунке 2.

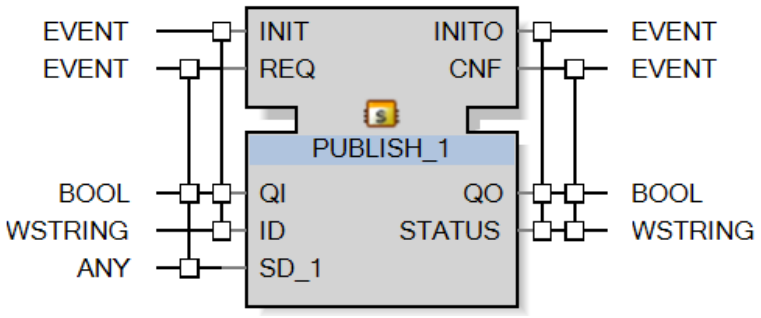


Рисунок 2 – СИФБ Publish_1

С назначением портов блоков можно ознакомиться во вкладке «Help» соответствующей библиотеки.

Для демонстрации работы коммуникационных блоков создадим приложение, включающее в себя 3 устройства. На первом устройстве будет располагаться блок-писатель и блок-читатель, на двух других по одному блоку-читателю.

Для ввода и вывода передаваемых данных создадим соответствующие CAT-блоки. Итоговое приложение примет вид, как на рисунке 3.

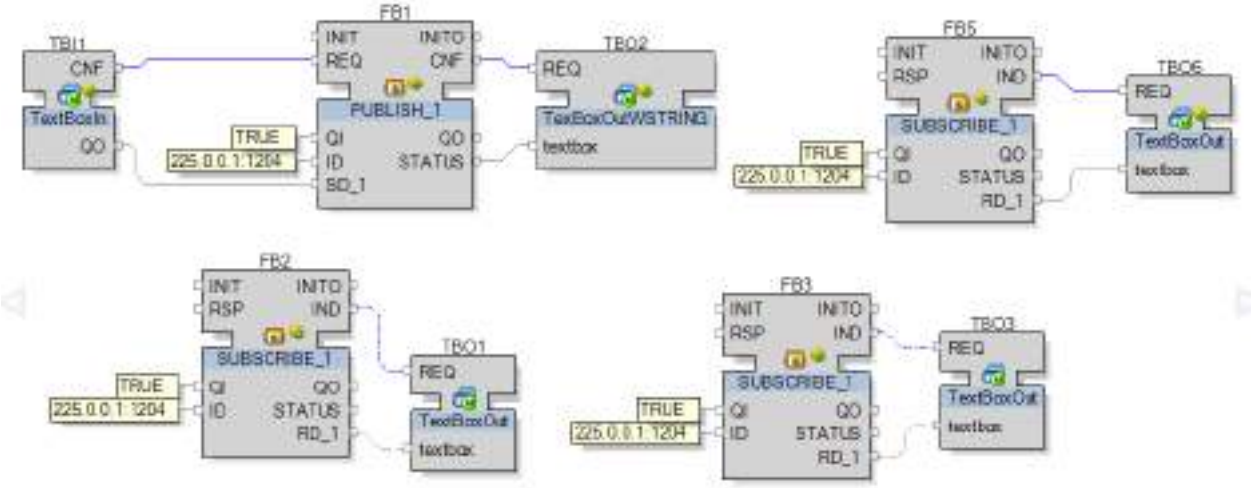


Рисунок 3 – Итоговое приложение

Блоки **FB2, FB3, FB5** – блоки-читатели.

Блок **FB1** – блок-писатель.

Блок **TBI1** – CAT-блок для ввода передаваемых данных.

Блоки **TBO1, TBO3, TBO2, TBO6** – CAT-блоки для отображения переданных данных.

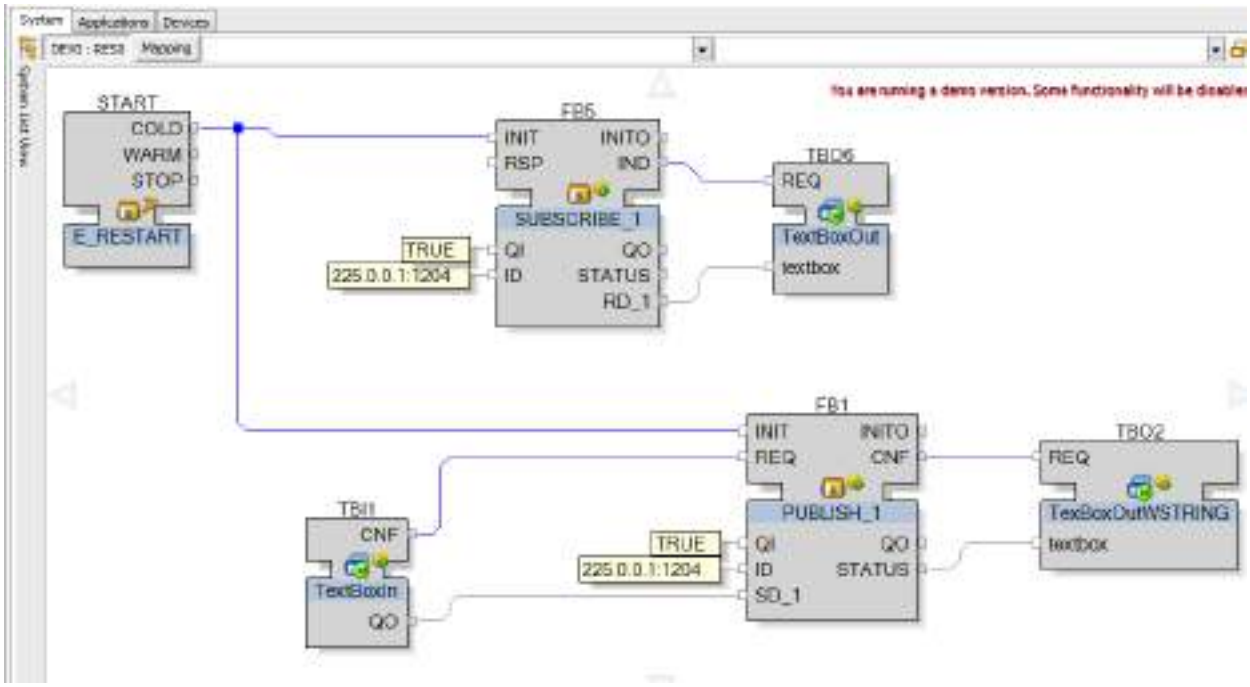


Рисунок 4 – Блоки, расположенные на ресурсе RES0 устройства DEV0

Из рисунка 3 видно, что блоки Publish и Subscribe никак не соединены друг с другом, т.к. связь между ними устанавливается непосредственно во время их инициализации при начале работы устройств DEV0, DEV1, DEV2. Для установки соединения друг с другом данные блоки должны иметь одно и тоже значение поля ID. При этом, если планируется широковещательная рассылка (более чем на один блок-читатель), то IP-адрес в поле ID должен быть в диапазоне адресов начиная с 224.0.0.0 до 239.255.255.255. Следует учитывать, что диапазон адресов с 224.0.0.0 до 224.0.0.255 зарезервирован для системных нужд.

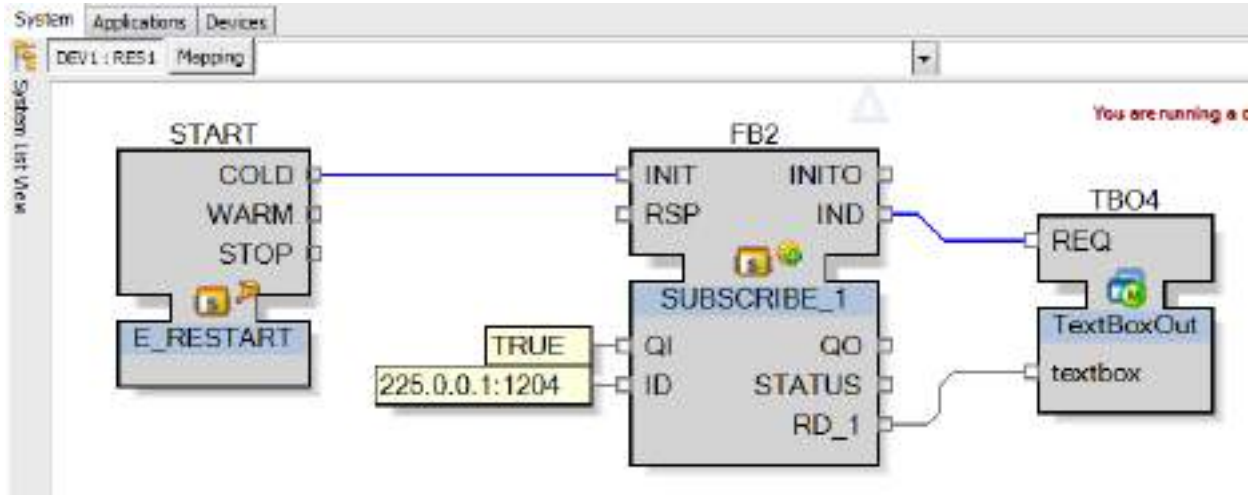


Рисунок 5 – Блоки, расположенные на ресурсе RES1 устройства DEV1

После создания приложения и расположения всех элементов на соответствующих устройствах необходимо реализовать человеко-машинный интерфейс для взаимодействия с программой.

Итоговый интерфейс выглядит в соответствии с рисунком 6.

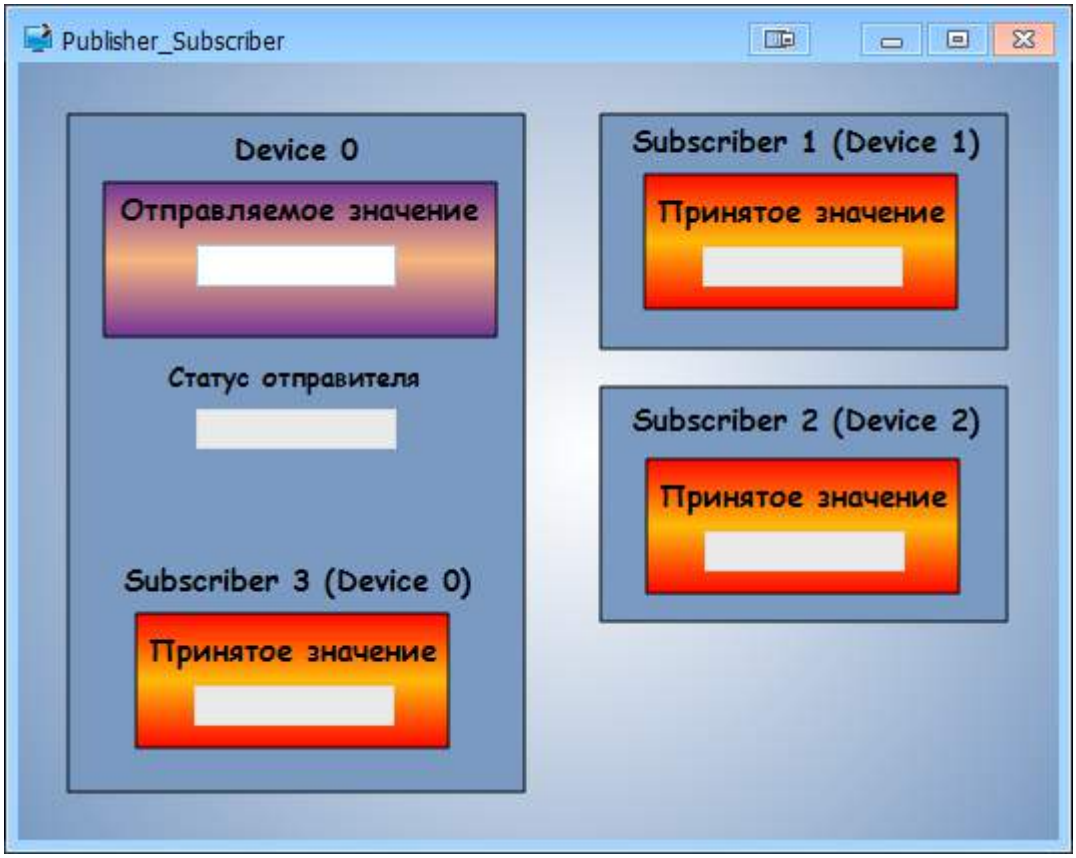


Рисунок 6 – Интерфейс программы

Результаты работы программы

Для проверки работоспособности созданного приложения был запущен эмулятор «Soft PLC» для каждого из устройств. При этом необходимо указать разные номера портов для каждого из устройств.

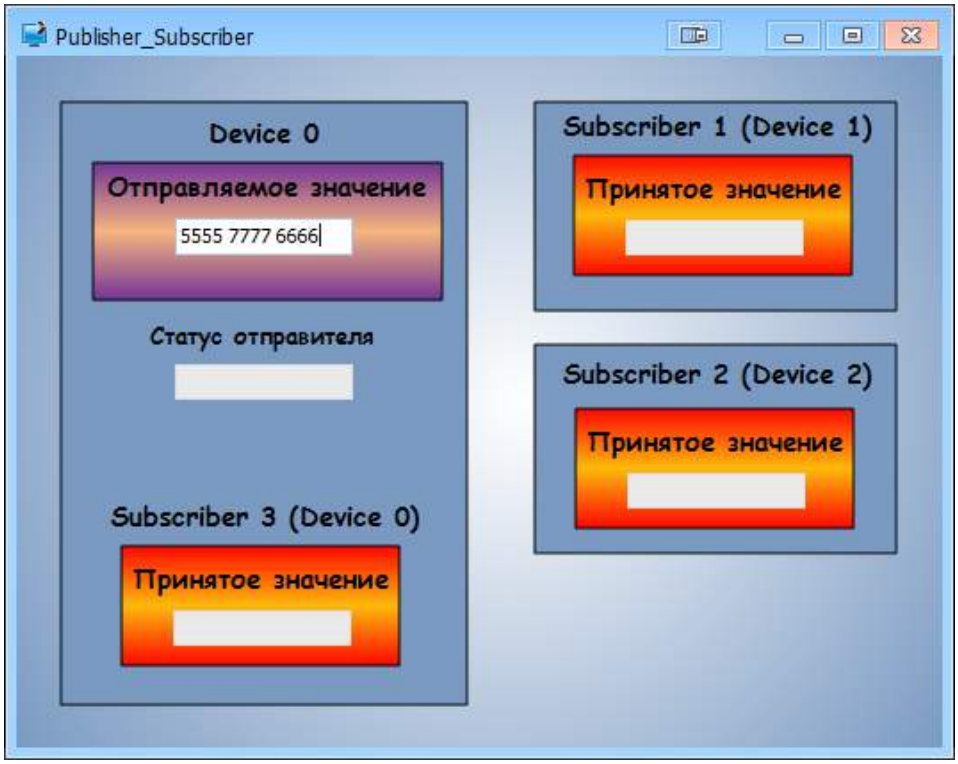


Рисунок 7 – Состояние интерфейса перед отправкой сообщения

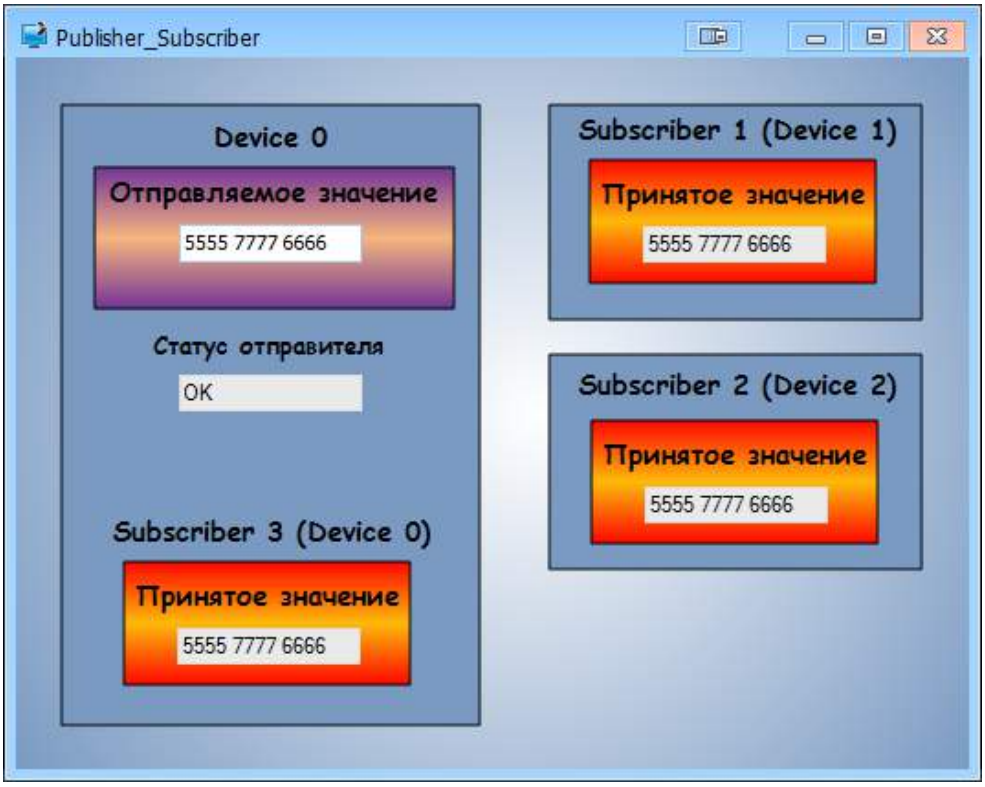


Рисунок 8 – Интерфейс после отправки сообщения

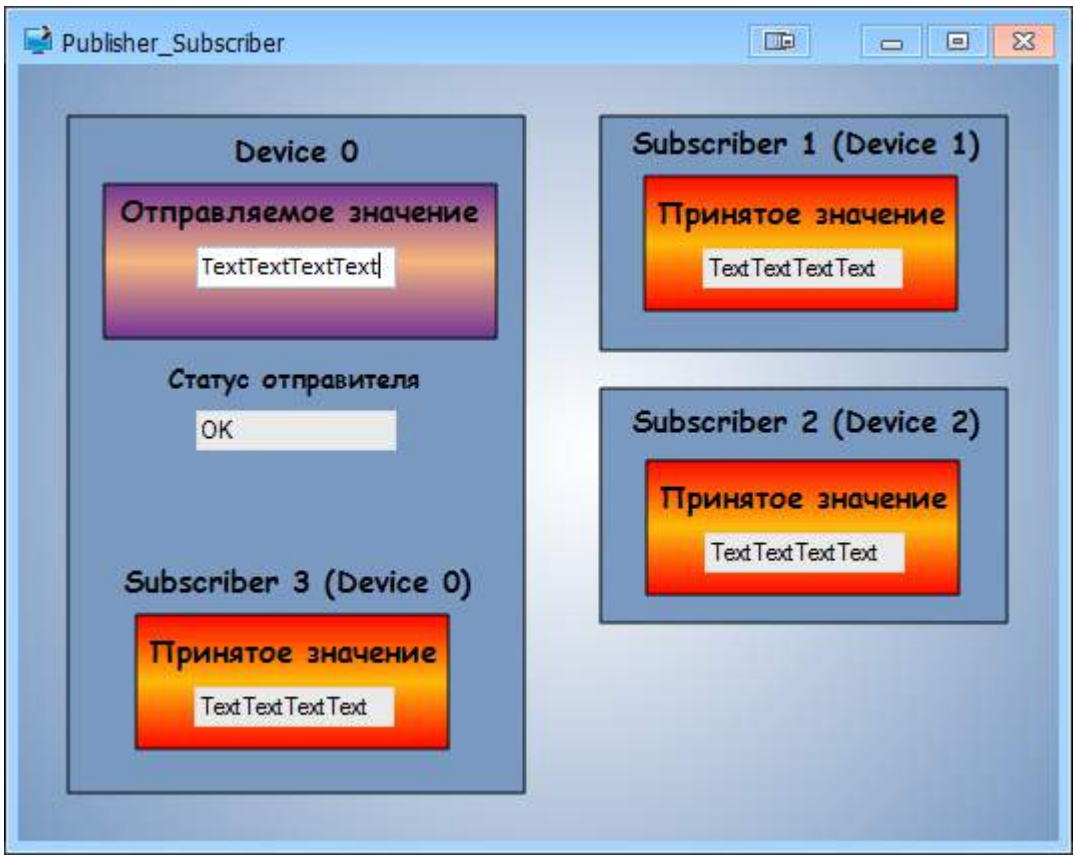


Рисунок 9 – Интерфейс после отправки следующего сообщения

В результате работы приложения можно сделать вывод, что передача данных между устройствами с использованием коммуникационных блоков реализована и сеть функциональных блоков работоспособна.

Вывод

В ходе выполнения данной лабораторной работы был изучен принцип подключения библиотек к проектам в среде NxtStudio и построена сеть функциональных блоков, выполняющая передачу данных между устройствами с использованием коммуникационных блоков библиотеки «IEC61499.Communication».

Лабораторная работа № 4

Создание интерфейса «человек-машина» для системы охлаждения дата-центра

Цель работы: 1) ознакомиться с принципами работы системы охлаждения дата-центров; 2) получить практические навыки разработки интерактивного человеко-машинного интерфейса (ЧМИ) на основе CAT-блоков в среде NxtStudio для управления, мониторинга и имитации работы локального и общего кулеров.

Примечание: данная лабораторная работа состоит из ряда последовательных заданий, выполнение которых приводит к созданию целевого проекта в среде NxtStudio.

Задание 1. Подготовка к лабораторной работе

1.1. Создайте новый проект (HMI and IEC Solution).

1.2. Назовите проект ServerRoom и сохраните его.

1..3. Распакуйте архив Lab4.zip[1].

1.4. Импортируйте CAT-блок (ServerRoomFB.20150215-1529.cat.zip[2]) в свой проект. Нажмите правой кнопкой мыши на CAT и выберите Import. Данный CAT-блок содержит шаблон интерфейса «человек-машина», в который не включены изображения кулеров и кнопок (рис. 4.1). В данной работе этот CAT-блок надо усовершенствовать для возможности его полноценного использования.

[1] Данный архив можно скачать по адресу:
https://drive.google.com/open?id=0B1_0eFPTb3VXdXR0NjNmcGxFSUE

[2] Этот файл можно найти в архиве Lab4.zip

1.5. Скопируйте папку SR-Images[3], содержащую графические элементы интерфейса, в папку HMI вашего проекта. Следует отметить, что данная папка содержит в том числе как статичные, так и анимационные изображения вентиляторов. В дальнейшем каждое изображение должно быть прикреплено к соответствующему элементу в рабочей области HMI CAT-блоков.

1.6. Нажмите на изображении прямоугольника под надписью Local Fan в левом верхнем углу экрана. Имя этого компонента rect_LF, вы можете найти его в Панели свойств (Properties Pane).

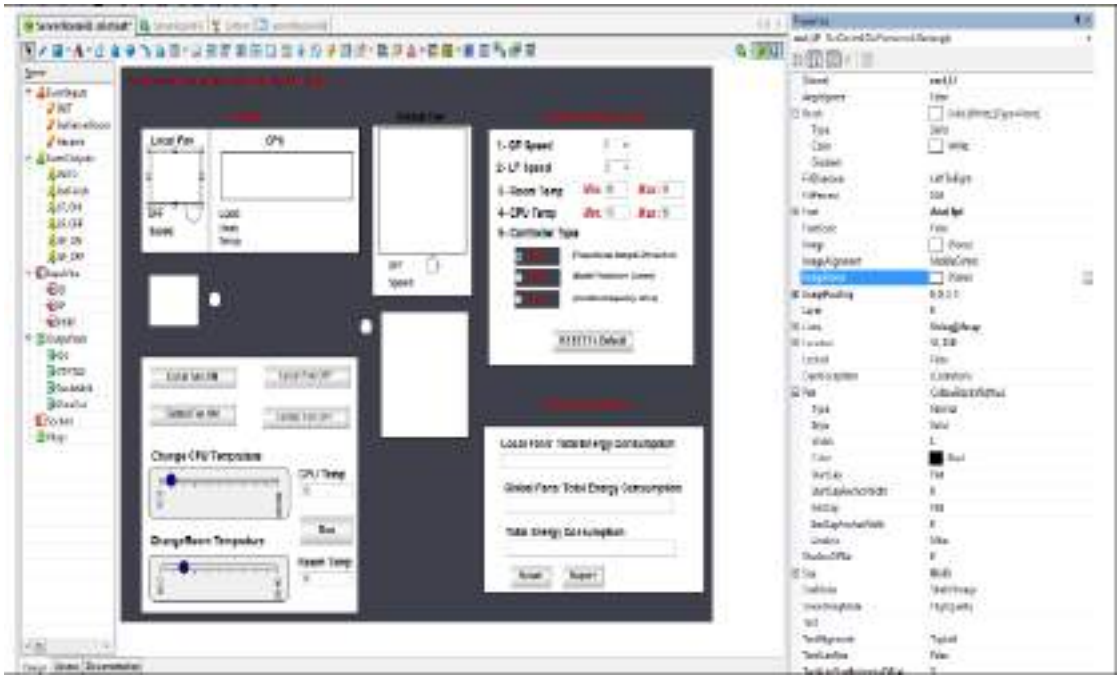


Рис. 4.1. Шаблон интерфейса «человек-машина», не содержащий изображения вентиляторов и кнопок

1.7. Для компонента rect_LF выберете свойство Image Name и щелкните мышью на поле ввода значений для изменения имени файла.

1.8. Откроется диалоговое окно редактора списка изображений (ImageList Editor). В списке Image Lists выберите HMI, затем кликните на кнопку Add («Добавить»), откроется диалоговое окно Catalogue Name. Введите в нем имя каталога, например, Images (рис. 4.2).

[3] Эту папку можно найти в архиве Lab4.zip

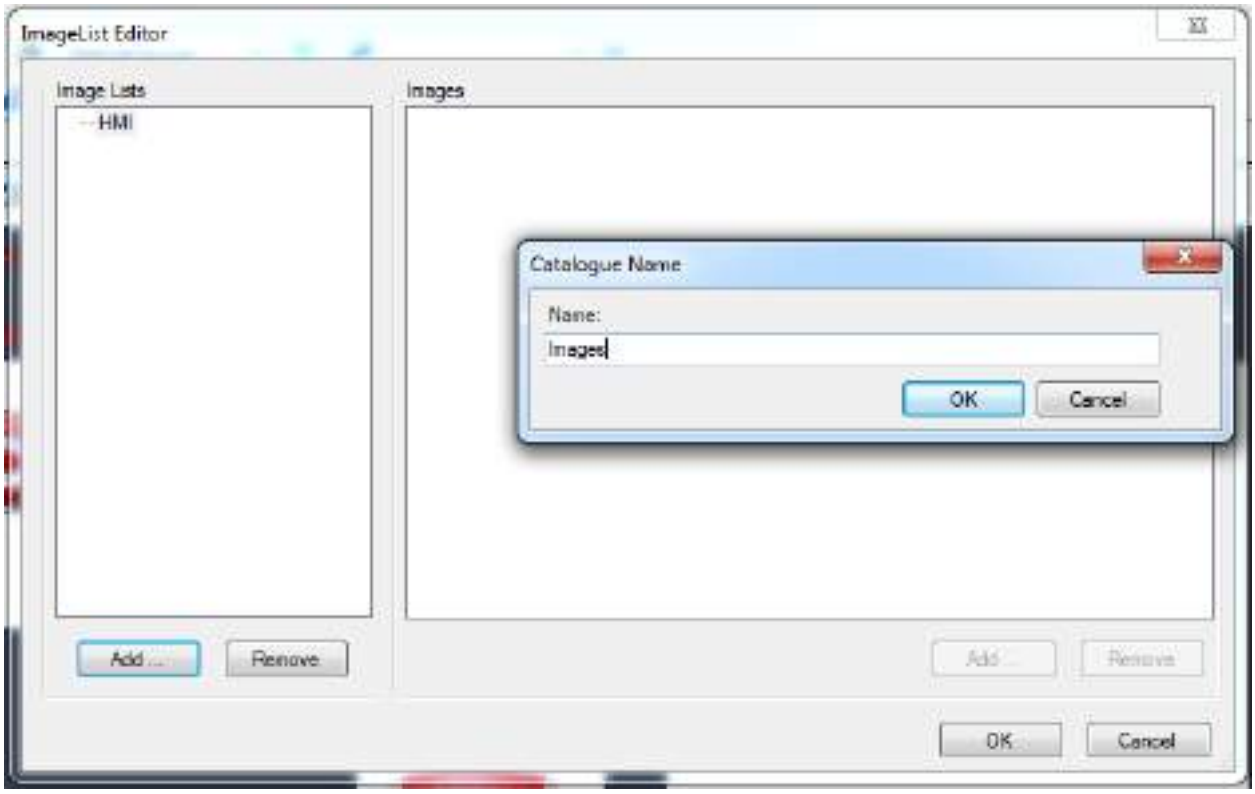


Рис. 4.2. Создание списка изображений в проекте

1.9. Под надписью HMI щелкните на только что созданном списке Images, затем нажмите на кнопку Add в правом нижнем углу. Откроется диалоговое окно для навигации по файловой системе, с помощью которого нужно найти ранее скопированную папку SR-Images. Выберите все файлы изображений из этой папки и нажмите Open (рис. 4.3). Добавленные изображения можно просмотреть в редакторе списка изображений (рис. 4.4).

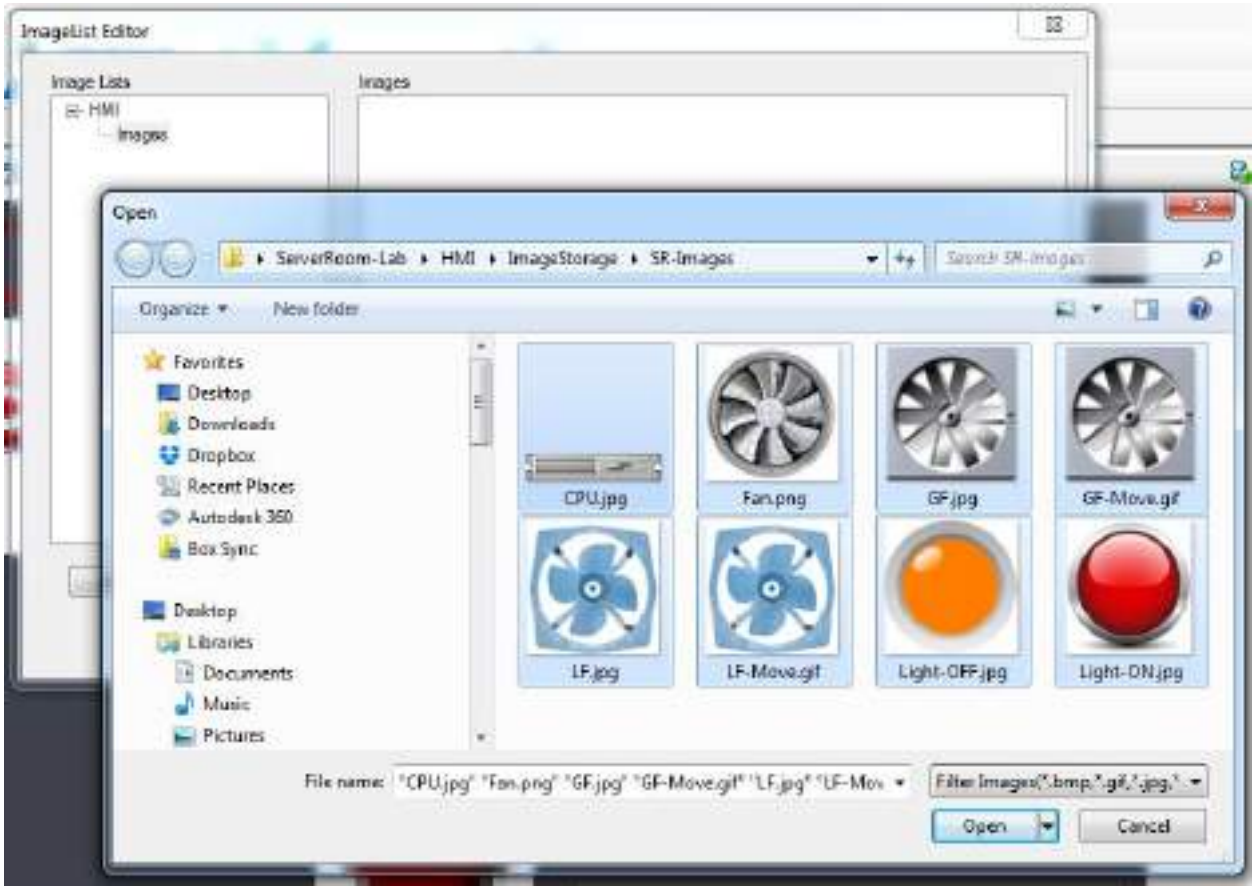


Рис. 4.3. Выбор изображений для добавления в созданный список

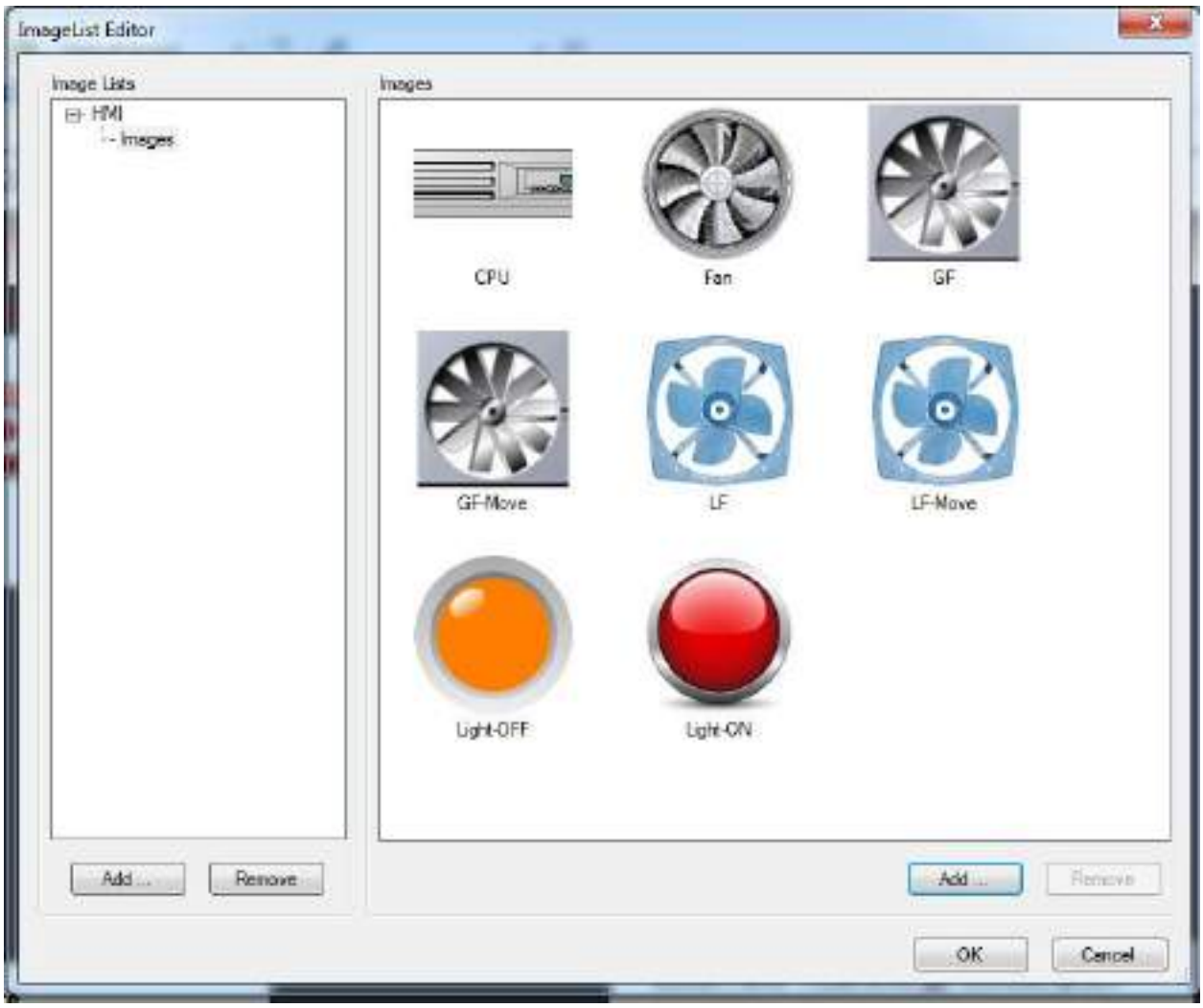


Рис. 4.4. Просмотр изображений в редакторе списка изображений

1.10. Выберите изображение с именем LF из списка изображений и нажмите на Ok (рис. 4.5). Это изображение представляет собой локальный кулер (вентилятор) в выключенном состоянии (состояние OFF).

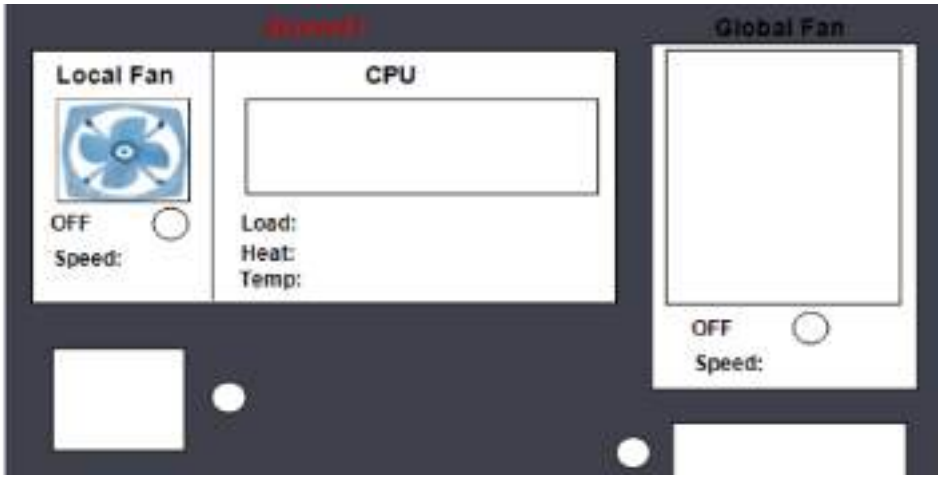


Рис. 4.5. Загрузка в пустой прямоугольник изображения кулера

1.11. Создайте новую рабочую область (Canvas) и назовите ее ServerRoomHMI. Для ее заголовка используйте то же самое имя.

1.12. Щелкните правой кнопкой мыши на пункте CAT Instances («Экземпляры CAT-блоков») и выберите NewItem->ServerRoom->ServerRoomFB. Введите в появившемся окне имя вашего CAT-приложения - SRFBAPP.

1.13. Добавьте SRFBAPP в ранее созданную рабочую область ServerRoomHMI. Вы можете регулировать размер рабочей области (холста), чтобы она подходила вашему CAT-блоку.

1.14. Протестируйте ЧМИ. Для этого щелкните правой кнопкой мыши на пункте 1280x980 и выберите пункт Test HMI Runtime on Local Computer.

Задание 2: Подготовка графики

2.1. Теперь, когда среда установлена, добавьте следующие графические изображения для соответствующих компонентов ЧМИ, как это было описано в Задании 1 (рис. 4.6):

a) свяжите изображение Light-OFF с компонентом LF_LightOFF;

b) свяжите изображение LF-Move с компонентом rect_LFMove;

c) свяжите изображение Light_ON с компонентом LF_LightON;

d) свяжите изображение CPU с компонентом rect_CPU;

e) свяжите изображение GF с компонентом rect_GF;

f) свяжите изображение GF-Move с компонентом rect_GFMove;

g) свяжите изображение Light-OFF с компонентом GF_LightOFF;

h) свяжите изображение Light-ON с компонентом GF_LightON.

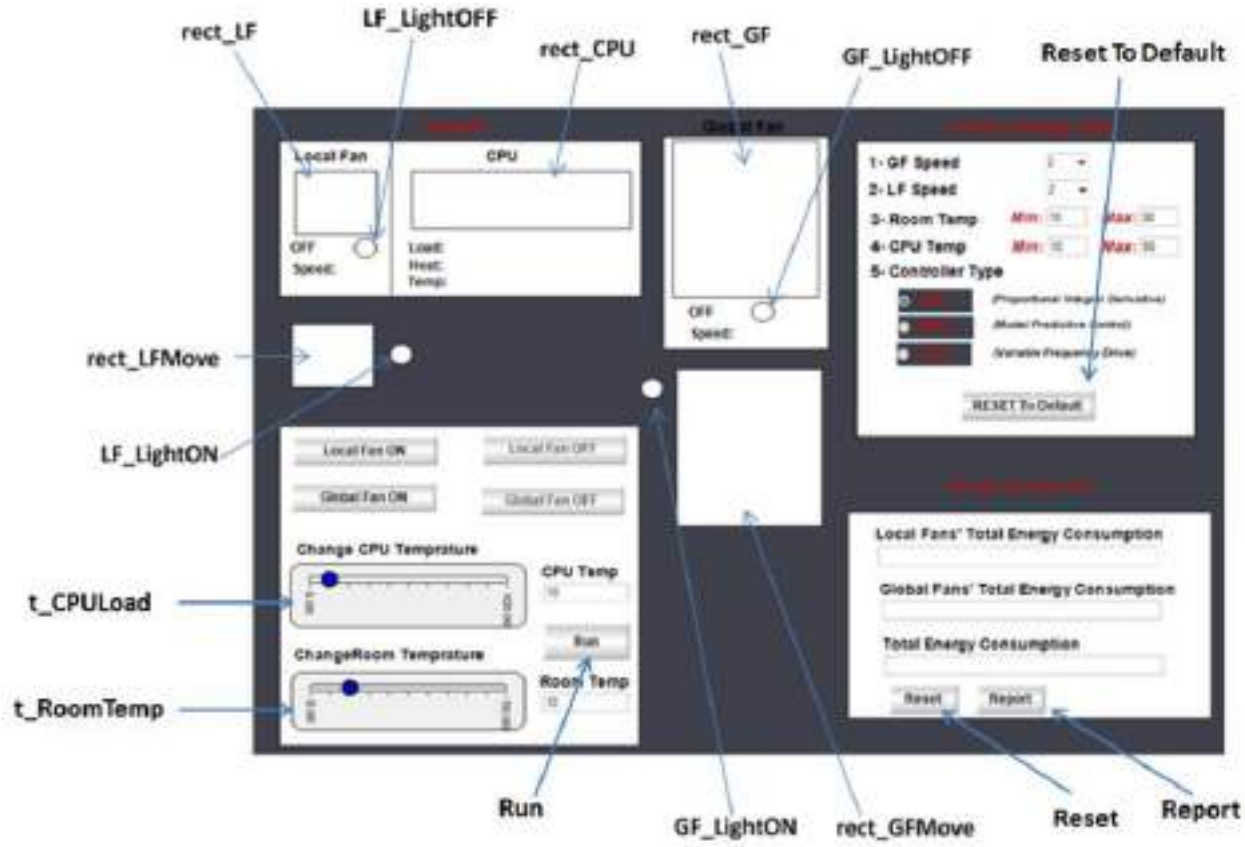


Рис. 4.6. Соответствие геометрических областей и компонентов ЧМИ

2.2. Снова проверьте ваш ЧМИ. Для этого щелкните правой кнопкой мыши на пункте 1280x980 и выберите Test HMI Runtime on Local Computer. Вы должны увидеть все изображения в вашем ЧМИ.

Задание 3: Разработка интерактивного CAT-блока для управления и мониторинга локального и общего кулера

Совет: щелкните на ServerRoomFB, затем найдите sDefault. Щелкните на нем два раза и выберите Source (Источник), чтобы найти код программы.

3.1. Напишите программу для кнопок Global Fan ON и Global Fan OFF так, чтобы они могли включать и выключать общий кулер. Вы можете использовать код, написанный для кнопок включения и выключения локального кулера Local Fan ON и Local Fan OFF. Примечание: измените следующие функции в исходном коде (Source Code) вашего CAT-блока.

Следует заметить, что для описания логики работы CAT-блоков используется язык программирования C#.

```
void GF_ON_BClick(object sender, EventArgs e)
{
// YOU HAVE TO COMPLETE THIS METHOD FOR TURNING ON THE GLOBAL FAN
}
void GF_OFF_BClick(object sender, EventArgs e)
{
// YOU HAVE TO COMPLETE THIS METHOD FOR TURNING OFF THE GLOBAL FAN
}
```

3.2. Можно заметить, что два объекта (типа Tracker), представляющие следящие системы - трекеры, уже включены в ЧМИ: t_CPULoad (под именем Change CPU Temperature) – для отслеживания температуры центрального процессора и t_RoomTemp (под именем Change Room Temperature) – для отслеживания температуры воздуха в помещении.

Код функции t_CPULoad() уже задан. Используя этот метод, измените метод t_RoomTemp().

```
void T_RoomTempValueChanged(object sender, EventArgs e)
{
// Complete This Method Wherever Rquired
// Use These Variables:
/*
1- GF_State
2- rect_GF.Visible
3- rect_GFMove.Visible
4- GF_ON_OFF.Text
5- GF_LightON.Visible
6- GF_LightOFF.Visible
*/
RoomTemp_Text.Text = t_RoomTemp.Value.ToString();
if (t_RoomTemp.Value > double.Parse(RTMax.Text) &&
GF_State.Equals(0)){
this.FireEvent_GF_ON("GF_ON");
// Complete this if block using given variables and follow the
t_CPULoad () for Help
//…
Thread threadRoom = new Thread(new ThreadStart(moveRoomTemp));
threadRoom.Start();
}
if (t_RoomTemp.Value == double.Parse(RTMin.Text) &&
GF_State.Equals(1)){
this.FireEvent_GF_OFF("GF_OFF");
// Complete this if block using given variables and follow the
t_CPULoad () for Help
```

```
//…
GF_State = 0;
rect_GF.Visible = true;
rect_GFMove.Visible = false;
GF_ON_OFF.Text = "OFF";
GF_LightON.Visible = false;
GF_LightOFF.Visible = true;
}
}
```

3.3. Теперь, когда вы убедились, что ЧМИ работает правильно, необходимо выполнить следующие шаги, чтобы этот интерфейс стал выглядеть более привлекательно:

a) переместите компонент rect_LFMove непосредственно на компонент rect_LF, чтобы они действовали бы как один объект;

b) переместите компонент LF_LightON непосредственно на компонент LF_LightOFF, чтобы они действовали бы как один объект;

c) переместите компонент rect_GFMove непосредственно на компонент rect_GF так, чтобы они действовали бы как один объект;

d) переместите компонент GF_LightOn непосредственно на компонент GF_LightOFF, чтобы они действовали бы как один объект.

3.4. После внесенных изменений, скомпилируйте заново рабочую область (можно использовать клавишу F8) и запустите ЧМИ. Это будет финальная версия вашего ЧМИ.

3.5. Запустите разработанный CAT-блок в рамках некоторого приложения на программном контроллере Soft PLC.

Задание 4. Расчет и вывод отчета о потреблении электроэнергии

4.1. Создайте две глобальные переменные и назовите их LF_EC_Total и GF_EC_Total.

4.2. Обновите программу так, что всякий раз, когда вы запускаете локальный кулер с использованием только одного трекера, значение константы C1 прибавлялось бы к значению переменной LF-EC-Total. Вы можете выбрать любое значение константы C1, которое имеет смысл.

4.3. Обновите программу так, что всякий раз, когда вы запускаете общий кулер с использованием только одного трекера, значение константы C2 прибавлялось бы к значению переменной GF-EC-Total. Вы можете выбрать любое значение константы C2, которое имеет смысл.

4.4. Измените следующий метод, чтобы выводить потребления электроэнергии в соответствующие текстовые поля (textbox). Необходимо выводить LF-EC-Total, GF-EC-Total и полное потребление энергии двумя этими вентиляторами.

```
void ReportClick(object sender, EventArgs e)
{
// YOU HAVE TO COMPLETE THIS METHOD TO DISPLAY ENERGY
CONSUMPTIONS IN APPROPRIATE TEXT BOX
}
```

4.5. Разработайте на языке C# полный набор функций (методов), определяющий логику работы CAT-блока. Должны быть реализованы следующие функции (методы):

- изменение цвета соответствующего светодиода при включении/выключении кулера;
- изменение соответствующего изображения (с анимацией/без анимации) при включении/выключении кулера;
- изменение значения температуры сервера/комнаты при перемещении ползунков трекеров. Если температуры достигают указанных предельных значений, то включаются соответствующие кулеры;
- добавление значения энергопотребления соответствующего кулера к глобальным переменным GF_EC_Total и LF_EC_Total при каждом автоматическом включении кулера;
- вывод отчета о потреблении электроэнергии при нажатии на кнопку Report.

4.5. Протестируйте приложение, включающее разработанный CAT-блок. На рис. 4.7 показано состояние системы после нажатия на кнопку Run и Report. Так как были указаны значений температур, выходящие за установленные ограничения, были включены оба кулера. После нажатия кнопки Report в соответствующие текстовые поля были выведены значения

энергопотребления кулеров. Примечание: для облегчения разработки CAT-блока можно воспользоваться листингом программ, приведенным в Приложении 1.

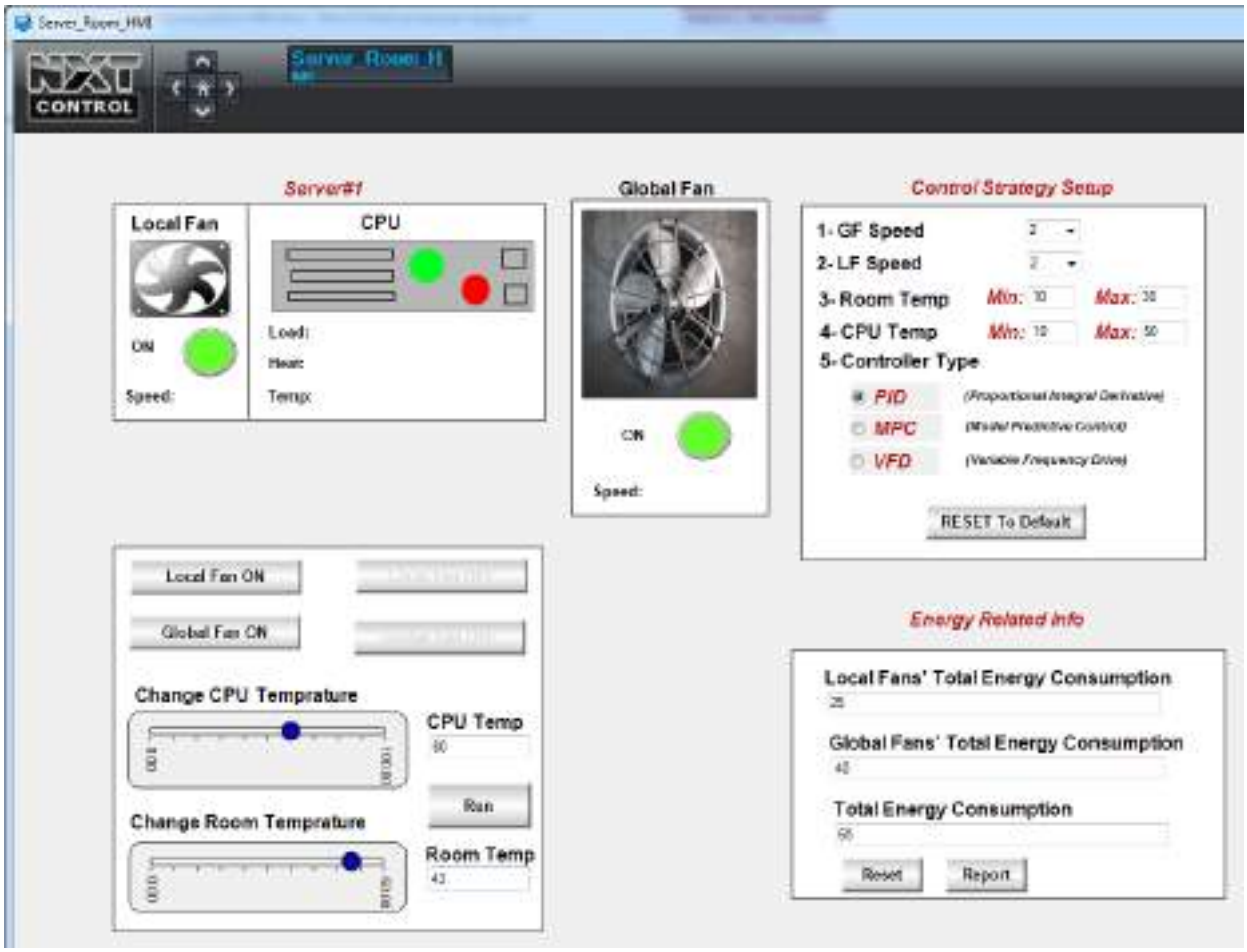


Рис. 4.7. Результаты работы программы

Задание 5. Разделение управляющего кода на два CAT-блока

5.1. Создайте два CAT-блока, первый из которых будет соответствовать локальному кулеру, а второй – общему. Управляющий код, который был разработан в ходе выполнения предыдущих пунктов, необходимо разделить на два этих блока.

5.2. Создайте приложение, содержащее эти два CAT-блока. Отобразите эти блоки на разные устройства. Соответственно, при запуске приложения используйте два программных контроллера Soft PLC.

Приложение 1. Листинг итогового CAT-блока для работы № 3

```
using System;
using System.Drawing;
using NxtControl.GuiFramework;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using System.Net;
using System.Net.Sockets;
using System.Diagnostics;
namespace HMI.Main.Symbols.ServerRoomFB
{
     /// <summary>
     /// Description of sDefault.
     /// </summary>
     public partial class sDefault :
NxtControl.GuiFramework.HMISymbol
     {
      int LF_State = 0;
      int GF_State = 0;
      double GF_EC_Total = 0;
      double LF_EC_Total = 0;
      double C1 = 5;
      double C2 = 10;

          public sDefault()
          {
               //
               // The InitializeComponent() call is required for
Windows Forms designer support.
               //
               InitializeComponent();
      this.InitServerRoom_Fired +=new EventHandler
<ServerRoomFB.InitServerRoomEventArgs>(InitServerRoom);
               this.Receive_Fired += new EventHandler
<ServerRoomFB.ReceiveEventArgs> (Receive);
          }

          public void InitServerRoom(object Sender,
ServerRoomFB.InitServerRoomEventArgs sr){

          }

  public void Receive(object Sender,
ServerRoomFB.ReceiveEventArgs r){

     }

/*---ON and OFF Buttons for Local and Global Fans
 /* Function to Turn the Local Fan On*/
     void LF_ON_BClick(object sender, EventArgs e)
```

```
            {
                this.FireEvent_LF_ON("LF_ON");
                LF_State = 1;
                rect_LF.Visible = false;
                rect_LFMove.Visible = true;
                LF_ON_OFF.Text = "ON";
           LF_LightON.Visible = true;
           LF_LightOFF.Visible = false;
           LF_ON_B.Enabled = false;
           LF_OFF_B.Enabled = true;
            }
/* Function to Turn the Global Fan Off*/
            void LF_OFF_BClick(object sender, EventArgs e)
            {
           this.FireEvent_LF_OFF("LF_OFF");
                LF_State = 0;
                rect_LF.Visible = true;
                rect_LFMove.Visible = false;
                LF_ON_OFF.Text = "OFF";
           LF_LightON.Visible = false;
           LF_LightOFF.Visible = true;
           LF_ON_B.Enabled = true;
           LF_OFF_B.Enabled = false;

            }

/* Function to Turn the Global Fan On*/
            void GF_ON_BClick(object sender, EventArgs e)
            {
          this.FireEvent_LF_ON("GF_ON");
               GF_State = 1;

         rect_GF.Visible = false;
         rect_GFMove.Visible = true;
         LF_ON_OFF.Text = "ON";
         GF_LightON.Visible = true;
         GF_LightOFF.Visible = false;

         GF_ON_B.Enabled = false;
         GF_OFF_B.Enabled = true;


            }
/* Function to Turn the Global Fan OFF*/
            void GF_OFF_BClick(object sender, EventArgs e)
            {
           this.FireEvent_LF_OFF("GF_OFF");
                GF_State = 0;

         rect_GF.Visible = true;
         rect_GFMove.Visible = false;
         LF_ON_OFF.Text = "OFF";
         GF_LightON.Visible = false;
```

```
        GF_LightOFF.Visible = true;

          GF_ON_B.Enabled = true;
          GF_OFF_B.Enabled = false;

           }



 /*  THIS  METHOD  RESPONDS  TO  CHANGES  ON  TRACKER  WHICH  IS
?CONNECTED TO A CPU
 WHEN THE TRACKER IS SLIDE TO THE RIGHT AND EXCEEDS THE MAXIMUM
LIMIT OF THE CPU TEMPERATURE
 ITS  CORRESPONDING  LOCAL  FAN  WILL  BE  ON  TO  REDUCE  ITS
TEMPERATURE
 AND TRACKER WILL SLIDE TO LEFT UNTIL CPU TEMPERATURE REACHES
ITS MINIMUM LIMIT   */
          void T_CPULoadValueChanged(object sender, EventArgs e)
          {

            CPULoad_Text.Text = t_CPULoad.Value.ToString();

            if (t_CPULoad.Value > double.Parse(CPUTMax.Text) &&
LF_State.Equals(0)){
              this.FireEvent_LF_ON("LF_ON");
              LF_State = 1;
              rect_LF.Visible = false;
              rect_LFMove.Visible = true;
              LF_ON_OFF.Text = "ON";
         LF_LightON.Visible = true;
         LF_LightOFF.Visible = false;
         Thread threadCPU = new Thread(new
ThreadStart(moveCPULoad));
         threadCPU.Start();
         LF_EC_Total=LF_EC_Total+C1;
          }
            if (t_CPULoad.Value == double.Parse(CPUTMin.Text) &&
LF_State.Equals(1)){
          this.FireEvent_LF_OFF("LF_OFF");
               LF_State = 0;
               rect_LF.Visible = true;
               rect_LFMove.Visible = false;
               LF_ON_OFF.Text = "OFF";
          LF_LightON.Visible = false;
          LF_LightOFF.Visible = true;

             }
           }

/*  THIS  METHOD  RESPONDS  TO  CHANGES  ON  TRACKER  WHICH  IS
?CONNECTED TO THE ROOM
```

```
 WHEN THE TRACKER IS SLIDE TO THE RIGHT AND EXCEEDS THE MAXIMUM
LIMIT OF THE ROOM TEMPERATURE
 CORRESPONDING TO THE GLOBAL FAN WILL BE ON TO REDUCE ITS
TEMPERATURE
 AND TRACKER WILL SLIDE TO LEFT UNTIL ROOM TEMPERATURE REACHES
ITS MINIMUM LIMIT   */
          void T_RoomTempValueChanged(object sender, EventArgs e)
          {
           RoomTemp_Text.Text = t_RoomTemp.Value.ToString();

           if (t_RoomTemp.Value > double.Parse(RTMax.Text) &&
GF_State.Equals(0)){
             this.FireEvent_GF_ON("GF_ON");
             // Complete this if block using given variables and
follow the t_CPULoad () for Help
             GF_State = 1;
             rect_GF.Visible = false;
             rect_GFMove.Visible = true;
             GF_ON_OFF.Text = "ON";
        GF_LightON.Visible = true;
        GF_LightOFF.Visible = false;
        Thread threadRoom = new Thread(new
ThreadStart(moveRoomTemp));
        threadRoom.Start();
        GF_EC_Total=GF_EC_Total+C2;

          }
           if (t_RoomTemp.Value == double.Parse(RTMin.Text) &&
GF_State.Equals(1)){
          this.FireEvent_GF_OFF("GF_OFF");
              GF_State = 0;
             rect_GF.Visible = true;
             rect_GFMove.Visible = false;
             GF_ON_OFF.Text = "OFF";
        GF_LightON.Visible = false;
        GF_LightOFF.Visible = true;
                  }
          }

// Function for Run Button
          void RunClick(object sender, EventArgs e)
          {
               t_RoomTemp.Value =
double.Parse(RoomTemp_Text.Text);
           t_CPULoad.Value = double.Parse (CPULoad_Text.Text);
          }

          public void moveCPULoad ()
          {
           try
           {
                string[] parts = CPULoad_Text.Text.Split('.');
        int y = int.Parse(parts[0]);
```

```
                t_CPULoad.Value = Convert.ToDouble(y);
        Thread.Sleep(500);
                for (int
i=y;i>=Convert.ToInt32(double.Parse(CPUTMin.Text));i--)
                {
                 t_CPULoad.Value = Convert.ToDouble(i);
          Thread.Sleep(100);
                }
            }catch (Exception ex){
            }
          }
          public void moveRoomTemp ()
          {
           try
           {
                string[] parts = RoomTemp_Text.Text.Split('.');
        int y = int.Parse(parts[0]);
        Thread.Sleep(500);

                for (int
i=y;i>=Convert.ToInt32(double.Parse(RTMin.Text));i--)
                {
                 t_RoomTemp.Value = Convert.ToDouble(i);
          Thread.Sleep(300);
                }
            }catch (Exception ex){

            }
          }
/*------------ Reseting the Control Strategy Setup to its
Default Value ----------- ---*/
 /*-------------------------------------------------------------
-------------------------*/
          void ReSetToDefaultClick(object sender, EventArgs e)
          {
               GFSpeed.Text = "2";
               LFSpeed.Text = "2";
               RTMin.Text = "10";
               RTMax.Text = "30";
               CPUTMin.Text = "10";
               CPUTMax.Text = "50";
          }
/*-------- Creating Energy Report and Reseting the Text Boxes
          void ReportClick(object sender, EventArgs e)
          {
               GF_Total_EC.Text = GF_EC_Total.ToString();
               LF_Total_EC.Text = LF_EC_Total.ToString();
               Total_EC_All.Text = (GF_EC_Total +
LF_EC_Total).ToString();
          }
          void ResetClick(object sender, EventArgs e)
          {
```

```
                t_RoomTemp.Value =  double.Parse(RTMin.Text);
                RoomTemp_Text.Text = RTMin.Text;
                t_CPULoad.Value = double.Parse (CPUTMin.Text);
                CPULoad_Text.Text = CPUTMin.Text;
                //ReportLog_listBox.Items.Clear();
                LF_Total_EC.Clear();
                GF_Total_EC.Clear();
                Total_EC_All.Clear();
               }
        }
}
```

Литература

1. Vyatkin V. IEC 61499 Function Blocks for embedded and distributed control systems design. Third Edition - ISA, 2014. – 261 p.

2. Дубинин В.Н., Вяткин В.В. Модели функциональных блоков IEC 61499, их проверка и трансформации в проектировании распределенных систем управления: монография. – Пенза: Изд-во ПГУ, 2012. – 348 с.

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ СИСТЕМНЫЙ И ИНТЕЛЛЕКТУАЛЬНЫЙ АНАЛИЗ ДАННЫХ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

Лекция 6. ПОИСК АССОЦИАТИВНЫХ ПРАВИЛ

1 Основные сведения

2 Характеристики ассоциативных правил

3 Значимость ассоциативных правил

4 Методы поиска ассоциативных правил

1 Основные сведения. Задача поиска ассоциативных правил (association rule mining) впервые была предложена для нахождения типичных шаблонов покупок, совершаемых в супермаркетах, поэтому эта задача имеет второе название: анализ рыночной корзины (market basket analysis). Транзакции являются достаточно характерными операциями для описания результатов посещений различных магазинов. Транзакция - это множество событий, которые произошли одновременно. Регистрируя все бизнес-операции в процессе своей деятельности, торговые компании накапливают огромные собрания транзакций. Каждая такая транзакция представляет собой набор товаров, купленных покупателем за один визит. Полученные в результате анализа шаблоны включают перечень товаров и число транзакций, которые содержат данные наборы.

Транзакционная или операционная база данных (Transaction database) представляет собой двумерную таблицу, которая состоит из номера транзакции (TID) и перечня покупок, приобретенных во время этой транзакции. TID - уникальный идентификатор, определяющий каждую сделку или транзакцию. Пример транзакционной базы данных, состоящей из покупательских транзакций, приведен в таблице 1.

Таблица 1. Транзакционная база данных

TID	Приобретенные покупки
100	Хлеб, молоко, печенье
200	Молоко, сметана
300	Молоко, хлеб, сметана, печенье
400	Колбаса, сметана
500	Хлеб, молоко, печенье, сметана

На основе имеющейся базы данных нужно найти закономерности между событиями, то есть покупками. Ассоциативное правило состоит из двух наборов объектов (продуктов, предметов), называемых условием и следствием, записываемых в виде X→Y (X имплицирует Y):

$$X \to Y = \bar{X} \vee Y.$$

Присвоим значениям товаров переменные: Хлеб = a , Молоко = b , Печенье = c, Сметана = d, Колбаса = e, Конфеты = f.

Таблица 2. Часто встречающиеся наборы товаров

TID	Приобретенные покупки	→	TID	Приобретенные покупки
100	Хлеб, молоко, печенье		100	a, b, c
200	Молоко, сметана		200	b, d
300	Молоко, хлеб, сметана, печенье		300	b, a, d, c
400	Колбаса, сметана		400	e, d
500	Хлеб, молоко, печенье, сметана		500	a, b, c, d
600	Конфеты		600	f

Рассмотрим набор товаров, включающий, например, {Хлеб, молоко, печенье}. Выразим этот набор с помощью переменных: **abc={a,b,c}.** Этот набор товаров встречается три раза в базе данных и поддержка этого набора товаров равна 3: **SUP(abc) = 3.** При минимальном уровне поддержки, равной трем, набор товаров **abc** является часто встречающимся шаблоном. Таким образом, поддержкой (обеспечением) называют количество или процент транзакций, содержащих определенный набор данных. Для набора товаров **abc** поддержка, выраженная в процентах, равна:

SUP(abc)=(3/6)*100%=50%.

Таким образом, набор представляет интерес, если его поддержка выше определенного пользователем минимального значения (min support). Эти наборы называют часто встречающимися (frequent).

2 Характеристики ассоциативных правил. Ассоциативное правило имеет вид: "Из события A следует событие B": A→B. Другими словами, утверждается закономерность следующего вида: "Если в транзакции

встретился набор товаров (или набор элементов) A, то можно сделать вывод, что в этой же транзакции должен появиться набор элементов B)" Установление таких закономерностей дает возможность находить простые и понятные правила, называемые ассоциативными. Основными характеристиками ассоциативного правила являются поддержка и достоверность правила. Рассмотрим правило "Из покупки молока следует покупка печенья".

Поддержка правила – это число транзакций, которые содержат как условие, так и следствие. Например, для ассоциации A→B поддержка определяется отношением количество транзакций, содержащих A и B к общему числу транзакций. Например, молоко - это товар A, печенье - это товар B. Поддержка правила "из покупки молока следует покупка печенья" равна 3, или 50%.

Достоверность правила оценивается как вероятность того, что из события A следует событие B. Другими словами, достоверность ассоциативного правила A→B представляет собой меру точности правила и определяется как отношение количества транзакций, содержащих условие и следствие, к количеству транзакций, содержащих только условие:

$$C(A \rightarrow B)=P(A|B)=P(A \cap B)/P(A).$$

Правило "Из A следует B" справедливо с достоверностью C, если количество транзакций из всего множества, содержащих набор элементов A, также содержат набор элементов B. Например, число транзакций, содержащих молоко, равно четырем, число транзакций, содержащих печенье, равно трем, достоверность правила равна (3/4)*100%, т.е. 75%. Таким образом, достоверность правила "из покупки молока следует покупка печенья" равна 75%, т.е. 75% транзакций, содержащих товар A, также содержат товар B.

Границы поддержки и достоверности. При помощи использования алгоритмов поиска ассоциативных правил аналитик может получить все возможные правила вида "Из A следует B", с различными значениями

поддержки и достоверности. Однако в большинстве случаев, количество правил необходимо ограничивать заранее установленными минимальными и максимальными значениями поддержки и достоверности.

Если значение поддержки правила слишком велико, то в результате работы алгоритма будут найдены правила очевидные и хорошо известные. Слишком низкое значение поддержки приведет к нахождению очень большого количества правил, которые, возможно, будут в большей части необоснованными, но не известными и не очевидными для аналитика. Таким образом, необходимо определить такой интервал, "золотую середину", который с одной стороны обеспечит нахождение неочевидных правил, а с другой - их обоснованность.

Если уровень достоверности слишком мал, то ценность правила вызывает серьезные сомнения. Например, правило с достоверностью в 3% только условно можно назвать правилом. Аналитики могут отдавать предпочтение правилам, которые имеют только высокую поддержку или только высокую достоверность, или оба этих показателя. Правила, для которых значения поддержки или достоверности превышают определенный, заданный пользователем порог, называются **сильными правилами**. Например, аналитика может интересовать, какие товары, покупаемые вместе в супермаркете, образуют ассоциации с минимальной поддержкой 20% и минимальной достоверностью 70%. А при анализе с целью обнаружения мошенничеств может потребоваться уменьшить поддержку до 1%, поскольку с мошенничеством связано сравнительно небольшое число транзакций.

3 Значимость ассоциативных правил. Методики поиска ассоциативных правил обнаруживают все ассоциации, которые удовлетворяют ограничениям на поддержку и достоверность, наложенным пользователем. Это приводит к необходимости рассматривать десятки и сотни тысяч ассоциаций, что делает невозможным обработку такого количества данных вручную. Число правил желательно уменьшить таким образом, чтобы проанализировать только наиболее значимые из них.

Значимость часто вычисляется как разность между поддержкой правила и в целом и произведением поддержки только условия и поддержки только следствия.

Если условие и следствие независимы, то поддержка правила примерно соответствует произведению поддержек условия и следствия, то есть SAB≈SASB. Это значит, что хотя условие и следствие часто встречаются вместе, не менее часто они встречаются и по отдельности. Например, если товар A встречался в 70 транзакциях из 100, а товар B – в 80, и в 50 транзакциях из 100 они встречаются вместе, то несмотря на высокую поддержку (SAB=0.5), это не обязательно правило. Просто эти товары покупаются независимо друг от друга, но в силу их популярности часто встречаются в одной транзакции. Так как SASB=0.7 0.8=0,56 отличается от SAB всего на 0.06, то предположение о независимости товаров A и B достаточно обоснованно.

Субъективными мерами значимости ассоциативных правил являются **лифт и левередж.**

Лифт вычисляется следующим образом:

$$L(A \rightarrow B)=C(A \rightarrow B)/S(B).$$

Таким образом, лифт – это отношение частоты появления условия в транзакциях, которые также содержат и следствие, к частоте появления следствия в целом. Значения лифта, большие, чем 1, Лифт показывает, что условие чаще появляется в транзакциях, содержащих следствие, чем в остальных. Можно сказать, что лифт является обобщенной мерой связи двух предметных наборов: при значениях лифта больше 1 связь положительная, при 1 она отсутствует, при значениях меньше 1 – отрицательная.

Левередж вычисляется следующим образом:

$$T(A \rightarrow B)=S(A \rightarrow B)-S(A)S(B).$$

Следовательно, левередж – это разность между наблюдаемой частотой, с которой условие и следствие появляются совместно (т.е. с поддержкой ассоциации), и произведением частот появления (поддержек) условия и

следствия по отдельности. Из ассоциаций с одинаковым лифтом ассоциация с большим левереджем представляет больший интерес, так как это говорит о том, что данное правило встречаются чаще.

Улучшение вычисляется следующим образом:

$$I(A \rightarrow B)=S(A \rightarrow B)/(S(A)S(B)).$$

Таким образом, улучшение показывает, полезнее ли правило случайного угадывания. Если $I(A \rightarrow B)>1$, это значит, что вероятнее предсказать наличие набора B с помощью правила, чем угадать случайно.

4 Методы поиска ассоциативных правил. В процессе поиска ассоциативных правил может производится обнаружение всех ассоциаций, оценка поддержки и достоверности значения которых превышают заданный минимум. Простейший алгоритм поиска ассоциативных правил рассматривает все возможные комбинации условий и следствий, оценивает для них поддержку и достоверность, а затем исключает все ассоциации, которые не удовлетворяют заданным ограничениям. Число возможных ассоциаций с увеличением числа предметов растет экспоненциально (экспоненциальная сложность алгоритма). Поэтому в процессе генерации ассоциативных правил широко используются методики, позволяющие уменьшить сложность алгоритма за счет снижения количества ассоциаций, которые требуется проанализировать.

Алгоритм AIS. Первый алгоритм поиска ассоциативных правил, называвшийся AIS, (предложенный Agrawal, Imielinski and Swami) был разработан сотрудниками исследовательского центра IBM Almaden в 1993 году. С этой работы начался интерес к ассоциативным правилам; на середину 90-х годов прошлого века пришелся пик исследовательских работ в этой области, и с тех пор каждый год появляется несколько новых алгоритмов. В алгоритме AIS кандидаты множества наборов генерируются и подсчитываются "на лету", во время сканирования базы данных. **Алгоритм SETM**. Создание этого алгоритма было мотивировано желанием использовать язык SQL для вычисления часто встречающихся наборов

товаров. Как и алгоритм AIS, SETM также формирует кандидатов "на лету", основываясь на преобразованиях базы данных. Чтобы использовать стандартную операцию объединения языка SQL для формирования кандидата, SETM отделяет формирование кандидата от их подсчета. Недостаток алгоритмов AIS и SETM состоит в излишнем генерировании и подсчете слишком многих вариантов правил, которые в результате не оказываются часто встречающимися. Для улучшения их работы был предложен алгоритм Apriori.

Алгоритм Apriori. В основе алгоритма Apriori лежит понятие частого набора. Под частотой понимается простое количество транзакций в которых содержится данный предметный набор. Частый предметный набор – предметный набор с поддержкой больше заданного порога либо равной ему. Этот порог называется минимальной поддержкой. Работа данного алгоритма состоит из нескольких этапов, каждый из этапов состоит из следующих шагов:

- Формирование кандидатов (candidate generation) - этап, на котором алгоритм, сканируя базу данных, создает множество i-элементных кандидатов (i - номер этапа). На этом этапе поддержка кандидатов не рассчитывается.

- Подсчет кандидатов (candidate counting) - этап, на котором вычисляется поддержка каждого i-элементного кандидата. Здесь же осуществляется отсечение кандидатов, поддержка которых меньше минимума, установленного пользователем (min_sup). Оставшиеся i-элементные наборы называем часто встречающимися.

Чтобы сократить пространство поиска ассоциативных правил, алгоритм Apriori использует свойство антимонотонности. Свойство утверждает, что если предметный набор Z не является частым, то добавление некоторого нового предмета A к набору Z не делает его более частым. Данное полезное свойство позволяет значительно уменьшить пространство поиска ассоциативных правил. На первом этапе алгоритма Apriori

формируется множество частых наборов F1, содержащих один предмет. Для поиска множества k-предметных наборов Fk, сначала создается множество Fk кандидатов в k-предметные наборы путем связывания множества Fk-1 с самим собой. Затем Fk сокращается с использованием свойства антимонотонности. Предметные наборы множества, которые остались после сокращения, формируют Fk. После того, как все частые предметные наборы найдены, можно переходить к генерации на их основе ассоциативных правил. Для этого к каждому частому предметному набору s можно применить следующую процедуру.

Генерируются все возможные поднаборы s 2. Если поднабор ss является непустым поднабором s, то рассматривается ассоциация R:ss→(s-ss), где s-ss представляет собой набор s без поднабора ss. R считается ассоциативным правилом, если удовлетворяет условию заданного минимума поддержки и достоверности. Данная процедура повторяется для каждого подмножества ss из s.

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ
СОВРЕМЕННЫЕ ТЕХНОЛОГИИ УПРАВЛЕНИЯ ДАННЫМИ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

Лабораторная работа 1.1 Нормализация отношений оперативной базы данных на основе функциональных зависимостей

Цель работы. Ознакомление с процессом нормализации схем реляционных баз данных, приобретение практических навыков анализа и построения эффективных схем оперативных баз данных для информационных систем класса OLTP.

1.1 Теоретическая часть. При проектировании отношений реляционной базы данных (РБД) могут возникать две серьезные проблемы:

- избыточность данных;
- потенциальная противоречивость (аномалии).

Под *избыточностью* понимают повторение данных в разных строках одной таблицы или в разных таблицах БД. Так, для каждого сотрудника отдела 128 повторяются данные «128, Отдел проектирования».

Аномалии – это проблемы, возникающие в данных из-за дефектов проектирования БД. Существуют три вида аномалий: вставки, удаления и модификации.

Аномалии вставки проявляются при вводе данных в дефектную таблицу. Добавляя информацию о новом сотруднике, мы должны добавить номер и название отдела. Если ввести данные, не соответствующие имеющимся в таблице (например, 42, отдел проектирования), будет не ясно, какая из строк БД содержит правильную информацию.

Аномалии удаления возникают при удалении данных из дефектной схемы. Предположим, что все сотрудники отдела 128 уволились в один и тот же день. После удаления записей этих сотрудников в БД больше не будет ни одной записи, содержащей информацию об отделе 128.

Аномалии модификации возникают при изменении данных дефектной схемы. Предположим, что отдел 128 решили переименовать в отдел передовых технологий. Необходимо изменить соответствующие данные о каждом сотруднике отдела. Если мы пропустим хотя бы одну запись, возникнет аномалия модификации.

Решение обеих проблем особенно важно для оперативных БД в системах оперативной обработки транзакций (OLTP-системах).

Правилом разработки хорошей структуры БД является необходимость избегать схем с большим числом *пустых атрибутов*. Если мы хотим указать, что один из ста служащих имеет особую квалификацию, для хранения этой информации не следует добавлять в таблицу еще один столбец, поскольку для остальных 99 работников значением столбца будет NULL. Вместо этого следует добавить новую таблицу, в которой будут храниться только кодовые номера и информация о квалификации тех работников, которых это касается.

Решение перечисленных проблем состоит в разделении данных и связей, что обеспечивается процедурой *нормализации*. Концепции и методы нормализации были разработаны Э. Ф. Коддом.

Нормализация отношений – это формальный аппарат ограничений на формирование отношений, который позволяет устранить дублирование и потенциальную противоречивость хранимых данных, уменьшает трудозатраты на ведение БД. Процесс нормализации заключается в декомпозиции исходных отношений на более простые отношения. Цель нормализации – получение такого проекта БД, в котором «каждый факт появляется лишь в одном месте».

Теория нормализации основана на наличии зависимостей между атрибутами отношения. Основными видами зависимостей являются:

- функциональные;
- многозначные;
- транзитивные.

Базовым является понятие *функциональной зависимости*, поскольку на его основе формируются определения всех остальных видов зависимостей. Атрибут B функционально зависит от атрибута A, если каждому значению A соответствует в точности одно значение B. Математически функциональную зависимость B от A обозначают A → B. Это означает, что во всех кортежах с одинаковым значением атрибута A атрибут B будет иметь также одно и то же значение. При этом A и B могут быть составными, то есть состоять из двух и более атрибутов.

Зависимость, при которой каждый неключевой атрибут зависит от всего составного ключа и не зависит от его частей, называется *полной функциональной зависимостью.* Если атрибут A зависит от атрибута B, а атрибут B зависит от атрибута C (C → B → A), но обратная зависимость отсутствует, то зависимость A от C называется *транзитивной.*

Многозначная зависимость. Говорят, что один атрибут отношения многозначно определяет другой атрибут того же отношения, если для каждого значения первого атрибута существует множество соответствующих значений второго атрибута. Многозначные зависимости могут быть:

- один-ко-многим (1:M);
- многие-к-одному (M:1);
- многие-ко-многим (M:M).

Каждая ступень процесса нормализации приводит схему отношений в последовательные нормальные формы. Для каждой ступени имеются наборы ограничений. Выделяют следующую последовательность нормальных форм:

- первая нормальная форма (1НФ);
- вторая нормальная форма (2НФ);
- третья нормальная форма (3НФ);
- усиленная 3НФ или нормальная форма Бойса-Кодда (БКНФ);
- четвертая нормальная форма (4НФ);
- пятая нормальная форма (5НФ).

Отношение находится *в первой нормальной форме (1НФ),* когда каждая строка содержит только одно значение для каждого атрибута (столбца), то есть все атрибуты отношения имеют единственное значение (являются атомарными).

В столбце *Квалификация* ненормализованной табл. 1 содержатся списки значений (C, Java и т. д.). Чтобы привести схему к 1НФ, необходимо разместить в этом столбце атомарные значения. Самый простой способ заключается в выделении по одной строке на каждый элемент квалификации (таблица 1).

Таблица 1

Код сотрудника	ФИО	Должность	Номер отдела	Наименование отдела	Квалификация
7513	Иванов И.И.	Программист	128	Отдел проектирования	C
7513	Иванов И.И.	Программист	128	Отдел проектирования	Java
9842	Сергеева С.С.	Администратор БД	42	Финансовый отдел	DB2
6651	Петров П.П.	Программист	128	Отдел проектирования	VB
6651	Петров П.П.	Программист	128	Отдел проектирования	Java
9006	Николаев Н.Н.	Системный администратор	128	Отдел проектирования	Windows
9006	Николаев Н.Н.	Системный	128	Отдел	Linux

		администратор		проектирования	

Такое решение далеко от идеального, поскольку порождает очевидную избыточность данных – для каждой комбинации сотрудник-квалификация приходится хранить все характеристики сотрудника.

Отношение находится *во второй нормальной форме (2НФ)*, если оно находится в 1НФ, и каждый неключевой атрибут полностью функционально зависит от всех составляющих первичного ключа. Если атрибут не зависит полностью от первичного ключа, то он внесен ошибочно и должен быть удален. Нормализация производится путем нахождения существующего отношения, к которому относится данный атрибут, или созданием нового отношения, в который атрибут должен быть помещен.

Таблица *Квалификации_сотрудников* (табл. 14) находится в 1НФ, но не удовлетворяет 2НФ. Первичный ключ должен уникальным образом идентифицировать каждую строку. Единственным вариантом является использование в качестве первичного ключа комбинации *Код сотрудника* и *Квалификация*. Это порождает схему: *Квалификации_сотрудников (Код сотрудника, ФИО, Должность, Номер отдела, Наименование отдела, Квалификация)*.

Одной из имеющихся здесь функциональных зависимостей будет: *Код сотрудника, Квалификация → ФИО, Должность, Номер отдела, Наименование отдела.* Но, кроме того, мы также имеем: *Код сотрудника → ФИО, Должность, Номер отдела, Наименование отдела.* Другими словами, можно определить имя, должность и отдел, используя только код сотрудника. Это значит, что указанные атрибуты функционально зависимы только от части первичного ключа, а не от всего первичного ключа. Следовательно, указанная схема не находится в 2НФ.

Для приведения этой схемы в 2НФ необходимо декомпозировать исходное отношение на два, в которых все неключевые атрибуты будут полностью функционально зависеть от ключа: *сотрудники (Код сотрудника, ФИО, Должность, Номер отдела, Наименование отдела)* и *Квалификации_сотрудников (Код сотрудника, Квалификация)* (таблицы 2-3).

Таблица 2

Код сотрудника	ФИО	Должность	Номер отдела	Наименование отдела
7513	Иванов И.И.	Программист	128	Отдел проектирования
9842	Сергеева С.С.	Администратор БД	42	Финансовый отдел
6651	Петров П.П.	Программист	128	Отдел проектирования
9006	Николаев Н.Н.	Системный администратор	128	Отдел проектирования

Таблица 3

Код сотрудника	Квалификация
7513	C
7513	Java
9842	DB2
6651	VB
6651	Java
9006	Windows
9006	Linux

Отношение находится *в третьей нормальной форме (3НФ)*, если оно находится во 2НФ и ни один из его неключевых атрибутов не связан функциональной зависимостью с любым другим неключевым атрибутом. Атрибуты, зависящие от других неключевых атрибутов, нормализуются путем перемещения зависимого атрибута и атрибута, от которого он зависит, в новое отношение.

Формально, для приведения схемы в 3НФ необходимо исключить все транзитивные зависимости. Схема отношения *сотрудники* (таблица 3) содержит следующие функциональные зависимости: *Код сотрудника → ФИО, Должность, Номер отдела, Наименование отдела* и *Номер отдела → Наименование отдела*.

Первичным ключом является *Код сотрудника*, и все атрибуты полностью функционально зависимы от него (первичный ключ определяется единственным атрибутом). При этом *Номер отдела* ключом не является.

Функциональная зависимость *Код сотрудника → Наименование отдела* является транзитивной, поскольку содержит промежуточный шаг (зависимость *Номер отдела → Наименование отдела*). Для приведения в 3НФ необходимо исключить эту транзитивную зависимость, декомпозируя отношение на два: *сотрудники (Код сотрудника, ФИО, Должность, Номер отдела)* и *отделы (Номер отдела, Наименование отдела)* (таблицы 4-5).

Таблица 4

Код сотрудника	ФИО	Должность	Номер отдела
7513	Иванов И.И.	Программист	128
9842	Сергеева С.С.	Администратор БД	42
6651	Петров П.П.	Программист	128
9006	Николаев Н.Н.	Системный администратор	128

Таблица 5

Номер отдела	Наименование отдела
42	Финансовый отдел
128	Отдел проектирования

Нормальная форма Бойса-Кодда (БКНФ) является развитием 3НФ и требует, чтобы в отношении были только такие функциональные зависимости, левая часть которых является потенциальным ключом отношения. *Потенциальный ключ* представляет собой атрибут (или множество атрибутов), который может быть использован для данного отношения в качестве первичного ключа. Фактически первичный ключ – это один из потенциальных ключей, назначенный в качестве первичного. *Детерминантом* называется левая часть функциональной зависимости. Отношение находится в БКНФ тогда и только тогда, когда каждый детерминант отношения является потенциальным ключом.

Алгоритм приведения ненормализованных схем в 3НФ показан на рисунке 1. На практике построение 3НФ в большинстве случаев является достаточным и приведением к ней процесс построения реляционной БД заканчивается.

Нормальные формы высших порядков (4НФ и 5НФ) представляют больший интерес для теоретических исследований, чем для практики проектирования БД. В них учитываются многозначные зависимости между атрибутами. *Полной декомпозицией отношения* называют такую совокупность произвольного числа его проекций, соединение которых позволяет получить исходное отношение.

Отношение находится *в пятой нормальной форме (5НФ)*, когда в каждой его полной декомпозиции все проекции содержат возможный ключ. Отношение, не имеющее ни одной полной декомпозиции, также находится в 5НФ.

Четвертая нормальная форма (4НФ) является частным случаем 5НФ, когда полная декомпозиция должна быть соединением ровно двух проекций. На практике непросто подобрать отношение, которое находится в 4НФ, не будучи в 5НФ.

Рисунок 1 – Алгоритм приведения ненормализованных схем в 3НФ

Нормализация – это процесс последовательной замены отношения его полными декомпозициями до тех пор, пока все они не будут находиться в 5НФ. Приведя отношения к БКНФ, можно с большой гарантией считать, что они находятся в 5НФ. Единственными функциональными зависимостями в любом отношении должны быть зависимости вида К → А, где К – первичный ключ, а А – атрибут.

1.2 Практический пример. Учебное заведение имеет некоторое количество факультетов и институтов. На каждом из факультетов есть определённое количество кафедр. Кафедры выпускают инженеров по специальностям. Специальность имеет уникальный код.

Практически ежегодно кафедрой принимается новый учебный план для обучения очередного потока поступивших в учебное заведение студентов. В соответствии с ним проводится обучение студентов, контроль успеваемости и подведение итогов. Учебные планы отличаются друг от друга перечнями преподаваемых дисциплин. Для каждой из дисциплин определено число аудиторных и практических часов, отведённых под преподавание, формы контроля знаний студентов (экзамены, зачёты, курсовые работы, курсовые проекты). Таким образом, по одному учебному плану может обучаться несколько потоков. В потоке может быть несколько групп студентов. Студент имеет следующие атрибуты: номер зачётной книжки, ФИО, пол, дата рождения, адрес, телефон, номер группы, год поступления, документ об образовании. В процессе обучения студент сдаёт экзамены и зачёты, выполняет курсовые работы и проекты. Он также проходит три вида практики (учебная, производственная, преддипломная). В конце обучения студент подтверждает свои знания на государственном экзамене и выполняет дипломное проектирование. Информация о дипломном проекте и дипломе студента включает в себя следующие сведения: регистрационный номер диплома, тема дипломного проекта, руководитель, дата защиты, дата выдачи, оценка.

При завершении обучения очередного потока студентов подготавливаются и выдаются дипломы (листы-вкладыши). Данные листы имеют определённую форму и печатаются на бланках государственного образца. Они должны содержать информацию о специальности, итогах обучения студента, выполненных курсовых работах, дипломном проекте, присвоенной квалификации. Помимо этого документа, в процессе обучения студентов часто требуется подготовка отчётов различного содержания. Например, отчёт по студентам, не закрывшим сессию, и другие отчёты.

Таблица 6 – Типы сущностей основной базы данных

Название	Идентификатор	Описание
Факультет	faculty	Существующие в учебном заведении факультеты
Кафедра	subfaculty	Существующие на факультете кафедры
Специальность	speciality	Преподаваемые кафедрами специальности
Учебный план	plan_of	Учебные планы, по которым преподаются специальности
Поток	stream	Потоки обучающихся групп
Группа	group_of	Группы студентов
Студент	student	Студенты, обучающиеся в учебном заведении
Дисциплина	discipline	Дисциплины, преподаваемые в учебном заведении
Дисциплина по плану	dis_for_plan	Дисциплина, входящая в перечень дисциплин учебного плана
Форма контроля	control	Формы контроля успеваемости студентов (экзамен, зачёт, курсовая работа, курсовой проект)
Итог студента	result	Итоги сдачи экзаменов и зачётов
Гос. экзамен	exam	Государственный экзамен
Итог гос. экзамена	result_of_exam	Итоги сдачи студентами государственных экзаменов
Вид практики	type_of_p	Виды проходимых студентами практик
Практика по плану	practice_for_plan	Продолжительности практик в соответствии с учебным планом
Практика	practice	Результаты прохождения студентами практик
Курсовая работа	course_work	Сведения о выполненных студентами в процессе обучения курсовых работах и проектах
Диплом	diploma	Сведения о дипломных проектах и дипломах студентов

Таблица 7 – Типы связей основной базы данных

Название	Сущности	Тип	Мощность
содержит	Факультет – Кафедра	1:M	один или более
выпускает	Кафедра – Специальность	1:M	один или более
преподаёт	Кафедра – Дисциплина	1:M	ноль или более
преподаётся по	Специальность – Учебный план	1:M	один или более
составлен для	Учебный план – Поток	1:M	один или более
содержит	Учебный план – Дисциплин по плану	1:M	один или более
включает	Поток – Группа	1:M	один или более
включает	Группа – Студент	1:M	один или более
защитил	Студент – Диплом	1:1	один
сдал	Студент – Итог государственного экзамена	1:M	один или более
получил	Студент – Итог студента	1:M	один или более

выполнил	Студент – Курсовая работа	1:M	один или более
прошёл	Студент – Практика	1:M	один или более
варьируется	Дисциплина – Дисциплина по плану	1:M	один или более
имеет	Дисциплина по плану – Форма контроля	1:M	один или более
выполнена в	Форма контроля – Итог студента	1:M	один или более
выполнена в	Форма контроля – Курсовая работа	1:M	один или более
варьируется	Вид практики – Практика по плану	1:M	один или более
завершена	Практика по плану – Практика студента	1:M	один или более
сдан с итогом	Государственный экзамен – Итог гос экзамена	1:M	один или более

Таблица 8 - Атрибуты сущности «Факультет» (faculty)

Имя атрибута	Идентификатор	Формат	Область допустимых значений	Роль атрибута
Идентификатор факультета	fac_id	Цел.	Полож.	Первичный
Сокращённое название	nam	Симв. 10	[А-Я], [а-я]	
Полное название	full_nam	Симв. 50	[А-Я], [а-я]	

Таблица 9 – Атрибуты сущности «Кафедра» (subfaculty)

Имя атрибута	Идентификатор	Формат	Область допустимых значений	Роль атрибута
Идентификатор кафедры	subf_id	Цел.	Полож.	Первичный
Идентификатор факультета	fac_id	Цел.	Полож.	Внешний
Сокращённое название	nam	Симв. 15	[А-Я], [а-я]	
Полное название	full_nam	Симв. 50	[А-Я], [а-я]	

Таблица 10 – Атрибуты сущности «Специальность» (speciality)

Имя атрибута	Идентификатор	Формат	Область допустимых значений	Роль атрибута
Идентификатор специальности	spec_id	Цел.	Полож.	Первич-ный
Код специальности	code	Симв.6	[А-Я], [а-я], [0-9]	
Сокращённое название	nam	Симв.15	[А-Я], [а-я]	
Полное название	full_nam	Симв.100	[А-Я], [а-я]	
Продолжительность обучения	time_of	Вещ.	Полож.	
Идентификатор кафедры	subf_id	Цел.	Полож.	Внешний

Таблица 11 – Атрибуты сущности «Учебный план» (plan_of)

Имя атрибута	Идентификатор	Формат	Область допустимых значений	Роль
Идентификатор учебного плана	plan_id	Цел.	Полож.	Первичный
Идентификатор специальности	spec_id	Цел.	Полож.	Внеш
Год принятия	year_of	Цел.	Полож.	

Таблица 12 – Атрибуты сущности «Поток» (stream)

Имя атрибута	Идентификатор	Формат	Область допустимых значений	Роль
Название потока	stream_no	Симв. 4	[А-Я], [а-я], [0-9]	Первичный
Идентификатор учебного плана	plan_id	Цел.	Полож.	Внешний

Таблица 13 – Атрибуты сущности «Группа» (group_of)

Имя атрибута	Идентификатор	Формат	Область допустимых значений	Роль
Номер группы	group_no	Симв. 5	[А-Я], [а-я], [0-9]	Первичный
Название потока	stream_no	Симв. 4	[А-Я], [а-я]	Внешний

Таблица 14 – Атрибуты сущности «Студент» (student)

Имя атрибута	Идентификатор	Формат	Область допустимых значений	Роль атрибута
Идентификатор студента	number	Симв.7	[а-я], [А-Я], [0-9]	Первичный
Фамилия, имя отчество	fio	Симв. 40	[а-я], [А-Я]	
Пол	sex	Симв.3	«муж»/«жен»	
Дата рождения	dat_of_birth	Дата		
Адрес	address	Симв.50	[А-Я], [а-я], ,-,,,.	
Телефон	phone	Симв.11	[0-9]	
Год поступления	year_in	Цел.	Больше 1990	
Документ об образовании	docum	Симв. 80	[А-Я], [а-я], , -,,,.	
Номер группы	group_no	Симв. 5	[а-я], [А-Я], [0-9]	Внешний

Таблица 15 – Атрибуты сущности «Диплом» (diploma)

Имя атрибута	Идент	Формат	Область допустимых значений	Роль атрибута
Регистрационный номер диплома	reg_no	Цел.	Полож.	Первичный
Идентификатор студента	number	Симв.7	[а-я], [А-Я], [0-9]	Внешний
Номер диплома	dip_id	Симв.12	[А-Я], [а-я], [0-9]	
Тема диплома	topic	Симв.200	[А-Я], [а-я], , -,,,.	
Руководитель	leader	Симв.100	[А-Я], [а-я]	
Дата защиты	dat_pro	Дата		
Дата выдачи	dat_recv	Дата		
Оценка	mark	Симв.17	«отлично» / «хорошо» / «удовлетворительно»	

Таблица 16 – Атрибуты сущности «Дисциплина» (discipline)

Имя атрибута	Идентификатор	Формат	Область допустимых значений	Роль атрибута
Идентификатор дисциплины	dis_id	Цел.	Полож.	Первичный
Идентификатор	subf_id	Цел.	Полож.	Внешний

кафедры				
Сокращённое название	nam	Симв.15	[А-Я], [а-я]	
Полное название	full_nam	Симв.150	[А-Я], [а-я]	

Таблица 17 – Атрибуты сущности «Дисциплина по плану» (dis_for_plan)

Имя атрибута	Идентификатор	Формат	Область допустимых значений	Роль атрибута
Идентификатор дисциплины по плану	dis_plan_id	Цел.	Полож.	Первичный
Идентификатор дисциплины	dis_id	Цел.	Полож.	Внешний
Идентификатор учебного плана	plan_id	Цел.	Полож.	Внешний
Количество аудиторных часов	aud_hours	Цел.	Неотриц.	
Количество практических часов	pr_hours	Цел.	Неотриц.	

Таблица 18 – Атрибуты сущности «Форма контроля» (control)

Имя атрибута	Идентификатор	Формат	Область допустимых значений	Роль атрибута
Идентификатор контроля	contr_id	Цел.	Полож.	Первичный
Идентификатор дисциплины по плану	dis_plan_id	Цел.	Полож.	Внешний
Форма контроля	type	Симв.7	«экзамен» / «зачёт» / «работа» / «проект»	
Семестр	term	Цел.	Полож.	

Таблица 19 – Атрибуты сущности «Государственный экзамен» (exam)

Имя атрибута	Идентификатор	Формат	Область допустимых значений	Роль атрибута
Идентификатор гос. экзамена	exam_id	Цел.	Полож.	Первичный
Название	nam	Симв.50	[А-Я], [а-я]	

Таблица 20 – Атрибуты сущности «Итог гос. экзамена» (result_of_exam)

Имя атрибута	Идентификатор	Формат	Область допустимых значений	Роль атрибута
Идентификатор гос. экзамена	exam_id	Цел.	Полож.	Первичный
Идентификатор студента	stud_id	Цел.	Полож.	Внешний
Оценка	mark	Симв.17	«удовлетворительно»/ «хорошо»/ «отлично»	
Дата сдачи	data	Дата		

Таблица 21– Атрибуты сущности «Учебный план – Государственный экзамен» (plan_exam)

Имя атрибута	Идентификатор	Формат	Область допустимых значений	Роль атрибута

Идентификатор учебного плана	plan_id	Цел.	Полож.	Первичный
Идентификатор гос. экзамена	exam_id	Цел.	Полож.	Первичный

Таблица 22 – Атрибуты сущности «Вид практики» (type_of_p)

Имя атрибута	Идентификатор	Формат	Область допустимых значений	Роль атрибута
Идентификатор вида практики	type_id	Цел.	Полож.	Первичный
Вид практики	type	Симв.30	[А-Я], [а-я]	

Таблица 23 – Атрибуты сущности «Практика по плану» (practice_for_plan)

Имя атрибута	Идентификатор	Формат	Область допустимых значений	Роль атрибута
Идентификатор практики по плану	pr_id	Цел.	Полож.	Первичный
Идентификатор вида практики	type_id	Цел.	Полож.	Внешний
Длительность практики (в неделях)	time_of	Цел.	Полож.	

Таблица 24 – Атрибуты сущности «Учебный план – Практика по плану» (plan_practice)

Имя атрибута	Идентификатор	Формат	Область допустимых значений	Роль атрибута
Идентификатор плана	plan_id	Цел.	Полож.	Первичный, внешний
Идентификатор практики по плану	pr_id	Цел.	Полож.	Первичный, внешний

Таблица 25 – Атрибуты сущности «Итог практики» (practice)

Имя атрибута	Идентификатор	Формат	Область допустимых значений	Роль атрибута
Идентификатор студента	number	Симв.7	[а-я], [А-Я], [0-9]	Первичный, внешний
Идентификатор практики по плану	pr_id	Цел.	Полож.	Первичный, внешний
Тема	topic	Симв.150	[А-Я], [а-я], „ , -	
Место прохождения	place	Симв.100	[А-Я], [а-я], „ , -	
Руководитель	leader	Симв.40	[А-Я], [а-я],	
Оценка	mark	Симв.17	«отлично»/ «хорошо»/ «удовлетворительно»	
Даты защиты	data	Дата		

Таблица 26 – Атрибуты сущности «Итог студента» (result)

Имя атрибута	Идентификатор	Формат	Область допустимых значений	Роль атрибута
Идентификатор студента	number	Симв.7	[а-я], [А-Я], [0-9]	Первичный, внешний

Идентификатор формы контроля	contr_id	Цел.	Полож.	Первичный, внешний
Оценка	mark	Симв. 17	«отлично»/ «хорошо»/ «удовлетворительно»/ «зачтено»	
Дата сдачи	dat	Дата		

Таблица 27 – Атрибуты сущности «Курсовая работа» (course_work)

Имя атрибута	Идентифи-катор	Формат	Область допустимых значений	Роль атрибута
Идентификатор студента	number	Симв.7	[А-Я], [а-я], [0-9]	Первичный, внешний
Идентификатор формы контроля	contr_id	Цел.	Полож.	Первичный, внешний
Тема работы	topic	Симв. 200	[А-Я], [а-я], **,** , -	
Оценка	mark	Симв.17	«отлично»/ «хорошо»/ «удовлетворительно»	
Дата защиты	dat	Дата		

Отношение Факультет

Схема отношения:

Факультет (идентификатор факультета, сокращённое название, полное название).

Возможные ключи:

1) идентификатор факультета;
2) сокращённое название;
3) полное название.

В качестве первичного ключа был выбран идентификатор факультета.

Функциональные зависимости:

1) идентификатор факультета → сокращённое название, полное название;
2) сокращённое название → идентификатор факультета, полное название;
3) полное название → идентификатор факультета, сокращённое название.

Отношение находится в третьей нормальной форме.

Отношение Кафедра

Схема отношения:

Кафедра (идентификатор кафедры, сокращённое название, полное название, идентификатор факультета).

Возможные ключи:

1) идентификатор кафедры;
2) сокращённое название;
3) полное название.

В качестве первичного ключа был выбран идентификатор кафедры.

Функциональные зависимости:

1) идентификатор кафедры → сокращённое название, полное название, идентификатор факультета;
2) сокращённое название → идентификатор кафедры, полное название, идентификатор факультета;
3) полное название → идентификатор кафедры, сокращённое название, идентификатор факультета.

Отношение находится в третьей нормальной форме.

Отношение Специальность
Схема отношения:
Специальность (идентификатор специальности, код специальности, сокращённое название, полное название, продолжительность обучения, идентификатор кафедры).
Возможные ключи:
1) идентификатор специальности;
2) код специальности;
3) сокращённое название;
4) полное название.
В качестве первичного ключа был выбран идентификатор специальности.
Функциональные зависимости:
1) идентификатор специальности → код специальности, сокращённое название, полное название, продолжительность обучения, идентификатор кафедры;
2) код специальности → идентификатор специальности, сокращённое название, полное название, продолжительность обучения, идентификатор кафедры;
3) сокращённое название → идентификатор специальности, код специальности, полное название, продолжительность обучения, идентификатор кафедры;
4) полное название → идентификатор специальности, код специальности, сокращённое название, продолжительность обучения, идентификатор кафедры.
Отношение находится в третьей нормальной форме.

Отношение Учебный план
Схема отношения:
Учебный план (идентификатор учебного плана, идентификатор специальности, год принятия).
Возможные ключи:
1) идентификатор учебного плана.
Функциональные зависимости:
1) идентификатор учебного плана → идентификатор специальности, год принятия.
В качестве первичного ключа был выбран идентификатор учебного плана.
Отношение находится в третьей нормальной форме.

Отношение Поток
Схема отношения:
Поток (название потока, идентификатор учебного плана).
Возможные ключи:
1) название потока.
Функциональные зависимости:
1) название потока → идентификатор учебного плана.
Отношение находится в третьей нормальной форме.

Отношение Группа
Схема отношения:
Группа (номер группы, название потока).
Возможные ключи:
1) номер группы.
Функциональные зависимости:
1) номер группы → название потока.
Отношение находится в третьей нормальной форме.

Отношение Студент
Схема отношения:

Студент (идентификатор студента, ФИО, пол, дата рождения, адрес, телефон, год поступления, документ об образовании, номер группы).
Возможные ключи:
1) идентификатор студента;
2) ФИО.
В качестве первичного ключа был выбран идентификатор студента.
Функциональные зависимости:
1) идентификатор студента → ФИО, пол, дата рождения, адрес, телефон, год поступления, документ об образовании, номер группы;
2) ФИО → идентификатор студента, пол, дата рождения, адрес, телефон, год поступления, документ об образовании, номер группы.
Отношение находится в третьей нормальной форме.

Отношение Диплом
Схема отношения:
Диплом (регистрационный номер, идентификатор студента, идентификатор диплома, тема, руководитель, дата защиты, дата выдачи, оценка).
Возможные ключи:
1) регистрационный номер;
2) идентификатор студента;
3) идентификатор диплома.
В качестве первичного ключа был выбран регистрационный номер диплома.
Функциональные зависимости:
1) регистрационный номер → идентификатор диплома, идентификатор студента, тема, руководитель, дата защиты, дата выдачи, оценка;
2) номер диплома → идентификатор студента, регистрационный номер, тема, руководитель, дата защиты, дата выдачи, оценка;
3) идентификатор студента → идентификатор диплома, регистрационный номер, тема, руководитель, дата защиты, дата выдачи, оценка.
Отношение находится в третьей нормальной форме.

Отношение Государственный экзамен
Схема отношения:
Государственный экзамен (идентификатор государственного экзамена, название).
Возможные ключи:
1) идентификатор государственного экзамена;
2) название.
В качестве первичного ключа был выбран идентификатор государственного экзамена.
Функциональные зависимости:
1) идентификатор государственного экзамена → название;
2) название → идентификатор государственного экзамена.
Отношение находится в третьей нормальной форме.

Отношение Учебный план – Итог государственного экзамена
Учебный план – Итог государственного экзамена (идентификатор учебного плана, идентификатор государственного экзамена).
Возможные ключи:
1) идентификатор учебного плана, идентификатор государственного экзамена.
Отношение находится в третьей нормальной форме.

Отношение Итог государственного экзамена
Схема отношения:

Итог государственного экзамена (идентификатор студента, идентификатор государственного экзамена, оценка, дата сдачи).
Возможные ключи:
1) идентификатор студента, идентификатор государственного экзамена.
Функциональные зависимости:
1) идентификатор государственного экзамена, идентификатор студента → оценка, дата сдачи.
Отношение находится в третьей нормальной форме.

Отношение Вид практики
Схема отношения:
Вид практики (идентификатор вида практики, вид практики).
Возможные ключи:
1) идентификатор вида практики;
2) вид практики.
Функциональные зависимости:
1) идентификатор вида практики → вид практики;
2) вид практики → идентификатор вида практики.
Отношение находится в третьей нормальной форме.

Отношение Практика по плану
Схема отношения:
Практика по плану (идентификатор практики по плану, идентификатор вида практики, длительность практики).
Возможные ключи:
1) идентификатор практики по плану;
2) идентификатор вида практики, длительность практики.
В качестве первичного ключа был выбран идентификатор практики по плану.
Функциональные зависимости:
1) идентификатор практики по плану → идентификатор вида практики, длительность практики;
2) идентификатор вида практики, длительность практики → идентификатор практики по плану.
Отношение находится в третьей нормальной форме.

Отношение Учебный план – Практика по плану
Учебный план – Практика по плану (идентификатор учебного плана, идентификатор практики по плану).
Возможные ключи:
1) идентификатор учебного плана, идентификатор практики по плану.
Отношение находится в третьей нормальной форме.

Отношение Итог практики
Схема отношения:
Итог практики (идентификатор студента, идентификатор практики по плану, тема, место прохождения, руководитель, оценка, дата защиты).
Возможные ключи:
1) идентификатор студента, идентификатор практики по плану.
Функциональные зависимости:
1) идентификатор студента, идентификатор практики по плану → тема, место прохождения, руководитель, оценка, дата защиты.
Отношение находится в третьей нормальной форме.

Отношение Дисциплина
Схема отношения:
Дисциплина (идентификатор дисциплины, сокращённое название, полное название, идентификатор кафедры).
Возможные ключи:
1) идентификатор дисциплины;
2) сокращённое название;
3) полное название.
В качестве первичного ключа был выбран идентификатор дисциплины.
Функциональные зависимости:
1) идентификатор дисциплины → сокращённое название, полное название, идентификатор кафедры;
2) сокращённое название → идентификатор дисциплины, полное название, идентификатор кафедры;
3) полное название → идентификатор дисциплины, сокращённое название, идентификатор кафедры.
Отношение находится в третьей нормальной форме.

Отношение Дисциплина по плану
Схема отношения:
Дисциплина по плану (идентификатор дисциплины по плану, идентификатор плана, идентификатор дисциплины, число аудиторных часов, число практических часов).
Возможные ключи:
1) идентификатор дисциплины по плану;
2) идентификатор плана, идентификатор дисциплины.
В качестве первичного ключа был выбран идентификатор дисциплины по плану.
Функциональные зависимости:
1) идентификатор дисциплины по плану → идентификатор дисциплины, число аудиторных часов, число практических часов, число экзаменов, число зачётов, число курсовых проектов, число курсовых работ;
2) идентификатор плана, идентификатор дисциплины → идентификатор дисциплины по плану, число аудиторных часов, число практических часов.
Отношение находится в третьей нормальной форме.

Отношение Форма контроля
Схема отношения:
Форма контроля (идентификатор формы контроля, идентификатор дисциплины по плану, форма контроля, семестр).
Возможные ключи:
1) идентификатор формы контроля;
2) идентификатор дисциплины по плану, форма контроля, семестр.
В качестве первичного ключа был выбран идентификатор формы контроля.
Функциональные зависимости:
1) идентификатор формы контроля → идентификатор дисциплины по плану, форма контроля, семестр;
2) идентификатор дисциплины по плану, форма контроля, семестр → идентификатор формы контроля.
Отношение находится в третьей нормальной форме.

Отношение Курсовая работа
Схема отношения:

Курсовая работа (идентификатор студента, идентификатор формы контроля, тема, дата защиты, оценка).
Возможные ключи:
1) идентификатор студента, идентификатор формы контроля.
Функциональные зависимости:
1) идентификатор студента, идентификатор формы контроля → тема, дата защиты, оценка.
Отношение находится в третьей нормальной форме.

Отношение Итог студента
Схема отношения:
Итог студента (идентификатор студента, идентификатор формы контроля, дата сдачи, оценка).
Возможные ключи:
1) идентификатор студента, идентификатор формы контроля.
Функциональные зависимости:
1) идентификатор студента, идентификатор формы контроля → дата сдачи, оценка.
Отношение находится в третьей нормальной форме.

1.3 Задание на лабораторную работу:
1 Провести анализ схем отношений имеющейся базы данных (созданной в курсе «Инструментальные средства информационных систем»). Определить нормальные формы для каждого из отношений, предварительно выделив функциональные зависимости.
2 Провести анализ возможных аномалий обновления и избыточности для каждого из отношений.
3 При наличии аномалий на основе анализа функциональных зависимостей провести декомпозицию отношений.

1.4 Содержание отчета. Отчет должен содержать следующие разделы:
1. Титульный лист;
2. Описание предметной области исходной базы данных, включая бизнес-ограничения, накладываемые на экземпляры сущностей.
Например:
1. Для каждого рейса назначается только одно время вылета.
2. Для данного пилота, даты и времени вылета возможен только один рейс.
3. Для данного рейса и даты назначается только один пилот.
3. Описание логической модели исходной базы данных:
- диаграмма логической модели исходной базы данных;
- описание схем отношений исходной базы данных.
4. Результаты анализа схем отношений исходной базы данных:
- наборы выделенных функциональных зависимостей для каждого отношения;
- выводы о соответствии схем отношений той или иной нормальной форме и наличия аномалий модификации данных.
5. Результаты модификации (декомпозиции) схем отношений для устранения выделенных аномалий (при их наличии).
6. Описание логической модели результирующей базы данных, полученной в процессе изменения схем отношений исходной базы данных на основе декомпозиции отношений (если изменения были выполнены):
- диаграмма логической модели результирующей базы данных;
- описание схем отношений результирующей базы данных.
7. Краткие итоги выполнения работы и выводы.

Лабораторная работа № 1.2 Разработка программных средств наполнения оперативной базы данных тестовыми данными

Цель работы. Разработка функций и хранимых процедур для автоматического заполнения оперативной базы данных тестовыми данными

1.1 Теоретическая часть. Перед разработчиком часто возникает задача провести тест базы данных на больших объемах данных, но откуда взять эти данные? Ведь всем известно, что структура базы может достигать несколько десятков таблиц, которые не очень удобно заполнять руками. А если учитывать внешние ключи и составные первичные ключи, значения которых связаны с другими таблицами, то задача серьезно усложняется. Существуют решения заполнения базы данных (БД) случайными значениями с использованием средств .NET, C++, Java и.д. Но часто, за отсутствием клиентских приложений на стадии отладки и тестирования, необходимо для заполнения БД разработать серверные программные средств, например, на языке T-SQL под управлением СУБД MS SQL Server.

Имеются следующие входные требования:

- Необходимо заполнить случайными данными БД с некоторым числом связанных таблиц;
- Все первичные ключи (PK) – суррогатные (искусственные) ключи со свойством автоинкремента;
- Существуют таблицы, содержащие в себе составной PK состоящие из внешних ключей (FK).

Задача состоит в следующем:

- Необходимость генерировать случайные данные в зависимости от типа атрибута (столбца);
- Необходимо пропускать заполнение первичных ключей – автоинкрементов;
- Заполнять FK дочерних таблиц случайными PK родительских таблиц.

Одним из возможных решений является разработка набора хранимых процедур, каждая из которых заполняет данными конкретную таблицу БД, например:

- AutoInsertCompanie – процедура заполнения случайными данными таблицы Компания;
- AutoInsertOrder – процедура заполнения случайными данными таблицы Заказ.

Возможно создание вспомогательных процедур для генерации случайных значений для атрибутов соответствующих таблиц, например:

- RandomCompanie – процедура генерации случайных данных для столбцов таблицы Компания;
- RandomOrder – процедура генерации случайных данных для столбцов таблицы Заказ.

Другим возможным решением является разработка набора «универсальных» хранимых процедур, вызывающих друг друга:

- randomString – генерация случайной строки символов заданной длины;
- randomInt – генерация случайного числа из заданного диапазона;
- generateDataByType – получает тип атрибута (столбца) таблицы и вызывает нужную процедуру генерации случайных значений.
- insertRandomData — основная процедура, на вход которой передается только имя таблицы и количество записей, которое необходимо добавить.

1.2 Практический пример. Пример реализации хранимой процедуры **AutoInsertCompanie** может быть следующий:

```
CREATE PROCEDURE AutoInsertCompanie
@count INT
```

```
AS
DECLARE @CompanieName NVARCHAR(MAX)
DECLARE @CompanieAddress NVARCHAR(MAX)
DECLARE @CompaniePhone NVARCHAR(MAX)
DECLARE @CompanieDiscount FLOAT
DECLARE @i INT = 0

WHILE @i < @count
BEGIN
EXEC RandomCompanie6 @CompanieName output,@CompanieAddress output,
@CompaniePhone output ,@CompanieDiscount output
INSERT INTO Companies VALUES
(@CompanieName,@CompanieAddress,@CompaniePhone,@CompanieDiscount)
SET @i = @i + 1
END
```

Пример реализации хранимой процедуры Random**Companie** может быть следующий:

```
CREATE PROCEDURE RandomCompanie
@OutputCompanieName nvarchar(max) output,
@OutputCompanieAddress nvarchar(max) output,
@OutputCompaniePhone nvarchar(max) output,
@OutputCompanieDiscount FLOAT output
AS
BEGIN
DECLARE @CompanieDiscount FLOAT
DECLARE @CompanieName NVARCHAR(MAX) = ''
DECLARE @CompanieAddress NVARCHAR(MAX) = ''
DECLARE @CompaniePhone NVARCHAR(MAX) = ''

SET @CompanieDiscount = ABS(CHECKSUM(NewId())) % 25.01 + 0.5
SET @CompanieName = CAST((SELECT TOP (1) a
      FROM(
            VALUES
            ('Рив Гош'), ('Море парфюма'), ('Yves rocher'),
            ('Dzintars'), ('Mary Kay'), ('ЛЭтуаль'),
            ('Phyts'), ('Для душа и души'), ('Национальный научно-
производственный центр технологии омоложения'),
            ('Креатив'), ('Фортуна'), ('Мир красоты'),
            ('Иль де Ботэ'), ('Profilook'), ('Дом Парфюмера')
            )AS Name(a)
      ORDER BY ABS(CHECKSUM(NewId())) AS VARCHAR(300))

SET @CompanieAddress ='ул. '+ CAST((SELECT TOP (1) b
      FROM(
            VALUES
            ('Московская'), ('Баумана'), ('Бакунина'),
            ('Сверлова'), ('Беляева'), ('Байдукова'),
            ('Бурденко'), ('Виражная'), ('Гоголя'),
            ('Грибоедова'), ('Горького'), ('Толстого'),
            ('Пушкина'), ('Ягодная'), ('Красная'),
            ('Жемчужная'), ('Дружбы'), ('Заводская'),
            ('Захарова'), ('Злобина'), ('Ижевская'),
            ('Тихомирова'), ('Красная'), ('Зеленая'),
            ('Воронова'), ('Оранжевая'), ('Ломоносова'),
            ('Южная'), ('Карамзина'), ('Лермонтова'),
            ('Северная'), ('Захарова'), ('Цветочная')
            )AS Street(b)
      ORDER BY ABS(CHECKSUM(NewId())) AS VARCHAR(200))
      + ' ' + CAST(((ABS(CHECKSUM(NewId())) % 99 + 1)) AS VARCHAR) + ',
'
      + CAST((SELECT TOP (1) c
```

```
      FROM(
            VALUES
            ('Кузнецк'), ('Белинский'), ('Нижний Ломов'),
            ('Пенза'), ('Заречный'), ('Каменка'),
            ('Никольск'), ('Сердобск'), ('Городище')
            )AS Region(c)
      ORDER BY ABS(CHECKSUM(NewId())) AS VARCHAR(200))

SET @CompaniePhone = '8' + SUBSTRING(CAST(((ABS(CHECKSUM(NewId())))) AS
VARCHAR(50)),1,9)
SET @OutputCompanieName  = @CcompanieName
SET @OutputCompanieAddress = @CompanieAddress
SET @OutputCompaniePhone  = @CompaniePhone
SET @OutputCompanieDiscount = @CompanieDiscount
END
```

Шаблон реализации хранимой процедуры **randomString** может быть следующим:

```
CREATE PROCEDURE [dbo].[randomString]
@inputSize int,
@outputRandomString nvarchar(max) output
AS
BEGIN
        -- Любой известный алгоритм реализации случайной строки
заданной длины.
END;
```

Процедура **randomString** получает на вход длину строки, а на выходе выдает строку случайных символов типа NVARCHAR(MAX) нужного размера. В данном случае реализация не является критичной по времени, так как не имеет серьезных временных затрат при больших объемах данных.

Процедура **randomInt** может быть реализована следующим образом:

```
CREATE PROCEDURE [dbo].[randomInt]
@inputSize int,
@outputInteger int output
AS
BEGIN
  DECLARE @TEMP bigint
  SET @TEMP = SUBSTRING('999999999999999999',1,@inputSize)
  SET @outputInteger = (ABS(CHECKSUM(NewId())) % @TEMP)
END
```

Возможно, процедура **randomInt** реализована не оптимальным образом, но зато она обладает приемлемым быстродействием.

Одним из возможных вариантов реализации процедуры **generateDataByType** может быть следующий:

```
CREATE PROCEDURE [dbo].[generateDataByType]
   @tableName nvarchar(40), -- имя таблицы, для который будем
генерировать данные
   @inputColumName nvarchar(40), -- имя столбца, для которого будем
генерировать данные
   @inputType nvarchar(10),
   @inputSize int,
        @outputString nvarchar(max) output   --готовая строка
AS
BEGIN
         DECLARE @isFK bit = 0;
  DECLARE @FKName NVARCHAR(MAX);
  DECLARE @ParentTable NVARCHAR(MAX);
```

```
        --Для @tableName получаем имена полей являющихся FK
(ccu.table_name) и мена родительских таблиц (references_table) на
которые эти ключи ссылаются и загоняем все под курсор
        DECLARE columnsCursor1 CURSOR FOR
          SELECT  kcu.column_name,
                 ccu.table_name AS references_table
                  FROM information_schema.table_constraints tc
         INNER JOIN information_schema.key_column_usage kcu
                   ON tc.constraint_catalog = kcu.constraint_catalog
                  AND tc.constraint_schema = kcu.constraint_schema
                  AND tc.constraint_name = kcu.constraint_name
         INNER JOIN information_schema.referential_constraints rc
                   ON tc.constraint_catalog = rc.constraint_catalog
                  AND tc.constraint_schema = rc.constraint_schema
                  AND tc.constraint_name  =  rc.constraint_name
                  AND  tc.constraint_type = 'FOREIGN KEY'
         INNER JOIN information_schema.constraint_column_usage ccu
                   ON rc.unique_constraint_catalog =
ccu.constraint_catalog
                  AND rc.unique_constraint_schema =
ccu.constraint_schema
                  AND rc.unique_constraint_name = ccu.constraint_name
                  WHERE tc.table_name = @tableName
        OPEN columnsCursor1;

  FETCH NEXT FROM columnsCursor1
         INTO @FKName,@ParentTable
        --пробегаемся по каждому внешнему ключу
  WHILE @@FETCH_STATUS = 0
  BEGIN
         -- проверяем, является ли пришедший на вход процедуре столбец
@inputColumName найденным ранее внешним ключом данной таблицы
         IF (@inputColumName = @FKName)
         BEGIN
                 SET @isFK = 1;   --устанавливаем флаг в true - работаем
с FK и других проверок на тип данных делать не нужно.
                 DECLARE @selectedPK NVARCHAR(MAX);
                 DECLARE @params NVARCHAR(MAX);
                  -- формируем динамический запрос и забираем случайный
первичный ключ из родительской таблицы, который и станет внешним ключом
для текущей таблицы
                 SET @selectedPK = N'SELECT TOP 1 @outputString =' +
@FKName + ' FROM ' + @ParentTable + ' ORDER BY NEWID(); ';
                 SET @params = N'@FKName NVARCHAR(MAX), @ParentTable
NVARCHAR(MAX), @outputString NVARCHAR(MAX) OUTPUT';
                 EXEC sp_executesql @selectedPK , @params, @FKName =
@FKName, @ParentTable = @ParentTable, @outputString = @outputString
OUTPUT;
         END

         FETCH NEXT FROM columnsCursor1
         INTO @FKName,@ParentTable
  END;
  CLOSE columnsCursor1;
  DEALLOCATE columnsCursor1;

        --если столбец таблицы не является внешним ключом заполняем его
случайными данными.
        IF (@isFK <> 1)
         BEGIN

                  IF (@inputType = 'nvarchar')
                  BEGIN
                         EXECUTE randomString
```

```
                        @inputSize, @outputRandomString = @outputString
OUTPUT ;
                END

  ELSE
            --тоже самое выполняем для других типов данных и вызываем
нужные процедуры.
END
```

Применение указанного варианта процедуры **generateDataByType** мы получаем **огромные** временные затраты при заполнении внешнего ключа таблицы данными из найденной родительской таблицы. Если данный поиск заменитьть подстановкой случайных чисел в заданном диапазоне производительность резко возрастает. Возможно дело в SELECT из системной таблицы и случайной сортировки. Для сравнения: запись 1 млн. строк в таблицу без FK занимает около 20 мин, запись 1 млн. строк в таблицу с FK занимает больше 17 часов. *Для справки, запись одного миллиона строк чистым INSERT в одно поле занимаем 6-10 сек.*

Хранимая процедура может быть реализована в соответствие со следующим шаблоном:

```
CREATE PROCEDURE [dbo].[insertRandomData]
@childTableName nvarchar(MAX),
@insertRowCount int
AS
BEGIN
  DECLARE @i int = 0
  /*ПЕРЕМЕННЫЕ ДЛЯ КУРСОРА*/
  DECLARE @columnName NVARCHAR(30); DECLARE @columnType NVARCHAR(10);
  DECLARE @columnLenght INT; DECLARE @columnUniq INT;

  /*ПЕРЕМЕННЫЕ ДЛЯ ДИНАМИЧЕСКОГО СОЗДАНИЯ ЗАПРОСА К БД*/
  DECLARE @insertQuery NVARCHAR(MAX);   DECLARE @insertColumnsQuery
NVARCHAR(MAX); DECLARE @insertValuesQuery VARCHAR(MAX);DECLARE @params
NVARCHAR(MAX);

  SET @insertColumnsQuery = '';
  SET @insertValuesQuery = '';

  begin transaction
  WHILE (@i < @insertRowCount)
  BEGIN
         DECLARE columnsCursor CURSOR FOR
                ----------Получаем в запросе типы и размеры столбцов
таблицы @childTableName и загоняем под курсор-----
                SELECT
                                all_columns.column_id,
                                all_columns.name,
                                systypes.name,
                                all_columns.max_length
                        FROM
                                SYS.all_objects
                                        join SYS.all_columns on
all_columns.object_id = all_objects.object_id
                                        join SYS.systypes on
all_columns.system_type_id = systypes.xtype
                        WHERE
                                all_objects.name like @childTableName
and
                                all_objects.type = 'U' AND
                                systypes.name <> 'sysname'/*ПОЧЕМУ-ТО
НА ОДНО ПОЛЕ ПРИХОДИТСЯ 2 ТИПА ДАННЫХ (ИСКЛЮЧАЕМ СИСТЕМНЫЕ)*/
```

```
                        ORDER BY
                                all_columns.column_id;

        OPEN columnsCursor;
        --Выполняет действие дважды, чтобы пропустить ID-автоинкремент
(не самое лучшее решение)
                FETCH NEXT FROM columnsCursor
                        INTO @columnUniq, @columnName, @columnType,
@columnLenght;
                FETCH NEXT FROM columnsCursor
                        INTO @columnUniq, @columnName, @columnType,
@columnLenght;

        DECLARE @tempLenght INT = 0;
        WHILE @@FETCH_STATUS = 0
                BEGIN
                        /*ОПРЕДЕЛЯЕМ ДЛИНУ ПОЛЯ (ЕСЛИ ТЕКСТ ТО -1
ЗАМЕНЯЕМ НА 30) , -1 возвращается при MAX размере типа, в такие поля я
буду вставлять строку из 30 символов. */
                        IF(@columnLenght >= 0)
                        BEGIN
                                SET @tempLenght = @columnLenght;
                        END
                        ELSE
                        BEGIN
                                SET @tempLenght = 30;
                        END

                        --формируем левую часть запроса INSERT,
содержит название столбцов.
                        SET @insertColumnsQuery = @insertColumnsQuery +
@columnName + ', ';

                        DECLARE @TEMPValues nvarchar(MAX) = ''
                        ---Вызываем процедуру generateStringByType,
записываем результат в TEMPValues---
                        EXECUTE generateDataByType

  @childTableName,

  @columnName,

  @columnType,

  @tempLenght,

  @outputString = @TEMPValues OUTPUT

                                -- формируем правую часть запроса
INSERT, содержит данные.
                        SET @insertValuesQuery = @insertValuesQuery
+'''' + @TEMPValues + ''','

                      FETCH NEXT FROM columnsCursor
                                INTO @columnUniq, @columnName,
@columnType, @columnLenght;
                END;
                --Убираем лишнюю запятую в конце каждой части запроса
типа INSERT
                SET @insertColumnsQuery =
SUBSTRING(@insertColumnsQuery, 1, LEN(@insertColumnsQuery)-1);
                SET @insertValuesQuery =
SUBSTRING(@insertValuesQuery,1, LEN(@insertValuesQuery)-1);
```

```
                -- Формируем запрос на вставку и выполняем его.
                SET @insertQuery = N'INSERT INTO ' + @childTableName +
N' (' + @insertColumnsQuery + N') VALUES (' + @insertValuesQuery + ')
;';
                EXEC(@insertQuery);
        --дальше ничего интересного
```

Данная процедура является «относительно» не затратной по времени хотя и обращается к системным таблицам, чтобы получить структуру пришедшей на вход таблицы, но содержит в себе несколько явных слабых мест. Например, прыжок через первый элемент таблицы в надежде на то, что именно он являет PK.

1.3 Задание на лабораторную работу.

1. Разработать набор хранимых процедур для заполнения случайным образом БД тестовыми данными в соответствие с одним из предлагаемых подходов.
2. Реализация хранимых процедур заполнения БД должна быть эффективной с точки зрения временных затрат.
3. Заполнить БД тестовыми данными из расчета, что в таблицах-справочниках (классификаторах) должно быть от нескольких десятков до нескольких сотен записей, в стержневых таблицах должно быть несколько сотен тысяч записей.

1.4 Содержание отчета. Отчет должен содержать следующие разделы:

1. Титульный лист;
2. Схему базы данных в графическом виде (диаграмму данных);
3. Спецификации и тексты хранимых процедур для заполнения БД тестовыми данными;
4. Скриншоты, демонстрирующие корректное выполнение хранимых процедур и заполнение БД;
5. Краткие итоги выполнения работы и выводы.

абораторная работа № 1.3 Разработка аналитических запросов к оперативной базе данных

Цель работы. Разработка аналитических запросов к оперативной базе данных на языке SQL, включающих промежуточные и общие итоги в результирующий набор данных

1.1. Теоретическая часть. В Transact-SQL имеется возможность формировать отчеты, как со строкой общего итога, так и со строками промежуточных итогов. Для этих целей в MS SQL Server существуют такие операторы как **ROLLUP, CUBE и GROUPING SETS**. Для того чтобы подсчитать общий или промежуточный итог необходимо использовать агрегатные функции и соответственно группировку данных, поэтому операторы ROLLUP, CUBE и GROUPING SETS относятся к конструкции GROUP BY и, фактически, являются расширением GROUP BY.

Некоторые результирующие наборы, сформированные запросами, в которых используются операторы ROLLUP и CUBE, а также некоторые совершаемые ими вычисления совпадают с вычислениями приложений OLAP. Оператор CUBE формирует результирующий набор, который может быть использован для перекрестных отчетов. Операция ROLLUP может вычислить эквивалент измерения или иерархии OLAP.

Например, если существует измерение времени с уровнями или атрибутами года, месяца и дня, то приведенная в таблице 1 операция ROLLUP сформирует следующие группировки.

Таблица 1

Операция	**Группирования**
ROLLUP (DATEPART(yyyy,OrderDate), DATEPART(mm,OrderDate), DATEPART(dd,OrderDate))	year, month, day year, month year ()

Если существует измерение местонахождения, имеющее уровни региона и города, объединенные с уровнями измерения времени – годом, месяцем и днем, то следующая операция ROLLUP выдаст следующие группирования (таблица 2).

Таблица 2

Операция	**Группирования**
ROLLUP (region, city, DATEPART(yyyy,OrderDate), DATEPART(mm,OrderDate), DATEPART(dd,OrderDate))	region, city, year, month, day region, city, year, month region, city, year region, city region, year, month, day region, year, month region, year region year, month, day year, month year ()

Операция CUBE, использующая те же уровни измерений местоположения и времени, выдаст следующие группирования (таблица 3).

Таблица 3

Операция	**Группирование**
CUBE (region, city, DATEPART(yyyy,OrderDate),	region, city, year, month, day region, city, year, month

DATEPART(mm,OrderDate), DATEPART(dd,OrderDate))	region, city, year region, city region, city, month, day region, city, month region, city, day region, city, year, day region, city, day region, year, month, day region, year, month region, year region, month, day region, month region, year, day region, day region city, year, month, day city, year, month city, year city, month, day city, month city, year, day city, day year, month, day year, month year year, day month, day month day ()

Значения NULL в результирующих наборах. Если результирующие наборы формируются операторами предложения GROUP BY, значения NULL используются следующим способом.

- Если в столбце, по которому производится группирование, содержится значение NULL, то все значения NULL считаются равными и помещаются в одну группу NULL.
- При статистической обработке столбца в строку в качестве значения этого столбца отображается значение NULL.

В следующем примере функция GROUPING используется для демонстрации двух этих способов применения значений NULL. Значение UNKNOWN заменяет значение NULL в тех строках, в которых значения NULL столбца были сгруппированы. Значение ALL заменяет значение NULL для столбца, в котором значение NULL означает то, что столбец был включен в статистическое выражение.

```
USE tempdb;
GO
CREATE TABLE dbo.GroupingNULLS (
   Store nvarchar(19)
   ,SaleYear nvarchar(4)
   ,SaleMonth nvarchar (7))
INSERT INTO dbo.GroupingNULLS VALUES
(NULL,NULL,'January'), (NULL,'2002',NULL), (NULL,NULL,NULL),
('Active Cycling',NULL ,'January'), ('Active Cycling','2002',NULL),
('Active Cycling',NULL ,NULL), ('Active Cycling',NULL,'January'),
('Active Cycling','2003','Febuary'), ('Active Cycling','2003',NULL),
('Mountain Bike Store','2002','January'), ('Mountain Bike Store','2002',NULL),
('Mountain Bike Store',NULL,NULL), ('Mountain Bike Store','2003','January'),
```

```
('Mountain Bike Store','2003','Febuary'), ('Mountain Bike Store','2003','March');
SELECT ISNULL (Store,
   CASE WHEN GROUPING(Store) = 0 THEN 'UNKNOWN' ELSE 'ALL' END)
   AS Store
   ,ISNULL(CAST(SaleYear AS nvarchar(7)),
   CASE WHEN GROUPING(SaleYear)= 0 THEN 'UNKNOWN' ELSE 'ALL' END)
   AS SalesYear
   ,ISNULL(SaleMonth,
   CASE WHEN GROUPING(SaleMonth) = 0 THEN 'UNKNOWN' ELSE 'ALL'END)
   AS SalesMonth
   ,COUNT(*) AS Count
FROM dbo.GroupingNULLS
GROUP BY ROLLUP(Store, SaleYear, SaleMonth);
```

В таблице 4 приводится результирующий набор данных.

Таблица 4

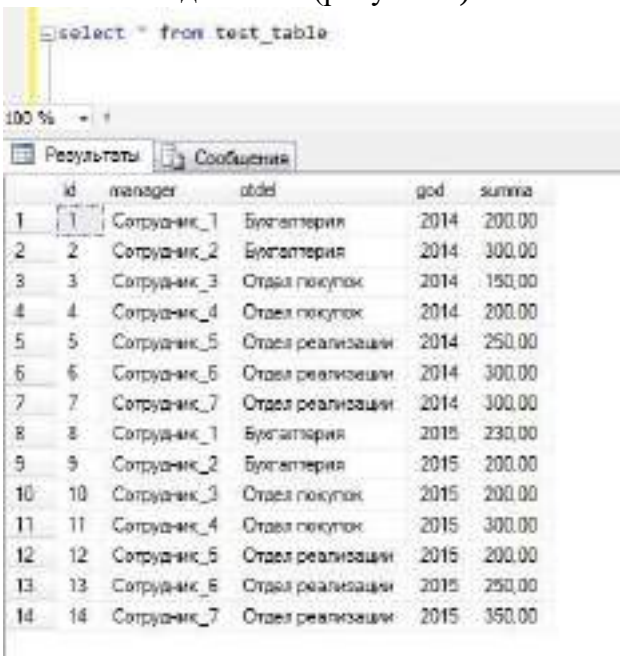
Store	SalesYear	SalesMonth	Count
Unknown	Unknown	Unknown	1
Unknown	Unknown	January	1
Unknown	Unknown	ALL	2
Unknown	2002	Unknown	1
Unknown	2002	ALL	1
Unknown	ALL	ALL	3
Active Cycling	Unknown	Unknown	1
Active Cycling	Unknown	January	2
Active Cycling	Unknown	ALL	3
Active Cycling	2002	Unknown	1
Active Cycling	2002	ALL	1
Active Cycling	2003	Unknown	1
Active Cycling	2003	February	1
Active Cycling	2003	ALL	2
Active Cycling	ALL	ALL	6
Mountain Bike Store	Unknown	Unknown	1
Mountain Bike Store	Unknown	ALL	1
Mountain Bike Store	2002	Unknown	1
Mountain Bike Store	2002	January	1
Mountain Bike Store	2002	ALL	2
Mountain Bike Store	2003	February	1
Mountain Bike Store	2003	January	1
Mountain Bike Store	2003	March	1
Mountain Bike Store	2003	ALL	3
Mountain Bike Store	ALL	ALL	6
ALL	ALL	ALL	15

1.2 Практический пример. Запросы создаются к таблице, которая содержит список сотрудников с указанием отдела, в котором они работают, а также сумму их заработка по годам:

```
CREATE TABLE [dbo].[test_table](
        [id] [int] IDENTITY(1,1) NOT NULL,
        [manager] [varchar](50) NULL,
        [otdel] [varchar](50) NULL,
        [god] [int] NULL,
```

```
        [summa] [money] NULL
) ON [PRIMARY]
GO
```

Таблица test_table заполнена данными (рисунок 1).

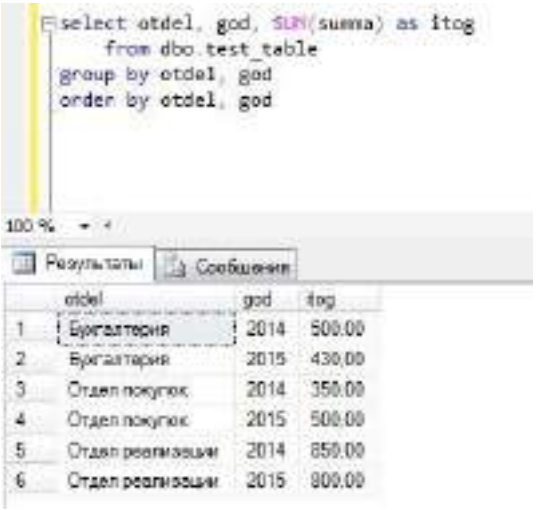


	id	manager	otdel	god	summa
1	1	Сотрудник_1	Бухгалтерия	2014	200.00
2	2	Сотрудник_2	Бухгалтерия	2014	300.00
3	3	Сотрудник_3	Отдел покупок	2014	150.00
4	4	Сотрудник_4	Отдел покупок	2014	200.00
5	5	Сотрудник_5	Отдел реализации	2014	250.00
6	6	Сотрудник_6	Отдел реализации	2014	300.00
7	7	Сотрудник_7	Отдел реализации	2014	300.00
8	8	Сотрудник_1	Бухгалтерия	2015	230.00
9	9	Сотрудник_2	Бухгалтерия	2015	200.00
10	10	Сотрудник_3	Отдел покупок	2015	200.00
11	11	Сотрудник_4	Отдел покупок	2015	300.00
12	12	Сотрудник_5	Отдел реализации	2015	200.00
13	13	Сотрудник_6	Отдел реализации	2015	250.00
14	14	Сотрудник_7	Отдел реализации	2015	350.00

Рисунок 1 – Исходные данные в таблице test_table

ROLLUP – оператор Transact-SQL, который формирует промежуточные итоги для каждого указанного элемента и общий итог.

Пусть необходимо получить сумму расхода на оплату труда по отделам и по годам, и сначала запрос с группировкой без использования оператора ROLLUP (рисунок 2).

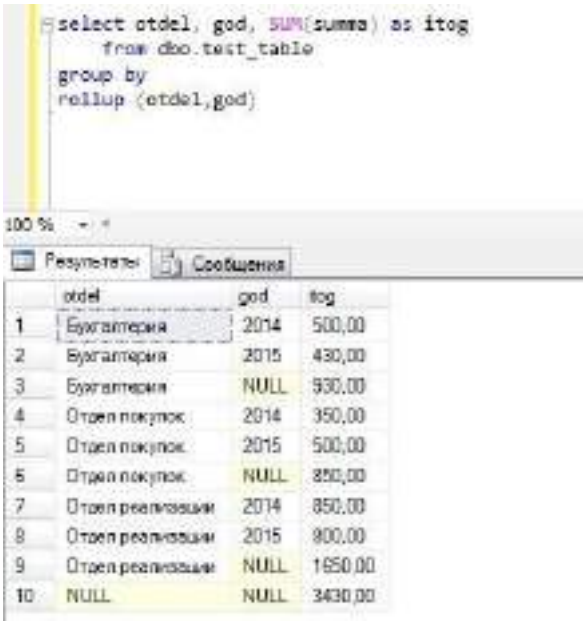


	otdel	god	itog
1	Бухгалтерия	2014	500.00
2	Бухгалтерия	2015	430.00
3	Отдел покупок	2014	350.00
4	Отдел покупок	2015	500.00
5	Отдел реализации	2014	850.00
6	Отдел реализации	2015	900.00

Рисунок 2 – Запрос суммы расхода на оплату труда по отделам и по годам

Иногда необходима общая информация, например общий расход по каждому отделу. Для этих целей мы можем использовать оператор ROLLUP.

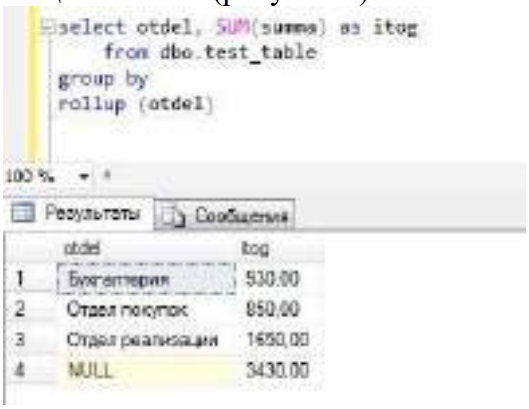
Строки со значением NULL и есть промежуточные итоги по отделам, а самая последняя строка общий итог (рисунок 3).



	otdel	god	itog
1	Бухгалтерия	2014	500,00
2	Бухгалтерия	2015	430,00
3	Бухгалтерия	NULL	930,00
4	Отдел покупок	2014	350,00
5	Отдел покупок	2015	500,00
6	Отдел покупок	NULL	850,00
7	Отдел реализации	2014	850,00
8	Отдел реализации	2015	800,00
9	Отдел реализации	NULL	1650,00
10	NULL	NULL	3430,00

Рисунок 3 – Запрос с группировкой по отделам и годам и выводом промежуточного и общего итогов по отделам

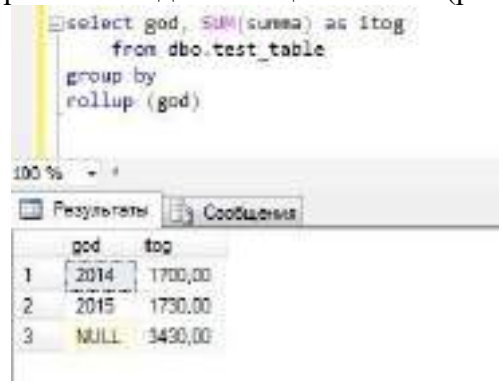
Можно также использовать rollup и с группировкой по одному полю. Например, группировка по отделам с общим итогом (рисунок 4).



	otdel	itog
1	Бухгалтерия	930,00
2	Отдел покупок	850,00
3	Отдел реализации	1650,00
4	NULL	3430,00

Рисунок 4 – Группировка по отделам с общим итогом

Группировка по годам с общим итогом (рисунок 5).



	god	itog
1	2014	1700,00
2	2015	1730,00
3	NULL	3430,00

Рисунок 5 – Группировка по годам с общим итогом

CUBE – оператор Transact-SQL, который формирует результаты для всех возможных перекрестных вычислений.

Создадим такой же SQL-запрос, как на рисунке 3, только вместо rollup укажем cube и посмотрим на полученный результат (рисунок 6).

```
select otdel, god, SUM(summa) as itog
    from dbo.test_table
group by
cube (otdel,god)
```

100 % - 4

Результаты | Сообщения

	otdel	god	itog
1	Бухгалтерия	2014	500.00
2	Отдел покупок	2014	350.00
3	Отдел реализации	2014	850.00
4	NULL	2014	1700.00
5	Бухгалтерия	2015	430.00
6	Отдел покупок	2015	500.00
7	Отдел реализации	2015	800.00
8	NULL	2015	1730.00
9	NULL	NULL	3430.00
10	Бухгалтерия	NULL	930.00
11	Отдел покупок	NULL	850.00
12	Отдел реализации	NULL	1650.00

Рисунок 6 – Запрос с группировкой по отделам и годам и выводом промежуточного и общего итогов по отделам и годам

В данном случае отличие от rollup заключается в том, что группировка и промежуточные итоги выполнены как для otdel так и для god.

GROUPING SETS – оператор Transact-SQL, который формирует результаты нескольких группировок в один набор данных, другими словами он эквивалентен конструкции UNION ALL к указанным группам.

На рисунке 7 приведен пример использования GROUPING SETS по отделам и годам.

```
select otdel, god, SUM(summa) as itog
    from dbo.test_table
group by
grouping sets (otdel,god)
```

100 % - 4

Результаты | Сообщения

	otdel	god	itog
1	NULL	2014	1700.00
2	NULL	2015	1730.00
3	Бухгалтерия	NULL	930.00
4	Отдел покупок	NULL	850.00
5	Отдел реализации	NULL	1650.00

)

Рисунок 7 – Запрос с GROUPING SETS, возвращающий итоги по отделам и годам

На рисунке 8 приведен тот же результат, но с использованием UNION ALL

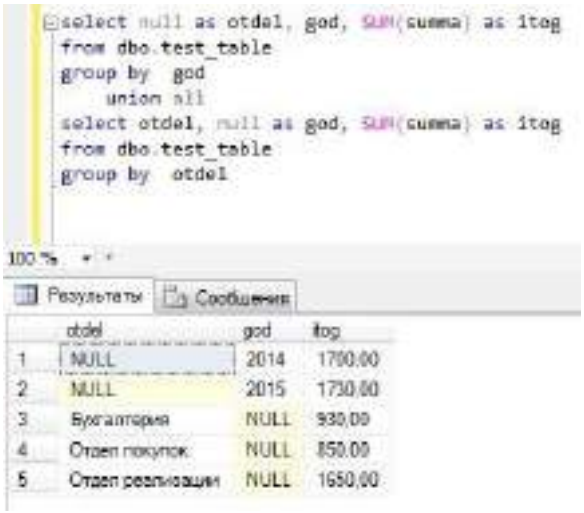


Рисунок 8 – Запрос с UNION ALL, возвращающий итоги по отделам и годам

GROUPING – функция Transact-SQL, которая возвращает истину, если указанное выражение является статистическим, и ложь, если выражение нестатистическое.

Данная функция создана, для того чтобы отличить статистические строки, которые добавил SQL сервер от строк, которые и есть сами данные, так как когда используется много группировок запутаться в строках очень легко.

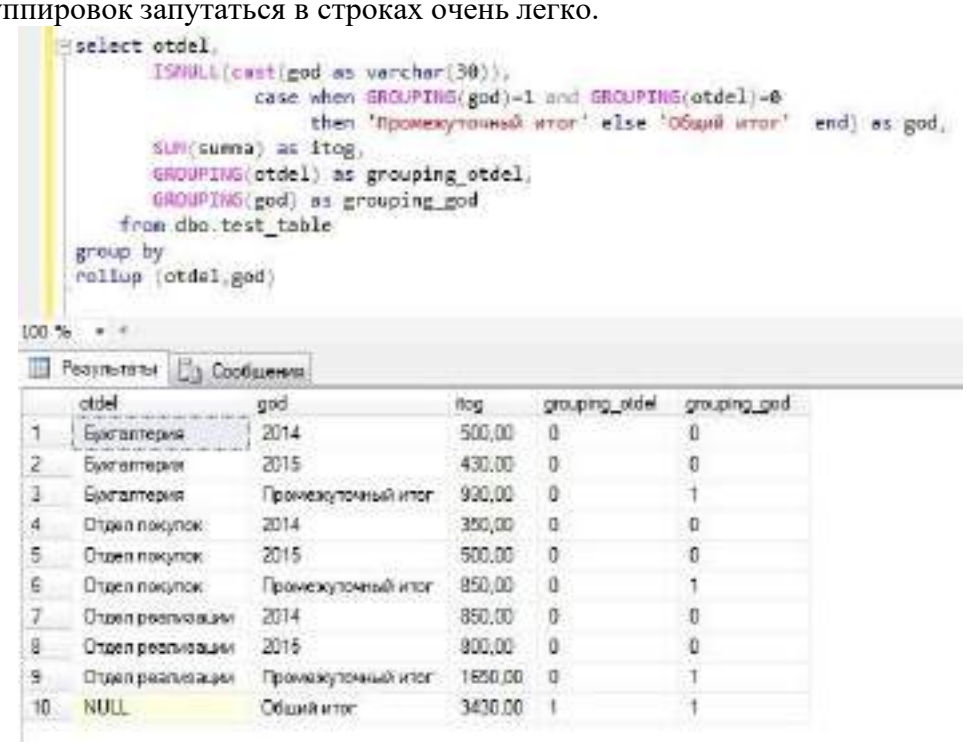


Рисунок 9 – Запрос с GROUPING, возвращающий итоги по отделам и годам и соответствующий признак

1.3 Задание на лабораторную работу.

1. Разработать набор аналитических запросов с использование ROLLUP, CUBE, GROUPING SETS и GROUPING.
2. Выполнить запросы и привести результаты их выполнения.

1.4 Содержание отчета. Отчет должен содержать следующие разделы:

1. Титульный лист;
2. Словесное описание запроса;
3. Тексты запросов на языке Transact-SQL;

3. Скриншоты, демонстрирующие выполнение запросов и возвращаемые наборы данных;

4. Краткие итоги выполнения работы и выводы.

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ ФИЛОСОФИЯ И МЕТОДОЛОГИЯ НАУЧНЫХ ИССЛЕДОВАНИЙ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

МАТЕРИАЛЫ К ЛЕКЦИЯМ

Тема 1. Предмет и основные концепции философии науки и техники.

Предметом курса «Философские проблемы науки и техники» является наука и техника как сферы бытия человека, как феномены общественной жизни и культуры в их динамике и взаимосвязи. В данном разделе будут рассмотрены некоторые проблемы философии науки. В наше время наука является объектом изучения многих дисциплин. Среди них - история науки, социология науки, наукометрия, науковедение, философия науки. У каждой из них имеется свой предмет изучения. Если социология науки, например, на исследовании взаимоотношения науки как социального института с социальной структурой общества, типологии поведения ученых в различных социальных системах, взаимодействии формальных и неформальных сообществ ученых, динамике их групповых взаимодействий, конкретных социокультурных условиях развития науки в разных типах общественного устройства, а наукометрия занимается применением методов математической статистики к анализу потоков научных публикаций, ссылочного аппарата, финансовых затрат, то *философия науки* изучает общие закономерности и тенденции научного позанания как особой деятельности по производству научныз знаний, взятых в их историческом развитии и рассматриваемых в исторически изменяющемся социокультурном контексте**.**

Философские проблемы возникают тогда, когда анализ проблем функционирования и развития науки доводится до мировоззренческого и методологического уровней: соотношения человека и мира, субъекта и объекта познания, природы науки и научной истины, оснований науки и ее ценностных ориентиров и т д.

Наука это один из древнейших, важнейших и сложнейших компонентов человеческой культуры. *Во-первых,* наука - это многообразный мир

человеческих знаний, который позволяет человеку преобразовывать природу и приспосабливать ее для удовлетворения своих все возрастающих материальных и духовных потребностей. *Во-вторых*, это - сложная система исследовательской деятельности, направленная на производство новых знаний. *В-третьих*, это и социальный институт, организующий усилия сотен тысяч ученых-исследователей, отдающих свои знания, опыт, творческую энергию постижению законов природы, общества и самого человека.

Наука теснейшим образом связана с материальным производством, с практикой преобразования природы, социальных отношений. Вся современная цивилизация, подавляющая часть материальной культуры общества создана на базе науки, и прежде всего достижений естествознания. Научная картина мира всегда была и важнейшей составной частью мировоззрения человека. Научное понимание природы и общества - особенно в настоящую эпоху - существенно определяет и содержание внутреннего духовного мира человека, сферу его представлений, ощущений, переживаний, динамику его потребностей и интересов.

Наука занимает особое и важнейшее место в духовной культуре, которое определяется значением познания в способе бытия человека в мире, в практике, т.е. в материально-предметном преобразовании мира.

Часто понятия наука и познание отождествляются. Но такое отождествление является упрощением. Ведь познание может быть *донаучным, вненаучным* и *научным*. Наука представляет собой лишь одну из исторических форм познания мира. Долгое время познание развивалось в донаучных формах (мифология, религия и др.). Вместе с тем, некоторый познавательный момент несомненно (был всегда и сейчас) свойственен и ненаучным формам духовной культуры - искусству, политическому сознанию, правосознанию, морали и даже религии.

Наука - это специфическая деятельность людей, главной целью которой является получение знаний о реальности. Знание - главный продукт научной деятельности, но не единственный. К продуктам науки можно отнести и научный стиль рациональности, который распространяется во все сферы деятельности людей; и различные приборы, установки, методики, применяемые за пределами науки, прежде всего в производстве. Научная деятельность является и источником нравственных ценностей. Хотя наука ориентирована на получение истинных знаний о реальности, *наука и истина не тождественны*. Истинное знание может быть и ненаучным. Оно может быть получено в самых разных сферах деятельности людей: в обыденной жизни, экономике, политике, искусстве, в инженерном деле. В отличие от науки, получение знания о реальности не является главной, определяющей целью этих сфер деятельности (в искусстве, например, такой главной целью являются новые художественные ценности, в инженерном деле - технологии, изобретения, в экономике - эффективность и т.д.). Важно подчеркнуть, что определение «ненаучный» не предполагает негативную оценку.

Научная деятельность специфична. Другие сферы деятельности человека - обыденная жизнь, искусство, экономика, политика и др. - имеют

каждая свое предназначение, свои цели. Роль науки в жизни общества растет, но научное обоснование не всегда и не везде возможно и уместно. История науки показывает, что научное знание не всегда является истинным. Понятие «научный» часто применяется в ситуациях, которые не гарантируют получение истинных знаний, особенно когда речь идет о теориях. Многие научные теории были опровергнуты. Иногда утверждают (например, Карл Поппер), что любое теоретическое высказывание всегда имеет шанс быть опровергнутым в будущем. Наука не признает паранаучные концепции - астрологию, парапсихологию, уфологию и т.п. Она не признает эти концепции не потому, что не хочет, а потому, что не может, поскольку, по выражению Т. Гексли, «принимая что-нибудь на веру, наука совершает самоубийство». А никаких достоверных, точно установленных фактов в таких концепциях нет. Возможны случайные совпадения. По поводу такого рода проблем Ф. Бэкон писал так: «И потому правильно ответил тот, который, когда ему показали выставленное в храме изображение спасшихся от кораблекрушения принесением обета и при этом добивались ответа, признает ли теперь он могущество Богов, спросил в свою очередь: «А где изображение тех, кто погиб после того, как принес обет?» Таково основание почти всех суеверий - в астрологии, в повериях, в предсказаниях и тому подобном. Люди, услаждающие себя подобного рода суетой, отмечают то событие, которое исполнилось, и без внимания проходят мимо того, которое обмануло, хотя последнее бывает гораздо чаще». Важные черты облика современной науки связаны с тем, что сегодня она является профессией. До недавнего времени наука была свободной деятельностью отдельных ученых. Она не была профессией и никак специально не финансировалась. Как правило, ученые обеспечивали свою жизнь за счет оплаты их преподавательской работы в университетах. Однако сегодня ученый - это особая профессия. В XX веке появилось понятие «научный работник». Сейчас в мире около 5 млн. людей профессионально занимаются наукой. Для развития науки характерны противостояние различных направлений. Новые идеи и теории утверждаются в напряженной борьбе. М. Планк сказал по этому поводу: «Обычно новые научные истины побеждают не так, что их противников убеждают, и они признают свою неправоту, а большей частью так, что противники эти постепенно вымирают, а подрастающее поколение усваивают истину сразу». Жизнь в науке - это постоянная борьба различных мнений, направлений, борьба за признание идей.

Каковы же *критерии* научного знания, его характерные признаки? Важнейшим критерием научности является стремление к максимальной *точности, объективности*. Результаты научного познания (теории, понятия и др.) организованы таким образом, чтобы исключить все личностное, привнесенное исследователем от себя. Одна из главных особенностей науки состоит в том, что она нацелена на *отражение объективных сторон мира*, т.е. на получение таких знаний, содержание которых не зависит ни от человека, ни от человечества. Наука стремится, прежде всего, построить объективную картину мира, т.е. отразить его так, как он существует как бы

«сам по себе», независимо от человека. Никакой другой компонент духовной культуры (ни искусство, ни идеология, ни религия и т.д.) такой цели перед собой не ставит.

Одним из важных отличительных качеств научного знания является его систематизированность. Это один из критериев научности. Но знание может быть систематизированным не только в науке. Кулинарная книга, телефонный справочник, дорожный атлас и т.д. и т.п. - везде знание классифицируется и систематизируется. Научная же систематизация специфична. Для нее свойственно стремление к полноте, непротиворечивости, четким основаниям систематизации. Наука представляет собой исторически сложившуюся систему познания объективных законов мира. Результатом научной деятельности выступает система развивающегося доказательного и обоснованного знания. *Обоснованность*, *доказательность* и *проверяемость* также являются критериями научности.

Научное знание как система имеет определенную структуру, элементами которой являются *факты*, *законы*, *теории*, *картины мира*. Отдельные научные дисциплины взаимосвязаны и взаимозависимы. Обоснование знания, приведение его в единую систему всегда было характерным для науки. Со стремлением к доказательности знания иногда связывают само возникновение науки. Применяются разные способы обоснования научного знания. Для обоснования эмпирического знания применяются многократные проверки, обращение к статистическим данным и т.п. При обосновании теоретических концепций проверяется их непротиворечивость, соответствие эмпирическим данным, возможность описывать и предсказывать явления. В науке ценятся оригинальные, «сумасшедшие» идеи. Но ориентация на новации сочетается в ней со стремлением элиминировать из результатов научной деятельности все субъективное, связанное со спецификой самого ученого. В этом - одно из отличий науки от искусства. Если бы художник не создал своего творения, то его бы просто не было. Но если бы ученый, пусть даже великий, не создал теорию, то она все равно была бы создана, потому что представляет собой необходимый этап развития науки, является *интерсубъективной*.

Научные знания достигаются посредством проверенных практикой методов познания. *Научное знание дает возможность предвидения и преобразования действительности в интересах общества и человека*. Современная наука это сложная и многообразная система отдельных научных дисциплин.

Для понимания предмета «философия науки» как учебной дисциплины и раздела философского знания необходимо дать ответ на вопрос, *что есть наука.* Классическое определение науки генетически связано с возникновением философского знания. Наука — это особый вид теоретического знания, точнее, совокупность теоретических знаний специального характера, т.е. направленного на постижение законов бытия

(общества, природы, мышления людей). Эти законы, необходимые, универсальные, постигаются разумом человека и носят объективный (т. е. независимый от человека или Бога) характер. Такой ход мысли позволяет сразу отграничить теоретическое научное знание от теоретического, но ненаучного знания (богословского, например), от знания опытного, обыденного, от образно-художественного знания. Начиная с XVI, а особенно с XVII века, наука приобрела новые сущностные, принципиально важные черты. Она отныне не только особый вид, тип знания, но и социально-практическая сила общества, т.е. сила, способная изменить общество в «лучшую» сторону, позволяющая человеку господствовать над силами природы и стихийными процессами в обществе. Наука определяет технические достижения в обществе и влияет на производство в целом. Зреет мысль, сформулированная позже американским философом, систематизатором прагматизма Дж. Дьюи о том, что наука не только теория, она — форма практики людей, определяющая производство сила.

Мыслители XVII—XVIII веков (Ф. Бэкон, Р. Декарт, Т. Гоббс, Ф. Вольтер, Д. Дидро и др.) сформулировали идею о науке как ценности общества — высшем проявлении разума человечества, определяющем, в конечном счете, и мораль, и политику, и историю, и религию. Возникают словосочетания «научная история», «научная политика» и даже «научное питание», «научный подход к образованию», «научная экономика» и т.д. Наука как «венец развития» разума смягчает нравы общества, устраняет жестокость, фанатизм, несправедливость — считали французские просветители XVIII века. В конечном счете, все общество может быть построено исключительно на принципах разума и науки.В XIX веке французским философом и социологом, основателем позитивизма О. Контом (1792—1857), немецкими философами, основателями идеологии «научного коммунизма» К. Марксом (1818-1883), и Ф. Энгельсом (1820-1895) были сформулированы новые подходы к определению науки. Она стала пониматься как неотъемлемая и существенная часть производства: наука не существует вне производства, а производство — вне науки. Маркс формулирует идею о науке как «всеобщей общественной производительной силе». И, наконец, наука выступает в виде социального института. Созданные специальные организации ученых (академии, научные институты) играют огромную роль в жизни общества, определяя техническую, экономическую, политическую, социальную, военную деятельность государств. С середины—конца XIX века наука понимается как важнейшая часть социальной структуры общества.

Предмет философии науки связан с вышеизложенным пониманием сущности и особенностей науки. Впервые термин «философия науки» ввел Уильям Юэлл в 1840 г. (Англия). Что такое *философия науки*? У. Селларс (сотрудник католического Нотрдамского университета, США) считал, что «это — философия, которая берет науку серьезно», так как наука неотъемлемая часть человеческого бытия. По мнению В.И. Вернадского (1863-1945), философия науки представляет собой «тесную связь философии

и науки в обсуждении общих вопросов естествознания» в силу того, что «вопросы философские и научные слились, как это было в эпоху эллинской науки». Если наука требует себе признания, то она, по словам Г. Гегеля, «обязательно должна оправдать себя перед разумением и мыслью».Философия науки – это то, что философы думают о науке, причём эти мнения диаметрально противоположны: от признания ценности науки до утверждения о губительном влиянии науки на европейскую цивилизацию.

Современный отечественный философ науки В. С. Степин *считает, что* предметом философии науки являются общие закономерности и тенденции научного познания как особой деятельности по производству научных знаний, взятых в их историческом развитии и рассмотренных в исторически изменяющемся социально-культурном контексте. Но это понимание предмета философии науки следует расширить: в него входят не только проблемы научного познания, но и место науки в нашей цивилизации, ее отношение к этике, политике, религии и т.д. Так, к области философии науки следует отнести и актуальную для современной науки проблему «возникновения слоя плебеев от науки», которые, как писал *К. Я*слерс, «создают в своих работах пустые аналогии, выдавая себя за исследователей, приводят любые установления, подсчеты, описания и объявляют их эмпирической наукой». Каждый обладающий «рассудком и прилежанием» «считает себя способным» к науке, «каждый безответственно смеет высказывать свое мнение, которое он вымучил», а в результате рождается бесконечное множество точек зрения, затрудняющих понимание сути дела.

Философия науки включает следующие основные концептуальные составляющие: естественнонаучные теории; историко-философское знание; логические, методологические и лингвистические концепции; историко-научные исследования.

Обосновывая необходимость философского осмысления науки, в частности математики и физики, Гегель неоднократно писал, что математические определения (бесконечность, бесконечно-малое, множители, степени и т.д.) находят свое истинное понятие только в философии. Установить смысл понятий, с которыми работают математики, может только философия. Философы переводят и «материал физики» «на язык понятия как нечто в самом себе необходимое целое», т.е. «материал, изготовленный физикой на основании опыта», философия «преобразовывает дальше». Философия «делает дальнейший шаг потому, что способ действия с понятием, употребляемый в физике, неудовлетворителен».

Идею Гегеля о том, что только философия может уяснить смысл употребляемых наукой понятий, «увидеть» в исторической смене понятийно-категориального аппарата движение научной мысли, поддерживал В.И. Вернадский. Он считал, что в истории научной мысли никто не учитывает значение понятийно-категориального аппарата науки «и истории его создания нет».

Необходимость философии науки признавал и один из крупнейших мыслителей современности немецкий философ М. Хайдеггер (1889—1976).

Эту необходимость он обосновывал тем, что наука ничего не может сказать сама о себе своими научными средствами, не может, например, обоснованно ответить на главный вопрос: «что значит знать?». Например, вопрос «как возможна наука?» не может быть решен в рамках самой науки и ее средствами познания.

Фундаментальное различие между вопросами, которые изучают философы и ученые, немецко-американский философ и логик Р. Карнап (1891—1970) иллюстрирует так: вопросы типа «как образовались лунные кратеры?», «существуют ли галактики, по-строенные из антиматерии?» решают астрономы и физики; вопросы же о том, как ученые строят понятия, каковы логические и гносеологические свойства этого построения, решают философы науки. Другими словами, если вопрос «касается не природы мира, а анализа фундаментальных понятий науки», то это вопрос для философии науки.

Интерес к философским аспектам науки проявили все выдающиеся физики XX века: Н. Бор, М. Борн, В. Гейзенберг, М. Планк, А. Эйнштейн и др. Так, Эйнштейн признавал, что «в наше время физики вынуждены заниматься философскими проблемами в гораздо большей степени, чем это приходилось делать физикам предыдущих поколений. К этому их вынуждают трудности их собственной науки; ученый должен стараться полностью понять, до какого предела используемые им понятия обоснованны и необходимы». Австрийский физик и философ, представитель неопозитивизма Ф. Франк (1884— 1966) писал, что «всякий, кто хочет добиться удовлетворительного понимания науки XX века, должен хорошо освоиться с философской мыслью».

Истоки философии науки можно искать в натурфилософии Р. Декарта, в работах Ф. Бэкона, в философии И. Канта), поставившего вопросы о том, как возможна наука, каковы границы науки, как связаны наука, основанная на разуме, и вера с ее внерациональными постулатами, является ли философия наукой и чем должна заниматься философия в науке?. Вне сферы науки находятся проблемы религии, существования Бога, жизни после смерти, религиозных догматов, происхождение и сущность души человека, свобода человека. Научное знание имеет пределы, за которыми открывается область веры. Видимая граница научного знания — положения, которые можно с одинаковой логической убедительностью доказать и опровергнуть средствами разума.

Со второй половины XIX века проблемы строения, оснований и функций научного знания становятся главными в философии. Появляется и утверждается понятие о специфической единице научного знания — научной теории. Одновременно из проблематики чувственного опыта, которую рассматривали практически все философы XVI—XVIII веков, выделяются специфическая для философии науки проблематика эмпирического знания и соответствующие понятия «наглядность», «аналогия». Формируются и другие специфические проблемы философии науки, такие как «научный закон», «математика» и т.д.

Дисциплинарно философия науки оформилась во второй половине XIXвека (Г. Гельмгольц, Э. Мах, Ч. Пирс и др.). Этому способствовал ряд обстоятельств: (1) наука к этому времени оформилась в важную и самостоятельную сферу общественной жизни, подкрепив свое значение развитием прикладных разработок и исследований; (2) математиками (французом О. Коши и чехом Б.Больцано) была поставлена проблема логического обоснования и изложения математического анализа. В том же направлении начал размышлять и немецкий философ, основатель феноменологии Э. Гуссерль (1859—1938); (3) кризис механистического мировоззрения потребовал переосмысления обоснования знания. Учёными были вынесены на обсуждение вопросы: что такое научная теория, какое место занимают в ней механические модели и математические уравнения, как соотносится она с экспериментом и т.д.; (4) стал осознаваться трагический для европейской цивилизации процесс расщепления культуры на научную и художественно-гуманитарную. Эта трагичность состояла в разведении знания и нравственности. Если античный философ Парменид был уверен, что знающий не может не быть по необходимости добрым и справедливым, так как Истина, Добро и Красота тождественны, то уже в философии Канта наука и мораль рассматривались в разных «Критиках»: разум отделился от морали, знание стало выше морали. Такая ситуация породила серьезные этические проблемы науки, особенно в конце XX века.

Институционализация и социализация философии науки как научной дисциплины начались в США, где еще до второй мировой войны начинает выходить журнал «Философия науки». В бывшем СССР сразу после войны в структуре Института философии Академии наук создается сектор философии естествознания (впоследствии философских вопросов естествознания). Содержание предмета философии науки (и понимание сущности науки) менялось исторически. Можно выделить следующие этапы развития философии науки, характеризующиеся спецификой изучаемых проблем.

1-й этап (вторая половина XIX века): исследуются психологические и индуктивно-логические процедуры эмпирического познания.

2-й этап (первые два десятилетия XX века): кризис классической физики вызвал потребность в переосмыслении самых фундаментальных понятий этой науки. Актуализировались проблемы связи физики с математикой.

3-й этап (20—40-е гг. XX века):в неопозитивизме создаются программы анализа языка науки, форм суждений и типов логики, которые применяются в ней.

4-й этап (40—50-е гг. XX века): в работах У. Куайна, К. Поппера, Т. Куна, М. Полани проводится тщательное изучение логики научного объяснения и научного исследования.

5-й этап (60—70-е гг. XX века): методологическая проблематика (вопросы строения, функционирования, обоснования знания) пополнилась вопросами методологии истории естествознания и истории математики. Обращение к истории науки, к анализу научных революций остро поставило вопрос о соотношении научного знания и реальности, т.е. вопрос о природе

истины.Началось изучение науки как социально-культурного феномена в рамках социологии знания и социологии науки. Философия науки оформляется как одна из наиболее сложных дисциплин в рамках профессиональной философии, опирающаяся на исследования в области теоретической философии, логики, социологии, культурологии, социальной психологии, истории науки и истории философии и др.

6-й этап (70—80-е гг. XX века): формируются два направления в развитии постпозитивизма: (а) анализ эпистемологических оснований выдвигаемых моделей наук (Т. Кун, И. Лакатос) и (б) критика философии науки, и даже отрицание философского и общекультурного ее значения (П. Фейерабенд и Р. Рорти).

Предметом философии науки можно считать принципы, структуру и методы научного познания, историю науки, социальные и культурологические аспекты научного знания. Центральной является проблема теории научного знания.

В 90-е гг. XX века интерес ученых к философии науки ослабевает, усиливается критика науки и научной рациональности, ставятся под сомнение претензии разума быть единственным средством постижения мира и т.д.Современную ситуацию в философии науки четко охарактеризовали отечественные философы науки И.Т. Касавин и В.Н.Порус: «Нужно в очередной раз дать ответ на сакраментальный вопрос — быть или не быть философии науки и, может быть, на еще более важный — если быть, то какова она должна быть, чтобы ею захотелось заниматься».

Основными концепциями философии науки выступают позитивизм, неопозитивизм, постпозитивизм. Основоположник позитивизма О. Конт выдвинул ряд интересных идей в области философии науки. Анализ роли современной мыслителю науки привел его к убеждению в том, что наука перестала быть делом одиночек или небольших групп исследователей. Роль науки изменилась, наука не просто сила, а сила производства, наука определяет производство и сама выступает как производящая сила и социальный институт. Научное знание опирается на опыт и эксперимент. Только опытное, экспериментальное знание способно дать нечто новое. Такое знание может быть названо положительным. Наука направлена только на реально существующие объекты действительности. Такие же теоретические и только теоретические «объекты», как, например, «первоначала бытия», изучает философия. Но никто и никогда в опыте с ними не встречался. Познать их средствами экспериментальной науки невозможно. Отсюда для Конта философия (как и религия, искусство, мифология) наукой не является. Критерии научности таковы: (а) наука познает факты, явления, процессы, которые «улавливаются» в опыте человечества; (б) наука связана с практикой, знание ценно только тогда, когда мы его можем использовать практически. Истина — это придаток практики, практика — критерий истины; (в) однако не любое знание, используемое в практике, автоматически становится научным. Например, обыденное знание людей, несомненно, носит практическую ценность и

характер, но не является научным. Научное знание содержит гипотезу, теорию, оно способно объяснять мир, а главное, предвидеть явления действительности. Научное знание имеет свой собственный понятийный аппарат, свои методы познания; (г) настоящая наука не может ничего сказать о причинах существования космоса или человека. Она может говорить «как» устроен космос, «как» устроен человек и прочее, но не «почему» существуют космос и человек. Построенная им новая классификация наук в труде «Общий обзор позитивизма» в число наук включает математику, астрономию, физику, химию, физиологию и призванные заменить философию — социальную физику (позже названную социологией) и мораль. Кстати, возможно и существование новой «позитивной философии», которая бы создала систему всех наук и знаний человечества, т.е. выступила бы в роли «систематизатора» всех наук. Сама же новая «позитивная философия» никаких законов и явлений действительности не открывает, а только использует данные позитивных наук.

К. Маркс и Ф. Энгельс обосновали идею о том, что при капитализме наука не только применена к процессу производства, не только стала производительной силой труда. Наука стала условием для развития всех других производительных сил общества, она стала силой капитала, она обогащает капиталиста, который смог поставить ее себе на службу, и укрепляет могущество буржуазии. Маркс и Энгельс показали, что наука — плод усилий всего человечества, выступает как сумма человеческого знания, ее продукты принадлежат всему человечеству. В терминах марксизма наука — «всеобщий духовный продукт общественного развития». Она выступает как особый вид общественной деятельности, как форма общественного сознания. Наука — знание объективное, логически систематизированное, выступающее как система знаний о законах общества природы и мышления; она опирается на общественную практику, и практика выступает как критерий истины научного знания; наука сложилась исторически в обществе, она есть часть общества и вне общественного развития его этапов, которые специфичны, не понятна; материальное производство ставит науке те или иные цели, определяет главную проблематику науки. Однако наука не только определяется производством, она способна идти впереди запросов производства и в значительной степени определяет производство. Наука определяется культурой общества, степенью развития экономики, техники, морали; но она же (наука) способна изменить культуру общества, его технику, экономику, мораль и пр. Неопозитивистские концепции философии науки.

Согласно неопозитивисту Л. Витгенштейну, наука нуждается в очищении своего языка. В современной науке очень много проблем, связанных с употреблением обыденного языка. Обыденный язык людей слишком многозначен, а потому его проникновение в язык науки губительно для последнего. Научное знание должно быть общезначимо, что достижимо при помощи особого языка - языка науки. Язык выражает мысль. Витгенштейн полагал, что «логика языка» определяет границу выражения мысли. То, что

не выразимо в слове, не существует в мире. Таким образом, для мыслителя границы языка совпадают с границами бытия. Язык науки тесно связан с опытом. Знания в науке тоже могут быть только из опыта. Только те высказывания могут иметь ценность для науки, которые связаны с опытом (или настоящим, или прошедшим, или непосредственным, или опосредованным). Тогда истины науки — это совпадение высказываний с опытом человека или человечества. Л.Витгенштейн выдвинул принцип «верификации», согласно которому любое высказывание в науке верифицируемо, т.е. подлежит опытной проверке на истинность. Однако сам мыслитель понимал, что некоторые высказывания не могут быть верифицируемы. Например, обобщающие высказывания типа «все люди смертны», некоторые исторические сведения и пр. Кроме того, принцип верификации затруднял выдвижение гипотез, которые, как правило, опытом не подтверждены. Другой неопозитивист, К. Поппер, в ходе исследования сущности науки, ее законов и методов пришел к идеям, несовместимым с принципом верификации. Он выдвинул идею о невозможности сведения содержания науки, ее законов только к утверждениям, основанным на опыте, т.е. к наблюдению, эксперименту и т.д. Наука не может сводиться к верифицируемым высказываниям, как это считал Л. Витгенштейн. Принцип верификации «асимметричен»: доказательства ряда положений добываются огромным трудом, а опровергаются одним фактом. Логическая непротиворечивость также не всегда доказывает истинность знания. Большинство богословских схоластических трактатов выполнены с точки зрения логики непротиворечиво, но это знаниененаучное. Опыт и факты тоже не всегда в науке ведут к истине. Например, в социальном знании опыт и факты имеют относительную ценность. Факты, например, рассматриваются в контексте статистики и пр. В науке, считает К. Поппер, легко и просто прийти ко лжи, но гораздо труднее и тяжелее — к истине.

Первый признак ложности теории как раз в том и состоит, что все факты объясняются в рамках одной теории. Те факты, которые необъяснимы в теории, либо замалчиваются, либо «подгоняются», искажаются. Вот это и есть ненастоящая наука. Настоящая наука какие-то факты и явления объясняет, а какие-то нет. Например, теория Эйнштейна. Наука тогда настоящая наука, когда ее принципы могут быть опровергаемы некоторыми фактами. В этом и состоит смысл нового принципа фальсификации (искажения), который выдвинул К. Поппер. Знание научно, когда оно фальсифицируемо.Методы науки — всегда методы проб и ошибок. Научные теории первоначально обычно выступают в виде необоснованных догадок, которые проверяются людьми и через некоторое время сменяются на другие догадки, не похожие на предыдущие. Мы всегда недовольны теориями, потому что они не способны все и вся объяснить, но это — ситуация естественная для существования науки. Научное знание должно стремиться к элиминации субъекта из процесса познания, т.е. к устранению всего, что относится к эмоционально-психологической стороне процесса научного знания.

Т. Кун особое внимание уделяет истории науки, полагая, что вне своей истории наука не понятна. Есть правила общения ученых между собой, а также неписаные правила отношений ученых и общества. Например, кодексы научной честности ученых, невозможность привлекать к решению научных споров власть, и т. д. Научная, академическая жизнь имеет свои ценности, отличные от ценностей массовых. Т. Кун выдвигает концепцию «парадигм», признаваемых большинством ученых совокупностей теорий, научных достижений, которые служат для сообщества ученых моделью постановки научных проблем и их решения. Например, таковой является парадигма классической науки. «Нормальная наука» — это эволюционная фаза в развитии науки, характеризующаяся наличием совокупности теорий, подтвержденных фактами. Когда появляются факты, необъяснимые в рамках «нормальной науки», и этих фактов накапливается достаточное количество, начинается революционная фаза в развитии науки. Появляется множество теорий, дающих различную интерпретацию научным аномалиям. Впоследствии научное сообщество выбирает наиболее приемлемый вариант теории, объясняющей область действительности или всю действительность. Эта теория и составляет теоретическое и методологическое основание новой «парадигмы». Начинается развитие следующего этапа «нормальной науки». Новая парадигма может включать в себя старую как фрагмент, частный случай новой парадигмы. Т. Куну удалось преодолеть некоторые недостатки, присущие позитивистским взглядам на науку. В науке нет непрерывного прогресса и кумуляции знания. Каждая парадигма формирует уникальное понимание мира и не имеет особых преимуществ перед другой парадигмой. Научный прогресс лучше понимать как научную эволюцию — прирост знания внутри парадигмы. Наука всегда социокультурно обусловлена.

Согласно И. Лакатосу, наука выглядит как поле борьбы, соперничества идей, что ведет к смене «научно-исследовательских программ» и «кодексов научной честности». Научно-исследовательская программа - это совокупность теорий, связанных генетически и методологически. Существуют принципы и правила, которые запрещают те или иные проблемы, методы, пути исследования. Лакатос называет их «отрицательная эвристика. Есть правила, определяющие выбор проблем, методов и путей их решения и применения. Это «позитивная эвристика». В научно-исследовательских программах мы можем различать «твердое» или «жесткое» ядро — основную теорию, главную общепринятую концепцию и «защитный пояс» — «вспомогательные теории», которые вытекают из главной теории, базируются на ее основе. Научно-исследовательская программа тогда прогрессивна, когда ее «защитный пояс» растет, когда предсказываются новые явления, объясняются не только известные, но и новые факты. Такая научно-исследовательская программа называется прогрессивной. Главное здесь — способность предвосхищения новых фактов. Если необъяснимых фактов больше, чем объяснимых, если теория не способна более предвидеть новые явления, то она регрессивна. В этом случае старую программу заменяет новая научно-исследовательская программа,

которая больше явлений объясняет и больше явлений предсказывает. Смена старой научно-исследовательской программы на новую происходит не вследствие того, что старая программа чего-либо не объясняет, а вследствие появления новой, более сильной, лучше объясняющей факты теории. В целом, наука — это бесконечное противоречие между старыми теориями и вновь возникшими фактами.

П.Фейерабенд выдвинул идею «пролиферации» (размножения) теорий и идею эпистемологического анархизма. Настоящий ученый всегда стремится создать свою научную теорию. Отсюда постоянное и хаотичное увеличение числа научных теорий, конкурирующих друг с другом. Несмотря на взаимную критику, борьба теорий полезна для науки, способствует ее развитию, обогащает научный инструментарий и т.д. Но тогда не может быть единого «языка науки», единой общепризнанной теории. Преемственность научного знания и способность кумуляции научных достижений также играют слабую роль в развитии науки. В теории познания господствует эпистемологический анархизм, который отрицает любые догмы в науке, признает ценность любой научной теории, любого категориального аппарата, отрицает возможность объективной истины, единые методологические стандарты и т.д. Имеются и иррациональные моменты в деятельности ученых. Новая теория сменяет старую не потому, что она более истинна, а потому, что она лучше пропагандируется, лучше организована автором, поддерживается властью, или капиталом, или средствами информации (чаще всего — всеми сразу). Наука гораздо ближе к мифу, чем это обычно считают. Наука выступает как одна из форм идеологии в обществе. М. Полани выдвигает концепцию «личностного знания в науке, доказывая, «что в каждом акте познания присутствует страстный вклад познающей личности и что это не добавка, не свидетельство несовершенства, но насущно необходимый элемент знания». Он делил знание на «явное», «центральное» и «неявное», «периферическое», «скрытое», «имплицитное». Если до Полани господствовала идея о том, что ученый полностью осознает свои ощущения, восприятия в процессе научного творчества, то Полани пишет, что человек всегда знает больше, чем он может сказать. Информация, которая идет через органы чувств, шире, чем то, что проходит через сознание. Мыслитель утверждает, что личность ученого играет огромную роль в процессе познания и то, что время одиночек в науке прошло, вовсе не снимает факта личного проникновения ученого в исследовательскую задачу. Наука не продвигается вперед одними методами, без личности исследователя, без общения творческих личностей, непосредственного обучения учителем ученика.

Теория тогда принимается ученым, когда он в нее верит. А уж потом начинается критическое осмысление теории, сопоставление ее с другими теориями. В теорию надо «вживаться». Здесь очень нужна «научная вера». Если исследователь разделяет научные убеждения, верит в «свою» теорию, он ее не променяет на другую, даже лучше обоснованную и

аргументированную. Вера, доверие есть условие научного познания; другое условие — необходимый уровень развития способностей личности.

Если философию науки рассматривать в связи не с общефилософскими направлениями, а с решением проблем научности, рациональности, истинности, то можно дать иную ее классификацию: релятивизм, фаллибилизм, конструктивный эмпиризм, эволюционная эпистемология и т.д. Релятивизм (от лат. relativus — относительный) в философии науки утверждает условность, ситуативность научного знания, отказывается от теоретических его дефиниций. Например, геометрии Лобачевского, Римана и Евклида имеют одинаковые права на описание реальности. «Никакая геометрия, — писал А. Пуанкаре,— не является более истинной, чем другая, та или иная геометрия может быть только более удобной». К философским истокам релятивизма в философии науки относятся философия прагматизма (Ч. Пирс, Дж. Дьюи, В. Джемс) и аналитическая философия,' сочетающая эпистемологию с психологией и лингвистикой. К релятивизму можно отнести философию науки Т. Куна.

Фаллибилизм (от англ. погрешимый), или «критический рационализм». Основателем фаллибилизма признан представитель философии прагматизма американец Ч. Пирс, который считал, что знание всегда гипотетично, вероятностно. К. Поппер сделал фаллибилизм артикулированной философской доктриной, радикальная позиция которой такова: «нельзя ошибиться только в том, что все теории ошибочны». Философия науки – это поиск знания, обеспечиваемого индуктивными умозаключениями. Научные положения в принципе ошибочны, какие бы строгие проверки они ни проходили. С фаллибилизмом концепции Поппера связано и введенное им понятие «фальсификация». Фаллибилизм допускает существование под покровом эмпирических данных реальности, по отношению к которой строятся теории с надеждой на истинность, т.е. соответствие реальности. Но реальность ускользает от познания, а потому все теории оказываются ложными. К фаллибилизму можно отнести и концепцию науки И. Лакатоса.

Такое направление, как эволюционная эпистемология, сформировалось в 40-е гг. XX века и его основоположники — австрийский биолог и философ К. Лоренц и немецкий естествоиспытатель и философ Г. Фоллмер. Эволюционная эпистемология ставит задачу объяснить биологические предпосылки человеческого познания, его сущность на основе современной синтетической теории эволюции. Люди, как все живые существа, являются продуктами естественных эволюционных изменений, и в силу этого формирование их когнитивных и ментальных способностей направляется механизмами органической эволюции, понимаемой в дарвиновском смысле. В центре «эволюционной теории познания» не просто эволюция человеческого познания, а эволюция органов познания и познавательных способностей. Классические гносеологические вопросы - почему мы познаем именно так и именно это; насколько надежно наше познание; на чем основана надежность познания и др., решаются на основе естественных наук. Существует две исследовательские программы: одна изучает, как в ходе

биологической эволюции формировались носители когнитивных процессов (нервная система, органы чувств) (К. Лоренц, Г. Фоллмер); другая пытается объяснить возникновение научных идей, гипотез и теорий в контексте биологической эволюции (К. Поппер, Т. Кун и др.)

Вопросы для самоконтроля

1. Какие определения науки вам известны?
2. Каковы критерии научности?
3. Каковы основные этические принципы науки и инженерного отношения к миру?
4. Каковы идеалы науки?
5. Каковы специфические особенности науки, отличающие её от обыденного познания, искусства, философии, религии?
6. Каковы основные черты современной науки?
7. Каковы критерии научности?
8. Каковы характерные черты лженауки, паранауки, квазинауки, псевдонауки?
9. В чем проявляется нерациональное в научном знании?
10. Какие существуют подходы к периодизации науки?
11. Что такое «научная революция»?
12. Какие бывают виды научных революций? Почему они возникают?
13. Какие глобальные научные революции вы знаете?
14. Что называется научной рациональностью?
15. Какие типы научной рациональности вам известны?
16. Каковы причины смены типов научной рациональности?
17. В чем отличие неклассической науки от классической?
18. В чем специфика постнеклассической науки ?
19. В чем проявляется сближение идеалов естественнонаучного и социально-гуманитарного познания в XX веке.

Список рекомендуемой литературы

Основная

1. История и философия науки: учебник /под ред. А.С. Мамзина, Е.Ю. Сиверцева. -2-е изд. перераб. и доп. М.: Юрайт, 2016.-360 с.
2. Канке, В. А. Философские проблемы науки и техники: учебник и практикум для магистратуры / В. А. Канке. — М. : Издательство Юрайт, 2016. — 288 с.

3.Философия науки: Учебное пособие.-2-е изд.-М.;Изд.-ство Юрайт.-2015, 296 с.

3. Мокий, М. С. Методология научных исследований : учебник для магистров/ М. С. Мокий, А. Л. Никифоров, В. С. Мокий ; под ред. М. С. Мокия. — М.: Изд.-тво Юрайт, 2015. — 255 с.

8. Никифоров А. Л.Философия и история науки: Учебное пособие / А.Л. Никифоров. - М.: НИЦ ИНФРА-М, 2014. - 176 с.

9. Прытков, В. П. Философские проблемы науки и техники : учебное пособие / В. П. Прытков. - Екатеринбург : Изд-во Урал, ун-та, 2013. - 63, с.

Дополнительная

10. Заблуждающийся разум? Многообразие вненаучного знания/Отв. Ред. И.Т. Касавин – М.: Политиздат, 1990.-404 с.

11. Кошарный В.П. Философия науки и техники: учебное пособие. Пенза, 2012.- 294 с.

12.Современная философия науки: Хрестоматия/Сост. А.А. Печенкин. – М., 1994.- 252 с.

13. Степин В.С. История и философия науки: учебник. М.: Академический проект. 2011.-423 с.

14.Философия науки: общие проблемы познания. Методология естественных и гуманитарных наук : хрестоматия / отв. ред.-сост. Л.А Микешина. — М. : Прогресс-Традиция : МПСИ : Флинта, 2005. - 992 с

15.Философия и методология науки /Под ред. В. И. Купцова и С. В. Девятовой. – М., 1996.- 551 с.

16. Шаповалов В.Ф. Философия науки и техники: Учебное пособие. – М.: ФАИР-ПРЕСС, 2004. – 320 с.

Тема 2. Возникновение науки и основные стадии ее исторической эволюции

Важную роль в становлении науки сыграла преднаука. Её способы познавательной деятельности, несмотря на принципиальное отличие от науки, имели общую с ней цель — познать вещи, их свойства и отношения. На этапе преднауки изучались те предметы и вещи, те их свойства и отношения, которые были задействованы в реальной практической деятельности людей. Работа с чертежом на бумаге, замещающим реальное расположение, формы и размеры земельных участков, есть идеальный план работы мышления, идеализированная схема практических преобразований материальных предметов. Из повседневной практики выводились и первичные идеальные объекты, т.е. схемы, чертежи, геометрические фигуры, которые представляли и замещали реальные объекты. На этапе преднауки мышление обрело идеальную форму своей деятельности, т.е. такую форму, когда все операции счета, измерения и т.д. проводились не с реальными вещами и явлениями, а с их «заместителями», в качестве которых выступали таблицы, чертежи, календари и т.д. Формирование идеальных моделей и схем всегда было вплетено в ткань непосредственных жизненно значимых практических действий, а потому характерной особенностью преднауки являлся рецептурный характер полученного знания. Преднаука существовала и в полной мере обслуживала потребности общественно-производственной деятельности Древнего Египта, Месопотамии, Индии и др. на протяжении

десятков веков, и потому нельзя оценивать ее негативно, как низшую ступень знания, «не дозревшую» до науки.

Наука не могла возникнуть как результат линейного процесса углубления, обобщения рецептурно-технологического характера преднаучного знания, так как наука принципиально отличается от преднауки тем, что является формой теоретического мышления и сознания. Теоретическое мышление работает с идеальными объектами, которые, в отличие от преднаучных идеальных планов ре- цептурно-технологического характера, превращаются в независимый от сиюминутных нужд практики предмет исследования. Идеальные объекты в науке, в отличие от преднауки, во-первых, не выводятся непосредственно из обыденной практики, не являются замещением реальных предметов этой практики. Они или заимствуются из ранее сложившихся систем знания, либо конструируются учеными заново путем «запределивания» реальных характеристик явлений и предметов, не связанных с обыденными повседневными практическими интересами и потребностями; во-вторых, выступают в качестве фундамента системы знания, которая начинает строиться «сверху» по отношению к практике; в-третьих, созданные из идеальных объектов конструкции сопоставляются с практикой с помощью ряда опосредований: эксперимента, который не является формой обыденно-повседневной практики, путем сопоставления выводов, полученных при использовании какой-то идеальной схемы, с выводами, уже прошедшими проверку; в-четвертых, идеальные объекты и схемы применяются в науке в качестве строительного материала при формировании новых знаний.

Важно, что только европейская наука в качестве главной цели имеет познание истинной сущности предметов и явлений внешнего мира, изменение этого мира для удовлетворения материальных потребностей людей. Однако, интеллектуальная традиция, например, Китая базируется на мировоззренческой установке, согласно которой окружающий мир не надо изменять, так как он изначально гармоничен и совершенен. Человек и общество в своем познавательном рвении представляют угрозу этой гармонии, а потому надо искать не пути изменения мира, а практики самосовершенствования человека. Это одна из причин, по которой культуры традиционных обществ (Древний Китай, Индия, Древний Египет, Вавилон) не создали предпосылок возникновения науки европейского типа.

В науке, в отличие от преднауки, формируется особый вид знания — теория, которая позволяет получать эмпирические знания из теоретических постулатов. Теоретические знания выступают как знания об объектах реальности вне зависимости от их непосредственного использования в практике.

Духовной родиной Европы и европейской науки многие философы (Г. Гегель, Г. Г. Гадамер, Э. Гуссерль, М. Хайдеггер и др.) считали античную Грецию, где в VII—VI вв. до н.э. возникла философия.

Первое открытие, сыгравшее роль в становлении науки — обнаружение способности мышления к теоретизированию, которое произошло в

античности на рубеже VI— V вв. до н.э. Направляя свой интерес на поиск первых начал бытия как такового, занимаясь поисками истинного содержания добра как такового, красоты как таковой, первые философы создали мир абстрактных понятий, которые не могли использоваться древними греками в их повседневной практике. Греки, писал Гуссерль, «заняты теорией и только теорией, развивают только ее». Теоретическая деятельность первых философов изначально характеризовалась «чистотой», т.е. предполагала «вынесение за скобки» практически-житейских интересов, психологических, социальных и прочих связей и мотивов. Гуссерль писал в этой связи: «Теоретическая установка ... целиком изъята из практики. Она основывается на намеренном воздержании от любой естественной практики, включая самые высокие ступени последней». Теоретическая позиция мышления превратила греческих философов в незаинтересованных наблюдателей мира, а их теоретическое знание по степени своей абстрактно-логической мощи далеко превосходило знания, полученные в ходе обыденной практики, а также знания религиозно-мифологические, эзотерические и др.

В отличие от других продуктов человеческой деятельности — ремесла, сельского хозяйства и т.д. — теоретические знания не уничтожаются временем, не поддаются порче, ибо они не реальны, а идеальны. С момента своего возникновения теоретический (идеальный) мир в своей попытке проникнуть в тайны природы «надстраивался» над обыденными житейскими представлениями и чувствами. Появилась тенденция «отчуждения» теоретического мира от обыденного: теоретическая деятельность начинает оцениваться как более высокая по сравнению с деятельностью ремесленников. Аристотель писал, что ремесленники действуют «по привычке», т.е. без помощи слова передают способность «знать руками», а не «умом». Использование новоевропейской наукой практики конструирования идеальных объектов Гадамер прокомментировал так: «Сохранилось фундаментальное начало греческого мышления бытия: современная физика предполагает античную метафизику». На Востоке, в отличие от Древней Греции, философия тяготела к идеологическим конструкциям, обслуживающим традицию. Так, философия конфуцианства и брахманизма одновременно являлась религиозно-идеологическим регулятивом поведения людей, что не способствовало открытию способности мышления к теоретизированию.

Второе интеллектуальное открытие античной мысли связано с превращением математики в теоретическую науку. В античности к математике относили арифметику, геометрию, стереометрию, астрономию, акустику, гармонику (теорию музыки). Гадамер был убежден, что «не естествознание, не говоря уже об истории, но математика была для греков подлинной наукой». Гуссерль особо выделял античную геометрию. Важную роль в формировании древнегреческой математики сыграла пифагорейская школа. Древние греки заимствовали алгебру и арифметику из культуры Востока, но заимствование было творческим: начиная с Пифагора с помощью

числа стали объяснять природу всего сущего, рассматривать числа и числовые отношения как ключ к пониманию вселенной и ее структуры. Родилось убеждение, что «все есть число». Эту мысль повторил спустя 2000 лет Галилей. Пифагорейцы поставили вопрос о сущности числа, т.е. превратили число из средства в предмет и цель исследования. Поиски связей и единства всех возможных закономерностей чисел становятся центральной задачей математических исследований. Греки перестроили математику из системы счета и измерения в систему знаний, построенную на применении правил вывода одних математических положений из других. Они превратили математику в теоретическую науку, так как сформулировали вопрос «как это доказать? и ввели процедуры систематического доказательства. Гадамер писал, что предметом античной математики являлось «чисто рациональное бытие, ее считали образцом для всякой науки, поскольку она представлена в замкнутой дедуктивной системе».

Учение пифагорейцев о том, что природа построена по законам математики, не принял Аристотель. Поэтому вплоть до Реформации средневековая теология, вобравшая в себя многие положения философии Аристотеля, также не одобряла постулат об онтологической значимости чисел и фундаментальности математики по сравнению с физикой. И только в XVII веке вновь актуализируется математическая программа античности, согласно которой «книга природы написана на языке математики». В Новое время изменилось понимание математики: во-первых, всеобщие математические формулы, с помощью которых стали выражать «законы природы», приобрели форму «функциональных» зависимостей чисел, потерявших свой мистический смысл; а, во-вторых, математика стала рассматриваться как образец для других наук не благодаря своему предмету (как это было в античности), но как «самый совершенный способ познания» (Гадамер).

Многие исследователи признают, что христианство, в отличие от религий Древнего Востока, потенциально содержало в себе предпосылки возникновения науки. Генезису науки в христианской культуре, как считают некоторые ученые, способствовала, во-первых, идея единого Бога, творящего по Слову законы Вселенной, во-вторых, христианский идеал святости, согласно которому достигшая высокой степени духовности и самосознания личность не разрывает с телесным миром, а «работает» в нем и для него, просветляя и одухотворяя его. В восточных же религиях постижение «света истины» означало воссоединение со своей духовной родиной («Единым», «нирваной», «Брахманом») и предполагало уход из телесного мира — темницы души, что не способствовало формированию ориентации на познание законов, которые Бог вложил в свое творение.

Некоторые философы, например К.Ясперс, высказывают предположение, что возникновение современной науки «немыслимо без той душевной направленности и тех импульсов, исторической основой которых является библейская религия». К таким импульсам он относил содержащееся в этосе библейской религии требование истинности любой ценой, что «заставляло

видеть в познании не игру, не благородное занятие для досуга, а серьезное дело, профессию, являющую собой самое важное для человека; вытекающее из идеи творения мира Богом убеждение, что все существующее является достойным познания». Даже позитивист Конт не отрицал, что в начале теология и наука не находились в открытой вражде в силу того, что занимались разными вопросами. Но многие ученые связывают генезис новоевропейской науки не с христианством вообще, а с конфессионально отпочковавшимся от него протестантизмом. Их аргументы в пользу своей версии генезиса науки таковы: иерархическая картина мира, господствовавшая в средневековом мировоззрении, была главным препятствием на пути возникновения науки. В этой картине мира доминировала качественная онтология, согласно которой пространство анизотропно (гр. anisos неравный + tropos свойство), место пребывания каждой вещи уникально и совпадает с ее предназначением в иерархии мироздания: Бог-планеты-ангелы-мужчины-женщины-животные-растения. Человечество занимало высшую ступеньку на иерархической лестнице. Вся природа служила человечеству, а человечество Богу. Природа мыслилась как статичная, иерархически упорядоченная властью Бога. В ней царили нравственные, а не механические законы: любовь и цель считались неотъемлемыми характеристиками всего сущего. Развитие вещи трактовалось как раскрытие данных ей Богом потенциальных возможностей. Вот эта картина мира и была разрушена в эпоху Реформации.

Наука — форма духовной деятельности, а потому ее генезис в XVII веке нельзя понять без рассмотрения религиозно-мировоззренческих изменений, вызванных Реформацией, и, прежде всего, изменения картины мира. Совершенно очевидно, что в эпоху господства религии любая новая форма духовной и познавательной деятельности должна была получить религиозное оправдание. А так как время (место) возникновения науки совпало с Реформацией, то появилась гипотеза об опосредованности научной революции XVII века протестантизмом. Ученые, придерживающиеся этой гипотезы, не отрицали того факта, что капиталистическое производство повлияло на возникновение науки, но при этом они считали, что нельзя объяснить генезис науки только запросами промышленности, навигации, военного дела. Эти запросы удовлетворительно объясняют появление в XVI—XVII веках отдельных дисциплин, таких как магнетизм, механика. Но они не объясняют появление новых теорий типа гелиоцентрической системы Вселенной или теории кровообращения.

Основоположник идеи влияния протестантизма на становление капитализма М. Вебер распространял эту идею и на генезис науки: «Хорошо известна явная склонность протестантского аскетизма к эмпиризму, рационализированному на математической основе... Поэтому излюбленной научной дисциплиной всех пуританских, баптистских или пиетических христиан была именно физика и близкие к ней, использующие сходную методику, математические дисциплины».

Американский социолог Р. Мертон, разрабатывая идею о наличии связи между аскетическим протестантизмом и естественными науками, пришел к следующим выводам. Классический протестантизм содействовал мотивации деятельности человека в направлении опыта. Пуританство религиозно оправдало занятия научной деятельностью, что способствовало ее легитимации.

Конечно, Реформация сознательно не ставила перед собой цель изобрести науку, легитимизировать и институционализировать ее. Известно негативное отношение протестантов к некоторым научным открытиям того времени. Лютер аргументировал против Коперника: «Этот болван затеял перевернуть все искусство астрономии. Но в Священном писании прямо сказано, что Иисус Навин остановил Солнце, а не Землю». Парадоксальность протестантизма состоит в том, что его адепты в процессе критики католической теологии бессознательно для самих себя создавали условия возможности возникновения и легитимации науки. Протестантские теологи разрушили средневековую картину мира, главным интегрирующим звеном которой был принцип иерархии, предполагающий ранжирование душ и движений, а также небесных духовных инстанций. Иерархия предполагала наличие строгих качественных различий между земными и небесными телами и явлениями. Протестантская теология отказалась признавать не только иерархию ангельских существ, но и само существование этих духовных инстанций между человеком и Богом, миром и Богом, что разрушало небесную иерархию, без которой не могла быть оправдана церковная иерархия. Разрушив иерархию, Реформация разрушила целостный христианский миропорядок, и на место духовной интеграции Вселенной теперь можно было придумать любую новую интеграционную составляющую, способную гарантировать целостность мира.

В результате разрушения иерархии появилась возможность построить новое понимание мира — без ангелов и духов и представить его как божественный механизм, части которого способны к согласованному самодвижению и самоопределению по законам, единожды установленным Богом в момент творения. Теперь «...окружающий мир оказался миром твердым, холодным, бесцветным, тихим и мертвым, миром количества, миром математически измеряемых и механически регулируемых движений» (Иен Барбур). Только по отношению к так понятому миру можно было обоснованно утверждать, что законы природы написаны на языке математики. Спиноза утверждал, что «если бы люди ясно познали весь порядок природы... они нашли бы все так же необходимым, как все то, чему учит математика». Так, Ньютон доказал закон обращения планет, открытый Кеплером, не физическим, а геометрическим способом, что и создало для него трудности в физическом истолковании введенного им понятия «сила».Так истолкованная Вселенная была как бы теологически «подготовлена» к познанию методами опытной науки, к формированию центральной для науки идеи «закона природы». Декарт первым использовал термин «закон природы», но проинтерпретировал его в контексте

религиозном: «Божество сначала сотворило материю и движение, а затем Вселенная управлялась «законами, установленными в природе Богом», а именно, законами механики». Механическая картина мира как основа научного знания XVII века не была религиозно нейтральной.

Время и пространство потеряли свою качественно-иерархическую определенность, когда каждая «точка» пространства-времени в силу своей уникальности обусловливала качественную специфику занимающего это «точку» тела, а потому время, пространство, тело были взаимосвязаны. Теперь «точки» пространства и времени рассматривались как равноправные и равноценные по своей бескачественности и безразличности к свойствам пребывающих в них тел. Ньютон ввел представление об абсолютном пространстве и времени, определял их так: «Абсолютное пространство по самой своей сущности, безотносительно к чему бы то ни было внешнему, остается всегда одинаковым и неподвижным», а «абсолютное, истинное математическое время... само по себе и по своей сущности, без всякого отношения к чему-либо внешнему, протекает равномерно». Абсолютное пространство и абсолютное время рассматривались как пустые вместилища тел, не оказывающие никакого влияния на их характеристики. Представление о пространстве как однородном и бесконечном лишало мир каких бы то ни было границ. Из Аристотелева понимания причин были элиминированы формальная и целевая причины, что по сути стало отказом признавать Бога как Высшее Благо, к которому все стремится. Причинность стала сводиться к «действующим» и «материальным» причинам. Теперь можно было объяснять все происходящие в мире процессы и перемены не в категориях цели, природных наклонностей, естественных мест, а в категориях массы и движения, связанных определенными количественными закономерностями. Все природные явления теперь определялись не целями, а механическими причинами. Это открыло возможность приписать материи способность к движению за счет собственных внутренних сил без постоянного подталкивания ангелами и духами, без стремления к цели. Эту возможность реализовал Галилей, сформулировав принцип инерции.

Конечным продуктом распада иерархий явилась идея равенства, которая утверждалась не только в протестантских странах, но также в социальной жизни и науке непротестантских стран: гелиоцентрическая теория католика Коперника установила равенство Земли с другими планетами перед Солнцем; абсолютная мо-нархия сделала всех равными в правах (или бесправии) прямым отношением «король—подданный»; Гарвей отменил иерархию сердца, печени и мозга в человеческом теле, уравнял все органы перед кровью.Главная книга философа Нового времени, испытавшего влияние протестантизма, Ф. Бэкона называлась «Великое восстановление наук», в которой он рассматривал процесс познания как «восстановление» языка Адама и той власти человека над природой, которую обеспечивал этот язык. Известен афоризм Ф. Бэкона «Знание — сила». Наука — «служанка теологии», а истинная цель научной деятельности — «прославление Творца и облегчение доли человека», утверждал Ф. Бэкон, способствуя этим

утверждением легитимизации научных исследований, а следовательно, безопасности ученых.

Протестанты приняли «теологию воли» Блаженного Августина, который не проводил принципиального различия между естественным и сверхъестественным (чудом), утверждая, что «весь мир есть чудо», хотя люди и склонны удивляться лишь явлениям редким и непривычным. Достойны удивления и внимания не столько отклонения от нормы, сколько сама норма, сам замысел творения. Поэтому путь к Богу лежит через познание любой твари. Лютер и Кальвин считали, что ум христианина должны занимать не природные отклонения от нормы (так называемые чудеса), а великое чудо сотворенной нормы бытия Вселенной в целом. Кальвин утверждал, что «куда ни бросить взгляд, нет ни единого места во Вселенной, где нельзя было бы обнаружить хотя бы искру величия Бога. Задача человека — выявить волю Творца, запечатленную в любой твари, начиная от червя и кончая человеком». Идеологи Реформации рассматривали научное познание как интерпретацию «почерка Божьего», а потому считали, что ориентация на познание истины, запечатленной Творцом в творении, должна быть неотъемлемым элементом религиозной жизни человека. Естественные науки могли теперь рассматриваться в качестве средства познания «Бога в природе», что явилось одним из главных условий легитимации науки.

Воля человека была переориентирована с созерцательного мироотношения на активную деятельность. Благодаря Лютеру религиозная вера обрела рефлексивность: взамен веры в авторитет провозвестия и предания протестантизм утверждает господство субъекта, настаивающего на собственном понимании Божественного писания. Это можно рассматривать в качестве интеллектуальной предпосылки формирования представлений о субъекте и объекте науки Нового времени: принцип активности субъекта («свободы субъективности» — Гегель) становится доминирующим. Переориентация воли человека с созерцательного отношения к истине на активный ее поиск в Книге природы зафиксирована в философии Ф. Бэконом: «Ведь речь идет не о созерцательном благе, но поистине о достоянии и счастье человеческом и о всяком могуществе в практике. Познавательная деятельность ученых стала базироваться на представлении, что субъект не пассивно созерцает объект, а «творит» его. Философски обосновывая идею активности субъекта познания, И. Кант назовет это обоснование «коперниковым переворотом» в гносеологии.

Непреднамеренными последствиями протестантской теологии явились социализация и институционализация научного исследования: оно было признано на правах достойной христианина деятельности, не вызывающей ни чувства вины, ни тем более чувства совершенного преступления перед религией. Этим можно объяснить тот факт, что общество, в котором уже действовал целый ряд социальных институтов, не предусматривающих появление науки, не воспротивилось институционализации и социализации науки. Идеология Реформации подготовила почву для легитимации (лат.

legitimus— законный) зарождающейся науки. Ни конфуцианство, ни ислам не обладали такой легитимизирующей функцией в отношении науки, как протестантизм в Англии.

Реформация в своей идеологии ставила цель — максимально развить самосознание человека, научить его опираться только на собственный внутренний духовный опыт. Для протестантизма (особенно кальвинизма) этика эпикуреизма была привлекательна, во-первых, требованием мужества и полной личной ответственности за свою жизнь в согласии с индивидуально осознанным ее смыслом; во-вторых, отрицанием авторитетов и полного доверия к собственному разуму в достижении внутренней уравновешенности и спокойствия духа; в-треть- их, отрицанием ценности абстрактного философствования и признанием практического интереса главным критерием истины; в-четвертых, представлением о разумном, договорном характере общественных связей. Следует учесть, что атомизм эпикурейцев имел нравственный смысл, а потому их физика была одновременно и основой этики. Из идеи атомистического устройства основ Космоса Эпикур делал этически-нравственные выводы: так как мироздание состоит из атомов и пустоты, то человек не может рассчитывать на помощь богов, а должен мужественно взять груз ответственности за свою жизнь только на свои плечи. Поэтому принять этику эпикурейцев нельзя без физики.

Обратившись к этике эпикуреизма, протестантизм совершил своеобразную интеллектуальную революцию, если учесть, что не только этика, но и эпикурейская физика (атомизм) осуждались христианской теологией. Хотя «языческая» этика соответствовала идеологии протестантизма, все же атомизм в науке должен был получить специальную религиозную легитимацию. П.Гассенди нашел специфические аргументы для религиозного оправдания атомизма, обвиняемого в атеизме. Авторство идей атомизма, утверждал он, принадлежит не Левкиппу и Демокриту, а финикийцу Моху, которые тот позаимствовал у иудеев. Конечный источник движения находится не в самих атомах, а привнесен в них извне Богом. Тем самым было обосновано соответствие атомизма идее креационизма. Ученые теперь безбоязненно могли использовать учение Эпикура об атомах, в частности, ту часть учения, в которой утверждалось, что атомам присуща одинаковая скорость, если при движении в пустом пространстве они не наталкиваются на сопротивление. Но в католических странах, в отличие от протестантских, распространение атомизма встречало сопротивление церкви. Так, Декарт, публикуя свои механико-корпускулярные идеи, избежал столкновения с католической церковью только потому, что объявил созданную им картину мира, в котором царят механические законы, принципиально гипотетической. Научное мышление как высшая нравственная ценность.

Перед нарождающейся наукой, мыслителями XVII века стояла задача — сделать научное познание самым главным и благородным, с религиозной точки зрения, делом человека. Природу нужно изучать, чтобы правильно жить. Этические вопросы, ставшие центральными в теологической и

житейской практике реформаторов, могли быть решены двумя способами: а) через непосредственное постижение природы человека; б) опосредованно через изучение тех смыслов, которые Творец вложил в физический мир. Оба способа равноправны, ибо и природа человека и Вселенная созданы единым Творцом, сообразно единому разумному плану творения. Но все же предпочтение было отдано второму способу по той причине, что человек своим эгоизмом, своеволием, буйством низменных страстей, являющихся следствием грехопадения, исказил замысел Божий. Физический же мир являет познающему уму божественные законы в более чистом виде, а потому их познание позволит более адекватно постичь мудрость Творца относительно назначения человека.

Но здесь возникала проблема: как может человек, обуреваемый низменными страстями и аффектами, адекватно познать божественные законы, вложенные Творцом в природу? Только очистившись от своеволия, аффектов и других искажающих ум человека пороков. Декарт выделял четыре рода предрассудков, а Ф. Бэкон — четыре группы «идолов», от которых следует избавить познающий ум. В пространстве становящейся науки XVII века познание законов природы и нравственная «работа» субъекта оказались первоначально неразрывно связанными. Мышление, освобожденное от эмоционально-аффективной составляющей, стало рассматриваться как высшая нравственная ценность. Так, Паскаль писал: «Будем же стараться хорошо мыслить: вот начало нравственности». Декарт был уверен, что всякий, понявший его принципы строения физического мира, убедится, «до какой высокой степени мудрости, до какого совершенства жизни, до какого блаженства могут довести нас эти начала». В человеке стала цениться его способность самостоятельно и безукоризненно, с точки зрения логики, мыслить. Такая позиция нашла оправдание в философии XVII века (Ф. Бэкон, Декарт, Спиноза, Гоббс, Бойль и др.). «Очищенное» от страстей и аффектов мышление было сведено к категориальной деятельности рассудка, которая, как предполагалось, совершалась в соответствии с определенными принципами и законами мышления, имеющими универсальный характер. Это породило представление о самотождественности субъекта познания, об инвариантности мыслительных познавательных процедур, что явилось основанием идеи трансцендентального субъекта, сформулированной впоследствии И. Кантом (философом-протестантом) и сыгравшей важную роль в формировании представления о субъекте научного познания.

Определённую роль в становлении науки Нового времени сыграли университеты. В Европе университеты появились в начале XIII века, а к концу века они превратились в центры культурной жизни Европы, где шла подготовка светской и церковной элиты, распространяющей в обществе научные, правовые и теологические идеи. По вопросу о влиянии средневековых университетов на становление науки существует две точки зрения.

1. Наука возникла вне традиционной академической системы, где господствовала натурфилософия Аристотеля, органично связанная с христианско-схоластической теологической картиной мира. Интеллектуальная элита этих университетов, состоящая в большинстве из схоластов и гуманитариев, не хотели давать естественным наукам статус, равный их собственным дисциплинам, и прежде всего теологическим. Фейербах приводит сведения о том, что парижский теологический факультет подал в курию жалобу на Фому Аквинского, обвинив его в неправомерном действии соединения теологии с философией. Аргументом в пользу непричастности университетов к становлению науки служит и тот факт, что «над созданием нового естествознания работало много людей со свободным духовным горизонтом, стоявших вдали от прочно придерживающихся старых традиций университетской жизни. Достаточно указать лишь на Коперника, Кеплера, Тихо, Герике, Агриколу, Левенгука, Грю и многих других... Университеты относились иногда прямо отрицательно к естественнонаучному исследованию».

2. Средневековые университеты повлияли на становление науки. Аргументы сторонников этой позиции таковы: (а) университеты развивали «книжную ученость», но, в отличие от монастырей, они воспринимали книгу не как сокровище божественной мудрости, а как инструмент познания, что породило возможность свободного толкования текстов. В средневековых университетах господствовала схоластика, т.е. тип религиозной философии, которая, опираясь на теологию и метод соединения догматических предпосылок с рационалистическими объяснениями, особое внимание уделяла разработке формально-логических процедур, жестко фиксированных правил мышления, что явилось необходимой базой для становления научного мышления; (б) отрицать возможность появления науки в средневековых университетах на том основании, что платоники критиковали механическую картину мира, не вполне корректно, если учесть, что один из главных основоположников науки Нового времени И. Ньютон также не согласился с предло-женной Декартом картиной мира, где все силы, действующие во Вселенной, сводились к механическим. Причиной такого несогласия была «метафизическая» вера Ньютона в существование во Вселенной помимо инертной материи активного начала, носящего немеханический характер. Рассматривая силы тяготения как математические, Ньютон не наносил «ни малейшего вреда славе Бога как творца и правителя Вселенной»; (в) в Оксфорде, где доминировала логика, тесно связанная с математикой и астрономией, работали ученые Р. Гроссетест, Роджер Бэкон, которые начали опытные исследования света. Р.Бэкон вводил в пространство научного доказательства наблюдение и опыт с целью нахождения математических соотношений в мире природы. Он пытался установить связь между свободными искусствами и механикой, между наукой и техникой. (г) в университетах XIII века были предприняты первые попытки создать теорию «двойственной истины», т.е. теорию сосуществования истин веры и разума.

И все же вывести классическую науку Нового времени только из средневековой и даже возрожденческой университетской учености невозможно, так как наука нового времени немыслима, как мы показали выше, без перехода от иерархически упорядоченной средневековой картины мира к открытой, лишенной иерархии многообразной развивающейся Вселенной. Такой переход не мог эволюционно созреть в мысли того или иного университетского ученого-схоласта, и даже, если предположить, что в чью-то голову пришла бы спонтанно мысль о необходимости отменить иерархию, то средневековая теология сразу же «заблокировала» бы распространение этой «крамолы». М. Хайдеггер писал в этой связи, что современное слово «наука» означает нечто иное по сравнению с «doctrina и scientia средневековья или episteme греков», а Гадамер считал, что «наука Нового времени осуществила решительный разрыв с формами знания греческого и христианского Запада».

Как известно, эмпирическое исследование появилось задолго до возникновения науки. Эмпиризм как методология восходит к Аристотелю, который, как и Демокрит, признавал значение опытного познания, сводящегося к наблюдению окружающей природы. До Аристотеля господствовал платоновский идеал постижения «чистых» идеальных сущностей, а единственным предметом, заслуживающим теоретического интереса, выступала некая вечная первооснова, которая определяла все совершающееся в мире. Изучение этой «первоосновы» служило средством для решения нравственно-мировоззренческих проблем. Применительно к изучению природы этот познавательный идеал мог реализоваться в полной мере только для изучения божественной сферы небесных движений.

Аристотель же исходил из того, что все единичные, меняющиеся и преходящие вещи достойны познавательного интереса сами по себе, а не ради решения каких-то морально-нравственных проблем. Будучи своеобразным методологом эмпирико-описательного стиля научного мышления, Аристотель не случайно был первым античным ученым, создавшим науку о природе — физику и выделившим в ней центральную проблему — движение. Признание того факта, что природе присуще движение, изменение, привело Аристотеля к убеждению о невозможности строить физику на базе математики, так как математика изучает «статические связи и отношения». Для Аристотеля фундаментом истинного знания о реальном мире является «восприятие — а не умозрительные математические построения; опыт — а не априорные геометрические рассуждения». Аристотелевская наука «основывалась на чувственном восприятии и была действительно эмпирической, она гораздо лучше согласовывалась с общепризнанным жизненным опытом, чем Галилеева или Декартова наука», — писал А. Койре. Методология эмпиризма — это методология созерцания, т.е. невмешательства в естественный ход вещей, признания неприкосновенности естества, а потому опытное постижение, проводимое в рамках такой методологии, нельзя отождествлять с экспериментальным.

Не исключала возможности использования опыта и наблюдения и средневековая, особенно позднесхоластическая натурфилософия, несмотря на доминирующую установку, согласно которой ответы на все человеческие вопросы уже содержатся в текстах Священного Писания и Предания и могут быть получены путем логического и филологического, рационального и нерационального проникновения в их смыслы. Опытное исследование «оправдывалось» так: Бог сотворил природу, и человек, обращаясь к опытному ее исследованию, постигает одновременно и Бога. Эмпирия (натуралистический опыт) выступала для средневековых ученых как «служанка» священного знания, как подготовительный этап перехода к мистическому опыту непосредственного постижения божественных истин внутренним созерцанием, озарением.

Методология экспериментализма, в отличие от методологии эмпиризма, признает возможность и правомерность вмешательства человека в естественный ход событий с целью вычленения в нем разумного «идеального объекта». Разум осознал, что он может и должен заставлять природу отвечать на вопросы, которые он ставит сам по собственному плану. Среднековье не может считаться «родиной» эксперимента, ибо от опыта и наблюдения ожидали, что бы они «навели» на мысль, подсказали проблему и пути ее решения. Эксперимент, на котором воздвигнуто здание естественной науки Нового времени, подразумевает «лабораторные» условия, позволяющие изолировать и контролировать исследуемую систему. В ходе эксперимента ученый управляет физической реальностью, вынуждает ее действовать по теоретическому «сценарию», т. е. отвечать на вопрошания разума. Возникновение эксперимента (лат. experimentum — опыт, проба) связывают с именем Галилея. В античности под опытом понимали жизненный опыт чувственного восприятия, а в средние века — конструкцию из эмпирических данностей, создаваемую в контексте истин откровения. Для Галилея «опыт» — это конструирование некоего явления из эмпирических данностей на основе определенной теоретической предпосылки, сформулированной самим ученым. Галилей воспроизвел формальную структуру построения теории, изобретенной Евклидом: вначале создавал первичные идеальные объекты (сила, материальная точка и т.д.), а затем использовал их для построения «вторичных» идеальных объектов, в качестве которых выступали модели явлений природы. Решая доставшуюся от Аристотеля задачу описания падения камня, Галилей начал не с эмпирического наблюдения, а с теоретической гипотезы, согласно которой природа «стремится применить во всяких своих приспособлениях самые простые и легкие средства». «Когда я замечаю, — писал он, — что камень, падающий со значительной высоты, приобретает все новое и новое приращение скорости, не должен ли я думать, что подобное приращение происходит в самой простой и ясной для всякого форме? ... Нет приращения более простого, чем происходящее всегда равномерно». Теоретическое предположение (которое могло быть заимствовано как у эпикурейцев, так и у стоиков) о том, что приращение скорости падающего тела происходит всегда равномерно, есть основание для

построения мысленного физического эксперимента, в ходе которого происходит создание первичных идеальных объектов: тела, движения в пустоте, среды и т.д. С помощью этих первичных идеальных объектов Галилей строит идеальную модель самого физического явления падения тел с высоты, воплощая его затем в инженерную конструкцию: гладкие наклонные плоскости, шары и т.д. В этом «воплощении» и заключена специфика эксперимента: теоретическая идеальная модель и «эмпирический материал» оказываются связанными. Эксперимент объединил сущность (идеальные математические конструкты объектов) и существование (реальные чувственные предметы). Галилей впервые понял, что в науке можно приписать существование лишь тем «сущностным» (идеальным) объектам, которые могут быть выполнены в эмпирическом опыте и выражены математически. В противном случае идеализации не могут быть с полной определенностью отнесены к научным. Опытная объективация идеализации — это гносеологический императив экспериментальных наук. Идеальная модель одновременно является способом «преобразования» реальных предметов (мысленный эксперимент) и способом их познания.

Эксперимент включает в себя два уровня: (1) мысленный эксперимент, который начинается с полагания в основу определенного теоретического принципа или предположения и протекает как искусственное целенаправленное воспроизведение природных явлений и процессов в виде идеальных моделей, предполагающих видоизменение и сознательно контролируемое устранение побочных, несущественных для этих явлений и процессов, компонентов и признаков; (2)реальный эксперимент, т.е. техническое исполнение мысленного эксперимента. Оба эти уровня неразрывно связаны и подчинены единой цели — подтвердить тот или иной закон, то или иное теоретическое предположение. Опытная объективация идеализации — это гносеологический императив экспериментальных наук. Идеальная модель одновременно является способом «преобразования» реальных предметов (мысленный эксперимент) и способом их познания.

Галилей соединил эксперимент с математическим объяснением, придал физическим идеализациям математический характер. Как и пифагорейцы, он считал, что законы природы записаны на языке математики. Но для античных греков математика была единственной подлинной наукой, которая, как писал немецкий математик Герман Вейль, изучала «теоретико-числовые свойства», служащие источником магической силы чисел, тогда как для Галилея в естествознании имеет значение не магия чисел, а «их свойства в качестве величин». Эту точку зрения раньше Вейля высказал Лейбниц, утверждая, что числа входят в объяснение природы благодаря тому, что имеют характер величин. Впоследствии Гуссерль также скажет, что Галилей превратил математику в «исследовательскую технику», в самый совершенный способ познания количественных, а не качественных параметров явлений и процессов.

Описывая с помощью математического уравнения ускоренное движение катившегося по наклонной плоскости шара, Галилей дал пример

математического объяснения эксперимента. Он пользовался такими абстракциями, как длина, время, скорость, которые можно измерить и выразить в математических символах, и ставил задачу — выразить законы природы через математические отноше-ния измеряемых переменных. Его интересовала проблема не почему движутся предметы, а как они движутся, его научный интерес был направлен не на нахождение целевых и формальных причин, относящихся к качественной сущности предмета, а на изучение действующих причин, дающих возможность количественного объяснения. Галилей заложил основы экспериментально-математического естествознания.

Но мир математически описываемых свойств не мог быть жизненным миром человека. Это механический универсум, в котором человек занимает случайное место, а мир при этом может функционировать и без него. Личный, осмысленный мир, мир духовных ценностей, чувств и мыслей, мир, осознанный и свободный, теряет свою связь с научным миром, в котором господствует детерминизм. Научная картина мира — это наступление на обширный мир человеческого опыта, на тот мир, в котором люди чувствуют себя уютно и с которым они имеют внутреннюю взаимосвязь. Мир науки, по словам Ф. Бэкона, — это мир, подвергающийся таким истязаниям, что он непременно раскроет свои тайны и секреты; и только такой мир — мир объектов и может быть предметом технологических манипуляций.

С развитием науки процесс объективации не ограничился природой. Шаг за шагом этот процесс охватывал собой все новые и новые области знания: и живую природу, и общество, и, наконец, человека. Гуссерль писал, что «Галилей осуществил замещение единственно реального, опытно воспринимаемого и данного в опыте мира — мира нашей повседневной жизни миром идеальных сущностей, который обосновывается математически». В этом «замещении» и состояла суть научного познания.

Начатое Галилеем использование математически выраженных теорий и экспериментов продолжил И.Ньютон. Тесную связь между экспериментальными явлениями и математическими структурами подтвердили и физики начала XX века. Эксперимент вошел в арсенал методов научного познания в качестве одного из главных его средств, и в XX веке физик Н.Бор, обсуждая с Эйнштейном гносеологические проблемы, поставленные атомной физикой, писал, что «словом «эксперимент» мы указываем на такую ситуацию, когда мы можем сообщить другим, что именно мы сделали и что именно мы узнали».

Открытие опытно-экспериментальной науки часто приписывают не ученому Галилею, а его современнику философу Ф. Бэкону, который выдвинул идею необходимости опытного естествознания, разделил научные опыты на «плодоносные», имеющие отношение к технологическому применению, и «светоносные», связанные с чистой наукой. Но считать Ф. Бэкона основателем экспериментальной науки было бы большой натяжкой. В отличие от Галилея Ф. Бэкон считал, что наука классифицирует наблюдения без теоретических предположений, что путь к научным открытиям лежит

через индукцию (наблюдение и обобщение его результатов), а потому может совершаться как бы механически, без создания абстрактных идеальных моделей. Теоретически он зафиксировал необходимость опираться в науке на опыт, но не довел опытное познание до формы научного эксперимента. Даже его идея «плодоносных» опытов не была доведена им до практического приложения.

В средние века схоластическая ученость существовала автономно от ремесленных практик, владеющих секретами технологий производства предметов и орудий труда. В эпоху Возрождения стирается граница между отвлеченной ученостью и ремесленной деятельностью. Рождается новый тип интеллектуала, соединившего «высокую» ученость с искусством ремесленника (например, Леонардо да Винчи), который стал толковаться как подражание творчеству Бога, что открывало широкий простор для экспериментов в науке, а деятельность ученого-экспериментатора стала рассматриваться как своеобразное подобие в малых масштабах актам творения, направленным на распознавание в природе божественных разумных законов. Получившее в XVII веке религиозную санкцию представление о «бездуховности» и механистичности природы способствовало утверждению мнения о том, что для распознавания разумных законов в природе необходимо не созерцание, а своеобразное «дознание», которое состояло в том, что человек задавал природе вопросы и для получения на них ответов активно преобразовывал природные объекты в соответствии с логикой задаваемых вопросов и ожидаемых ответов. Также укоренившийся в культуре протестантизма принцип пользы создал этические предпосылки возникновения эксперимента. Представление об однородности пространства и времени, разрушение противопоставления небесной и земной сфер стало условием становления физического эксперимента, предполагающего его принципиальную воспроизводимость, повторяемость в любой точке пространства и в любой момент времени. В этом мировоззренческом контексте Галилей сформулировал невероятную, с точки зрения средних веков, эвристическую программу — исследовать закономерности движения небесных тел по законам механики, проверяемым в эксперименте.

Открытие научного эксперимента породило много идеологических и методологических споров. В.И. Вернадский писал, что открытие экспериментально-математического метода научного познания не было принято сразу всеми учеными XVII века. «Пытка естества» (выражение Г. Галилея), характерная для экспериментального метода (русское слово «естествоиспытатель» также содержит корень «пыт», который составляет основу слова «пытка»), осуждалась не только многими теологами, но и философами. Так, Ж.-Ж. Руссо писал: «Все хорошо, выходя из рук Творца вещей, все вырождается в руках человека». Гете, будучи крупным натуралистом, до конца жизни не принял ньютоновскую картину мира, базирующуюся на признании экспериментально-математической идеи, хотя

уже при его жизни идеи Ньютона оказались плодотворными в физике и небесной механике.

Среди ученых XX века длительное время велись дискуссии по вопросу: открывает или преграждает себе человек путь к познанию предметов и явлений природы в эксперименте? В классической физике господствовало молчаливое предположение, что средства наблюдения не влияют на объект и его поведение, и объект изучается сам по себе. Но уже Гегель отверг эту методологическую и мировоззренческую установку, доказав, что эксперимент «возмущает» объект в силу деятельности субъекта с его приборами и собственным исследовательским методом. Предположение, что наука изучает объект в его независимом от субъекта познания бытии, есть абстракция. Квантовая физика обнаружила невозможность познания атомов самих по себе, вне зависимости от экспериментально поставленного вопроса. Н. Бор утверждал, что «согласно квантовому постулату, всякое наблюдение атомных явлений включает такое взаимодействие последних со средствами наблюдения, которым нельзя пренебречь. Соответственно этому невозможно приписать самостоятельную реальность в обычном физическом смысле ни явлению, ни средствам наблюдения... Поведение атомных объектов невозможно резко отграничить от их взаимодействия с измерительными приборами, фиксирующими условия, при которых происходят явления».

Возникла проблема: что изучается в эксперименте — сама вещь или ее взаимодействие с прибором? В дискуссии по этому вопросу участвовали Н. Бор, А. Эйнштейн, В. Гейзенберг, М. Планк и другие физики. Многие из них пришли к принципиально отличному от классического мировоззрению. Так, В. Гейзенберг писал: «Мы с самого начала находимся в средоточии взаимоотношений природы и человека, и естествознание представляет собой только часть этих отношений, так что общепринятое разделение мира на субъект и объект, внутренний мир и внешний, тело и душу больше не приемлемо и приводит к затруднениям». Ученые-физики, таким образом, пришли к выводу, что «физика описывает реальное в той мере, в какой оно дано субъекту, не описывая самого субъекта» (А. Кожев).

Идея извлечения пользы из знания, власти над природой была вычитана протестантами из Библии, но там ничего не говорилось о том, как приложить знания к технологиям, т.е. как извлечь практическую пользу из научного знания. Известно, что ученые предприняли ряд попыток реализовать такого рода приложения, но успеха не имели. Так, в 1670-е гг. Гюйгенс и Гук пытались усовершенствовать часы, но хронометр открыл примерно в это же время плотник Хэррисон. Паровая машина в ее технологически приемлемых формах была разработана независимо военным инженером Сейвери и кузнецом Ньюкоменом. Парадокс этой ситуации состоит в том, что наука, став экспериментальной, соединила теорию и практику, в роли которой выступал эксперимент. Но это была эксклюзивная практика. Что же касается выхода экспериментальной науки в область широкого технологического приложения, то здесь существовал разрыв.

Технологическое применение научного знания началось с конца XIX века. До этого времени практики делали технологические изобретения, не опираясь на научные знания, и только задним числом наука начинала выяснять принципы и законы, лежащие в их основе. Так, Сади Карно в начале XIX века для научного обоснования цикла работы двигателя изучал работу паровых машин, созданных без помощи науки и независимо от нее. В конце XIX века Дизель идет обратным путем: он вначале исследовал научно-теоретическую сторону проблемы, а затем разработал двигатель.

Технологическое приложение научного знания может быть реализовано практиками, имеющими научную подготовку. Но как осуществить такую подготовку? Путь изучения практиками оригинальных трудов ученых очень трудный и по времени долгий: практик должен вначале освоить научные труды, стать вровень с теоретиком, сделавшим открытие, а затем уж заниматься технологическими разработками. Совершенно очевидно, что в таком случае неизмеримо возрастет время от научного открытия до его практически-технологического приложения, что чревато многими негативными последствиями как технологического, так и экономического порядка. Поэтому надо было облегчить практикам процесс усвоения научного знания и ускорить этот процесс. Для этого научно-теоретическое знание следовало «упаковать», т.е. привести в соответствие с физическими и ментальными возможностями его сравнительно легкого и быстрого усвоения. Сделать это можно двумя способами: во-первых, подготовить штат преподавателей, которые бы занимались такой «упаковкой» знания и затем передавали его учащимся; во-вторых, сжать весь корпус научного знания до уровня словарей и учебников и научить пользоваться ими.

И то и другое было реализовано впервые в Германии. В организованном усилиями Гумбольдта Берлинском университете был создан «профессорский» или «приват-доцентский» штат, задача которого и состояла в том, чтобы весь корпус оригинальных научных исследований свести к представлениям, удобным для быстрого усвоения, а созданная Либихом (1826) в Гисене химическая лаборатория стала местом подготовки химиков высшего класса и соответствующего уровня химических исследований. Но необходимо отметить, что знаменитый химик Либих говорил своим студентам: «Не забывайте, что мы при всех наших знаниях и исследованиях остаемся близорукими людьми, сила которых коренится в том, что мы имеем опору в высшем существе». Академические новации Гумбольдта и Либиха вывели Германию в разряд ведущих стран мира в области технического приложения научных знаний.

Начавшееся широкое технологическое приложение научного знания способствовало формированию современного взгляда на науку как неразрывное триединство (а) исследовательской, (б) прикладной и (в) академической составляющих. Такого понимания науки не было в XVII веке.

В конце XVIII – начале XIX в. происходят радикальные перемены в естествознании. Начинает развиваться биология, химия и другие области знаний, что приводит к выделению науки из натурфилософии,

формированию дисциплинарно организованной науки. Натурфилософские системы природы, созданные до XIX в. И. Кантом, Ф. Шеллингом, Г. В. Ф. Гегелем, в XIX в. не могли уже выполнять функции теоретического анализа и обобщения новых научных данных. Это было обусловлено, с одной стороны, тем, что натурфилософия давала умозрительную картину мироздания, в формировании которой участвовали этические, эстетические и религиозные взгляды, она часто опиралась на антропоморфные аналогии, эмоциональные аргументы и фантазии. С другой стороны, тем, что натурфилософия XVII–XIX вв. опиралась на механистическую картину мира. При этом механика прямо отождествлялась с точным естествознанием, и ее задачи, сфера ее применимости казались безграничными.

Переход к дисциплинарному естествознанию ограничил сферу идеалов механики и сформировал новую систему разнообразных, специфических для каждой дисциплины идеалов и норм, отражающих особенности различных предметов исследования. В биологии, химии и других областях знания формируются специфические картины реальности, не редуцируемые к механистической картине мира. Накапливаются факты, которые все труднее было согласовывать с ее принципами. Начинается процесс расшатывания механистической картины мира, она теряет свой универсальный характер, расщепляясь на ряд частно-научных картин.

В этот период формируется система прикладных и инженерно-технических наук как посредник между фундаментальными знаниями и производством. Различные сферы научной деятельности специализируются, и формируются соответствующие этой специализации научные сообщества.

Новые открытия в науке не укладывались в господствующую механистическую картину мира, свидетельствовали об ее ограниченности. Встал вопрос об абсолютной истинности классической механики как теоретической базы естествознания и основанной на ней картины мира и об адекватности эпистемологических идей и представлений, лежащих в основе научного познания. Фундаментальные естественнонаучные представления о материи, пространстве, времени, причинности потребовали серьезного философского анализа. Это привело к осознанию кризиса в естествознании (прежде всего, в физике). Он проявлялся и на уровне понятий и принципов, и на уровне философско-методологических оснований, и на мировоззренческом уровне (материализм, идеализм).

Осознание кризиса в естествознании приводит к необходимости коренной перестройки оснований науки – перестройки научной картины мира, идеалов и норм познания, философских оснований науки. Становление новой научной картины мира во многом связано с формированием нового образа детерминизма.

В конце XIX – начале XX вв. начался переход к новому типу рациональности, в основе которого лежит представление о неразрывности субъекта и объекта исследования, невозможности устранения субъекта из научной картины мира, изображение мира самого по себе, без учета средств и методов познания. Квантовая механика дала первые наглядные примеры и

неопровержимые доказательства о включенности познающего субъекта в тот предметный мир, который он исследует. Поведения атомных объектов «самих по себе» невозможно резко отграничить от их взаимодействий с измерительными приборами, со средствами наблюдений.

Наряду с естествознанием и техническими науками в XIX в. осуществляется становление социальных и гуманитарных наук. Поднимается вопрос о специфике методов гуманитарного познания. Все это заставляет задуматься о фундаментальных основаниях научного знания. Классический этап в развитии науки уступает место новому – неклассическому, который продолжался с конца XIX в. до последней трети XX столетия. Открытие радиоактивности, сложной структуры атома, создание квантовой механики, теории относительности составили ядро научной революции конца XIX – начала XX вв., привели к появлению электродинамической и квантово-механической картин мира, изменился и стиль научного мышления. В 70-е гг. XX в. научное знание претерпело новые качественные трансформации, началось формирование постнеклассической науки.

Вопросы для самоконтроля

1. Каковы особенности развития научных знаний в античности?
2. В чем состоят особенности развития рациональности в эпоху средневековья?
3. Какие факторы содействовали духовной революции в эпоху Возрождения?
4. Каковы основные этапы становления классической науки?
5. В чем особенности научной программы Р. Декарта?
6. Что характерно для ньютоновской научной программы?
7. Что объединяет все научные программы, существовавшие в классической науке?
8. Какие факторы повлекли за собой научную революцию на рубеже XIX–XX вв.?
9. Каковы характерные особенности неклассической науки?
10. Чем обусловлено становление постнеклассической науки?

Основная литература

Бернал, Дж. Наука в истории общества / Дж. Бернал. – М., 1955.
Вернадский, В. И. Избранные труды по истории науки / В. И. Вернадский. – М., 1981.
В поисках теории развития науки. – М., 1982.
Гайденко, П. П. Эволюция понятия науки (становление и развитие первых научных программ) / П. П. Гайденко. – М., 1980.
Гайденко, В. П. Западноевропейская наука в средние века / В. П. Гайденко, Г. А. Смирнов. – М., 1989.

Гайденко, П. П. Эволюция понятия науки (формирование научных программ Нового времени XVII–XVIII вв.) / П. П. Гайденко.– М., 1987.
Галилео Галилей. Диалог о двух главнейших системах мира : Птолемеевой и Коперниковой / Галилео Галилей. – М. ; Л., 1948.
Койре, А. Очерки истории философской мысли. О влиянии философскихконцепций в развитии теорий / А. Койре. – М., 1985.
Коперник, Н. О вращениях небесных сфер / Н. Коперник. – М., 1964.
Косарева Л.М. Социокультурный генезис науки Нового времени (Философский аспект проблемы).-М.,1989.
Кузнецов, Б. Г. Идеи и образы Возрождения / Б. Г. Кузнецов. – М., 1979.
Маркова, Л. А. Наука, история и историография XIX–XX вв. / Л. А. Маркова. – М., 1987.
Нейгебауэр, О. Точные науки в древности / О. Нейгебауэр. – М., 1968.
Рожанский, И. Д. Античная наука / И. Д. Рожанский. – М., 1980.
Степин, В. С. Научная картина мира в структуре техногенной цивилизации / В. С. Степин, Л. В. Кузнецова. – М., 1994.
Типы рациональности в культуре : сборник статей. – М., 1992.
Философия и методология науки. – М., 1996.
Хореев, Н. В. Философия как фактор развития науки / Н. В. Хореев. – М., 1979.
Эпистемология и постнеклассическая наука : сборник статей. – М., 1992.

Дополнительная литература

Ахутин, В. А. История принципов физического эксперимента: от античности до 17 века / В. А. Ахутин. – М., 1976.
Ахутин, В. А. Понятие «природы» в античности и в новое время /В. А. Ахутин.– М., 1988.
Наука и культура. – М., 1984.
Ван дер Варден Б. Пробуждающаяся наука: рождение астрономии / Ван дер Варден Б. – М., 1991.
Ван-дер-Верден Б. Пробуждающаяся наука: математика Древнего Египта, Вавилона и Греции / Ван-дер-Верден Б. – М., 1957.
Дмитриев, И. С. Религиозные искания Исаака Ньютона / И. С. Дмитриев //Вопросы философии. – 1991. – № 6.
Дорфман, Я. Г. Всемирная история физики с древнейших времен до конца XVIII в. / Я. Г. Дорфман. – М., 1974.
Кирсанов, В. С. Научная революция XVII века / В. С. Кирсанов. – М.,1987.
Кудрявцев, В. С. История физики. Т. 1 / В. С. Кудрявцев. – М., 1956.
Надточаев, А. С. Философия и наука в эпоху античности / А. С. Надточаев. – М., 1990.
Петров, М. К. Язык, знак, культура / М. К. Петров. – М., 1981.
Спасский, Б. И. История физики : в 2 т. / Б. И. Спасский. – М., 1963.
Степин, В. С. Философская антропология и философия науки / В. С. Степин. – М., 1992.
Катасонов, В. Н. Метафизическая математика XVII в. / В. Н. Катасонов. М., 1993.

Григорьян, А. Г. Очерки развития основных понятий механики /А. Г. Григорьян, В. П. Зубов. – М., 1962.
Маркова, Л. А. Конец века – конец науки / Л. А. Маркова. – М., 1992.
Жмудь, Л. Я. Пифагор и его школа / Л. Я. Жмудь. – Л., 1990.
Рабинович В. Л. Алхимия как феномен средневековой культуры .-М.,1979.

Тема 3. Уровни, формы и методы научного познания

В научном познании выделяются два уровня: *эмпирический* и *теоретический*. Они отличаются друг от друга: глубиной, полнотой, степенью всесторонности постижения объекта; целями, методами постижения и способами выражения знаний; степенью значимости в них чувственного и рационального моментов.

На эмпирическом уровне осуществляется наблюдение объектов, фиксируются факты, проводятся эксперименты, устанавливаются эмпирические соотношения и закономерные связи между отдельными явлениями. На теоретическом – создаются системы знаний, теорий, в которых раскрываются общие и необходимые связи, формулируются законы в их системном единстве и целостности.

Эмпирический и теоретический уровни различаются также и тем, с какой стороны они исследуют объект, каким образом получено основное содержание знаний, что является логической формой его выражения, научной и практической значимостью содержания знания.

На эмпирическом уровне научного познания объект отображается со стороны его внешних связей и проявлений, которые доступны, в основном. Живому созерцанию. Логической формой выражения знаний эмпирического уровня является система суждений и умозаключений, с помощью которых формулируются законы, которые отображают взаимосвязи и взаимодействия явлений действительности в их непосредственной данности. Практическое применение знаний, полученных эмпирическим путем, ограничено, а если говорить о развитии научного знания в целом, то они являются начальными, исходными для построения теоретического знания. На эмпирическом уровне основное содержание знания получается, как правило, из непосредственного опыта, их научного эксперимента. Рациональными здесь являются форма знаний и понятия, язык науки, в которых выражен результат данного уровня научного познания. На теоретическом уровне объект отображается со стороны его внутренних связей и закономерностей, которые постигаются путем рациональной обработки данных эмпирического познания, а субъект с помощью мышления выходит за пределы того, что дается непосредственно опыту, и осуществляет переход к новому знанию, не обращаясь к

чувственному опыту. На теоретическом уровне раскрываются глубинные причины и связи между явлениями. Абстрактное мышление является здесь не только формой выражения результатов познавательной деятельности, но и способом получения нового знания.

Главной познавательной функцией эмпирического уровня является *описание* явлений, теоретического – *объяснение* их.

Для выяснения специфики теоретического познания важно подчеркнуть, что теория строится с явной направленностью на объяснение объективной реальности, но описывает непосредственно она не окружающую действительность, а идеальные объекты, которые в отличие от реальных объектов характеризуются не бесконечным, а вполне определенным числом свойств. Например, такие идеальные объекты, как материальные точки, с которыми имеет дело механика, обладают очень небольшим числом свойств, а именно, массой и возможностью находиться в пространстве и времени. Идеальный объект строится так, что он полностью интеллектуально контролируется.

Результатом эмпирического уровня являются научные факты, определенная суммативность знания, совокупность эмпирических обобщений. На теоретическом уровне знания фиксируются в форме законов, теорий, теоретических систем и системных законов.

Однако, несмотря на отмеченные различия, эмпирический и теоретический уровни органически связаны друг с другом в целостной структуре научного познания. Теоретический уровень существует не сам по себе, а опирается на данные эмпирического уровня. Но существенно то, что и эмпирическое знание неотрывно от теоретических представлений; оно обязательно погружено в определенный теоретический контекст.

Кроме эмпирического и теоретического в структуре научного знания можно выделить еще один уровень, содержащий общие представления о действительности и процессе познания - *уровень философских предпосылок, философских оснований*. Например, известная дискуссия Бора и Эйнштейна по проблемам квантовой механики, по сути, велась именно на уровне философских оснований науки, поскольку обсуждалось, как соотнести аппарат квантовой механики с окружающим нас миром. Эйнштейн считал, что вероятностный характер предсказаний в квантовой механике обусловлен тем, что квантовая механика неполна, поскольку действительность полностью детерминистична. А Бор считал, что квантовая механика полна и отражает принципиально неустранимую вероятность, характерную для микромира.

Определенные идеи философского характера вплетены в ткань научного знания, воплощены в теориях. Теория из аппарата описания и предсказания эмпирических данных превращается в знания тогда, когда все ее понятия получают онтологическую и гносеологическую интерпретацию. Иногда философские основания науки ярко проявляются и становятся предметом острых дискуссий (например, в квантовой механике, теории относительности, теории эволюции, генетике и т.д.). В то же время в науке

существует много теорий, которые не вызывают споров по поводу их философских оснований, поскольку они базируются на философских представлениях, близких к общепринятым.

Необходимо отметить, что не только теоретическое, но и эмпирическое знание связано с определенными философскими представлениями. На эмпирическом уровне знания существует определенная совокупность общих представлений о мире (о причинности, устойчивости событий и т.д.). Эти представления воспринимаются как очевидные и не выступают предметом специальных исследований. Тем не менее, они существуют, и рано или поздно меняются и на эмпирическом уровне.

Большое значение для ученых, особенно для теоретиков, имеет философское осмысление сложившихся познавательных традиций, рассмотрение изучаемой реальности в контексте картины мира. Обращение к философии особенно актуально в переломные этапы развития науки. Великие научные достижения всегда были связаны с выдвижением философских обобщений. Философия содействует эффективному описанию, объяснению, а также пониманию реальности изучаемой наукой. Важные особенности научного знания отражает понятие «стиль научного мышления». М. Борн писал так: «... Я думаю, что существуют какие-то общие тенденции мысли, изменяющиеся очень медленно и образующие определенные философские периоды с характерными для них идеями во всех областях человеческой деятельности, в том числе и в науке. Паули в недавнем письме ко мне употребил выражение «стили»: стили мышления - стили не только в искусстве, но и в науке. Принимая этот термин, я утверждаю, что стили бывают и у физической теории, и именно это обстоятельство придает своего рода устойчивость ее принципам». Известный химик и философ М. Полани показал в конце 50-х годов нашего века, что предпосылки, на которые ученый опирается в своей работе, невозможно полностью вербализировать, т.е. выразить в языке. «То большое количество учебного времени, которое студенты-химики, биологи и медики посвящают практическим занятиям, свидетельствует о важной роли, которую в этих дисциплинах играет передача практических знаний и умений от учителя к ученику. Из сказанного можно сделать вывод, что в самом центре науки существуют области практического знания, которые через формулировки передать невозможно». Знания такого типа Полани назвал неявными. Эти знания передаются не в виде текстов, а путем непосредственной демонстрации образцов. Термин «менталитет» применяется для обозначения тех слоев духовной культуры, которые не выражены в виде явных знаний, но тем не менее, существенно определяют лицо той или иной эпохи или народа. Но и любая наука имеет свой менталитет, отличающий ее от других областей научного знания, но тесно связанный с менталитетом эпохи.

В научном познании формируются и приобретают относительную самостоятельность такие основные *формы и способы познания*, как проблема, гипотеза и теория.

Проблема – это форма и способ научного познания, соединяющая в себе два элемента: знание о незнании и предвидение возможности открытия нового. Проблема – это форма выражения необходимости развития знания, которая отображает противоречие между знанием и действительностью или противоречие в самом познании. Постановка проблемы – это выход из сферы уже изученного в сферу того, что необходимо изучить. Это поиск ответа на вставший перед исследователем вопрос в условиях отсутствия необходимой и достаточной информации для этого. Проблема включает в себя новое знание, но оно имеет характер допущений и наряду с истинными положениями может содержать также заблуждения. Это этап зарождения новых знаний, который имеет активный, поисковый характер. Это также начальный этап становления научной теории. Проблема – источник развития теории, поиска путей ее использования для решения практических задач, а также выявления границ ее применения. Развитие познания можно представить как переход от постановки одних проблем к их решению, а затем к постановке новых проблем и их дальнейшего решения.

И при постановке проблеме и, еще больше, при ее решении требуются факты. Фактов обычно очень много и при желании их всегда можно собрать или подобрать. Понятие факта неоднозначно. В обыденном сознании, и не только, факты привычно отождествляются с вещами, благодаря соответствию с которыми знание якобы и становится истинным. Но в действительности все выглядит несколько иначе. *Факты* не вещи сами по себе, а их адекватное, точное воспроизведение в нашем сознании. То есть это тоже знание, знание реального положения вещей. Доступный нам мир поэтому состоит не из вещей, а из фактов.

Факты, как принято говорить, - упрямая вещь. Их приходится принимать, нравятся они нам или нет, двигают или, наоборот, тормозят наше исследование. Но это, однако, не означает их пригодности для всех возможных случаев. Факты должны быть релевантны конкретной ситуации. Иначе они не проясняют предмета исследования, не могут служить доказательством его истинности. На тенденциозно подобранных фактах можно строить все что угодно.

Именно проблема вскрывает релевантность фактов, устанавливает границы их применимости. Проблема содержит в себе определенную стратегию отыскания и сбора фактов. В этом смысле проблема служит объединению фактов в некую определенную систему, в одно целое. Благодаря проблеме мы впервые начинаем понимать факты, которые всегда, казалось бы, лежали перед нами.

Гипотеза – это форма и способ научного познания, с помощью которой формируется один из возможных вариантов решения проблемы. Гипотеза является формой развития научного познания, способом перехода от неизвестного к известному, от незнания к знанию, от неполного, неточного знания к знанию более полному, точному. Выдвижение гипотезы - это, прежде всего, ответ на требование объяснить собранные факты. Необходимо отметить, что одни и те же факты могут быть объяснены различным образом,

разными гипотезами. Как правило, выдвигается несколько гипотез, а затем начинается их выбраковка - устранение неподходящих. После всестороннего и критического рассмотрения оставляется одна («выжившая») гипотеза для дальнейшего продвижения исследования. При прочих равных условиях, предпочтение отдается той, которая достигает своей цели самым ясным, простым и экономичным образом. В самом общем виде гипотеза может быть определена как догадка, предположение о существовании некоторых вещей или о законе, который может объяснить связь вещей, уже известных как существующие. В нашем же случае под гипотезой будем понимать положение или систему положений, выдвигаемых для объяснения ранее собранных фактов, а также для следствий, которые в дальнейшем будут извлечены из них.

Никто еще не открыл метод выдвижения гипотез. Гипотезы - это работа нашего продуктивного воображения. Хороший ученый, также как и поэт, рождается, а не делается. В то же время ни один даже самый талантливый ученый не может игнорировать факты. Направляемое аккуратным следованием фактам, научное воображение старается изобрести некий закон или принцип, который позволит дать им объяснение.

Гипотеза имеет какую-то ценность только тогда, когда ее можно подтвердить или опровергнуть. В этом плане из гипотезы принято дедуцировать определенные следствия или выводы и сопоставлять их с фактами. Вообще-то есть множество требований, которым гипотеза обязана удовлетворять. Гипотеза, несомненно, должна подтверждаться теми фактами, которые с ее помощью собрали, с которыми собственно и связана стартовая проблема исследования. Но этого мало, вернее, это только начало. Далее, выдвинутая гипотеза не должна противоречить фундаментальным законам природы - разумеется, известным науке. Речь идет именно о фундаментальных законах. Их действительно нельзя трогать. Но на достаточно революционные гипотезы («сумасшедшие идеи» в науке) это требование не распространяется. Абсолютно непогрешимых законов, даже среди фундаментальных, не бывает. Но главное подтверждение гипотеза находит в тех фактах, экспериментальных данных например, которые с ее помощью (как следствия или выводы из гипотезы) были предсказаны, о существовании которых мы ничего не знали да ее выдвижения.

Важно учитывать все факты, нельзя пренебрегать ни одним из них. А соблазн ограничиться только работающими на гипотезу, т.е. однозначно подтверждающими, фактами всегда есть. Они, как ни странно, всегда находятся. Но ученый должен любить истину больше, чем гипотезу, какой бы привлекательной и интересной она для него ни была. Здесь важно принять во внимание то обстоятельство, что подтверждения, сколь бы много их ни было, не способны превратить гипотезу в незыблемую истину, но достаточно одного опровержения, т. е. негативного факта, чтобы ее, эту гипотезу, подорвать и забраковать. Испытание гипотез негативными фактами принято называть фальсификацией, соответственно позитивными фактами, подтверждениями - верификацией. Верификация страдает всегда неполной и

экстенсивностью индукции: подтверждающие примеры часто физически невозможно перебрать. В отличие от верификации, фальсификация интенсивна, она экономит силы и позволяет, если контрпримеров не находится, настаивать, с достаточной степенью уверенности, на истинностной состоятельности гипотезы.

Теория – это наиболее адекватная форма научного познания, система достоверных, глубоких и конкретных знаний о действительности, которая имеет строгую логическую структуру и дает целостное, синтетическое представление о закономерностях и сущностных характеристиках объекта. Теория в отличие от гипотезы является знанием достоверным, истинность которого доказана и проверена практикой. Она дает истинные знания и объяснения определенной сферы объективной реальности, дает возможность понять ее всеобщие, необходимые, существенные, внутренние закономерные свойства и связи. Теория характеризуется системной, логической организацией и своим объективным содержанием. Она имеет соответственно и свои познавательные функции. Теория дает возможность понять объект познания в его внутренних связях и целостности, объясняет многообразие существующих фактов и позволяет предвидеть новые, еще неизвестные, прогнозировать поведение систем в будущем. Две важнейшие функции теории – *объяснение* и *предсказание*.

Итак, процесс научного исследования предстает перед нами в виде следующей стадиальной цепочки: формулировка или постановка проблемы - проблемно ориентированное собирание фактов - выдвижение и проверка гипотезы - теория как доказанная гипотеза.

Рассматривая специфику научного познания, необходимо охарактеризовать основные методы, которые в нем используются. Но прежде всего нужно выяснить содержание понятий «*научный метод*» и «*методология*». В широком смысле, *метод* - это способ организации средств (инструментов, приемов, операций и др.) теоретической и практической деятельности. Любое разумное действие подчиняется некоторым регулятивным принципам, от выбора которых существенно зависит результат деятельности. Любая форма деятельности опирается на некоторые методы. Метод оптимизирует деятельность человека, вооружает человека наиболее рациональными способами организации его деятельности.

Понятие метода тесно связано с понятием методологии. *Методология* - это учение о закономерностях, которым подчиняется метод деятельности, о происхождении, сущности методов, их эффективности. Методология призвана выработать принципы создания наиболее совершенных методов в каждой форме деятельности.

Научное познание - это особая форма человеческой деятельности. И, как каждая деятельность, познание также опирается на определенный набор средств деятельности, средств познания, научный метод.

Научный метод - это организация средств познания (приборов, инструментов, приемов, предметных и теоретических операций и др.) для достижения научной истины, система регулятивных принципов

познавательной деятельности. Научный метод рационализирует и оптимизирует научное познание. Он, по словам Ф. Бэкона, подобен фонарю, освещающему дорогу бредущему в темноте путнику. Только верный метод может привести к получению истинного знания, подлинной картины познаваемого предмета.

Метод объединяет теорию и практику. Это возможно в силу того, что в научном методе аккумулирован обобщенный практикой исторический опыт познания мира. Такой опыт и позволяет методу направлять построение научной теории.

Обычно методы подразделяются на *эмпирические* и *теоретические* соответственно двум основным уровням научного познания. Кроме того, методы научного познания подразделяются на три группы: специальные методы, применимые в отдельных науках; общенаучные методы и универсальные, которые характеризуют человеческое мышление в целом.

Исходным методом эмпирического познания является *наблюдение*. Это целенаправленное изучение предметов, опирающееся на ощущение, восприятие и представление, в ходе которого получается знание о внешних сторонах, свойствах, признаках объектов. Особо различают качественное наблюдение и количественное. Последнее предполагает измерение как процесс определения отношения одной величины к другой, принятой за единицу. Качественное наблюдение предшествует количественному.

Более активным методом эмпирического познания является *эксперимент*. Он представляет собой активный целенаправленный метод изучения явлений в точно фиксируемых условиях, которые могут воссоздаваться и контролироваться исследователем. Итогом эксперимента является фактуальное знание и эмпирические закономерности. Различают исследовательский эксперимент, проверочный, воспроизводящий, изолирующий, и т.д. Эксперимент называется решающим, если проводится с целью опровержения одной и подтверждения другой гипотезы. При мысленном эксперименте все условия и сам объект являются воображаемыми, но воображение при этом строго регулируется известными законами науки и правилами логики.

Кроме названных специфических методов эмпирического уровня, применяются также *общенаучные методы*. К ним относятся: анализ и синтез, индукция и дедукция, абстрагирование, идеализация, моделирование.

Анализ - это прием разложения объекта на составные части с целью их самостоятельного изучения. *Синтез* - противоположная анализу операция объединения выделенных ранее частей с целью получения нового знания о целом.

При *индукции* мысль движется от знания частного к знанию общего, от фактов к законам, от одних законов к более общим законам. Индукция проблематична и не дает достоверного знания, за исключением полной индукции. Противоположной индукции является *дедукция,* в ней мысль движется от общего к частному знанию. Дедукция дает достоверное знание. Индукция и дедукция органически связаны. Индукция дает гипотезы,

дедукция позволяет проверять гипотезы путем выведения из них логических следствий, сопоставимых с фактами.

Абстрагирование есть процесс мысленного выделения отдельных составляющих предмета и одновременно отвлечения от других составляющих, которые в данном отношении несущественны. Различают абстракцию отождествления, аналитическую абстракцию (изолирующую), другие.

Идеализация - это процесс предельного отвлечения от всех реальных свойств предмета с одновременным введением в содержание образуемых понятий признаков, нереализуемых в действительности. В ходе идеализации образуется так называемый идеальный объект, которым можно оперировать на теоретическом уровне для выявления сущности, глубинных законов реальных предметов.

Моделирование - это такой метод, при котором один объект замещается другим, находящимся в отношении подобия к первому (первый - оригинал, второй - модель). Моделирование применяется там, где изучение оригинала невозможно или затруднительно, например, связано с большими расходами и риском. Выделяют две группы моделей: материальные и идеальные. Материальные модели – это физические объекты, которые подчиняются в своем функционировании природным закономерностям. Идеальные – фиксируются в соответствующей знаковой форме и функционируют по законам логики. К идеальным моделям относятся результаты логико-математического и информационного моделирования, которые осуществляются средствами математики и кибернетики. С возникновением новых поколений ЭВМ в науке все большее распространение получает компьютерное моделирование. Метод моделирования чрезвычайно расширяет возможности научного познания, поскольку позволяет наглядно представить исследуемые явления, устранить влияние сопутствующих посторонних факторов, т.е. исследовать объекты в «чистом виде».

К методам *теоретического* уровня научного познания относятся: аксиоматический, гипотетико-дедуктивный, метод восхождения от абстрактного к конкретному и единства логического и исторического.

Аксиоматический метод – это метод теоретического исследования и построения научной теории, согласно которому некоторые утверждения принимаются в качестве исходных аксиом, а все иные положения выводятся из них посредством определенных логических правил.

К системе знания, которая строится на основе аксиоматического метода, предъявляются такие требования: 1) требование непротиворечивости, согласно которому в системе аксиом не может быть однозначно выведено какое-либо положение одновременно с его отрицанием; 2) требование полноты, согласно которому любое положение, которое формулируется в данной системе аксиом, можно доказать или опровергнуть в данной же системе; 3) требование независимости аксиом, согласно которому любая аксиома системы не может выводиться из других аксиом этой же системы. Аксиоматично построенная теория может быть истинной лишь в том случае,

если истинны и сами аксиомы и те правила, по которым получены все остальные положения теории. Аксиоматический метод способствует: 1) точному определению научных понятий и соответственному их использованию; 2) точному и четкому рассуждению; 3) упорядочению знания, исключению из него лишних элементов, устранению противоречий и многозначности терминов.

Гипотетико-дедуктивный метод – это метод научного исследования, который состоит в выдвижении гипотез о причинах исследуемых явлений и выведении из этих гипотез выводов путем дедукции. Если полученные результаты соответствуют всем фактам, данным в гипотезе, то эта гипотеза признается достоверным знанием. Гипотетико-дедуктивный метод является важной составной частью методологии научного познания, он дает возможность проверить любую научную гипотезу в составе гипотетико-дедуктивной теории. Впрочем, эмпирическое подтверждение гипотезы еще не гарантирует ее истинности, а отрицание одного из ее следствий еще не свидетельствует о ее ошибочности в целом. Знакомство с общей структурой гипотетико-дедуктивного метода дает возможность определить его как сложный комплексный метод познания, который включает в себя все многообразие форм и методов научного познания, и направленный на открытие и формулировку законов, принципов, теорий.

Метод единства исторического и логического в познании. В основе исторического подхода к исследованию лежит изучение реальной истории, разнообразия ее проявлений. Логический подход раскрывает логику развития реальной истории, бытия. Логический и исторический методы выступают как приемы построения теоретических знаний о природной и социальной действительности, отображают одни и те же процессы и поэтому совпадают по содержанию, но отличаются по форме. Единство исторического и логического позволяет рассматривать объект или процесс в единстве прошлого, настоящего и будущего.

Метод восхождения от абстрактного к конкретному. Каждый объект имеет большое количество свойств и отношений. Теоретическое познание начинается с изучения отдельных сторон объекта (абстрактного), а заканчивается воссозданием полного теоретического образа объекта (конкретного). Этот метод используется как в естествознании, так и в обществоведении.

В реальном научном познании все методы могут проявлять себя одновременно, во взаимодействии.

Вопросы для самоконтроля

1. Назовите и охарактеризуйте формы эмпирического познания.
2. Что такое «метод» и «методология»?
3. Какие существуют подходы к классификации научных методов?
4. Какие частно-научные, общенаучные и всеобщие методы познания вам известны?
5. Назовите важнейшие методы эмпирического познания?

6. Чем отличается эксперимент от наблюдения?
7. Что такое решающий эксперимент и какую он играет роль в научном познании?
8. Дайте определения основным формам теоретического познания. Приведите известные Вам из истории науки примеры проблем, гипотез, теорий.
9. Что называется научными фактами?
10. Что такое гипотеза? Каким требованиям она должна удовлетворять?
11. Имеют ли процедуры подтверждения и опровержения гипотезы одинаковый познавательный статус?
12. Что такое теория? Какие виды теорий вам известны?
13. Что такое развитая научная теория?
14. Является ли неопровержимость теории свидетельством ее истинности?
15. Нагружено ли эмпирическое знание теоретическим?
16. В чем состоит смысл идеализаций?
17. Каковы характерные черты аксиоматического и гипотетико-дедуктивного методов построения научных теорий?

Список рекомендуемой литературы

Основная

1. История и философия науки: учебник /под ред. А.С. Мамзина, Е.Ю. Сиверцева. -2-е изд. перераб. и доп. М.: Юрайт, 2016.-360 с.
2. Лебедев С. А.Методы научного познания: Учебное пособие / С.А. Лебедев. - М.: Альфа-М: НИЦ ИНФРА-М, 2014. - 272 с. .
3. Мокий, М. С. Методология научных исследований : учебник для магистров/ М. С. Мокий, А. Л. Никифоров, В. С. Мокий ; под ред. М. С. Мокия. — М.: Изд.-тво Юрайт, 2015. — 255 с.
4. Новиков А.С. Философия научного поиска. М.: Либроком, 2014. – 330 с.
5. Никифоров А. Л.Философия и история науки: Учебное пособие / А.Л. Никифоров. - М.: НИЦ ИНФРА-М, 2014. - 176 с.

Дополнительная

6. Заблуждающийся разум? Многообразие вненаучного знания/Отв. Ред. И.Т. Касавин – М.: Политиздат, 1990.-404 с.
7. История информатики и философия информационной реальности: Учебное пособие для вузов/Под ред. Р.М. Юсупова, В.П, Котенко.-М.: Академический проект, 2007- 429 с.
8. Кошарный В.П. Философия науки и техники: учебное пособие. Пенза, 2012.- 294 с.
9. Микешина Л.А. Диалог когнитивных практик. Из истории эпистемологии и философии науки. М.: РОССПЭН, 2010.-575 с.
10. Современная философия науки: Хрестоматия/Сост. А.А. Печенкин. – М., 1994.- 252 с.

11. Прытков В.П. Философские проблемы науки и техники: учебное пособие/ В.П. Прытков.- Екатеринбург: Изд-во Урал. Ун-та, 2013.-63 с.8

12. Степин В.С. История и философия науки: учебник. М.: Академический проект. 2011.-423 с.
13. Степин В. С. Философия науки и техники / В. С. Степин, В. Г. Горохов, М. А. Розов. – М., 1995.- 372 с.
Философия науки: общие проблемы познания. Методология естественных и гуманитарных наук: хрестоматия / отв. ред.-сост. Л.А Микешина. — М. : Прогресс-Традиция : МПСИ : Флинта, 2005. - 992 с.
14. Философия и методология науки /Под ред. В. И. Купцова и С. В. Девятовой. – М., 1996.- 551 с.
15. Шаповалов В.Ф. Философия науки и техники: Учебное пособие. – М.: ФАИР-ПРЕСС, 2004. – 320 с.

Тема 4.Особенности современного этапа развития науки

1. Проблема детерминизма в современной науке и философии.
2. Глобальный эволюционизм и синергетика.
3. Природа теоретических объектов науки и их соотношение с объективной действительностью.
4. Осмысление социальных и внутринаучных ценностей как условие современного развития науки.
5. Ценностные ориентации ученого и этические проблемы науки XXI в.

1 Проблема детерминизма в современной науке и философии

Детерминизм – учение о всеобщей закономерной связи и взаимообусловленности всех явлений. В философии детерминистические концепции описываются с помощью категорий причина и следствие, необходимость и случайность, возможность и действительность, функциональная зависимость, связь состояний (в физике) и др. Идеи детерминизма появляются уже в античной философии (Демокрит). Дальнейшее развитие и обоснование детерминизм получает в естествознании и философии Нового времени (Бэкон, Декарт, Ньютон, Лаплас, Спиноза и др.).

Известен вариант детерминизма, который был распространен в классической науке и представление о котором наиболее отчетливо сформулировано известным французским физиком и математиком П. Лапласом (1749–1827) в работах «Опыт философии теории вероятностей» и «Аналитическая теория вероятности» и получило название лапласовского детерминизма. Значение координат и импульсов всех частиц во Вселенной в данный момент времени, с его точки зрения, совершенно однозначно определяет ее состояние в любой прошедший или будущий момент. Случайному как объективному явлению места нет. Только ограниченность наших познавательных способностей заставляет рассматривать отдельные

события как случайные. Исключая случайность из категориального аппарата теории, Лаплас абсолютизировал необходимость как единственно возможное основание фундаментальных теорий типа классической механики.

В соответствии с уровнем развития естествознания этого времени детерминизм находит свое выражение в такой форме всеобщей связи и взаимной обусловленности, которая описывается динамическими закономерностями. Динамические закономерности выражают строго однозначную обусловленность изменений одних элементов другими, при которой данное состояние системы однозначно определяет ее последующее состояние, и описывают их абсолютно точно в форме связи вполне определенных физических величин. Лаплас допускал существование объективной случайности только в эмпирическом материале, признавал эвристическую ценность вероятностных методов исследования. Но он считал, что теория должна исключать из своих построений случайность; зная в любой момент координаты и скорость каждой ее части, можно точно предсказать дальнейшее развитие системы. Траектория любого объекта (эволюция любой ситуации) однозначно определяется начальными условиями.

В механистической детерминистической концепции предполагалось, что для поведения каждой частицы, каждого элемента имеется только одна с необходимостью осуществляющаяся возможность. Понятый таким образом детерминизм ведет к фатализму, принимает мистический характер и фактически смыкается с верой в божественное предопределение. Во второй половине XIX в. наука приступила к изучению статистических закономерностей. Статистические закономерности выражают такие связи, когда данное состояние системы определяет все ее последующие состояния не однозначно, а лишь с определенной вероятностью, являющейся объективной мерой возможности реализации заложенных в прошлом тенденций изменения.

Осознание ограниченности механистического детерминизма при объяснении явлений на рубеже XIX–XX вв. привело к формированию философского и естественно-научного индетерминизма. Индетерминизм полностью или частично отрицает существование причинно-следственных связей и возможность их детерминистического объяснения. Однако развитие науки и философии в XX в. показало необходимость сохранения принципа детерминизма и его дальнейшего развития.

Важно понять, что современная наука фиксирует открытость систем, возможность реализации множества тенденций развития, заложенных в прошлых состояниях систем, возникновение в процессе развития возможностей и тенденций качественно новых состояний. То есть всякий достаточно сложный процесс развития подчиняется статистическим закономерностям, т.к. динамические закономерности являются лишь приблизительным выражением отдельных этапов этого процесса. Различие динамических и статистических закономерностей относительно, т.к. всякая динамическая закономерность представляет собой статистическую

закономерность с вероятностью осуществления, близкой или равной единице.

С расширением пространственно-временных интервалов развития связь между предшествующими и последующими состояниями системы все в большей степени подчиняется законам вероятностной детерминации. Приведите примеры тех областей знаний, где невозможно обойтись без анализа статистических закономерностей. В отличие от классической науки, стремившейся сводить все к простому и предсказуемому, современная наука работает с непредсказуемым, неопределенным, неточным и сложным, широко использует вероятностные методы и признает важную роль случайного и непредсказуемого. В ближайшем будущем, по-видимому, науку ожидает расширение и переосмысление многих классических понятий.

Таким образом, развитие науки в течение последних ста лет привело к тому, что представления о детерминизме становятся все более сложными и гибкими, ученые осознают ограниченность классического физического детерминизма, стремятся снять противопоставление необходимости и случайности.

Одной из разновидностей детерминации является целевая детерминация (teleos – цель); принцип «конечных причин» (causa finalis), согласно которому идеально постулируемая цель, конечный результат, оказывает объективное воздействие на ход процесса, принимает различные формы в различных телеологических концепциях.

В современной науке сформировался целевой подход, суть которого заключается в том, что научное исследование обращается к конечной стадии, результату процесса как его цели, отправляясь от которой аналитически устанавливаются причины по их следствию. Особое внимание, которое современная наука уделяет целевой детерминации, обусловлено рядом новых открытий в физике, космологии.

В связи с этим в науке возникла своеобразная телеологическая проблема. Она состоит в необходимости объяснить чрезвычайно высокую и тонкую взаимосогласованность ряда фундаментальных свойств и характеристик нашей Вселенной. Выясните содержание так называемого *антропного принципа*, обсуждаемого в современной науке. Антропный космологический принцип несет определенную философскую нагрузку – вызывает различные мировоззренческие интерпретации. В мировоззренческом плане антропный принцип воплощает в себе идею взаимосвязи человека и универсума, высказанную еще в античности.

2. Глобальный эволюционизм и синергетика

Становление эволюционных идей имеет достаточно длительную историю. Уже в XX в. они нашли применение в геологии, биологии и других областях знаний, но воспринимались, скорее, как исключения по отношению к миру в целом. Вплоть до конца XX в. принцип эволюции не был доминирующим в естествознании. Во многом это было связано с тем, что

лидирующей научной дисциплиной была физика, которая на протяжении большей части истории в явном виде не включала в число своих фундаментальных постулатов принцип развития.

Наука второй половины XX в. ликвидировала противоположность биологии и физики в понимании эволюции. Выяснилось, что процессы усложнения организации присущи не только биологическим системам, но и системам неорганической природы (концепция эволюции Вселенной Фридмана и Хаббла, неравновесная термодинамика Пригожина, идея самоорганизации в кибернетике Винера и Эшби). Эволюция затрагивает не только макроскопические тела, но и мир элементарных частиц, основные типы физических взаимодействий. Таким образом, идея развития эволюции приобретает глобальное космическое значение, пределы применимости ее расширились от объектов микромира до метагалактики. Это привело к формированию концепции глобального эволюционизма как системы представлений о всеобщем процессе развития природы во всех его многообразных естественно-исторических формах: социальной и биологической эволюции, эволюции Земли, солнечной системы, Вселенной. Она обеспечивает экстраполяцию эволюционных идей, получивших обоснование в биологии, астрономии и геологии, на все сферы деятельности и исследования неживой, живой и социальной реальности как единого универсального эволюционного процесса. Утверждается мысль, что Вселенная – это не механизм, однажды заведенный Внешним Наблюдателем (Разумом), судьба которого определена раз и навсегда, а непрерывно развивающаяся и самоорганизующаяся система; человек не просто активный внутренний наблюдатель, а действующий элемент системы.

В обоснование глобального эволюционизма свою лепту внесли многие естественно-научные дисциплины. Но определяющее значение в его утверждении сыграли три важнейших концептуальных направления в науке XX в.: теория нестационарной Вселенной, синергетика, теория биологической эволюции и развитая на ее основе концепция биосферы и ноосферы.

Существенно важным для становления концепции глобального эволюционизма явилось исследование механизмов самопроизвольного возникновения упорядоченных структур в открытых нелинейных системах, что привело к формированию нового научного направления – синергетики.
Проблемное поле синергетики вращается вокруг понятий «неустойчивость», «нестабильность», «неравновесность», «хаос», «случайность» и т.п. Одной из важных идей, которую синергетика вносит в современную науку и картину мира, является идея необратимости и нелинейности. Она открывает необычные стороны мира: его нестабильность и режимы с обострением (режимы гиперболического роста, когда характерные величины многократно, вплоть до бесконечности, возрастают за конечный промежуток времени), нелинейность и открытость (различные варианты будущего), возрастающую сложность формообразований и способов их объединения в эволюционирующие целостности (законы коэволюции). Она дает

возможность шире взглянуть на процессы развития и глобальной эволюции и сформировать основные принципы современной концепции самоорганизации.

На основе этих исследований формируется ныне новый образ мира – открытого и сложноорганизованного; мира, который является не ставшим, а становящимся, не просто существующим, а непрерывно возникающим миром. Понятия «бытие» и «становление» объединяются в одни понятийные рамки, идея эволюции органично входит не только в науки о живом, но и в физику и космологию. Современная наука окончательно разбивает миф о жестко детерминированной и безвременной Вселенной. Мир рассматривается как процесс, как последовательность деструктивных и креативных процессов, в котором важную роль играют как детерминистические, так и стохастические процессы. Он полон неожиданных поворотов, связанных с выбором путей дальнейшего развития.

Введение нестабильности, неустойчивости, открытие неравновесных структур – важная особенность постнеклассической науки.

3 Природа теоретических объектов науки и их соотношение с объективной действительностью

Совсем не прост вопрос о природе теоретических объектов современной науки и их соотношения с объективной действительностью. Одной из особенностей современной науки является ее теоретизация.
«Картина», создаваемая наукой, – это «изображение» искусственно сконструированных идеальных объектов и экспериментально-измерительных процедур, интерпретируемых как данные непосредственно эмпирического опыта. Она изменяется по мере развития познания и практики, и ее можно рассматривать в качестве теоретической модели исследуемой реальности. Таким образом, определяющая роль в формировании теории принадлежит идеализированному объекту – теоретической модели существенных связей реальности, представленных с помощью гипотетических допущений и идеализаций.

Наука XX в., особенно современная теоретическая физика, предметом исследования которой становятся объекты, далекие от нашего макроскопического опыта, сделала актуальным вопрос о существовании и реальности этих объектов. Эта проблема является одной из наиболее острых в современной философии науки. Например, проблема существования виртуальных частиц, кварков, тахионов и других объектов, которые были введены чисто теоретически, но претендуют на включение в физическую картину мира.

Классическая наука исходила из того, что существует только одна изучаемая ею реальность. Соответственно, может существовать только одна истина, относящаяся к этой реальности. Между тем сегодня ясно, что существует вовсе не одна, а много разных реальностей. Это не только та реальность, с которой имеет дело наука, но и реальность повседневной

жизни, обыденного знания. Есть субъективная реальность: то, что я сейчас имею конкретные переживания или определенные мысли, – это реальные факты моего сознания. Есть реальность идеальных объектов культуры: научных и философских теорий, произведений искусства – то, что Поппер называет «третьим миром». Есть реальность межиндивидуальных отношений, коммуникаций (с нею иногда связывают мир интерсубъективности).

Наверное, можно говорить сегодня о появлении виртуальной реальности как особого типа межчеловеческой коммуникации с помощью компьютера. Возможно, есть и какие-то еще типы реальностей.

В современной эпистемологии осознается тот факт, что природа исследуемой реальности связана не только с объективной реальностью самой по себе, но и с деятельностью человека. При анализе понятия «реальность» учитывают важную роль, которую играют в формировании содержания данного понятия конструктивно-теоретическая деятельность и практическиэкспериментальная деятельность человека как субъекта познания. Субъект познания не устраним из содержания физической реальности.

Значительно различается понимание реальности в классической и современной науке. Каждая физическая теория изучает определенный вид физической реальности при специфических ее упрощениях, огрублениях, идеализациях, неизбежных в процессе отображения этой реальности. Поэтому законы теории лишь опосредованно и с определенной степенью точности относятся к объективной реальности.

Таким образом, реальность в научном исследовании – это не объективная реальность в смысле денотата философской категории материи, но, скорее, когнитивное образование, содержание которого наполняется объективно-реальными факторами и, вместе с тем, субъективными факторами – теоретической и экспериментальной деятельностью исследователя.

Исследуемая реальность, как утверждает современная наука, имеет множество теоретических описаний. Это обусловлено многоаспектностью, многокачественностью самой действительности, что создает объективные предпосылки для этого. Вполне понятно, что, когда естествознание было занято накоплением эмпирической информации, классификацией и систематизацией многочисленных новых фактов и результатов, легко возникало искушение построить такую логику открытия, с помощью которой можно было бы получать новые истины в науке чуть ли не чисто механическим путем. В 30–40-е гг. XIX в. начинается критика концепций логики открытия, которые были распространены в XVII и XVIII вв., и постепенно происходит переход к новой концепции методологии научного исследования – к гипотетико-дедуктивной ее модели. Согласно такой модели, логика не играет какой-либо роли в процессе генерирования новых научных идей, гипотез и теорий, ее задача сводится лишь к дедукции

следствий из гипотез, найденных нелогическим путем, и проверке их с помощью данных наблюдения и эксперимента.
Важнопонять, что теории современной науки создаются не просто путем индуктивного обобщения опыта (хотя такой путь не исключается), а за счет первоначального движения в поле ранее созданных идеализированных объектов, которые используются в качестве средств конструирования гипотетических моделей новой области взаимодействий. Обоснование таких моделей опытом превращает их в ядро будущей теории.

Недооценка, а тем более полное игнорирование творческой активности субъекта в познании, стремление «изгнать» из процесса познания эту активность закрывают дорогу к истине, к объективному отражению реальности. Научное исследование – не монолог, а диалог с природой. А это значит, что «активное вопрошание природы» есть лишь неотъемлемая часть ее внутренней активности. Тем самым объективность в современной науке обретает более тонкое значение, ибо научные результаты не могут быть отделены от исследовательской деятельности субъекта. И это не отход от объективности, а все более полное приближение к ней, ибо она открывается только в процессе активной деятельности людей.

Многие исследователи считают, что мало связать субъективность с объективностью. Нужно еще наполнить субъективность социальным содержанием. А это значит, что в проблемное поле гносеологии должны войти проблемы культурной детерминации объектов познания, проблемы трансляции знаний и коммуникации между субъектами познания, многосложные взаимозависимости между деятельностью людей и ее культурно-историческим контекстом.

Таким образом, развитие науки XX в. – как естествознания, так и обществознания – убедительно показывает, что независимого наблюдателя, способного только пассивно наблюдать и не вмешиваться в «естественный ход событий», просто не существует. Человека – «единственного наблюдателя», которого мы способны себе представить, невозможно вычленить из окружающего мира, сделать его не зависимым от его собственных действий, от процесса приобретения и развития знаний. Все это ведет к изменению представлений о содержании научной рациональности.

Процесс теоретизации современной науки тесно связан с процессом ее формализации. Метод формализации – это перевод содержательных фрагментов знания (в математике, физике, логике, химии и других науках) на искусственные символические, логико-математические и математические языки, подчиненные четким правилам построения формул и их преобразований. При формализации суждения об объектах переносятся в плоскость оперирования с символами и знаками. Потребность в формализации возникает перед той или иной наукой на достаточно высоком уровне ее развития, когда задача логической систематизации и организации наличного знания приобретает первостепенное значение.

С появлением вычислительной техники появилась возможность автоматизации этого процесса.Формализация позволяет:

- однозначно определить входные термины, уяснить существенные связи и отношения в структуре научного знания;
- вычленить и уточнить логическую структуру теории, т.е. установить исходные посылки теории, в качестве которых в математике выступают аксиомы, а в эмпирических науках – фундаментальные принципы

или законы. Точное перечисление логических правил вывода также весьма важно для выявления структуры теории; обеспечить стандартизацию используемого языка и понятийного

аппарата, которые используются в данной теории постановку новых проблем и поиск их решения.

Формализация играет важную роль:

- в выявлении и уточнении содержания научной теории;
- в систематизации той суммы знаний, которая накоплена содержательной теорией;
- в синтезе смежных наук.

Усиление процессов теоретизации и формализации научного познания органично связано с его математизацией – проникновением математических методов и языка математики в разные науки.

Роль математики в развитии познания была осознана довольно давно. Уже в античности были созданы предпосылки для становления математической программы научного исследования, которая опиралась на две фундаментальные идеи:

- −об особом месте математического знания в системе научного познания в целом;
- об органическом родстве, существенной близости собственно математического и философского знания.

Определяющей причиной математизации современной науки является переход многих ее отраслей на теоретический уровень исследования, изучение более глубоких внутренних механизмов, процессов, происходящих в природе и обществе. Конечно, математические методы применяются и на эмпирической стадии исследования при измерении и количественном сравнении исследуемых величин, для выражения целого ряда эмпирических законов (например законы Бойля–Мариотта, Гей–Люссака, Ньютона), которые устанавливают связь между эмпирически наблюдаемыми свойствами, но не объясняют причины этих свойств.

Вторая причина математизации научного знания связана с качественными изменениями в самой математике – с разработкой нового математического аппарата, который дает возможность выражать количественные и структурные закономерности объектов познания современной науки.

Важной причиной математизации современной науки является возможность использовать электронно-вычислительную технику и другие средства автоматизации некоторых сторон интеллектуальной деятельности.

Масштаб и эффективность процесса проникновения количественных методов в частные науки, успехи математизации и компьютеризации во многом связаны с совершенствованием содержания самой математики, с качественными изменениями в ней. Современная математика развивается достаточно бурно, в ней появляются новые понятия, идеи, методы, объекты исследования и т.п., что, однако, не означает «поглощения» ею частных наук. Эффективность математизации всегда основывается на глубоком анализе качественных особенностей исследуемых явлений, ибо только в таком случае возможно обнаружить качественно однородное и существенно общее в них.

4 Осмысление социальных и внутринаучных ценностей как условие современного развития науки.

Теперь обратимся к проблеме социальных и внутринаучных ценностей как условиям современного развития науки.

Наука XX в. формирует новые идеалы и нормы описания и объяснения исследуемых объектов. В классической науке идеалом объяснения и описания считалась характеристика объекта «самого по себе», без указания на средства его исследования. Современная физика в качестве необходимого условия объективности описания выдвигает требование четкой фиксации взаимодействий объекта со средствами наблюдения и учета при его описании особенностей средств наблюдения (типов измерительных устройств).

Широкое применение в постнеклассической науке приемов и методов теоретического описания уникальных, индивидуально неповторимых объектов ставит методологическую задачу анализа типов такого описания. Известная оппозиция – «либо генерализирующий, либо индивидуализирующий подход» – снимается, когда речь идет об исторических реконструкциях.

В методологии современной науки активно обсуждается проблема соотношения описания и объяснения как функций науки. С другой стороны, осознается ограниченность представлений о необходимости противопоставления функций описания и объяснения, характерных для классической науки.

Реальной проблемой методологии современной науки является проблема соотношения объяснения и понимания. Длительное время существовало противопоставление естественных и гуманитарных наук. Естествознание ориентировалось на постижение природы самой по себе, безотносительно к субъекту деятельности. Его задачей было достижение объективно истинного знания, не отягощенного ценностно-смысловыми структурами. Ученые стремились выявить и объяснить наличие причинных связей, существующих в природном мире, и, раскрыв их, достичь объективно-истинного знания, установить законы природы. Гуманитарные науки были ориентированы на постижение человека, человеческого духа, культуры. Для них приоритетное значение имело раскрытие смысла, не столько объяснение, сколько понимание.

Важно отметить, что соединение объективного мира и мира человека в современных науках – как природных, так и гуманитарных – с

неизбежностью ведет к трансформации идеалов «ценностно-нейтрального исследования». Объективно истинное объяснение и описание применительно к «человекоразмерным» объектам не только допускают, но и предполагают включение аксиологических (ценностных) факторов в состав объясняющих положений.

Одной из центральных проблем самосознания современной науки стала проблема соотношения истинности и ценности. В отличие от познавательного, ценностное отношение неизбежно включает в себя наряду с характеристикой объекта также и выражение присущих субъекту идеалов и устремлений. Ценностно-оценочный компонент в структуре познавательного образа выражает его социальность, включенность в сложную систему общественных отношений. В процессе социализации отдельной личности ценностное отношение к действительности формируется значительно раньше, чем личность активно включается в специализированную познавательную деятельность. Ценностные установки входят в предпосылочное знание, образующее своеобразный «мост» между социокультурными реалиями и содержанием научного знания. Аксиологические проблемы науки, таким образом, – это проблемы социальной, нравственной, эстетической и культурной ценностной ориентации научных исследований и их результатов.

Современная фундаментальная наука стоит перед необходимостью не только осознания отдельных последствий своих результатов, но и установления аксиологического контроля за процессом постижения истины. Тип рациональности, который формируется в постнеклассической науке характеризуется соотнесенностью знания не только со средствами познания, но и с ценностно-целевыми структурами деятельности. Наблюдается неуклонный рост интереса к социальным, человеческим, гуманистическим аспектам науки. Все шире в научный оборот внедряется понятие «этос науки», обозначающее совокупность моральных императивов, нравственных норм, принятых в данном научном сообществе и определяющих поведение ученого, складывается особая дисциплина – этика науки, укрепляются представления о необходимости соответствия научных концепций добру, благу, гармонии и т.п. Наука, как и другие формы человеческого постижения мира, такие как философия и религия, содержит интересы и ценности, отображающие культуру, человеческие чувства и устремления. Объективно истинное описание и объяснение применительно к медико-биологическим системам, объектам экологии, объектам биотехнологии (генной инженерии), системам «человек-машина» и т.п. не только допускает, но и предполагает включение аксиологических факторов в состав объясняющих положений. То есть при изучении «человекоразмерных» объектов поиск истины оказывается связанным с определением стратегий и возможных направлений практического преобразования такого объекта, что непосредственно затрагивает гуманистические ценности.

Можно говорить о двух типах ценностных ориентаций в науке:

− ценностных ориентациях науки как социального института; − ценностных ориентациях работающих в науке людей. В современной культуре существуют две диаметрально противоположные мировоззренческие позиции, в основе которых лежит одинаково абсолютизированное представление о науке: сциентизм (от лат. «сциенция» – наука) и антисциентизм. Сциентизм связан с преувеличением роли науки как способа постижения мира и социального института, с верой в безграничные возможности науки как единственной основы социального прогресса. При этом в качестве образца для сциентизма выступает не наука вообще, а, прежде всего, развитые естественные, математические науки и технические науки. Такой позиции противостоит антисциентизм – социокультурная ориентация, обосновывающая антигуманитарную сущность науки и научно-технического прогресса в его современных формах, рассматривающая ее как угрозу существованию самой человеческой цивилизации. В качестве альтернативы науке, научному познанию, в некоторых случаях даже вообще рациональному взгляду на мир, выдвигаются различного рода вненаучные или внерациональные (иррациональные) способы постижения бытия. Необходимо уяснить для себя основные формы сциентизма и антисциентизма. В последние десятилетия XX в. и в начале XXI в. происходит изменение нравственно-этической ориентации естествоиспытателей, гуманизация естественно-научного и технического знания. Рост уровня гуманитарного сознания обусловлен целым рядом обстоятельств: осознанием смертельной опасности, которую создает для человека разработка оружия массового уничтожения; обострением глобальных проблем, появлением генной инженерии; использованием химических средств с целью воздействия на структуру личности; экспериментами на человеке и т.п. Сегодня важно органическое соединение ценностей научно-технического мышления с теми социальными ценностями, которые представлены нравственностью, искусством, философским и, возможно, религиозным постижением мира. Соединение Истины, Блага и Красоты – извечная мечта гуманистически ориентированных мыслителей, в том числе и видных представителей философской мысли России. Главный путь здесь – преодоление негуманного образа науки, выработка образа науки с «человеческим лицом», понимания того, что наука – не только отражение природы и общества, но и форма самореализации личности.

5. Ценностные ориентации ученого и этические проблемы науки XXI в.

Вопрос «Ценностные ориентации ученого и этические проблемы науки XXI века» предполагает выявление ценности научного познания как особого вида деятельности (когнитивные ценности) и ценностей, которыми руководствуется ученый как личность (экзистенциональные и социальные ценности). Традиционно главная когнитивная ценность науки истина (объективное, доказанное знание). И до недавнего времени ученые были убеждены, что этика науки состоит в соблюдении таких норм научной

деятельности, как чистота проведения эксперимента, научная добросовестность в теоретических исследованиях, отрицательное отношение к плагиату, высокий профессионализм, бескорыстный поиск и отстаивание истины. Смысл соблюдения этих норм в том, что в стремлении к истине ученый не должен считаться ни со своими симпатиями и антипатиями, ни с какими бы то ни было иными привходящими обстоятельствами. Нормы научной этики редко формулируются в виде специфических перечней и кодексов. Однако известны попытки выявления, описания и анализа этих норм. Наиболее популярна в этом отношении концепция английского социолога науки Р. Мертона, представленная в работе «Нормативная структура науки» (1942). В ней Р. Мертон дает описание этоса науки, который понимается им как комплекс ценностей и норм, воспроизводящихся от поколения к поколению ученых и являющихся обязательными для человека науки. С точки зрения Р. Мертона, нормы науки строятся вокруг четырех основополагающих ценностей:

- универсализм – убежденность в том, что изученные наукой природные явления протекают повсюду одинаково, и истинность научных утверждений должна оцениваться независимо от возраста, пола, расы, авторитета, званий тех, кто их формулирует. Наука, стало быть, внутренне демократична;
- общность – научное знание должно свободно становиться общим достоянием;
- бескорыстность – стимулом деятельности ученого является поиск истины, свободной от соображений личной выгоды (славы, денежного вознаграждения и т.п.);
- организованный скептицизм – уважение к предшественникам и критическое отношение к их результатам.

В классической науке, эпицентром которой, как уже было сказано, был абстрактный идеал самоценной истины, научная истина и этические ценности (экзистенциальные и социальные ценности) были разделены непроходимой гранью. Концепция «этической нейтральности» науки стала едва ли не догмой позитивистски ориентированной философии науки, в которой разграничивается контекст открытия и обоснования и контекст познания и применения. С позиций здравого смысла науки ясно, что законы природы, выраженные математическими уравнениями, сами формализмы языка науки совершенно не зависимы от страстей, которые бушевали по поводу их поиска и обоснования, от субъективных вкусов и аффектаций теоретиков. Знание, опредмеченное в знаково-семиотических структурах, пребывает «по ту сторону добра и зла», ибо отображает объективное состояние, не зависимое ни от человека, ни от человечества. Все как будто так. Но ведь наука – это не только фиксация добытого знания, но и процесс живой продуктивной деятельности человека. Не учитывать социальное и антропологически-личностное измерение познания современная наука не может. Иначе человеческая личность неизбежно предстанет как орудийно-

инструментальный исполнитель безличной воли некоего абсолютного субъекта, природа которого совершенно неясна и иррациональна.

В формировании типа личности ученого, его поведенческих и ментальных навыков участвуют ценностные ориентации той или иной эпохи. Ученый разделяет основополагающие ценности взрастившей его культуры – гуманизм, уважение к личности, служение обществу, демократическое право каждого человека на свободу выбора, право на жизнь и т.д.

Особенностью современного, формирующегося стиля научного мышления можно считать признание принципиальной неустранимости ценностной основы познания. Осознание роли нравственного, этического начала в науке актуализирует вопрос о социальной ответственности ученого. Еще В. И. Вернадский подчеркивал, что ученые не должны закрывать глаза на возможные последствия их научной работы, научного прогресса. Они должны себя чувствовать ответственными за последствия их открытий. Особую остроту проблема нравственной ответственности приобрела в последнее время, в частности, в связи с прогрессом в области генной инженерии, поскольку это затрагивает интимные механизмы жизни.

В отличие от профессиональной ответственности, которая реализуется между членами научного сообщества, социальная ответственность реализуется во взаимоотношениях науки и общества. В реальной деятельности ученых эти формы ответственности тесно переплетены.

Постановка проблемы и требование социальной ответственности ученых – необходимость нашего времени. Но в какой степени ей могут следовать ученые в своей деятельности? Самосознание современной науки раздвоено. С одной стороны, она еще не утратила память о том, что научное исследование есть движение к истине, с другой, став «профессией», она приняла на себя все характерные черты этого рода деятельности.

Следует подчеркнуть принципиальное суждение о том, что при изучении «человекоразмерных объектов поиск истины непосредственно затрагивает и гуманистические ценности. С системами такого типа нельзя свободно экспериментировать. В процессе их исследования и практического освоения особую роль начинают играть знания запретов на некоторые стратегии взаимодействия.

Социальная ответственность ученых не есть нечто внешнее, некий довесок, который неестественным образом связывается с научной деятельностью. Напротив – это органическая составляющая научной деятельности, достаточно ощутимо влияющая на проблематику и направление исследований. Осознание этого приводит к гуманизации естественно-научного и технического знания, росту уровня гуманитарного сознания ученых. Расширение и углубление практических возможностей человека, меры его давления на биосферу, его вторжение в микромир – все это в совокупности принудительным образом побуждает науку (в том числе и посредством воздействия общественного сознания с его тревожностью) формировать новые этологические и этические ориентиры.

Итак, можно сказать, что формирование нового этоса современной науки происходит в лоне широкого процесса – процесса становления нового планетарно-экологического сознания, соответствующего практике взаимодействия цивилизации и природы в современных условиях.

Контрольные вопросы

1. Каковы особенности развития детерминизма в современной науке и философии?
2. Что такое телеологические концепции? Какие философские толкования антропного принципа Вы знаете?
3. Какое влияние вызывает синергетика на становление современного миропонимания?
3. Что такое аксиологические проблемы науки?
4. Как решается в современной науке проблема соотношения истинности и ценности?
3. Что такое сциентизм и антисциентизм в оценке роли науки в современной культуре?
5. Каковы особенности ценностных ориентаций ученого в процессе научного поиска? Что такое этос науки?
6. Чем обусловлена актуальность проблемы соотношения свободы научного поиска и социальной ответственности ученого?
7. В чем, по Вашему мнению, состоит социальная ответственность ученого?
8. В чем сущность современной концепции глобального эволюционизма?
9. Что такое теоретические объекты современной науки? Как они соотносятся с реальностью?
10. Как трансформируется в современной эпистемологии представление об объекте и субъекте познания?
3. Каковы изменения идеалов и норм познания, характерных для неклассической и постнеклассической науки?
4. Каковы особенности формализации науки? Чем обусловлены границыформализации научных знаний? В чем состоит философский смысл теорем Геделя?
5. Какие формы и методы математизации современной науки Вы знаете?
6. Какую роль играют новейшие информационные технологии в современной науке?

Основная литература

Агацци, Э. Моральное измерение науки и техники / Э. Агацци. – М.,1998.
Аршинов, В. И. Синергетика как феномен постнекласической науки /В. И. Аршинов. – М., 1999.
Вернадский, В. И. Размышления натуралиста. Научная мысль как планетарное мышление / В. И. Вернадский. – М., 1978.

Кохановский, В. П. Основы философии науки / В. П. Кохановский, Т. Г. Лешкевич, Т. П. Матяш, Т. Б. Фатхи. – Ростов-на-Дону, 2005.
Мертон, Р. Амбивалентность ученого / Р. Мертон. – М. 1965.
Пригожин, И. Порядок из хаоса / И. Пригожин, И. Стенгерс. – М., 1986.
Степин, В. С. Философия науки. Общие проблемы / В. С. Степин. – М.,2004.
Философия и методология науки. – М., 1996.
Миронов, В. В. Образы науки в современной культуре и философии /В. В. Миронов. – М., 1997.
Моисеев, Н. Н. Универсальный эволюционизм / Н. Н. Моисеев // Вопросы философии. – 1991. – № 3.
Моисеев, Н. Н. Естественно-научное знание и гуманитарное мышление /Н. Н. Моисеев // Общественные науки и современность. – 1993. – № 2.
Наука, техника, культура: проблемы гуманизации и социальной ответственности (Материалы «круглого стола») // Вопросы философии. – 1989. – № 1.
Косарева, Л. М. Социокультурный генезис науки Нового времени: философский аспект проблемы / Л. М. Косарева. – М., 1989.
Лесков, Н. С. Наука как самоорганизующаяся система / Н. С. Лесков //Общественные науки и современность. – 2003. – № 4.
Моисеев, Н. Н. Человек и ноосфера / Н. Н. Моисеев. – М., 1990.
Пригожин, И. Порядок из хаоса / И. Пригожин, И. Стенгерс. – М., 1986.
Рузавин, Г. И. Математизация научного знания / Г. И. Рузавин. – М.,1984.
Фролов, И. Т. Этика науки: проблемы и дискуссии / И. Т. Фролов,Б. Г. Юдин. – М., 1986.

Дополнительная литература

Балашев, Ю. В. «Антропные аргументы» в современной космологии /Ю. В. Балашев // Вопросы философии. – 1988. – № 7.
Глобальный эвоюционизм. Философский анализ. – М., 1994.
Канке В. А. Основные философские направления и концепции науки. Итоги XX столетия. – М., 2000.
Князева, Е. Н. Основания синергетики / Е. Н. Князева, С. П. Курдюмов. – СПб., 2003.
Лазар, М. Г. Этика науки / М. Г. Лазар. – Л., 1985.
Лэйси, Х. Свободна ли наука от ценностей. Ценности и научное понимание / Х. Лэйси. – М., 2001.
Микешина, Л. А. Философия науки / Л. А. Микешина. – М., 2005.
Проблемы методологии постнеклассической науки : сборник статей. –М., 1992.
Степин, В. С. Теоретическое знание / В. С. Степин. – М., 2000.
Степин, В. С. Эпоха перемен и сценарии будущего / В. С. Степин. – М.,1996.
Хакен, Г. Синергетика / Г. Хакен. – М., 1980.
Черникова, И. В. Философия и история науки / И. В. Черникова. – Томск,2001.

Штанько, В. И. Философия и методология науки : учебное пособие дляаспирантов и магистрантов естественно-научных и технических вузов /В. И. Штанько. – Харьков, 2002.

Тема 5.Предмет философии техники

Под *техникой* понимается *система созданных средств и орудий* производства*, и также приемы и операции, умение и искусство осуществления трудового процесса*. В технике человечество аккумулировало свой многовековой опыт, приемы, методы познания и преобразования природы, воплотило все достижение человеческой культуры. В формах и функциях технических средств отразились формы и способы воздействия человека на природу. Будучи продолжением органов человеческого тела, их усовершенствованием (глаза, мозга и т.д.), определенные технические средства в свою очередь диктуют человеку приемы и способы их применения. Техника возникает, когда для достижения цели вводятся промежуточные средства. Таким образом, техника как «производственные органы общественного человека» есть результат человеческого труда и развития знания и одновременно их средство.

Цель и функции техники – преобразовать природу и мир человека в соответствии с целями, сформулированными людьми на основе нужд и желаний. Следовательно, техника – это необходимая часть человеческого существования на протяжении всей истории. Человеческие существа могут реально воспроизводить лишь себя, и в этом самовоспроизведении индивидуальные параметры и функции остаются аналогичными, даже если они возрастают или уменьшаются экстенсивно или интенсивно. Техника не есть цель сама по себе. Она имеет ценность только как средство. По Хайдеггеру, техника не просто конструирует «технический мир», она подчиняет своему императиву едва ли не все пространство социального бытия, оказывая влияние на осмысление истории.

Человек всегда был связан с техникой; он производит и использует или потребляет продукты техники. Исторический процесс развития техники включает три основных этапа: орудие ручного труда, машины, автоматы, искусственный интеллект. Техника в своем развитии сейчас, пожалуй, начинает приближаться к человеческому уровню, двигаясь от аналогии с физическим трудом и его организации к аналогиям с ментальными свойствами человека. По словам немецкого философа А. Хунига, техника во все исторические моменты выражает людей и идею человечности данного времени.

Большую роль в осмыслении феномена техники играет философия техники. Несмотря на то, что техника является настолько же древней, как и само человечество, и хотя она так или иначе попадала в сферу интересов философов, как самостоятельная философская дисциплина *философия техники* возникла лишь в XX столетии.

В заглавии книги немецкого философа *Эрнста Каппа* мы находим словосочетание «Философия техники». Его работа «*Основные направления философии техники. К истории возникновения культуры с новой точки зрения*» вышла в свет в 1877 г. Позднее другой немецкий философ *Фред Бон* одну из глав своей книги *«О долге и добре»* (1898 г.) также посвятил «философии техники». В конце XIX века российский инженер *Петр Климентьевич Энгельмейер* формулирует задачи философии техники в своей брошюре *«Технический итог XIX века»* (1898 г.). Его работы были опубликованы также на немецком языке.

Однако можно утверждать, что лишь в XX веке техника, ее развитие, ее место в обществе и значение для будущего человеческой цивилизации становится предметом систематического изучения. Не только философы, но и сами инженеры, начинают уделять осмыслению техники все большее внимание. Особенно интенсивно эта тематика обсуждалась на страницах журнала Союза германских дипломированных инженеров «Техника и культура» в 30-е гг. Можно сказать, что в этот период в самой инженерной среде вырастает потребность философского осознания феномена техники и собственной деятельности по ее созданию. Часто попытки такого рода осмысления сводились к исключительно оптимистической оценке достижений и перспектив современного технического развития. Одновременно в гуманитарной среде возрастало критическое отношение к ходу технического прогресса современного общества, и внимание привлекалось, прежде всего, к его отрицательным сторонам. Так или иначе, в обоих случаях техника стала предметом специального анализа и исследования.

Таким образом, *философия техники* уже сравнительно давно выделилась в самостоятельную область философского исследования. Что же представляет собой *философия техники*? На этот вопрос можно ответить двояким образом*: во-первых*, определив, что особенного изучает философия техники по сравнению с другими дисциплинами, изучающими технику, и, *во-вторых*, рассмотрев, что представляет собой сама техника.

Философия техники, исследует:

- во-первых, феномен техники в целом,
- во-вторых, место в общественном развитии.

Как феномен техника выступает не только в виде машин и орудий, но и как технические сооружения и даже техническая среда. К характеристикам техники относятся также знания, используемые в технике, и различные культурные «тексты», в которых обсуждается техника, и техническое поведение людей. Осмысление *сущности техники* – это ответ на такие фундаментальные вопросы как: в чем природа техники? Как техника относится к другим сферам человеческой деятельности – науке, искусству, инженерии, проектированию, практической деятельности? Когда техника возникает, и какие этапы она проходит в своем развитии? Действительно ли техника угрожает нашей цивилизации, как это утверждают многие

философы? Каково влияние техники на человека и природу? Наконец, каковы перспективы развития и изменения техники?

Необходимо отметить, что эти вопросы заинтересовали мыслителей относительно недавно. Хотя техника как создание орудий и «техника» в смысле технологической стороны всякой деятельности (техника земледелия, техника изготовления вещей, техника в спорте т.д.) возникла на заре человечества, несколько десятков тысяч лет тому назад, феномен техники в его современном понимании был выделен и осознан только в XIX столетии. Глубокое философское же осмысление техники относится ко второй половине XX столетия.

Философский характер размышлениям о технике придает уяснение идеи и сущности техники, понимание места техники в культуре и социальном универсуме, исторический подход к исследованиям техники.

Параллельно с философией техники возникла также некая междисциплинарная область знаний, представляющая собой вообще широкую рефлексию над техникой. Поэтому очень важно выявить именно предмет философии техники, органически связанный с вопросами мировоззренческого и методологического характера.

Философия техники ориентирована на две основные задачи.

Первая задача – осмысление техники, уяснение ее природы и сущности – была вызвана кризисом не столько техники, сколько всей современной «техногенной цивилизации». Постепенно становится понятным, что кризисы нашей цивилизации – экологический, эсхатологический, антропологический (деградация человека и духовности), кризис культуры и другие – взаимосвязаны. Причем техника и, более широко, техническое отношение ко всему является одним из факторов этого глобального неблагополучия. Именно поэтому нашу цивилизацию все чаще называют «техногенной», имея в виду влияние техники на все ее аспекты и на человека, а также глубинные технические истоки ее развития.

Вторая задача имеет скорее методологическую природу: это *поиск в философии техники путей разрешения кризиса техники.*

Вопрос о статусе и природе философии техники связан с еще одной проблемой, а именно, включать ли в философию техники прикладные задачи и проблемы. Фактически это уже происходит: к философии техники сегодня, например, относят такие проблемы как определение основ научно-технической политики, разработка методологии научно-технических и гуманитарно-технических экспертиз, методология научно-технического прогнозирования и др.

Отечественные исследователи В.Г. Горохов и В.М. Розин справедливо полагают, что для определения предмета и объекта философии техники необходимо различать техническое действие, техническое знание и техническое сознание. Результатом технического сознания является выявление мета и роли техники, технической деятельности и технического знания в культуре. Техника, техническая деятельность и техническое знание

как феномены культуры – это объект философии техники. *Развитие технического сознания, осмысливающего этот объект - это ее предмет*.

П. К. Энгельмейер следующим образом формулирует ее задачи:

1. В любой человеческой активности, при всяком переходе от идеи к вещи, от цели к ее достижению мы должны пройти через некоторую специальную технику. Но все эти техники имеют между собой много общего. Одна из задач философии техники как раз и состоит в том, чтобы выяснить, что же такое это общее?
2. В каких отношениях находится техника со всей культурой?
3. Соотношение техники с экономикой, наукой, искусством и правом.
4. Разработка вопросов технического творчества.

Одной из важнейших проблем, которой занимается философия техники, является концепция человека, создающего и использующего технику. В конечном счете, речь идет о влиянии техники на человека. Несомненно, сложилась абсолютно новая ситуация: никогда прежде техника не обладала такой мощью, чтобы была в состоянии уничтожить жизнь в частичной экологической системе и, в конечном счете, довести дело до глобальной катастрофы.

Изобретя компьютер (кибернетическую систему, моделирующую различные виды мыслительной деятельности, оперирующую сложными видами информации) человек произвел свой интеллектуально-информационный аналог (как бы пока прототип). Конечно, компьютер – это орудие труда, но в то же время человек, взаимодействуя с ним, испытывает на себе его влияние. По мнению Н. Винера, проблема информатизации и глобальной коммуникации является сутью проблем кибернетики. Взаимодействие между машиной и человеком – это взаимодействие между думающим, чувствующим, наделенным волей и сознанием существом и неодушевленным предметом. Демонизм – это символ злого начала. Когда речь идет о демонизме применительно к технике, то имеются в виду непредсказуемые последствия ее использования для человека, общества в целом. Ясперс считал, что все возрастающая доля труда ведет к механизации и автоматизации деятельности работающего человека: труд не облегчает бремя человека в его упорном воздействии на природу, а превращает его в часть машины. Техника, будучи созданной человеком, направлена на то, чтобы в ходе преобразования всей трудовой деятельности преобразовать и самого человека: мышление, весь склад души становится «технократическим». Вся судьба человечества, по словам Ясперса, зависит от того способа, посредством которого он подчинит себе последствия технического развития и их влияние на его жизнь.

Как уже отмечено, сущностное назначение техники – быть средством для достижения целей. Средством весьма специфическим, особым. С его помощью человек опосредствует и тем усиливает (совершенствует) свое преобразовательное воздействие на мир. Поскольку строительным материалом для техники служит механико-физико-химическая действительность, то можно сказать, что с помощью техники человек

заставляет одну силу природы (воду, нефть, ветер и т. д.) воздействовать на другую (другие) и реализовать при этом его (человека) сознательные цели.
В технике воплощается рациональная природа (сущностная сила) человека. А это, помимо прочего, означает, что технические средства позволяют достигать наибольшего результата при наименьшей трате сил. То, что на стороне человека выглядит как рациональность, на стороне техники обретает форму экономичности, оптимальности и эффективности. Как средство техника аккумулирует и передает человеку власть. Поначалу для освобождения от власти природы, которая долго не отпускала от себя человека, не позволяла ему оторваться от своей животности. Затем уже для доминирования, господства над природой, которое содержит в себе не только положительные, но и отрицательные моменты в виде, скажем, разрушения эволюционно отработанных зависимостей и балансов природы. В дальнейшем процесс технического «освобождения – господства» перекинулся на общество. Картина получилась следующая: с одной стороны, освобождение от стихийных сил истории, а с другой – широкое использование социальных технологий господства (манипулирование общественным сознанием, идеологическое подавление духа критики и сопротивления, новое, «технически-компетентное», расслоение общества). На очереди теперь сам человек. И, видимо, дело времени – освободить его от естественно-природного (удалить все природное из природы человека) и перейти полностью (это уже господство) на искусственно-техническое.

Формирование техники в современной культуре привело к тому, что *современный человек видит в технике действие законов природы и свое собственное инженерное творчество*.

Таким образом, *философия техники – направление современной философии, призванное исследовать наиболее общие закономерности развития техники, технологии, инженерной и технической деятельности, а также их место в человеческой культуре и в современном обществе.*

Основные сферы философии техники:

1) культура и техника (историко-культурный и социокультурный аспекты);

2) методологические проблемы философии техники, методология технических наук и проектирования;

3) социальная оценка техники и ее последствий;

4) инженерная этика.

Главная задача философии техники - исследование технического отношения человека к миру, то есть технического миропонимания. Философия техники не имеет ничего общего с технократизмом, более того, она противостоит ему и с самого своего возникновения ориентирована на гуманизацию техники.

Вопросы для самоконтроля

1. Классическая и неклассическая традиции в интерпретации понятия «техника».

Предмет философии техники.

2. Основные исследовательские программы в философии техники.

3.Техника как важнейший фактор цивилизационной динамики.

4.Генезис и социодинамика техносферы.

5. Человек и техносфера. Становление техноструктуры XXI века.

6. Образы техники в культуре: традиционная и проектная культуры.

7.Перспективы и границы современной техногенной цивилизации.

8.Технический оптимизм и технический пессимизм: апология и культуркритика техники.

Список рекомендуемой литературы

Основная

4. Канке, В. А. Философские проблемы науки и техники: учебник и практикум для магистратуры / В. А. Канке. — М. : Издательство Юрайт, 2016. — 288 с.
5. Мареева Е. В.Философия науки: учебное пособие для аспирантов и соискателей/Мареева Е. В., Мареев С. Н., Майданский А. Д. - М.: НИЦ ИНФРА-М, 2016. - 332 с.
6. Мокий, М. С. Методология научных исследований : учебник для магистров/ М. С. Мокий, А. Л. Никифоров, В. С. Мокий ; под ред. М. С. Мокия. — М.: Изд.-тво Юрайт, 2015. — 255 с.
7. Прытков В.П. Философские проблемы науки и техники: учебное пособие/ В.П. Прытков.- Екатеринбург: Изд-во Урал. Ун-та, 2013.-63 с.8

Дополнительная

8.Аль-Ани Н.М. Философия техники: очерки истории и теории: учебное пособие.-СПб.: А-принт, 2004.-184 с.

9.Горохов В.Г., Розин В.М. Введение в философию техники: Учебное пособие.-М.: Инфра-М, 1998.-224 с.

10.Горохов В.Г. Техника и культура: возникновение техники и теории технического творчества в России и Германии в конце XIX –начале XX столетия.-М., 2009.-376 с.

11.Горохов В.Г. Основы философии техники и технических наук.

Учебник. М.: Гардарика, 2007.-335 с. 9.

12. Кошарный В.П. Философия науки и техники: учебное пособие. Пенза, 2012.- 294 с.

13. Митчем К. Что такое философия техники?- М.: Аспект-пресс, 1995.- 149 с.

14. Новая технократическая волна на Западе/ Гуревич П. С. (отв. ред.). Издательство: М.: Прогресс .- 1986 - 453 с.

15.Поносов, Ф.Н. Современные философские проблемы техники и технических наук : учебное пособие / Ф.Н. Поносов. – Ижевск : ФГБОУ ВПО Ижевская ГСХА, 2013. – 262 с.

16.Попкова Н.В. Философия техносферы. – М.: URSS; ЛКИ, 2008. - 344 с**.**

17. Современные философские проблемы естественных, технических и социально-гуманитарных наук/под. ред. В.В.Миронова.-М.:Гардарики, 2006.-639 с.

18.Степин В. С. Философия науки и техники / В. С. Степин, В. Г. Горохов, М. А. Розов. – М., 1995.- 372 с.

18. Философия математики и технических наук / Под общ. ред. С.А. Лебедева: Учебное пособие для вузов.- М., 2006.-779 с

19. Философия техники в ФРГ/ Арзаканян Ц.Г., Горохов В.Г. (сост.).-М.: Прогресс, 1989.

20. Философия техники: история и современность/Ред. Розин В.М.-М.: Изд-во: ИФ РАН, 1997.- 283 с.

21. Шаповалов В.Ф. Философия науки и техники: Учебное пособие. – М.: ФАИР-ПРЕСС, 2004. – 320 с.

Тема 6. Становление философии техники

Теперь рассмотрим творчество первых представителей философии техники с момента ее зарождения: прежде всего в Германии и России в конце XIX – начале XX в. В их работах уже содержалась в зачаточной форме вся будущая проблематика философии техники.

Еще в 1903 г. русский инженер и философ техники *П. К. Энгельмейер*, делая доклад *«Библиографический очерк философии техники»* Политехническому обществу, попытался обрисовать зарождение этой новой отрасли философской науки. Он отмечал, что современную эпоху недаром называют технической: машинная техника распространяет свое влияние далеко за пределы промышленности, и воздействие ее сказывается чуть ли не на всех сторонах современной жизни культурных государств. И вот: мыслители и ученые самых разнообразных сфер начинают изучать этот, до сих пор не вполне еще оцененный фактор. И здесь по мере изучения открываются все новые и новые умозрительные горизонты. Тем не менее все, что до сих пор сделано, можно назвать только расчисткой места для будущего здания, которое можно пока, за недостатком более подходящего слова, назвать философией техники. В этом очерке российский ученый указал на множество работ, так или иначе касающихся различных сторон этой проблематики. Однако среди них можно выделить две линии: первая идет от философствующих инженеров (это – *Эрнст Гартиг* (1836–1900), *Иоганн Бекманн* (1739–1811), *Франц Рело* (1829–1905) и *Алоиз*

Ридлер (1850–1936)), вторая – от философов – *Э. Каппа* (1808–1896), *А. Эспинаса(1844-1922), Ф. Бона*.

Обратимся к идеям одного из них. *Эрнст Капп* был первым, кто совершил смелый шаг – в заголовке своей работы *«Основные направления философии техники»* он соединил вместе два ранее казавшиеся несовместимыми понятия «философия» и «техника». Главными понятиями его философии техники являются «антропологический критерий» и концепция органопроекции.

Формулируя свой антропологический критерий, Эрнст Капп подчеркивал: каковы бы ни были предметы мышления, то, что мысль находит в результате всех своих исканий, всегда есть человек. Поэтому содержанием науки в исследовательском процессе вообще является не что иное, как возвращающийся к себе человек. Капп считает, что именно в словах древнегреческого мыслителя Протагора – «Человек есть мера всех вещей» – были впервые сформулированы антропологический критерий и ядро человеческого знания и деятельности. Именно благодаря тому, что человек мыслит себя в природе и из природы, а не над ней и вне ее, мышление человека становится согласованием его физиологической организации с космическими условиями.

Осмысливая понятие внешнего мира человека, Э. Капп замечает, что для него недостаточно слова «природа» в обычном понимании. К внешнему миру, окружающему человека, принадлежит также множество вещей, которые являются его созданием. Будучи искусственными произведениями в отличие от естественных продуктов (природа доставляет для них материал), они образуют содержание мира культуры. Э. Капп проводит четкое разграничение «естественного» и «искусственного»: то, что вне человека, состоит из созданий природы и созданий человека.

Этот исходящий от человека внешний мир является, с точки зрения Каппа, реальным продолжением его организма, перенесением вовне, воплощением в материи своих представлений, т.е. части самого себя, своего собственного «Я». Это – отображение вовне, как в зеркале, внутреннего мира человека. Но созданный человеком искусственный мир становится затем средством самопознания в акте обратного перенесения отображения из внешнего мира во внутренний. Таким образом, человек познает процессы и законы своей бессознательной жизни. Короче говоря, «механизм», бессознательно созданный по органическому образцу, сам служит для объяснения и понимания «организма». В этом и состоит суть принципа *органической проекции* Эрнста Каппа.

Капп отмечает, что человек бессознательно делает свое тело масштабом для природы. Так возникла, например, десятичная система счисления (десять пальцев рук). Однако принцип *органопроекции* легко объясняет только возникновение первых простейших орудий. При его применении к сложным орудиям и машинам возникают проблемы. Хотя Капп и предупреждал, что органическая проекция может и не позволять распространять формальное сходство и что ее ценность – в преимущественном выражении основных связей и отношений организма, этим проблемы не снимаются. В качестве примера возьмем, вслед за Каппом, паровую машину. Форма ее как целого не имеет ничего общего с человеком, схожи лишь отдельные части. Но когда паровая машина начинает функционировать, например в локомотиве, то сразу обнаруживается сходство ее общего целесообразного механического действия с органическим единством жизни: питание, изнашивание частей, выделение отбросов и продуктов сгорания, остановка всех функций и смерть, если, скажем, разрушена важная часть машины – сходны с жизненными процессами животного. Капп подчеркивает, что это уже не бессознательное воспроизведение органических форм, а проекция живого и действующего как организм существа. Именно эта видимость самостоятельной деятельности и поражает больше всего в паровой машине.

Затем Капп переходит от отдельных творений техники к тем продуктам культуры, которые не укладываются в понятие аппаратов и имеют характер систем. Таковы, например, железные дороги и телеграф, покрывшие сетью весь земной шар. Первые, особенно при соединении рельсовых путей и пароходных линий в одно целое, являются отражением системы кровеносных сосудов в организме. Это коммуникационная артерия, по которой циркулируют продукты, необходимые для существования человечества. Второй естественно сравнить с нервной системой. Здесь, по мнению Каппа, *органопроекция* празднует свой триумф: сначала бессознательно совершающееся по органическому образцу построение, затем взаимное узнавание оригинала и отражения (по закону аналогии) и, наконец, подобно искре вспыхивающее сознание совпадения между органом и орудием вплоть до тождества.

Косвенным подтверждением принципа органопроекции, понятого, конечно, не буквально, является развитие современной микроэлектроники, которая, перепробовав (бессознательно) всевозможные материалы, выбрала для интегральных схем в качестве наиболее оптимального материала кремний. Но именно его еще раньше эволюция «выбрала» исходным материалом органических тел.

Послойный синтез твердотельных интегральных структур, развитый в современной технологии производства микроэлектронных схем, также наиболее распространен в живой и неживой природе (например, рост кристаллов, годичный рост деревьев, образование кожи). Здесь органопроекция имеет тенденцию к отображению, по крайней мере, нижних уровней структуры биосинтеза. Причем технологические приемы послойного синтеза эффективно (и бессознательно) применялись в первобытных технологиях, начиная с неолита, например, при производстве украшений, в полиграфии, при изготовлении корабельной брони.

Концепция органопроекции – первая попытка философской экспликации генезиса техники и ее «антропных» начал.

Особая роль в развитии философии техники в России принадлежит *Петру Климентьевичу Энгельмейеру* (1855–1942), который был первым философом техники в России. Петр Энгельмейер окончил гимназию в Москве и посещал лицей в Ницце. Он владел немецким, французским, английским и итальянским языками. В 1874–1881 гг. он учился в Императорском Московском техническом училище и по окончании его получил диплом инженера-механика. Он увлекался также различными другими областями техники (электротехникой, авиатехникой и т.д.). Был редактором и издателем журнала «Техник», учителем механики в средней технической школе, в воскресной и вечерней школе для рабочих, инженером на машиностроительном заводе в Москве и т.д. Перу Энгельмейера принадлежит около ста статей, брошюр и книг. Еще в 1898 г. в брошюре *«Технический итог XIX века»* П. К. Энгельмейер следующим образом формулирует задачи философии техники:

«1. В любой человеческой активности, при всяком переходе от идеи к вещи, от цели к ее достижению, мы должны пройти через некоторую специальную технику. Но все эти техники имеют между собой много общего. Одна из задач философии техники как раз и состоит в том, чтобы выяснить, что же такое это общее?

2. В каких отношениях находится техника со всей культурой?

3. Соотношение техники с экономикой, наукой, искусством и правом.

4. Разработка вопросов технического творчества».

Он полагал, что техника есть только одно из колес в гигантских часах человеческой общественности. Внутреннее устройство этого колеса исследует технология, но она не в силах выйти за свои пределы и выяснить место, занимаемое этим колесом, и его функцию в общем механизме. Эту задачу может выполнить только *философия техники*.

П. К. Энгельмейер следующим образом определяет сущность техники: «Сущность техники заключается не в фактическом выполнении намерения, но в возможности воздействия на материю... Природа не преследует никаких целей, в человеческом смысле этого слова. Природа автоматична. Явления природы между собой сцеплены так, что следуют друг за другом лишь в одном направлении: вода может течь только сверху вниз, разности потенциалов могут только выравниваться. Пусть, например, ряд А – В – С – Д – Е представляет собой такую природную цепь. Является фактически звено А, и за ним автоматически следуют остальные, ибо природа фактична. А человек, наоборот, гипотетичен, и в этом лежит его преимущество. Так, например, он желал, чтобы наступило явление Е, но не в состоянии его вызвать своей мускульной силой. Но он знает такую цепь А – В – С – Д – Е, в которой видит явление А, доступное для его мускульной силы. Тогда он вызывает явление А, цепь вступает в действие, и явление Е наступает. Вот в чем состоит сущность техники» (Энгельмейер, П. К. Философия техники. – М., 1912. – Вып. 2. – С. 85).

П. К. Энгельмейер показывает тесную связь философии техники (техницизма) с теорией деятельности, которую он впоследствии называет «активизм». На этом пути, считает он, философия техники разрастается в философию человеческой деятельности.

Одно из центральных мест в философии техники П. К. Энгельмейера занимает *теория технического творчества*, суть которой выражается в так называемом «трехакте». Творчество зарождается из *желания* (потребности, наклонности) и выявляется в некоторой обстановке, которую оно изменяет сообразно с желанием. Таким образом, творчество выражается, в конце концов, в прямом воздействии на окружающую обстановку. Но тут замечается еще и промежуточный момент: *составление плана действия*. В составлении плана действуют два агента, существенно различных: один бессознательный, внелогический – это интуиция; другой сознательный, логический – это рассуждение. А *выполнение плана* на деле совершается за счет третьего агента, телесного, двигательного, способного воздействовать на окружающую материю.

Отсюда видно, что *механизм творчества есть «трехакт»*, три акта которого суть функции трех вышеназванных агентов. Первый акт есть функция *интуиции,* второй – *рассуждения*, третий – *организованного рефлекса*. В первом акте под давлением первоначального желания возникает идея, которая ставит цель. Во втором акте рассуждение

вырабатывает из идеи план действий. В третьем акте этот план приводится в исполнение.

Теория технического творчества, разработка которой была начата в трудах Энгельмейера, активно развивалась в течение XX в. в трудах А. Осборна, Г. С. Альтшуллера и др. Выявление философской основы технического творчества позволяет не только повысить его эффективность, но также выявляет его смысл, особенно в контексте нарастания глобальных проблем современности.

Наиболее выдающейся фигурой в дискуссиях по проблемам философии техники как до Второй мировой войны, так и непосредственно после нее, был, несомненно, *Фридрих Дессауер* (1881–1963), деятельность которого охватывает всю первую половину XX в. Фридрих Дессауер создал концепцию техники как сопричастности божественному творению. Еще в юношеские годы он был буквально загипнотизирован работами Рентгена, открывшего *X*-лучи, в девятнадцать лет бросил школу и основал заводы, на которых производились машины для получения *X*-лучей. В качестве изобретателя и предпринимателя он разработал и развил технику терапии глубокого проникновения *X*-лучей. Как предприниматель он был связан с университетскими исследованиями по проблеме создания трансформаторов высоких энергий, которые намеревался сочетать с мощными машинами, производящими *X*-лучи. На основании этих работ ему была присвоена докторская степень в области прикладной физики в 1917 г. в университете Франкфурта-на-Майне. Позднее, получив должность в университете и оставив свою практическую деятельность в этой области, в 1922 г. как популярный лектор университета и писатель Дессауер сумел убедить целую группу промышленников финансировать создание исследовательского Института биофизики, директором которого он и стал. Умер Дессауер в 1963 г.

В своей философии техники Дессауер был всесторонен и открыт другим сферам знания. Он, хотя и был сторонником защиты техники с помощью строгих понятий, все же пытался вступать в диалог с экзистенциалистами, с социологами и теологами. Благодаря последовательности и строгости в анализе проблем философии техники Дессауер стал именно тем исследователем, которого чаще всего упоминают и цитируют представители философии науки, когда касаются вопросов философии техники. В работе *«Философия техники»* и три десятилетия спустя в книге *«Споры вокруг техники»* он пытается, для выявления силы и значения техники, заново обосновать кантовскую концепцию трансцендентальных условий технической деятельности, а также показать этические следствия при ее применении. К трем кантовым

критикам – научного знания, морали и эстетического восприятия (критика способности суждения) – Дессауер добавляет четвертую – критику технической деятельности.
В отличие от Канта Дессауер утверждает, что делание, особенно в виде технических изобретений, может как раз установить позитивный контакт с «вещами самими по себе». Сущность техники не проявляется ни в промышленном производстве (которое лишь в массовом порядке производит результаты тех или иных открытий), ни в самих продуктах техники (которые только лишь используются потребителями), но в самом акте технического творчества. Анализ акта технического творчества показывает, что оно реализуется в полной гармонии с естественными законами, однако эти природные законы и цели, будучи необходимыми, не являются одновременно достаточными условиями изобретения. Помимо них существует и нечто другое, что Дессауер называет «внутренней обработкой, которая и приводит сознание изобретателя к контакту с “четвертым царством” – сферой, в которой пребывают “предданные решения технических проблем”». Именно эта внутренняя обработка и есть то, что делает возможным технические изобретения. То обстоятельство, что эта внутренняя обработка и реализует контакт с трансцендентными «вещами самими по себе» технических объектов, подтверждается следующими двумя фактами: 1) изобретение в качестве артефакта не есть нечто такое, что можно обнаружить в мире явлений; 2) лишь когда оно появляется в качестве феноменальной реальности как данное изобретение посредством творчества изобретателя и через него, только тогда оно вступает в действие, «работает». Изобретение не есть нечто выдуманное, продукт человеческого воображения без реальной силы; оно появляется лишь после и в результате встречи в сознании со сферой предданных решений технических проблем. Техническое изо-бретение олицетворяет «реальное бытие идей», т.е. порождает и формирует условия для «существования сущности», для материального воплощения трансцендентальной реальности. В концепции Канта подобного рода трансценденция, переход через границы опытного знания, если она возможна, существует только в сфере морального и эстетического опыта. Дессауер, в отличие от Канта, видит переход через границы опыта именно в той практической сфере, которую Кант полностью игнорировал, во всяком случае, никогда не рассматривал всерьез. И в этом Дессауер достаточно последователен и решителен. В соответствии с таким метафизическим анализом Дессауер формулирует определенную теорию моральной значимости техники. Большинство концепций техники ограничиваются рассмотрением практических выгод и пользы. Для Дессауера же создание техники носит характер кантовского категорического императива или божественной заповеди. Люди создают

технику, однако ее могущество переходит грань всякого ожидания. Современная техника не должна восприниматься как средство облегчения условий человеческого бытия (как утверждал Ф. Бэкон); в действительности техника есть «участие в творении». Согласно концепции Дессауера, техника становится религиозным переживанием и опытом, и само религиозное переживание приобретает техническую значимость.

Инженерная философия техники осуществляла анализ техники как бы изнутри: интерпретация технического способа бытия человека в мире как главного для понимания других типов человеческого мышления и действия вызвали к жизни иной подход в философии техники – так называемую «гуманитарную» философию техники.

Признанным лидером в исследовании социальных аспектов технического прогресса является *Карл Маркс* (1818–1883). В пятой главе «Капитала» Маркс проводит основательный анализ человеческого труда, поскольку именно он «потребляется» (т.е. имеет потребительную стоимость), а технические средства – лишь его проводники. Мыслитель с симпатией относился к идее органопроекции Э. Каппа. Маркс считал, что предмет, которым человек овладевает непосредственно, есть не предмет труда, а средство труда. Так, данное самой природой становится органом его деятельности, органом, который он присоединяет к органам своего тела, удлиняя таким образом его естественные размеры. Для Маркса орудия труда – это «овеществленная сила знания». Вытеснение ручного труда машинным привело к революционным преобразованиям трудового процесса. Характер новой эпохи Маркс определял через прогресс средств труда, представляющих собой не только мерило развития рабочей силы, но и показатель самих общественных отношений. Последствия революции в развитии средств труда, которая привела к вытеснению ручного труда и связанному с этим массовому увольнению тех, кого заменила машина, Маркс подробно рассматривает в восьмой главе «Капитала». При переходе от ремесленной техники к технике машинной карликовое орудие человеческого организма – мускульная энергия – была заменена силами природы, а на смену традиционным знаниям, использовавшимся в процессе ручного труда, пришли естественно-научные знания точных наук. Промышленный труд вытесняет труд ремесленный, тем самым машина становится кровным врагом ремесленника. «Мертвый» (машинный) труд полностью господствует над «живым» и успешно конкурирует с ним, делая его придатком машинного производства. В мануфактуре и ремесле рабочий заставляет орудие служить себе, а на фабрике он служит машине, являясь ее живым придатком. За этими техническими переменами следует вторая

степень зависимости рабочего: он зависит не только от работодателя, но и от средств труда, что придает его отчужденности явно техническое измерение. Очень скоро обнаруживается, что работодателю теперь не нужно столько рабочих: многие трудовые операции делают «умные» машины. Наступает время массовых увольнений, миллионы тружеников становятся безработными. Кровным врагом рабочего становится машина – средство труда. Но Маркс утверждает, что машины сами по себе не ответственны за то, что они «освобождают» рабочего от жизненных средств существования. Причина, по Марксу, – в капиталистическом применении машин. Машина «враждебна» к ремеслу не сама по себе. Она просто оказалась не в тех руках, следовательно, необходимо передать ее в другие руки – в руки ставших безработными рабочих, а работодателя экспроприировать как экспроприатора, отдав политическую и экономическую власть рабочим, пролетариату. Такова логика учения Маркса. Следует отметить еще один важный аспект в технофилософской концепции Маркса. Опираясь на диалектику, Маркс полагал, что любой из системообразующих элементов этого процесса непременно должен содержать в себе и относительный регресс. Речь идет о техническом развитии как важной составляющей социального прогресса.

Французский философ техники и культуролог *Жак Эллюль* (1912–1994) приобрел известность, опубликовав книгу *«Техника»* (1954). Все работы Эллюля были посвящены анализу и изучению современного ему технического общества. Основное исследовательское кредо автора сводится к оспариванию марксистской концепции о решающей роли способа производства в историческом развитии общества. По мнению Эллюля, в основе классификации исторических эпох должна быть положена степень развития техники. Эти идеи последовательно освещались в его книгах *«Техническое общество»* (1965), *«Политические иллюзии»* (1965), *«Метаморфоза буржуазии»* (1967), «Империя нелепости» (1980). Сюжетами их были проблемы современного технического общества, техники, технической личности, политики, положение общественных классов и искусство. Центральные понятия в теории Эллюля – «техника» и «технофилософия». Технику он определяет как «совокупность рационально выработанных методов, обладающих абсолютной эффективностью в каждой области человеческой деятельности». «Феномен техники», согласно Эллюлю, характеризуется такими важными особенностями, как рациональность, артефактность, самонаправленность, саморазвитие, неделимость, универсальность и автономность. Эти семь признаков, по мнению автора, образуют характерное содержание техники в качестве основной господствующей формы человеческой деятельности.

Таким образом, именно техника определяет все другие формы деятельности, всю человеческую технологию и все общественные структуры – экономику, политику, образование, здравоохранение, искусство, спорт и т.д. Технику на современном этапе ее развития Эллюль рассматривает в широком мировоззренческом плане как тип рациональности. Она замещает природу техносферой, технической средой, подменяя собой среду естественную. Техника – это навязанная извне сила, данность, с которой человеку приходится считаться; она навязывает себя просто тем, что существует. Техника как данность, как нечто самодовлеющее «ведет» весьма опасную и рискованную игру. В этой игре человек должен сделать ставку только на те действия, которые он предпринимает, чтобы достичь своих добрых целей и осуществить свои благие намерения.

С точки зрения автора, техника призвана помочь людям построить свой дом здесь, на Земле. Претензиям техники превратиться во всеобусловливающее и всепорождающее начало, ее стремлению к всевластию подлежит оказывать активное противодействие и давать серьезный отпор. На роль такого противодействия претендует сформулированная Эллюлем этическая концепция отказа от власти техники. Эта концепция практически основывается на прямом и полном отрицании так называемого «технологического императива», согласно которому люди могут, а следовательно, должны делать все, что технически им доступно и принципиально выполнимо. Эллюль фактически требует отказаться от подобной установки. Этика отказа от власти техники требует не просто ограничения указанного императива, а полного его отрицания. Исходным принципом этой этической концепции является идея самоограничения человека, которая неизбежно ведет к замене «технологического императива» противоположной установкой, согласно которой люди должны договариваться между собой, не делать всего того, что они вообще в состоянии технически осуществить. Эту установку можно назвать «антитехнологическим императивом», она становится и актуальной, и судьбоносной, так как на фоне непомерного усиления власти техники приходит убеждение в полном отсутствии внешних сил, способных противостоять технике и активно противодействовать ее всевластию.

Однако реальной альтернативы технике все же не существует, поэтому приходится с ней «уживаться». В этих условиях остается одно: следовать этике отказа от власти техники. Такая этика требует не только самоограничения, но и отказа от техники, разрушающей личность. Для этого, по мнению Эллюля, необходима революция: только она сможет обратить технику из фактора порабощения человека в фактор его

освобождения. Философ называет эту революцию «политико-технической» – это своеобразная утопическая модель развития современного западного общества. «Политико-техническая» революция обусловлена необходимостью решения пяти проблем (аспектов) развития общества.

Во-первых, необходимо оказать безвозмездную помощь странам «третьего мира» с целью предоставить им возможность для извлечения всей пользы из западной технологии, самостоятельного строительства своей истории. Во-вторых, следует отказаться от применения силы в «какой бы то ни было форме» и от «военных арсеналов, подавляющих нашу экономику», а также полностью ликвидировать «централизованное бюрократическое государство». При этом автор полагает, что это не приведет ни к падению организованности, ни к неразберихе, так как, по его мнению, некому будет создавать путаницу, беспорядок и замешательство. Далее необходимы отказ от роста цен, поощрение малого бизнеса. Снижение уровня жизни должно компенсироваться повышением ее качества. В-третьих, необходимо добиться всестороннего развертывания способностей и диверсификации занятий. С этим связаны расцвет национальных дарований, признание всех автономий, создание свободной и достойной жизни малым народам, обеспечение им подъема образования, причем не обязательно с созданием своей государственности. В-четвертых, необходимо добиться резкого сокращения рабочего времени, замены 35-часовой рабочей недели 2-часовой ежедневной работой. Кроме того, предполагается вести пропаганду по вопросам о смысле жизни, о новой культуре, открывая простор для нового размаха творческих способностей и т.д. Наконец, в-пятых, критерием прогресса предлагается считать количество «сэкономленного» человеком времени. Оплату труда предполагается вести не деньгами, а путем продуктообмена, причем независимо от количества вложенного труда.

Целью «политико-технической» революции признается не захват власти, а реализация позитивных потенций современной техники, ориентированных на полное освобождение человека. В социально-утопическом проекте Эллюля предусматривается налаживание самоуправления на уровне коммун.

Многочисленные труды американского философа и социолога *Льюиса Мэмфорда* (1895–1990) были посвящены философии техники: *«Техника и цивилизация»* (1934), *«Искусство и техника»* (1952), *«Миф о машине»* (в 2 т., 1970). Мэмфорд считается представителем негативного технологического детерминизма. Главную причину всех социальных зол и потрясений он видел в возрастающем разрыве между уровнем технологии

и нравственностью. Научно-технический прогресс, совершенный со времен Г. Галилея и Ф. Бэкона, он называл «интеллектуальным империализмом», «жертвой» которого пали гуманизм, социальная справедливость. Наука – это суррогат религии, а ученые – сословие новых жрецов, – так оценивал Мэмфорд науку и ее служителей. Он считал, что нельзя понять действительную роль техники, рассматривая человека как «животное, делающее орудия». Древний человек обладал единственным орудием – своим телом, управляемым мозгом, умом. Его умственная энергия превосходила его потребности, и орудийная техника была частью биотехники мозга. Истоки этой «добавочной умственной энергии» Мэмфорд видит не только в труде, но и в других составляющих коллективного существования и общения, в таких, как игровая, эстетическая и религиозная стороны жизни человека, прочие нетрудовые формы, детерминированные опытом добывания средств существования.

Историю европейской цивилизации он делит на три основных этапа. Первый этап (с 1000 по 1750 г.) характеризуется культивированием так называемой интуитивной техники, связанной с применением силы падающей воды, ветра и использованием природных материалов: дерева, камня и т.д., которые не разрушали природу, а были с ней в гармонии. Второй этап (XVIII – XIX вв.) основан на палеотехнике (т.е. ископаемой технике) – это эмпирическая техника угля и железа. Данный этап характеризовался отходом от природы и попыткой господства человека над природой. Мэмфорд называет этот период «рудниковой цивилизацией». Третий этап (с конца XIX в. по настоящее время) – это завершающая фаза функционирования и развития западной цивилизации, в пределах которой происходит на строго научной основе восстановление нарушенной в предыдущей фазе гармонии техники и природы. Анализу этого периода Мэмфорд посвятил книги *«Миф машины»* (1969, 1970), *«Человек как интерпретатор»* (1950) и другие произведения. Дистанцируясь от ставших популярными определений типа «homo faber», он отстаивает понятие «homo sapiens», так как сущность человека, по его мнению, заключается в мышлении, а основой человечности является дух – разум. Человек – главным образом интерпретатор. Это его качество обнаруживается в самотворчестве: человек проецирует сам себя и сам себя создает.

Примечателен подход Мэмфорда к истории развития техники. Он выделяет два ее главных типа: биотехнику и монотехнику. Биотехника – это тип техники, который ориентирован на удовлетворение жизненных запросов и естественных потребностей и устремлений человека.

Монотехника ориентируется главным образом на экономическую экспансию, материальное насыщение и военное производство. Ее цель – укрепление системы личной власти, и поэтому она носит авторитарный характер. Она враждебна не только природе, но и человеку. Ее авторитарный статус восходит в своих истоках к раннему периоду существования человеческой цивилизации, когда впервые была изобретена «мегамашина» – машина социальной организации нового типа, способная повысить человеческий потенциал и вызвать изменения во всех аспектах существования. Человеческая машина с самого начала своего существования объединила в себе два фактора: 1) негативный, принудительный и разрушительный; 2) позитивный, жизнетворный, конструктивный. Оба эти фактора действовали во взаимной связке. Понятие машины, идущее от Франца Рело, означает комбинации «...строго специализированных способных к сопротивлению частей, функционирующих под человеческим контролем, для использования энергии и выполнения работы». В этой связи Мэмфорд пишет: «… Огромную рабочую машину можно с полным основанием называть настоящей машиной, тем более, что ее компоненты, пусть они состоят из человеческих костей, жил и мускулов, сводились к своим чисто механическим элементам и жестко подгонялись для выполнения строго ограниченных задач» (Мамфорд, Л. Миф машины. Техника и развитие человечества / Л. Мамфорд. – М., 2001. – С. 256). Все типы современной машины представляют собой трудосберегающие устройства. Предполагается, что они выполняют максимальный объем работы при минимальных затратах человеческих усилий.

В древние времена вопрос об экономии труда не стоял, и, как пишет Мэмфорд, в древности машины можно было бы называть трудоиспользующимися устройствами. Для нормального функционирования «человеческой машины» были необходимы два средства: надежная организация знаний (естественных и сверхестественных) и развитая система отдачи, исполнения и проверки исполнения приказов. Первое воплощалось в жречестве, без активной помощи которого институт монархии не мог бы существовать; второе – в бюрократии. Обе организации были иерархическими, на вершине иерархии стояли первосвященник и царь. Без их объединенных усилий институт власти не смог бы эффективно функционировать. Следовательно, первое из указанных двух средств – знание, как естественное, так и сверхестественное,– должно было оставаться в руках жреческой элиты, т.е. быть жреческой монополией или жреческой собственностью. Только при таком условии, а значит, и при жестком тотальном контроле над информацией и ее дозированием для широких

слоев населения можно было обеспечить слаженность работы мегамашины и сберечь ее от разрушения. В противном случае, т.е. при разглашении «тайн храма» и обнаружении «закрытой информации», «мегамашина» непременно приходит в упадок и в конечном счете разрушается и гибнет. В этой связи Мэмфорд обращает внимание на то обстоятельство, что язык высшей математики в лице компьютеризации восстановил сегодня и секретность, и монополию знаний с последующим воскрешением тоталитарного контроля над ними. Мэмфорд указывает и еще на одну черту «мегамашины»: слияние монополии власти с монополией личности. Автор мечтает о разрушении подобной «мегамашины» во всех ее институциональных формах. От этого, по его мнению, зависит, будет ли техника функционировать «на службе человеческого развития» и станет ли мир биотехники более открытым человеку.

В 1915 г. *Н. А. Бердяев* (1874–1948) в статье «Дух и машина» делает свою первую попытку сформулировать проблему соотношения человека и техники. В ней Н. А. Бердяев рассматривает технику как освобождающее «дух человека» начало. В начале 1920-х гг. в книге *«Смысл истории»* он вновь возвращается к этой теме, пишет о поворотном значении техники в судьбе человека. Техника, утверждает он, покоряет не только природу, но и человека. Наконец, в 1933 г. он пишет статью «Человек и машина», где трезво оценивает кризис человека и человечества, вызванный бурным развитием техники, рассматривает технику как фактор, определяющий жизнедеятельность человека. Н. Бердяев считает, что человечество стоит перед основным парадоксом: без техники невозможна культура, с ней связано само возникновение культуры, и окончательная победа техники в культуре, вступление в техническую эпоху ведет культуру к гибели. В культуре всегда есть два элемента – элемент технический и элемент природно-органический. И окончательная победа элемента технического над элементом природно-органическим означает перерождение культуры во что-то иное, на культуру уже не похожее. Романтизм есть реакция природно-органического элемента культуры против технического его элемента. Поскольку романтизм восстает против классического сознания, он восстает против преобладания технической формы над природной. Возврат к природе есть вечный мотив в истории культуры, в нем чувствуется страх гибели культуры от власти техники, гибели целостной человеческой природы. Техническая эпоха требует от человека фабрикации продуктов и в наибольшем количестве при наименьшей затрате сил. Человек делается орудием производства продуктов. Вещь ставится выше человека. Трагедия в том, что творение восстает против своего творца, более не повинуется ему. Это видно во

всех процессах рационализации в техническую эпоху, когда человек заменяется машиной, техника заменяет органически-иррациональное организованно-рациональным. Машина и техника, – отмечает он, – наносят страшные поражения душевной жизни человека, и прежде всего, жизни эмоциональной, человеческим чувствам. Душевно-эмоциональная стихия угасает в современной цивилизации... Машинная, техническая цивилизация опасна прежде всего для души. «Сердце с трудом выносит прикосновение холодного металла, оно не может жить в металлической среде. Для нашей эпохи характерны процессы разрушения сердца как ядра души. Все разложилось на элемент интеллектуальный и на чувственные ощущения». Техника наносит страшные удары гуманизму, гуманистическому миросозерцанию, гуманистическому идеалу человека и культуры. Машина по природе своей антигуманистична. Исключительная власть технизации и машинизации влечет именно к этому пределу, к небытию в техническом совершенстве. Невозможно допустить автономию техники, предоставить ей полную свободу действия, она должна быть подчинена духу и духовным ценностям жизни... «Дух человеческий справится с грандиозной задачей в том лишь случае, если он не будет изолирован и не будет опираться лишь на себя, если он будет соединен с Богом. Только тогда сохранится в человеке образ и подобие Божие, т.е. сохранится и человек». Но она порождает новые иррациональные последствия в социальной жизни. Так, рационализация промышленности порождает безработицу, величайшее бедствие нашего времени. Труд человека заменяется машиной, что есть положительное завоевание, которое должно было бы уничтожить рабство и нищету человека. Но машина, по мнению Н. Бердяева, совсем не повинуется тому, что требует от нее человек, она диктует свои законы. Машина хочет, чтобы человек принял ее образ и подобие. Но человек есть образ и подобие Бога, рассуждает Бердяев, и не может стать образом и подобием машины, не перестав существовать. Самый дух, создавший технику и машину, не может быть технизирован и машинизирован без остатка, в нем всегда останется иррациональное начало. И это титаническая борьба человека и технизируемой им природы.

Немецкий философ-экзистенциалист *Карл Ясперс* (1883–1969) был профессором психологии Гейдельбергского университета. В работе «Современная техника» (русское издание – в 1989 г.) впервые с позиций технофилософии анализируются труды Фихте, Гегеля и Шеллинга, посвященные обоснованию так называемого осевого времени, начало которому было положено с возникновением христианства. Отличительной чертой этого времени становятся катастрофическое обнищание в области духовной жизни,

человечности, любви и одновременное нарастание успехов в области науки и техники. Духовная нищета многих ученых-естественников и техников характеризуется их скрытой неудовлетворенностью на фоне исчезающей человечности. Технику Ясперс рассматривает как совокупность тех действий, которые знающий человек совершает с целью господства над природой, т.е. ради того, чтобы придать своей жизни «такой облик, который позволил бы ему снять с себя бремя нужды и обрести нужную форму окружающей среды». Соглашаясь с оценкой К. Маркса совершившейся промышленной революции, Ясперс пишет о переменах во взаимоотношении человека и природы, о его подчиненности природе и последствиях этой «тирании». Планета, как пишет Ясперс, стала единой фабрикой! Свое понимание техники Ясперс конкретизирует следующим образом. По его мнению, она характеризуется двумя особенностями: с одной стороны, рассудком, с другой, – властью. Техника покоится на деятельности рассудка, потому что является частью общей рационализации. Но в то же время она есть умение, способность делать, применяя природу против самой природы. Именно в этом смысле знание – это власть. Основной смысл техни-ки – освобождение человека от власти природы. Принцип техники – это манипулирование силами природы для реализации назначения человека, под углом его зрения. Ясперс выделяет два главных вида техники – технику, производящую энергию, и технику, производящую продукты, а также три фактора, влияющих на развитие научно-технического знания:

1) естественные науки, которые создают свой искусственный мир и являются предпосылками к его дальнейшему развитию;

2) дух изобретательства, способствующий усовершенствованию уже существующих изобретений;

3) организация труда, направленная на повышение рационализации научной и производственной деятельности. Труд человека также оказывается в трехмерном измерении: как затраты физических сил, как планомерная деятельность и как существенное свойство человека. В совокупности труд – это планомерная деятельность, направленная на преобразование предметов труда с помощью средств труда. Собственный мир человека – созданная им искусственная среда обитания и существования – есть результат не индивидуального, а совместного человеческого труда. Ясперс, вслед за Марксом, заключает: «от характера труда и его разделения зависит структура общества и жизнь людей во всех ее измерениях и разветвлениях». В ходе развития человечества социальная оценка труда менялась. Греки презирали физический труд, считая его уделом невежественной массы. По

христианской версии человек был обречен добывать хлеб свой в поте лица, искупая свое грехопадение, т.е. труд ассоциируется с наказанием. Исключением в этом смысле являются протестанты, которые видят в труде благословение, и в частности кальвинисты, считающие труд богоугодным делом, доказательством избранности.

Однако отношение к технике не столь положительное даже среди протестантов. «В течение последних ста лет, – пишет Ясперс, – технику либо прославляли, либо презирали, либо взирали на нее с уважением». Но сама по себе техника нейтральна: она не является ни злом, ни добром. Все зависит от того, чего можно добиться с ее помощью. В этом Ясперс полагается на сознание человека. Естественное человеческое чувство рода вкупе с рационально-теоретическим, научным знанием дает человеку синтетическую интуицию как фундаментальное преимущество в противоборстве со стихийным характером космических и исторических процессов. В конце XVIII в. произошел великий исторический перелом в развитии техники, чему способствовали три фактора: естественные науки, дух изобретательства и организация труда. Техника, хотя и отдалила человека от природы, породила новую близость с ней – родила красоту технических изделий, расширила реальное видение мира. Ныне наступление нового «осевого времени» связано с бурным научно-техническим прогрессом.

Достаточно широко вопрос сущности техники освещен в работах К. Ясперса, в которых он начинает с описания кризиса. Природа меняет облик под воздействием техники и обратно – на человека – оказывает воздействие окружающая его среда. Перед лицом непокоренной природы человек представляется относительно свободным, тогда как во второй природе, которую он технически создает, он может задохнуться. Техника превратила все существование в действие некоего технического механизма, всю планету – в единую фабрику. Произошел полный отрыв человека от его почвы, от подлинного бытия. Значимость вопроса «к чему может прийти человек?» стала настолько велика, что техника становится центральной темой. Вместе с необычайно усилившимся господством человека над природой возникает угроза того, что природа, в свою очередь, подчинит себе человека. Под воздействием действующего в технических условиях человека природа становится подлинным его тираном. Возникает опасность того, что человек задохнется в той своей второй природе, которую он технически создает. Ясперс утверждает, что эпоха преобразований носит, прежде всего, разрушительный характер. Техника стала сегодня едва ли не главной проблемой для понимания нашей ситуации. Техника, для К. Ясперса, – это совокупность действий знающего человека, направленных

на господство над природой; цель их – придать жизни человека такой облик, который позволил бы ему снять с себя бремя нужды и обрести нужную ему форму окружающей среды. В такой трактовке техники на первом плане – цель, которую преследует человек. Техника возникает, когда для достижения цели вводятся промежуточные средства.

Обращаясь к выявлению сущности техники, К. Ясперс рассматривает технику как средство. Она возникает, когда для достижения цели вводятся промежуточные цели. Непосредственная деятельность, подобно дыханию, движению, принятию пищи, еще не называется техникой. Лишь в том случае, если эти процессы совершаются неверно, принимаются преднамеренные действия, говорят о «технике дыхания». Развитие техники К. Ясперс связывал с изменением труда: сокращением затрат и усилением интенсивности труда, эволюцией самого характера труда, в процессе которого техническое творчество противостоит нетворчеству. Техника открывает перед человеком новый мир. Но она имеет свои границы, определяемые тем, что техника – лишь средство господства над безжизненными, органическими силами и людьми, которые подчас смотрят на технику с ужасом.

Характеризуя черты техники,К. Ясперс выделяет следующие.

Рассудочность. Техника покоится на деятельности рассудка, на исчислении в сочетании с предвидением возможностей и с догадками. Техника оперирует механизмами, превращает свои данные в количества и отношения. Она является частью общей рационализации как таковой.

Властность. Использование техники дает человеку власть над природой. Техника господствует над природой посредством самой природы. Она дает умение, методы которого являются внешними по отношению к цели. Господство техники основывается на знании – вот почему говорится, что знание – это сила.

Телеономичность. Власть имеет смысл только при наличии цели. Целями властвования над природой являются облегчение жизни человека, сокращение ежедневных усилий, затрачиваемых на существование, увеличение удобств. Смысл техники состоит в освобождении от власти природы. Ее назначение – освободить человека как живое существо от подчинения природе с ее бедствиями, угрозами, оковами. Поэтому принцип техники заключается в целенаправленном манипулировании материалами и силами для реализации назначения человека. Однако это еще не все. Животное неразрывно связано со своей средой обитания, которую оно принимает, не сознавая этого. Человек же выводит созданную им среду в беспредельность. Он ощущает эту среду не только вследствие освобождения от нужды, но и

вследствие воздействия на него красоты. По мере расширения человеческой среды он утверждает свою реальность.

Репродуктивность. Техника – это совокупность открытых человеком приемов и действий, которые можно затем повторять сколько угодно раз. В этом заключается различие между творческой и трудовой деятельностью. Ясперс различает технику, производящую энергию, и технику, производящую продукты. Он считал, что возможны технические искажения, которые возникают тогда, когда орудия и действия перестают быть опосредующими действиями и становятся самоцелью, и тогда абсолютной целью становятся средства.

Другой представитель философии экзистенциализма – *Мартин Хайдеггер* (1889–1976) – полагал, что техника является нейтральным средством в руках человека или человеческой активности. Он доказывает, что техника лишь часть истины или откровения, что, с одной стороны, современная техника является откровением, при котором человек использует природу, не нарушая ее естественного состояния, с другой – бросает ей вызов тем, что из природного материала производит тот или иной вид энергии и, не будучи зависимым от природы, накапливает и передает их.

М. Хайдеггер показал, что техника не просто конструирует «технический мир», в котором она победоносна и универсальна, она подчиняет своему диктату едва ли не все пространство бытия. Присущая ей логика развития проникает в социальное и человеческое измерение истории. Ее инструментальный разум поражает все сознание эпохи. Он не приемлет представление о технике как о средстве и как о воплощении человеческой деятельности. Хайдеггер видит в технике способ конструирования мира. На место уникальной вещи наподобие изготовленного гончаром глиняного горшка современная техника порождает мир, который Хайдеггер называет Bestand («резервы на длительное время», «запасы») – объекты, готовые для продажи. Этот Bestand состоит из предметов, которые за рамками человеческих потребностей не представляют никакой ценности. По его мнению, «за спиной» или на «изнанке» современной техники в качестве способности открытия стоит нечто, что полагает мир и бросает ему вызов. Это нечто Хайдеггер называет Gestell (постав). Способ обнаружения сущности техники, форма раскрытия потаенности бытия, правящего современной техникой, не будучи чем-то техническим, Хайдеггер называет «поставом». Gestell выражает объединенное содержание тех ориентаций, которые направляют человека, бросают ему вызов, зовут его к раскрытию реального, подобно приказу. Gestell означает тот способ открытия, который определяет сущность современной техники,

но сам он не имеет технической природы. Этот каркас, или Gestell, не является частью техники; он является той установкой, которая лежит в самой основе современной техники, пребывает внутри технической деятельности. Проще говоря, этот термин означает техническое отношение к миру.

Причина возникновения современной техники – вовсе не потребности человека. Хайдеггер хочет сказать: сам тот факт, что реальность «позволяет» человеку манипулировать ею техническими средствами, в известном смысле означает, что сама действительность поощряет человека к такого рода действиям, призывает к манипулированию природой. Реальность поэтому должна нести как бы определенную ответственность за ее же эксплуатацию человеком, подобно тому (если позволителен такой пример), как хозяин, когда он уходит из дома и оставляет дверь открытой, в какой-то мере провоцирует на грабеж.

В воззрениях Хайдеггера, современная техника может быть охарактеризована как овеществленный догматизм. Техника «хорошо знает», как надо что-то конструировать и как надо производить. Хайдеггер утверждает, что вместо того, чтобы рассматривать технику в качестве прикладной науки, точнее было бы рассматривать науку как теоретическую технику. Техника обладает эффективным методом, исключающим все другие методы. И в этом отношении техника не обладает знанием собственных границ, она не признает их. Она неспособна познать саму себя. Техника, сущностью которой является само бытие, никогда не может позволить людям преодолеть ее. Это, в конечном счете, означает, что именно человек является господином бытия. Через сущность техники человек говорит с бытием, слышит его зов. Но импульс может быть угадан неверно, ибо техника провоцирует человека на ложное самораскрытие. По мнению Хайдеггера, бытие невозможно без человеческого существования. Техника вырастает из природного материала, но она входит в экзистенциальную структуру бытия человека, который обладает способностью объектировать свои замыслы. Техника несет с собой и выражает в себе новое отношение человека к миру, новый способ раскрытия бытия. В этом техника родственна искусству и сопряжена с истинным познанием. Однако современная техника связана с забвением бытия и его открытости. В этом исток той угрозы, которую несет с собой техника. Она формирует сугубо технический способ конструирования мира, где природа оказывается поставщиком энергии и материалов, ставится на службу производству по добыванию новых материалов, новой энергии, нового сырья. Техника из раскрытия потаенности бытия превращается в

производящее, добывающее раскрытие. Своеобразие современного сознания заключается в том, что онтологическая природа техники, ее сопряженность с Тайной Бытия исчезает, элиминируется из сознания. Вещь конституируется в горизонте техники, утрачивает свою индивидуальность и самостоятельность.

Для того чтобы охарактеризовать современную технику как обладающую особым характером «полагания», «вызова», Хайдеггер сопоставляет традиционную ветряную мельницу и электростанцию. Каждое из технических сооружений как бы обуздывает природную энергию и используется человеком для осуществления тех или иных своих целей. Однако ветряная мельница и мельница с водяным колесом находятся в таком отношении с природой, которое дает основание сравнивать их с произведениями искусства, утверждает Хайдеггер. Какая-нибудь электростанция редко вписывается в естественный ландшафт или дополняет его – все это похоже друг на друга и в таком виде как бы накладываются на любой ландшафт независимо от его характера. Это одна из последних реальностей технического прогресса, связывающая вопрос о технике с вопросом о вещи. Хайдеггер пытается показать, что технологические процессы, в отличие от традиционной техники, никогда не создают вещей в строгом смысле этого слова. Особенностями этого способа отношения человека к бытию, которое с наибольшей силой выражено в технике и которое, по Хайдеггеру, составляет миссию и судьбу человека, являются:

1) превращение природы в материал и источник энергии;

2) унификация, не постигающая многообразие и дифференцированность бытия;

3) функционализация, умаляющая индивидуальную самостоятельность вещи;

4) превращение субъекта лишь в средство развития техники; подчинение всего и вся планирующему и проектирующему расчету;

5) установка на господство, которая не ограничивается лишь осуществлением воли, а является способом перестраивания природы;

6) замещение природных вещей эрзацами (заменителями); нарастание риска вместе с техническим прогрессом и опасности техники для всей цивилизации.

Учение Хайдеггера о технике, с особой силой подчеркнуло риск и опасность техники для современной цивилизации и одновременно неустранимость технического орудования человеком вещами. Главная

опасность, по Хайдеггеру, не в технике, не в «технизации жизни». Нет никакой демонии техники, но есть опасность непонимания ее сущности. Техника – определенный способ бытия, отношения человека к миру, который связан с превращением бытия в сущее, творчества – в добывающее производство, с господством утилитарно добывающего производства и труда, с забвением истины бытия – его Тайны. Таким образом, Хайдеггер резко порвал с традицией европейской философии техники, которая акцентировала свое внимание на непосредственных, «очевидных» достояниях прогресса. Он показал, что последствия вторжения техники многообразны и в отдаленной перспективе даже труднопредсказуемы. Технологическая предопределенность едва ли не фатальна для человека в том смысле, что содержит в себе некую непредсказуемую заданность мышления, поведения, сознания.

Философские воззрения испанского публициста, общественного деятеля и философа *Хосе Ортеги-и-Гассета* (1883–1955) сложились под влиянием концепций Марбургской школы. Решающую роль в этом сыграли идеи *Германа Когена* (1842–1918), *Пауля Наторпа* (1854–1924), *Эрнста Кассирера* (1874–1945), *Николая Гартмана* (1882–1950). Целью Марбургской школы был анализ философских категорий, концепций этического социализма. Ортега-и-Гассет увлекался тезисом о самополагании познающего субъекта в процессе развития культуры. Положительно относился к теории переживания духовного опыта как вслушивания в жизнь (М. Хайдеггер), был озабочен проблемой разобщенности творцов культуры и ее «потребителей», негативными результатами культуры, проявляющимися в виде социальной дезориентации в системе «массового общества». Его перу принадлежит книга *«Размышления о технике»* (1933). Рассматривая жизнь как «потребность потребностей», Ортега-и-Гассет выступал в защиту автономности личности в ее отношении к собственной судьбе как репертуару жизненного действия. В этом своеобразном списке находятся как естественные, органические, биологические потребности, так и действия, удовлетворяющие эти потребности. По сути, в этом ассортименте и для животных, и для человека все едино. Разница, однако, состоит в том, что человек предпринимает определенные действия – он сам производит то, чего нет в природе. В этом состоит его «репертуар». Но это не самое главное его действие: освободившись от дефицита витальных потребностей, человек имеет возможность расширить круг своих потребностей, т.е. расширить «репертуар». Из этого свойства человеческой натуры автор делает вывод о противоречивости человеческих потребностей. «Репертуар» потребностей человека не совпадает с меню витальных потребностей. Это личное его желание действовать по второму (расширенному) репертуару и составляет

то, что называется деятельностью по преобразованию природы. С целью удовлетворения своих потребностей в угоду им человек навязывает природе свои желания, если она еще не готова послужить им. В этом услужении Ортега-и-Гассет наблюдает, как сама природа преобразуется. Она предъявляет человеку требования в виде естественных нужд. Человек на них отвечает тем, что навязывает ей изменения, преобразовывает ее с помощью техники. Осуществляя это преобразование, техника поддерживает человеческое желание. И эта связь, соединяющая природу с человеком, и наоборот, есть некий посредник – сверхприрода, надстроенная над «первой» природой.

Животному его собственная природа предзадана. Оно – существо нетехническое – именно из-за отсутствия в нем активного начала. Человек же благодаря природному техническому дару творит недостающее, создает новые обстоятельства, приспосабливая природу к своим нуждам. Человек и техника сливаются. Технические действия предназначены для того, чтобы, во-первых, что-то изобрести, во-вторых, обеспечить условия, в-третьих, создать новые возможности. Задача техники – совершать усилия ради сбережения усилий. По мнению автора, именно тогда у человека возникает проблема, как распорядиться освободившимся временем после преодоления им той, животной жизни. Благодаря технике человеческая жизнь выходит за рамки природы, человек ослабляет свою зависимость от природы. Но перед ним возникает новая проблема – как жить дальше? На этот вопрос Ортега-и-Гассет отвечает так. Реальность состоит в том, что мир одновременно и предоставляет человеку удобства, и чинит ему препятствия. Именно в таком мире пребывает человек; его существование окружено и удобствами и трудностями. Именно это придает человеческому бытию онтологический смысл. Человеку предначертано быть существом «сверхъестественным» и одновременно естественным – онтологический кентавр! Таким образом, человеческое «Я» – это непрерывное стремление реализовать определенный проект, программу существования, включающую то, чего еще нет, а также то, что мы должны для себя создать. Обстоятельства даны человеку как «сырье» и механизм. Человек-техник пытается обнаружить в мире скрытое устройство, потребное для его жизни. Для автора жизненная программа имеет нетехническое, т.е. дотехническое, происхождение. Ее корни уходят вглубь, в эпоху дотехнического изобретения. Следовательно, вероятность технократии является крайне низкой: человек-техник, по определению, не может управлять, быть высшей инстанцией, его роль второстепенна. Техника предполагает наличие, с одной стороны, существа, у которого есть желание, но еще нет проекта, замысла, программы, а с другой стороны, имеются связи между развитием техники и способом бытия человека. В

этом контексте Ортега-де-Гассет рассматривает индийского бодхисатву, испанского идальго и английского джентльмена образца 1950-х гг. Бодхисатва сводит свои материальные потребности к минимуму и к технике безразличен. Активен только английский джентльмен, который стремится жить в подлинном мире максимально насыщенной жизнью. В описании автора джентльмен уверен в себе, честен, ему свойственны чувство справедливости, искренность, самообладание, ясное понимание своих прав и прав других, а также и своих обязанностей по отношению к другим. Подобный анализ был нужен для того, чтобы определить периодизацию истории техники, где существенны взаимоотношения человека к человеку и человека к технике. Автор выделяет три значительных этапа в историческом развитии техники:

1) техника случая – это исторически первая форма существования техники, присущая первобытному обществу и характерная для доисторического человека. Она отличается простотой и скудостью исполнения и крайней ограниченностью технических действий (об этом писали Л. Нуаре и др.);

2) техника ремесла – это техника Древней Греции, доимператорского Рима, европейского Средневековья. В этот период существенно расширяется набор технических действий, усвоение которых требует специальной выучки, а занятие технической деятельностью становится профессией и передается по наследству;

3) техника человека-техника – это машинная техника с техническими устройствами, которая берет свое начало со второй половины XVIII в., когда был изобретен механический ткацкий станок Эдмунда Картрайта (1743). Машина существенно меняет отношения между человеком и орудием. «Работает» машина, а человек ее обслуживает. Он – придаток машины. Побочным явлением этого процесса становятся «кризис желаний», бездуховность. Ортега-де-Гассет свое учение называет рационализмом, хотя он близок к экзистен-циализму.

Интересны технофилософские поиски Франкфуртской школы. Франкфуртская школа философии техники представлена именами известных философов *Макса Хоркхаймера* (1895–1973), *Герберта Маркузе* (1898–1979), *Теодора Адорно* (1903–1969). Школа под таким названием сложилась в 1930–1940-х гг. вокруг возглавлявшегося с 1931 г. Хоркхаймером Института социальных исследований при университете во Франкфурте-на-Майне. В связи с приходом фашистов к власти большинство сотрудников института вынуждены были эмигрировать. Теоретическое наследие представителей Франкфуртской школы связано с выработкой критической теории общества с целью совершить «высший суд над классовым подавлением» ради создания

«общества без несправедливостей». Заявленная «критическая теория общества» исходит из того, что человек как активное, творческое и свободное существо в условиях современного общества разочаровывается («аннигилируется»), лишается своего «второго измерения», каким является его духовность. Вместе с этим он теряет и свою самость и спонтанность своего существования, отчуждается от самого себя, своей подлинной сущности.

Определенную долю ответственности за эти процессы несет техника. Она выступает генератором массовой культуры, лишенной духовности, рассчитанной на усредненение культурных образцов, т.е. их удешевление, на «массовый обман» (Т. Адорно).

По мнению Теодора Адорно, в так называемой «массовой» культуре теряются уникальность, самостоятельность человека, происходят унификация всех людей, превращение их в серую некритическую массу. Причем ограничивается вся культура, проектируется историческая тотальность, к человеку предъявляются требования, порабощающие его. При этом поработителем выступает не техника, а ее хозяин. Адорно исходит из того, что никоим образом нельзя противопоставлять технику и гуманизм: такое противопоставление – продукт ложного сознания. Можно сказать, что сам разрыв между техникой и гуманизмом, каким бы ни оказался неизлечимым, является образчиком созданной обществом видимости, пишет Адорно.

Философа интересует вопрос, каким образом следует приобщать техников к философии техники? Отвечая на него, он отвергает бытовавшую в то время мысль о том, что предмет преподается им как бы извне. Он предлагает апеллировать к самосознанию: «С помощью наших понятийных средств мы должны побудить их к этому самосознанию». Но на этом пути нас встречают трудности, такие, как «профессиональная ограниченность, патриотизм», чувство отторжения гуманитарного знания.

Адорно замечает, что «техники труднее воспринимают культуру», так как предпочитают расслабление делом, «не позволяют пичкать себя массовой продукцией, которую поставляет индустрия культуры». С другой стороны, техники страдают из-за односторонности, сухости, нечеловеческого характера своей рациональности. В книге *«О технике и гуманизме»* Адорно ставит вопрос об ответственности техников за плоды своего труда. По мнению философа, при решении данного вопроса необходимо исходить из того, что каждый из нас может оказаться не самим собой, а только носителем специально предписанных функций.

Та область, которую обычно называют этикой, лишь опосредованно проникает в то, что выполняется на работе. Адорно отвергает возможность существования моральных норм, препятствующих познанию. Согласно

Адорно, противоречие между общественным и техническим разумом нельзя игнорировать, от него невозможно просто открещиваться, его необходимо предметно решать. В конечном счете вопрос о том, принесет ли современная техника пользу или вред человечеству, зависит «не от техников и даже не от самой техники, а от того, как она используется обществом». Может оказаться, что в определении социальной роли техники наиболее четкие мысли содержатся в марксистской оценке техники.

В данном аспекте весьма интересны суждения Адорно, касающиеся проблемы «нового идеала образования». Он считает, что этот идеал разрушен, что культуре не удалось создать свою собственную человечность. Культура расплачивается за неистинность, за видимость, оторванность от гуманистической идеи, что «люди сбрасывают с себя культуру». Адорно был прекрасным музыкантом, литератором, социологом. В книге «Негативная диалектика», не претендуя на создание принципиально новой философской методологии, он пытался показать на примере своих творческих интересов анатомию жизни.

Основной вклад Адорно в философию состоит в его эстетических воззрениях, в которых он рассматривает опыт постижения индивидуального, нетождественного. В философско-эстетической концепции новой музыки как протокольной фиксации «непросветленного страдания» в противоположность гармоническому преобразованию страстей, характерному для классики, Адорно ориентируется на творчество композиторов «новой венской школы». Концепция «новой музыки» у него тесно связана с критикой массовой стандартизованной современной культуры и формирующегося в ее лоне «регрессивного слышания», которое диссоциирует восприятие на стереотипные элементы. Работы Адорно оказали положительное влияние на эстетику, музыковедение, идеологию молодежных движений своего времени.

Проблемы социализации личности в Германии активно рассматривались в работах другого представителя Франкфуртской школы, философа и социолога *Юргена Хабермаса* (р. 1929). Последователь Т. Адорно, сторонник раннебуржуазных просветительских идей, идеолог студенческих движений 1960-х гг., формирования правового государства в послевоенной Германии, Хабермас считается видным представителем «второго поколения» теоретиков Франкфуртской школы. Исходя из концепции «свободы и коммуникативного действия», он формирует свое негативное отношение к западной философии техники, склонной к технократическому мышлению. Хабермас придерживается концепции, согласно которой техника объявляется силой, отнимающей у человека его свободный творческий дух, лишающей его возможности свободного действия, самовыражения и самоорганизации и, в

конечном счете, обращающей его в раба собственных творений. Эмансипацию человека Хабермас связывает с вытеснением «инструментального разума», подчинением его человеческому разуму как целостности, объединяющей индивидуальный и общественный разум. Он связывает ее с установлением «коммуникативной демократии», сочетающей научно-технической прогресс с ценностями и нормами человеческой цивилизации, «лингвистическим поворотом» в философии и социальных науках, который влечет за собой отказ от субъективистской феноменологии, основанной на анализе внутреннего сознания времени. Рациональность сосредоточивается не в сфере разума, а в языковых формах взаимопонимания. Коммуникативную парадигму Хабермас противопоставляет производственной парадигме марксизма. Исследования теории коммуникативного действия он проводит по пяти основным направлениям.

Во-первых, им предлагается новая теория общества, отличная от проекта Адорно и Хоркхаймера. Во-вторых, разрабатывается концепция коммуникативной рациональности средствами герменевтики, различных теорий языка. В-третьих, ведется разработка теории социального (коммуникативного) действия. В-четвертых, проводятся исследования на основе новых понятий «жизненного мира» и «системы» с анализом их взаимоотношения в исторической перспективе. Наконец, в-пятых, с помощью этих понятий анализируются тенденции и кризисы современности.

Отношение Хабермаса к теории К. Маркса с годами изменилось от восторженного до критического. Маркс усматривал в капитализме черты политизированного общества, основанного на коллективном труде. Социализм, согласно Марксу, должен развиваться стабильно благодаря системному управлению. Однако, по мнению Хабермаса, вне поля зрения Маркса остались проблемы, касающиеся формы коммуникации, но именно они дают ключ к разумному переустройству общества. Исправить теорию Маркса Хабермас попытался с помощью своей концепции «коммуникативного действия». Хабермас критикует и теорию Т. Адорно как имеющую пессимистическую окраску и неплодотворную, неспособную к преодолению существующих в обществе противоречий. Как пишет Хабермас, Адорно и Хоркхаймер пытались придуманный ими «инструментальный разум» спасти с помощью самого инструментального разума, т.е. навязать ему задачи, заведомо непосильные. В своем двухтомнике *«Теория коммуникативного действия»* Хабермас уповает на правила коммуникаций в условиях речевого действия, разговора, дискуссии, дискурса. В его понимании дискурс – это больше, чем свободный разговор. Это – диалог на основе нормативного высказывания на уровне высокой теоретической зрелости, т.е. разговор «совершеннолетних», с

участием как можно большего количества народа. Подобный диалог, дискурс, как общение врача с пациентом, должен привести к излечению от недугов. По мнению автора, такой дискурс является образцом, моделью, для выработки коммуникативной компетенции.

Все философские интерпретации техники и технологии можно поместить между двумя полюсами: на одном – безудержный оптимизм, на другом – безнадежный пессимизм. Концептуально-методологическое оформление оптимистического толкования техники дает нам *технологический детерминизм*. Его соседями в этом случае оказываются экономический, геополитический, демографический и прочие столь же односторонние виды детерминизма.

Технологический детерминизм – это попытка объяснить все общественные явления, в особенности же логику, глубинные токи, тенденции и конфигурации истории в терминах техники и технологии. Техника и технология наделяются всепроникающей силой, им приписываются невероятные – демиургические – возможности и перспективы. На что опирается, чем питается технологический детерминизм? Сразу скажем, что беспочвенным его назвать никак нельзя. В самом деле, техника и технология составляют ядро такого фактора производства, как капитал (заметим, в его реальном, а не денежном выражении). От техники в решающей степени зависят содержание и характер труда, его производительность, эффективность, историческая действенность. Техника и технология существенным образом влияют на социальную динамику и сдвиги власти в обществе. Технико-технологическая компетентность работает сегодня на повышение социального статуса человека.

Техническая детерминация прорывается также в сферу культуры. Не всегда удается противостоять ей даже изящным искусствам. Как заметил Ж. Эллюль, «художник уже не может оставаться творцом перед реальностью этого колоссального продуцирования вещей, материалов, товаров, потребностей, символов, выбрасываемых ежедневно техническим производством. Теперешнее искусство – отражение технической реальности». Техника с ее динамизмом и страстью к инновациям превращает изменение в архетип социального бытия людей. Таким образом, и технология – важный, если не важнейший, фактор социального прогресса; незаменимы они и в конституировании общественной целостности (общества как целого). К. Маркс справедливо отмечал, что ручная мельница дает общество с сюзереном во главе, а паровая мельница – общество с промышленным капиталом. Оценивая, следует сказать, что технологический детерминизм явно преувеличивает силу и возможности техники. Она социальна по определению и раскрывает свой потенциал в социальном же контексте. А социальный контекст – это логика целого (техника – его часть), потребности и интересы человека, принимающие форму целевых ориентиров и социальных заказов для техники, ценностно-нормативная система и многие другие издержки и противовесы, ограничивающие детерминационные

аппетиты техники. Так что «гулять самой по себе» технике никак нельзя, вернее, не получится.

Как идеология технологический детерминизм – это технократизм и в теории, и на практике. Но прежде всего, конечно, в теории – на то он и идеология. Идеология фетишистского преклонения перед техникой, целерационального, линейно-объектного отношения к человеку, функциональной оптимизации отношений между людьми, аналитико-механического понимания общества и его институтов, исключительно рационального управления социумом и т. д. Технократизм делает человека объектом калькуляции или превращает технику из средства в цель, навязывает жизни определенно технические цели. Выражаясь патетически, можно, вслед за Бердяевым, сказать, что технократическая идеология есть стремление «заменить в человеке образ и подобие Божие образом и подобием машины».

У технократизма нет перспективы. Под давлением антропо- и культурогенных процессов современности, гуманистических запросов нового, XXI в., он будет медленно, но неуклонно сдавать свои позиции, отступать, уходить в прошлое. Будущее все-таки за человеком, а не машиной.

Вопросы для самоконтроля

1. Как понимали технику древние философы? Каковы значения античного понятия «технэ»?
2.Когда и где зарождается философия техники?
3. Какова сущность технического отношения к миру?
4. Как определить понятие «технология»
5. Каковы основные направления философии техники?
6. В чем состоит принцип органопроекции?
7. Каковы по П.К. Энгельмейеру основные задачи философии техники?
8. Каково соотношение науки и техники?
9. В чем состоит социальная оценка техники?
10. Каковы основные черты информационного общества?
11. Технократические концепции в социальной философии XX века.
12.. Феномен техники в осмыслении русской классической философии.
13.Образы техники в культуре: традиционная и проектная культуры.

Список рекомендуемой литературы

Основная

1.Канке, В. А. Философские проблемы науки и техники: учебник и практикум для магистратуры / В. А. Канке. — М. : Издательство Юрайт, 2016. — 288 с.

3.Кошарный В.П. Философия науки и техники: учебное пособие. Пенза, 2012.- 294 с.

4. Прытков В.П. Философские проблемы науки и техники: учебное пособие/ В.П. Прытков.- Екатеринбург: Изд-во Урал. Ун-та, 2013.-63 с.8

Дополнительная

5. Аль-Ани Н.М. Философия техники: очерки истории и теории: учебное пособие.-СПб.: А-принт, 2004.-184 с.

6. Горохов В.Г., Розин В.М. Введение в философию техники: Учебное пособие.-М.: Инфра-М, 1998.-224 с.

7. Горохов В.Г. Техника и культура: возникновение техники и теории технического творчества в России и Германии в конце XIX –начале XX столетия.-М., 2009.-376 с.

8. Горохов В.Г. Основы философии техники и технических наук. Учебник. М.: Гардарика, 2007.-335 с.

9. Митчем К. Что такое философия техники?- М.: Аспект-пресс, 1995.- 149 с.

10. Поносов, Ф.Н. Современные философские проблемы техники и технических наук : учебное пособие / Ф.Н. Поносов. – Ижевск : ФГБОУ ВПО Ижевская ГСХА, 2013. – 262 с.

11. Попкова Н.В. Философия техносферы. – М.: URSS; ЛКИ, 2008. - 344 с**.**

12. Современные философские проблемы естественных, технических и социально-гуманитарных наук/под. ред. В.В.Миронова.-М.:Гардарики, 2006.-639 с.

13. Степин В. С. Философия науки и техники / В. С. Степин, В. Г. Горохов, М. А. Розов. – М., 1995.- 372 с.

14. Философия математики и технических наук / Под общ. ред. С.А. Лебедева: Учебное пособие для вузов.- М., 2006.-779 с

15. Философия техники: история и современность/Ред. Розин В.М.-М.: Изд-во: ИФ РАН, 1997.- 283 с.

16. Шаповалов В.Ф. Философия науки и техники: Учебное пособие. – М.: ФАИР-ПРЕСС, 2004. – 320 с.

Тема 7. Наука и техника. Специфика технического знания

В современной литературе по философии техники можно выделить следующие основные подходы к решению проблемы изменения соотношения науки и техники:

1) техника рассматривается как прикладная наука;

2) процессы развития науки и техники рассматриваются как автономные, но скоординированные процессы;

3) современная наука рассматривается в ее ориентации на развитие техники.

Первоначально, с момента возникновения техники, в узком значении этого термина, – техника действительно во многом выполняла функцию применения знаний, открытых в науке. Однако со временем, по мере накопления собственно технических знаний, техника и технические науки образовали собственную сферу. В силу быстрого развития последней может сложиться впечатление, что современная наука служит технике. Однако это не совсем так, скорее можно говорить о взаимовлиянии данных областей человеческой деятельности. Современная техника немыслима без глубоких теоретических исследований, которые проводятся сегодня не только в естественных, но и в особых, технических, науках.

Если вплоть до конца XIX в. регулярного применения научных знаний в технической практике не было, то теперь это стало повседневной практикой. Начиная с XIX в. наблюдается «сциентизация техники», сопровождающаяся «технизацией науки». В целом выделяют следующие этапы взаимодействия науки и техники, приведшие к развитию технических наук:

– в первый период (донаучный) последовательно формируются три типа технических знаний: *практико-методические, технологические и конструктивно-технические*;

– во втором периоде происходит зарождение технических наук (со второй половины XVIII в. до 70-х гг. XIX в.): во-первых, формирование научно-технических знаний на основе использования в инженерной практике знаний *естественных наук* и, во-вторых, появление первых *технических наук*;

– третий период – классический (до середины XIX в.) – характеризуется построением ряда *фундаментальных технических теорий*;

– для четвертого этапа (настоящее время) характерны осуществление *комплексных исследований*, *интеграция технических наук* не только с естественными, но и с общественными науками.

Выявление специфики технических наук обычно осуществляется на основе их сопоставления с другими науками. К настоящему времени чаще всего выделяются *естественные, гуманитарные, математические и технические науки,* рассматриваемые как равноправные партнеры. Наиболее тесная связь наблюдается между техническими и естественными науками. Сегодня никого не удивит тот факт, что исследования, которые проводятся в промышленных лабораториях исследователями, получившими инженерное образование, приводят к важным научным прорывам или что ученые, работающие в университетах или академических центрах, приходят к важным технологическим открытиям. Поэтому технические науки должны в

полной мере рассматриваться как самостоятельные научные дисциплины наряду с общественными, естественными и математическими науками.

Технические и естественные науки имеют одну и ту же предметную область, они могут исследовать одни и те же объекты, но различным образом.

Технические явления в экспериментальном оборудовании естественных наук играют решающую роль, а большинство физических экспериментов являются искусственно созданными ситуациями. Объекты технических наук также представляют собой своеобразный синтез «естественного» и «искусственного». Искусственность объектов технических наук заключается в том, что они являются продуктами сознательной целенаправленной человеческой деятельности. Их естественность обнаруживается, прежде всего, в том, что все искусственные объекты в конечном итоге создаются из естественного (природного) материала. Осуществление эксперимента – это деятельность по производству технических эффектов и может быть отчасти квалифицирована как инженерная, т.е. как конструирование машин, как попытка создать искусственные процессы и состояния, однако с целью получения новых научных знаний о природе или подтверждения научных законов, а не исследования закономерностей функционирования и создания самих технических устройств.

К началу XX столетия технические науки составили сложную систему знаний – от систематических наук до собрания правил в инженерных руководствах. К этому времени технические науки, выросшие из практики, приняли качество подлинной науки, признаками которой являются систематическая организация знаний, опора на эксперимент и построение математизированных теорий. В технических науках появились также особые фундаментальные исследования.

Таким образом, естественные и технические науки – равноправные партнеры. В то же время нельзя не видеть, что в технических науках все заимствованные из естествознания элементы претерпели существенную трансформацию, в результате чего и возник новый тип организации теоретического знания. Кроме того, технические науки со своей стороны в значительной степени стимулируют развитие естественных наук, оказывая на них обратное воздействие.

Однако сегодня такой констатации уже недостаточно. Для определения специфики технического знания и технических наук необходимо анализировать их строение. В настоящее время научно-технические дисциплины представляют собой широкий спектр различных дисциплин – от самых абстрактных до весьма специализированных, которые ориентируются на использование знаний

не только естественных наук (физики, химии, биологии и т.д.), но и общественных (например экономики, социологии, психологии и т.п.). Относительно некоторых научно-технических дисциплин вообще трудно сказать, принадлежат они к чисто техническим наукам или представляют какое-то новое, более сложное единство науки и техники. Кроме того, некоторые части технических наук могут иметь характер фундаментального, а другие – прикладного исследования.

Прикладное исследование – это такое исследование, результаты которого адресованы производителям и заказчикам и которое направляется нуждами или желаниями этих клиентов. *Фундаментальное* – адресовано другим членам научного сообщества. Современная техника является не только применением существующего научного знания, но и имеет творческий компонент. Поэтому в методологическом плане исследование в технической науке не очень сильно отличается от общенаучного. Для современной инженерной деятельности требуются не только краткосрочные исследования, направленные на решение специальных задач, но и широкая долговременная программа фундаментальных исследований в лабораториях и институтах, специально предназначенных для развития технических наук.

Для современного этапа развития науки и техники характерно использование методов фундаментальных исследований для решения прикладных проблем. Тот факт, что исследование является фундаментальным, еще не означает, что его результаты не имеют непосредственного практического значения. Работа же, направленная на прикладные цели, может быть фундаментальной.

В научно-технических дисциплинах необходимо четко различать исследования, включенные в непосредственную инженерную деятельность и теоретические исследования, которые приводят к созданию *технической теории*.

Техническая теория имеет дело с более сложной реальностью, поскольку не может исключить сложное взаимодействие физических факторов, имеющих место в машине. Техническая теория является менее абстрактной и идеализированной, она более тесно связана с реальным миром инженерии.

За последние десятилетия возникло множество технических теорий, основывающихся не только на физике, и они могут быть названы абстрактными техническими теориями (например, системотехника, информатика или теория проектирования), для которых характерно включение в фундаментальные инженерные исследования общей методологии. Для трактовки отдельных сложных явлений в технических

разработках могут быть привлечены часто совершенно различные, логически не связанные теории. Такие теоретические исследования становятся по своей сути комплексными и непосредственно выходят не только в сферу «природы», но и в сферу «культуры». Необходимо брать в расчет не только взаимодействие технических разработок с экономическими факторами, но и связь техники с культурными традициями, а также психологическими, историческими и политическими факторами. Таким образом, мы попадаем в сферу анализа социального контекста научно-технических знаний.

Первые технические теории строились по образцу физических, в которых, наряду с концептуальным и математическим аппаратом, важную роль играют теоретические схемы, образующие своеобразный «внутренний скелет» теории.

Теоретические схемы представляют собой совокупность абстрактных объектов, ориентированных, с одной стороны, на применение соответствующего математического аппарата, а с другой – на мысленный эксперимент, т.е. на проектирование возможных экспериментальных ситуаций. Они представляют собой особые идеализированные представления (теоретические модели), которые часто выражаются графически. Примером их могут быть электрические и магнитные силовые линии, введенные М. Фарадеем в качестве схемы электромагнитных взаимодействий. Г. Герц использовал и развил далее эту теоретическую схему Фарадея для осуществления и описания своих знаменитых опытов. В научном поиске всегда имеются подобные идеализированные представления объекта исследования. В технической же теории такого рода графические изображения играют еще более существенную роль. Теоретические схемы выражают особое видение мира под определенным углом зрения, заданным в данной теории. Галилей, проверяя закон свободного падения тел, выбрал для бросаемого шарика очень твердый материал, что позволяло практически пренебречь его деформацией. Стремясь устранить трение на наклонной плоскости, он оклеил ее отполированным пергаментом. В качестве теоретической схемы подобным образом технически изготовленный объект представлял собой наклонную плоскость, т.е. абстрактный объект, соответствующий некоторому классу реальных объектов, для которых можно пренебречь трением и упругой деформацией. Одновременно он представлял собой *объект оперирования*, замещающий в определенном отношении реальный объект, с которым осуществлялись различные математические действия и преобразования.

Таким образом, абстрактные объекты, входящие в состав теоретических схем математизированных теорий, представляют собой

результат идеализации и схематизации экспериментальных объектов или более широко – любых объектов предметно-орудийной (в том числе инженерной) деятельности.

Особенность технических наук заключается в том, что инженерная деятельность, как правило, заменяет эксперимент. Именно в инженерной деятельности проверяется адекватность теоретических выводов технической теории и черпается новый эмпирический материал.

Специфика технической теории состоит в том, что она ориентирована на конструирование технических систем. Научные знания и законы, полученные естественнонаучной теорией, требуют еще длительной «доводки» для применения их к решению практических инженерных задач, в чем и состоит одна из функций технической теории.

Теоретические знания в технических науках должны быть обязательно доведены до уровня практических инженерных рекомендаций. Выполнению этой задачи служат в технической теории правила соответствия, перехода от одних модельных уровней к другим, а проблема интерпретации и эмпирического обоснования в технической науке формулируется как задача реализации. Поэтому в технической теории важную роль играет разработка особых операций перенесения теоретических результатов в область инженерной практики.

Эмпирический уровень технической теории образуют конструктивно-технические, технологические и практико-методические знания, являющиеся результатом обобщения практического опыта при проектировании, изготовлении, отладке и т.д. технических систем. Это эвристические методы и приемы, разработанные в самой инженерной практике, но рассмотренные в качестве эмпирического базиса технической теории.

Конструктивно-технические знания преимущественно ориентированы на описание строения (или конструкции) технических систем, представляющих собой совокупность элементов, имеющих определенную форму, свойства и способ соединения. Они включают также знания о технических процессах и параметрах функционирования этих систем.

Технологические знания фиксируют методы создания технических систем и принципы их использования.

Практико-методические знания представляют собой практические рекомендации по применению научных знаний, полученных в технической теории, в практике инженерного проектирования. Это фактически те же самые технологические и конструктивно-технические знания, только являющиеся уже не результатом обобщения

практического опыта инженерной работы, а продуктом теоретической деятельности в области технической науки, и поэтому сформулированы в виде рекомендаций для еще неосуществленной инженерной деятельности. В них также формулируются задачи, стимулирующие развитие технической теории.

Теоретический уровень научно-технического знания включает в себя три основных уровня, или слоя, теоретических схем: функциональные, поточные и структурные.

Функциональная схема фиксирует общее представление о технической системе, независимо от способа ее реализации, и является результатом идеализации технической системы на основе принципов определенной технической теории. Такие схемы совпадают для целого класса технических систем. Блоки этой схемы фиксируют только те свойства элементов технической системы, ради которых они включены в нее для выполнения общей цели. Каждый элемент в системе выполняет определенную функцию. Как правило, функциональные элементы выражают обобщенные математические операции, а функциональные связи, или отношения, между ними – определенные математические зависимости.

Функциональные схемы, например в теории электрических цепей, представляют собой графическую форму математического описания состояния электрической цепи. Каждому функциональному элементу такой схемы соответствует определенное математическое соотношение, например, между силой тока и напряжением на некотором участке цепи, или вполне определенная математическая операция (дифференцирование, интегрирование и т.п.). Порядок расположения и характеристики функциональных элементов адекватны электрической схеме.

Однако функциональные схемы могут быть и не замкнуты на конкретном математическом аппарате. В этом случае они выражаются в виде простой декомпозиции взаимосвязанных функций, направленных на выполнение общей цели, предписанной данной технической системе. С помощью такой функциональной схемы строится алгоритм функционирования системы и выбирается ее конфигурация (внутренняя структура).

Поточная схема, или схема функционирования, описывает естественные процессы, протекающие в технической системе и связывающие ее элементы в единое целое. Блоки таких схем отражают различные действия, выполняемые над естественным процессом элементами технической системы в ходе ее функционирования. Такие

схемы строятся исходя из естественнонаучных (например физических) представлений.

Теория электрических цепей, к примеру, имеет дело не с огромным разнообразием конструктивных элементов электротехнической системы, отличающихся своими характеристиками, принципом действия, конструктивным оформлением и т.д., а со сравнительно небольшим количеством идеальных элементов и их соединений, представляющих эти идеальные элементы на теоретическом уровне.
К таким элементам относятся прежде всего емкость, индуктивность, сопротивление, источники тока и напряжения.

Для применения математического аппарата требуется дальнейшая идеализация: каждый из перечисленных выше элементов может быть рассмотрен как активный (идеальные источники тока или напряжения) или пассивный (комплексное – линейное омическое и нелинейные – индуктивное и емкостное – сопротивления) двухполюсник, т.е. участок цепи с двумя полюсами, к которым приложена разность потенциалов и через которую течет электрический ток. Все элементы электрической цепи должны быть приведены к указанному виду. Причем в зависимости от режима функционирования технической системы одна и та же схема может принять различный вид. Режим функционирования технической системы определяется прежде всего тем, какой естественный (в данном случае физический) процесс в ней протекает, т.е. например, какой электрический ток (постоянный или переменный, периодический или непериодический и т.д.) течет через цепь. В зависимости от этого и элементы цепи на схеме функционирования меняют вид: например, индуктивность представляется идеальным омическим сопротивлением при постоянном токе, при переменном токе низкой частоты – последовательно соединенными идеальными омическим сопротивлением и индуктивностью (индуктивным сопротивлением), а при переменном токе высокой частоты ее поточная схема дополняется параллельно присоединяемым идеальным элементом емкости (емкостным сопротивлением). Для каждого вида естественного (физического) процесса применяется наиболее адекватный ему математический аппарат, призванный обеспечить эффективный анализ поточной схемы технической системы в данном режиме ее функционирования. Заметим, что для разных режимов функционирования технической системы может быть построено несколько поточных и функциональных схем. В предельно общем случае поточные схемы отображают не только естественные процессы, но и вообще любые потоки субстанции (вещества, энергии, информации). Причем в частном случае эти процессы могут быть

редуцированы к стационарным состояниям, но последние могут рассматриваться как вырожденный частный случай процесса.

Структурная схема технической системы фиксирует те узловые точки, на которых замыкаются потоки (процессы функционирования). Это могут быть единицы оборудования, детали или даже целые технические комплексы, представляющие собой конструктивные элементы различного уровня, входящие в данную техническую систему, которые могут отличаться по принципу действия, техническому исполнению и ряду других характеристик. Такие элементы обладают, кроме функциональных свойств, свойствами второго порядка, т.е. теми, которые привносят с собой в систему определенным образом реализованные элементы, в том числе и нежелательные (например усилитель – искажения усиливаемого сигнала). Структурная схема фиксирует конструктивное расположение элементов и связей (т.е. структуру) данной технической системы и уже предполагает определенный способ ее реализации. Такие схемы, однако, сами уже являются результатом некоторой идеализации, отображают структуру технической системы, но не являются ни ее скрупулезным описанием в целях воспроизведения, ни ее техническим проектом, по которому может быть построена такая система. Это – пока еще теоретический набросок структуры будущей технической системы, который может помочь разработать ее проект.

Структурные схемы в классических технических науках отображают в технической теории именно конструкцию технической системы и ее технические характеристики. В этом случае они позволяют перейти от естественного модуса рассмотрения технической системы, который фиксируется в его поточной схеме (в частности, физического процесса), к искусственному модусу. Поэтому в частном случае структурная схема в идеализированной форме отображает техническую реализацию физического процесса. В классической технической науке такая реализация, во-первых, является всегда технической и, во-вторых, осуществляется всегда в контексте определенного типа инженерной деятельности и вида производства. В современных человеко-машинных системах такая реализация может быть самой различной, в том числе и нетехнической. В этом случае наряду с терминами «технические параметры», «конструкция» и т.п. используются термины из гуманитарных и социально-экономических наук.

Таким образом, в технической теории на материале одной и той же технической системы строится несколько оперативных пространств, которым соответствуют различные теоретические схемы. В каждом таком «пространстве» используются разные абстрактные объекты и

средства оперирования с ними, решаются особые задачи. Механизмы взаимодействия этих оперативных пространств могут быть раскрыты в результате анализа функционирования технической теории.

Вопросы для самоконтроля

1. Раскройте соотношение естествознания и техники, научного и технического знания.

2. В чем состоят особенности теоретико-методологического синтеза знаний в технических науках?

3. Какова структура технической теории?

4. Технические науки и прикладное естествознание. Специфика технической рациональности.

5. Инженерные исследования и проектирование. Философско-методологические аспекты проектирования сложных антропотехнических комплексов.

6. Особенности методологии технических наук и методологии проектирования.

7. Роль техники в становлении классического математизированного и экспериментального естествознания и в современном неклассическом естествознании.

8. Специфика соотношения теоретического и эмпирического в технических науках, особенности теоретико-методологического синтеза знаний в технических науках.

9. Техническая теория: специфика построения, особенности функционирования и этапы формирования.
10. Дисциплинарная организация технической науки: понятие научно-технической дисциплины и семейства научно-технических дисциплин.
11.. Сущность классических научно-технических дисциплин.

12. Сущность современных (неклассических) научно-технических дисциплин.
13. Особенности теоретических исследований в современных (неклассических) научно-технических дисциплинах.
14. Развитие системных и кибернетических представлений в технике.

Список рекомендуемой литературы

Основная

1. Канке, В. А. Философские проблемы науки и техники: учебник и практикум для магистратуры / В. А. Канке. — М. : Издательство Юрайт, 2016. — 288 с.
2. Кошарный В.П. Философия науки и техники: учебное пособие. Пенза, 2012.- 294 с.

3.Прытков В.П. Философские проблемы науки и техники: учебное пособие/ В.П. Прытков.- Екатеринбург: Изд-во Урал. Ун-та, 2013.-63 с.

Литература дополнительная:

4. Аль-Ани Н.М. Философия техники: очерки истории и теории: учебное пособие.-СПб.: А-принт, 2004.-184 с.

5.Горохов В.Г., Розин В.М. Введение в философию техники: Учебное пособие.-М.: Инфра-М, 1998.-224 с.

6.Горохов В.Г. Техника и культура: возникновение техники и теории технического творчества в России и Германии в конце XIX –начале XX столетия.-М., 2009.-376 с.

7.Горохов В.Г. Основы философии техники и технических наук. Учебник. М.: Гардарика, 2007.-335 с.

8.Заблуждающийся разум? Многообразие вненаучного знания/Отв. Ред. И.Т. Касавин – М.: Политиздат, 1990.-404 с.

9. История информатики и философия информационной реальности: Учебное пособие для вузов/Под ред. Р.М. Юсупова, В.П, Котенко.-М.: Академический проект, 2007- 429 с.

10. Микешина Л.А. Диалог когнитивных практик. Из истории эпистемологии и философии науки. М.: РОССПЭН, 2010.-575 с.

11. Митчем К. Что такое философия техники?- М.: Аспект-пресс, 1995.- 149 с.

12. Новая технократическая волна на Западе/ Гуревич П. С. (отв. ред.). Издательство: М.: Прогресс .- 1986 - 453 с.

13. Поносов, Ф.Н. Современные философские проблемы техники и технических наук : учебное пособие / Ф.Н. Поносов. – Ижевск : ФГБОУ ВПО Ижевская ГСХА, 2013. – 262 с

14. Попкова Н.В. Философия техносферы. – М.: URSS; ЛКИ, 2008. - 344 с.

15.Современная философия науки: Хрестоматия/Сост. А.А. Печенкин. – М., 1994.- 252 с.

16. Современные философские проблемы естественных, технических и социально-гуманитарных наук/под. ред. В.В.Миронова.-М.:Гардарики, 2006.- 639 с.

15.Степин В.С. История и философия науки: учебник. М.: Академический проект. 2011.-423 с.

17. Степин В. С. Философия науки и техники / В. С. Степин, В. Г. Горохов, М. А. Розов. – М., 1995.- 372 с.

18. Философия математики и технических наук / Под общ. ред. С.А. Лебедева: Учебное пособие для вузов.- М., 2006.-779 с

16.

19. Философия техники в ФРГ/ Арзаканян Ц.Г., Горохов В.Г. (сост.).-М.: Прогресс, 1989.

20. Философия техники: история и современность/Ред. Розин В.М.-М.: Изд-во: ИФ РАН, 1997.- 283 с.

22. Шаповалов В.Ф. Философия науки и техники: Учебное пособие. – М.: ФАИР-ПРЕСС, 2004. – 320 с.

Тема 8. Философские проблемы информатики

Развитие компьютерной техники, все более широкое ее использование в различных областях, формирование новых научных дисциплин, связанных с автоматизированной обработкой информации, способствуют осознанию новых вопросов, касающихся человеческого знания, роли знания в жизни общества, видов знания и способов его существования, вопросов, касающихся того, что может быть названо теоретико-познавательным контекстом компьютерной революции.

Человеческое познание, мышление, знание, разум в течение многих веков были предметом философского исследования. С появлением кибернетики, компьютеров и компьютерных систем, которые стали называть интеллектуальными системами, с развитием учения об искусственном интеллекте, мышление, знание стали предметом интереса математических и инженерно-технических дисциплин. Это побудило людей по-новому взглянуть на ряд традиционных гносеологических проблем, наметить новые пути их исследования, обратить внимание на многие, остававшиеся ранее в тени аспекты познавательной деятельности, механизмов и результатов познания.

В ходе бурных споров 1960–70-х гг. на тему «Может ли машина мыслить?» были представлены различные варианты ответа на вопрос о том, кто может быть субъектом познания: только ли человек (или также машина) может считаться субъектом мыслящим, обладающим интеллектом и, следовательно, познающим. Сторонники последнего варианта пытались сформулировать такое определение мышления, которое позволяло бы говорить о наличии мышления у машины, например, мышление определялось как решение задач. Противники защитников идеи «компьютерного мышления», напротив, стремились выявить такие характеристики мыслительной деятельности человека, которые никак не могут быть приписаны компьютеру и отсутствие которых не позволяет говорить о мышлении в полном смысле этого слова. К числу таких характеристик относили, например, способность к творчеству, эмоциональность.

Компьютерное моделирование мышления дало мощный толчок исследованиям механизмов познавательной деятельности в рамках такого направления, как когнитивная психология. Здесь утвердилась мысль о

необходимости изучения познавательной деятельности человека по аналогии с переработкой информации на компьютере. Исследуя устройство человеческой памяти, например, стали различать, по аналогии с компьютерной системой, долгосрочную и оперативную (кратковременную) память. Вообще на этом пути были получены ценные результаты, обогатившие наши представления о человеческом мышлении и механизмах его функционирования.

Компьютерное моделирование мышления, использование методов математических и технических наук в его исследовании породили надежды на создание в скором будущем строгих теорий мышления, столь полно описывающих данный предмет, что это сделает излишними всякие философские спекуляции по его поводу. Мечтам такого рода, однако же, не суждено было сбыться, и сегодня мышление, будучи предметом изучения ряда частных наук (психологии, логики, теорий искусственного интеллекта, когнитивной лингвистики), остается также важнейшим объектом философских размышлений.

В последние два десятилетия в компьютерных науках заметное внимание стали уделять такому традиционно входившему в сферу философии предмету, как знание (эпистема – от греч. *episteme* – знание). Слово «знание» стало использоваться в названиях направлений и составляющих компьютерных систем, а также самих систем (системы, основанные на знаниях; базы знаний и банки знаний; представление, приобретение и использование знаний, инженерия знаний). Тема «компьютер и знание» стала предметом обсуждения и в значительно более широком контексте, где на первый план вышли ее философско-эпистемологические, социальные и технологические аспекты.

Подходы к исследованию знания и познания условно можно разделить на две большие группы: экзистенциальные и технологические. Последние стали доминировать со второй половины XX столетия. В 1980-е гг. в области разработки искусственного интеллекта понятие знания потеснило понятия мышления и интеллекта. Теория искусственного интеллекта стала иногда характеризоваться как «наука о знаниях, о том, как их добывать, представлять в искусственных системах, перерабатывать внутри системы и использовать для решения задач», а история искусственного интеллекта, исключая ее ранние этапы, – как история исследований методов представления знаний.

Расширение сферы применения искусственного интеллекта, переход к таким более сложным областям, как медицина, геология и химия, потребовали интенсивных усилий по формализации соответствующих знаний. Создатели искусственного интеллекта столкнулись с

необходимостью выявить, упорядочить разнообразные данные, сведения эмпирического характера, теоретические положения и эвристические соображения из соответствующей области науки или иной профессиональной деятельности и задать способы их обработки с помощью компьютера таким образом, чтобы их можно было успешно использовать в решении задач, для которых они предназначались (поиск информации, постановка диагноза и т.д.). Это привело к изменениям в характере данных, находящихся в памяти компьютера, – они стали усложняться, появились структурированные данные – списки, документы, семантические сети, фреймы. Для элементарной обработки данных, их поиска, записи в отведенном месте и ряда других операций стали использовать специальные вспомогательные программы. Процедуры, связанные с обработкой данных, постепенно усложнялись. Появился такой компонент интеллектуальной системы, как база знаний.

Термин «знания» приобрел в концепциях искусственного интеллекта специфический смысл, который Д. А. Поспелов характеризует следующим образом. Под знаниями понимается форма представления информации в ЭВМ, которой присущи такие особенности, как: а) внутренняя интерпретируемость (когда каждая информационная единица должна иметь уникальное имя, по которому система находит ее, а также отвечает на запросы, в которых это имя упомянуто); б) структурированность (включенность одних информационных единиц в состав других); в) связность (возможность задания временных, каузальных, пространственных или иного рода отношений); г) семантическая метричность (возможность задания отношений, характеризующих ситуационную близость); д) активность (выполнение программ инициируется текущим состоянием информационной базы). Именно эти характеристики отличают знания в искусственном интеллекте от данных.

Вместе с тем приведенная характеристика знаний в искусственном интеллекте не является совершенно изолированной от того, что мы обычно понимаем под знанием. Такие черты, как внутренняя интерпретируемость, структурированность, связность, семантическая метрика и активность, присущи любым более или менее крупным блокам человеческих знаний, и в этом смысле знания в компьютерной системе можно рассматривать как модель или образ того или иного фрагмента человеческого знания.

Однако связь знаний в специфическом для искусственного интеллекта смысле со знанием в более привычном смысле не ограничивается лишь сходством некоторых структурных характеристик. Ведь значительная часть

информации, представляемой в базе знаний интеллектуальной системы, есть не что иное, как знания, накопленные в той области, где должна применяться данная система. Исследование этого знания под углом зрения задач построения искусственного интеллекта определяет технологический подход искусственного интеллекта к знанию как таковому, под которым понимается следующее:

1. Извлечение знаний из социума в ходе неформальных интервью с экспертом и анализа специальной литературы.

2. «Представление знаний» – кодирование знаний с помощью специалистов-экспертов и создание машинной модели «порождения» знаний.

3. Создание «сверхбыстрого прототипа» экспертной системы и ее последующих версий.

4. Контроль над модификациями базы знаний.

В более широком смысле технологический подход к знанию предполагает постановку, исследование и решение технологических вопросов о знании. К последним относятся вопросы типа «Каким образом следует (можно, допустимо) обращаться (иметь дело) со знанием, имея в виду достижение такой-то цели?».

Технологические вопросы о знании противопоставлены, в определенном смысле, экзистенциальным вопросам – т.е. вопросам о том, как существует знание, каково оно есть. К вопросам последнего типа относятся, например, вопросы о соотношении знания с мнением или верой, о структуре знания и его видах, об онтологии знания, о том, как происходит познание.

При самом широком истолковании технологический подход к знанию является неотъемлемым элементом жизни любого человека. В этом смысле и первобытный человек, использующий для передачи информации примитивные сигналы, и наш современник, выбирающий между почтой, телеграфом, телефоном и факсом, могут считаться решающими технологические вопросы относительно знания. Расцвет технологических исследований знания связан с развитием эпистемической логики (раздел модальной логики, исследующий логические связи высказываний) и искусственного интеллекта. Эпистемическая логика как интенсивно развивающееся направление ставит, в частности, вопросы, связанные с определением истинности знания в контексте формул, содержащих такие эпистемические операторы, которые соответствуют словам «знает», «полагает», «сомневается», «отрицает» и др.

В эпистемической логике возможны различные подходы к трактовке знания. Знания могут пониматься как только истинные; в других случаях

выделяют различные степени истинности знания (строгое, гипотетичное и др.). Однако прежде всего в учениях об искусственном интеллекте уделяют внимание не вопросу об истинности знания, а проблеме его структурно-функциональных характеристик. Поэтому, говоря о знаниях, нередко указывают на такие их черты, как структурированность, активность, наличие метапроцедур, противопоставляя в этом отношении базу знаний в компьютерной системе базе данных, компоненты которой не обладают перечисленными свойствами.

Вместе с тем правомерно поставить вопрос о более широком контексте рассмотрения искусственного интеллекта в плане соотношения знания и истины. Очевидно, что проблема оценки когнитивного статуса познавательных результатов и методов, представляемых в компьютерной системе, связана с более общей проблемой истинности знания.

Большой интерес для философско-гносеологических исследований представляют технологические подходы к знанию. Технологические вопросы об истинности знания, исследуемые в рамках искусственного интеллекта, касаются в значительной степени способов представления знаний и методов приобретения знаний.

Проблемы представления знаний связаны в значительной степени с с разработкой соответствующих языков и моделей. Существуют различные типы моделей: логические, продукционные, фреймовые, семантические сети и др. Логические модели предполагают представление знаний в виде формальных систем (теорий), и в качестве языка представления знаний в таких моделях обычно используется язык логики предикатов. Продукционные представления можно охарактеризовать (упрощенным образом) как системы правил вида «Если А, то В», или «Предпосылка – действие». Сетевые модели предполагают выделение некоторых фиксированных множеств объектов и задание отношений на них (это могут быть отношения различного рода: пространственные, временные, отношения именования и др.). Фреймовые представления иногда рассматривают как разновидность семантических сетей, однако для первых характерно наличие фиксированных структур информационных единиц, в которых определены места для имени фрейма, имен слотов и значений слотов.

Каждая из упомянутых моделей имеет свои достоинства и недостатки в отношении того или иного круга задач. Преимущества логических моделей, использующих язык логики предикатов, связаны с дедуктивными возможностями исчисления предикатов, теоретической обоснованностью выводов, осуществляемых в системе. Однако такого рода модели в сложных

предметных областях могут оказаться слишком громоздкими и недостаточно наглядными в качестве моделей предметной области или соответствующих фрагментов знания. Продукционные модели получили широкое распространение благодаря таким достоинствам, как простота формулировки отдельных правил, пополнения и модификации, а также механизма логического вывода. В качестве недостатка продукционного подхода отмечают низкую эффективность обработки информации при необходимости решения сложных задач. Преимущества семантических сетей и фреймовых моделей заключаются, с одной стороны, в их экономичности, позволяющей сократить время автоматизированного поиска информации, а с другой стороны – в их удобстве для описания определенных областей знания (и соответствующих фрагментов реальности, изучаемых в данных областях), когда выделяются основные (с точки зрения задач, для которых создается искусственный интеллект) объекты предметной области и (или) система понятий, в которых будут анализироваться конкретные ситуации, а также описываются свойства объектов (понятий) и отношения между ними. И хотя в целом для этих типов моделей существуют значительные проблемы с организацией вывода, фреймовые системы многими были оценены как перспективные благодаря возможностям подведения под них достаточно строгих логических и математических оснований. Продукционный и фреймовый подходы объединяются иногда под общим названием эвристического, или когнитивного, подхода.

Эти подходы в представлении знаний достаточно тесно связаны с развитием когнитивной психологии. Однако само это направление сложилось под влиянием «компьютерной метафоры», когда познавательные процессы стали рассматриваться по аналогии с работой вычислительных машин. Неудивительно поэтому, что происходящее в учениях об искусственном интеллекте оказывало и оказывает заметное воздействие на когнитивную психологию (как и на еще более молодое направление – когнитивную лингвистику). Это справедливо и в отношении собственно представления знаний. И фреймовые, и сетевые модели основываются на соответствующих концепциях структур человеческого восприятия и памяти. Вместе с тем эта концепция имеет самостоятельное значение, как концепции психологическая и гносеологическая, и используется в исследовании проблем, выходящих за рамки собственно разработок компьютерных систем.

В книге *«Философия компьютерной революции»* (М., 1991) А. И. Ракитов выдвигает концепцию «информационной эпистемологии». «Возникновение "интеллектуальной технологии" и жгучий интерес к природе и возможностям машинного мышления, порожденный компьютерной

революцией, – пишет он, – привели к формированию нового, нетрадиционного раздела эпистемологии – эпистемологии информационной. Она исследует не те или иные виды научного знания, а знания вообще, но под особым углом зрения, с позиции переработки и преобразования информации в ее высшую форму – знания. Информационная эпистемология исследует различные способы представления и выражения знаний и возможности построения знаний с помощью технических систем». Процесс познания и мышления, считает А. И. Ракитов, рассматривается в информационной эпистемологии под углом зрения «инженерного фундаментализма» как процесс машинной трансформации информации. К основным проблемам информационной эпистемологии он относит следующие: «что такое информация; как она передается, трансформируется; каковы функции и соотношения сигналов и кодов; какова познавательная функция компьютеров, могут ли они мыслить; как из информации создаются знания; как соотносятся информация, смысл и значение; каковы способы машинного представления знаний; какова связь информации и языка; как осуществляются машинное понимание и взаимопонимание машины и человека; можно ли редуцировать мыслительные процессы к вычислительным функциям или представить через них; в чем сущность инженерного подхода к познавательной деятельности; и, наконец, каково соотношение компьютера и мозга?».

Очевидно, что в круг перечисляемых А. И. Ракитовым проблем входят как технологические, так и экзистенциальные вопросы о знании. Соотнесенность между собой этих вопросов обусловлена тем, что все они так или иначе связаны с проблемами компьютерной переработки информации. Для осмысления происходящего в разработке компьютерных систем с позиций эпистемологии характерно также наличие тенденции к определенной трансформации эпистемологии с учетом потребностей компьютерной революции.

Исследование знания в контексте искусственного интеллекта особенно остро поставило вопрос о неявной, личностной компоненте знания, а также о культурных предпосылках общения людей при посредстве ЭВМ. Заполнение базы знаний, осуществленное инженером в результате работы с экспертом, предполагает, конечно, формулировку правил, которые входят в базу знаний и необходимы для выполнения системой ее функций. Однако существенная часть этих правил может быть недоступна самому эксперту для рефлексии, в силу их неявности, либо может быть затруднена вербализация этих правил.

В принципе утверждение о существовании невербализуемого, неявного личностного знания (М. Полани) не противоречит утверждению о

возможности вербализации или иного рода экспликации той части неявного знания, которая это допускает. Существуют две крайние точки зрения на знание, одна из которых предполагает принципиальную эксплицируемость всей познавательной деятельности человека, а другая – принципиальную неэксплицируемость того, что не эксплицировано на данный момент.

Другой важной группой технологических вопросов о знании, изучаемых при рассмотрении проблемы искусственного интеллекта, являются вопросы приобретения знаний – т.е. вопросы о способах получения и ввода в ЭВМ информации, необходимой для наполнения структур представления знаний конкретным содержанием. Источниками этой информации могут быть как тексты (книги, статьи, архивные документы или уже созданные базы знаний, которые могут считаться текстами в широком смысле этого слова), так и не зафиксированные в текстах (или даже неартикулированные) знания, которыми обладает человек (специалист, эксперт). В некоторых случаях система может приобретать знания непосредственно благодаря наблюдению за окружающей средой. Многие исследователи считают, что ключевой проблемой при построении экспертных систем является получение знаний от экспертов.

Существуют разнообразные методики так называемого извлечения знаний из эксперта. Ранее других возникшие и наиболее распространенные из них – методики интервьюирования экспертов. Режим интервью, когда инженер по знаниям ведет активный диалог с экспертом, предполагает как предварительное ознакомление его с предметной областью, для работы в которой создается система, так и ознакомление эксперта с некоторыми вопросами построения искусственного интеллекта. От интервью отличаются такие способы взаимодействия инженера по знаниям с экспертом, как протокольный анализ и игровая имитация. Протокольный анализ предполагает фиксацию действий (видеозапись) или «мыслей вслух» (запись на магнитофонную ленту) эксперта в ходе решения проблемы. Эта запись впоследствии подвергается анализу.
В случае игровой имитации инженер по знаниям наблюдает за поведением эксперта в искусственно созданных ситуациях, моделирующих те, которые действительно имеют место в работе эксперта. Однако и эти способы требуют диалога инженера с экспертом. Такой диалог бывает необходим при анализе полученной информации, для ее уточнения, восстановления картины работы эксперта в том объеме, который требуется для построения искусственного интеллекта. Работа с экспертом может быть в значительной степени автоматизирована, когда функции инженера по знаниям выполняет искусственный интеллект.

Так или иначе, методы извлечения знаний, как и методы их представления, нередко базируются на когнитивно-психологических и эпистемологических соображениях, в том числе на экзистенциальном взгляде на когнитивную структуру экспертного знания (иногда вместо выражения «экспертное знание» предпочитают употреблять выражение «опыт эксперта»).

Трудности приобретения знаний – это в значительной степени трудности изучения структуры экспертного знания и механизмов его функционирования. Размышление эксперта над собственным знанием не может решить этой проблемы, поскольку, во-первых, не все эксперты обладают достаточно развитой способностью к рефлексии, во-вторых, далеко не всегда могут осуществлять ее в тех концептуальных рамках, которые обеспечивают возможность заполнения базы знаний, и, в-третьих, известны случаи, когда эксперты в силу каких-либо соображений не желают делиться информацией. Кроме того, имеются трудности принципиального характера, связанные с вербализацией неявного знания. Некоторые исследователи придают большое значение интуиции эксперта-человека, возможностям ее учета или «компенсирования» при разработке искусственного интеллекта. Проблема «знание и компьютер», таким образом, оказывается связанной с вопросами явного и неявного, вербализуемого и невербализуемого знания, а также с проблемой интуиции.

Вопросы для самоконтроля

1. Каково место информатики в системе современной науки, ее предмет и этапы становления?
2. Раскройте понятие информации в контексте теории информации, кибернетики, теории систем и синергетики.
3. Что такое киберпространство и виртуальная реальность?
4. Каково место моделирования и вычислительного эксперимента в информатике?
5. Что такое искусственный интеллект и инженерия знаний?
6. Назовите философские проблемы информатики.
7. Что такое информационное общество как информационная экономика, постиндустриальное общество, общество знания?
8. В чем состоит концепция информационной безопасности?
9. Каковы особенности и задачи социальной информатики?

10. В чем состоит проблема личности в информационном обществе?

11. Виртуальная реальность как социокультурный феномен информационного

общества.

12. Информационно-компьютерная революция в социальном контексте.

13.Информатизация, медиатизация современного общества и социальный контроль

над человеком.

14. Какую роль играет распространение информационных и коммуникационных технологий во все сферы общественной и частной жизни?

15. Раскройте понятие и особенности компьютерной революции.

Список рекомендуемой литературы

Основная

1.История и философия науки: учебник /под ред. А.С. Мамзина, Е.Ю. Сиверцева. -2-е изд. перераб. и доп. М.: Юрайт, 2016.-360 с.

2.Канке, В. А. Философские проблемы науки и техники: учебник и практикум для магистратуры / В. А. Канке. — М. : Издательство Юрайт, 2016. — 288 с.

3.Кошарный В.П. Философия науки и техники: учебное пособие. Пенза, 2012.- 294 с.

4.Лебедев С.Философия науки: Учебное пособие.-2-е изд.-М.;Изд.-ство Юрайт

2015, 296 с.

Дополнительная

5.Горохов В.Г., Розин В.М. Введение в философию техники: Учебное пособие.-М.: Инфра-М, 1998.-224 с.

6.Горохов В.Г. Техника и культура: возникновение техники и теории технического творчества в России и Германии в конце XIX –начале XX столетия.-М., 2009.-376 с.

7.Горохов В.Г. Основы философии техники и технических наук. Учебник. М.: Гардарика, 2007.-335 с.

8. История информатики и философия информационной реальности: Учебное пособие для вузов/Под ред. Р.М. Юсупова, В.П, Котенко.-М.: Академический проект, 2007- 429 с.

9. Попкова Н.В. Философия техносферы. – М.: URSS; ЛКИ, 2008. - 344 с**.**

10. Современные философские проблемы естественных, технических и социально-гуманитарных наук/под. ред. В.В.Миронова.-М.:Гардарики, 2006.-639 с.
11. Степин В.С. История и философия науки: учебник. М.: Академический проект. 2011.-423 с.
12. Философия математики и технических наук / Под общ. ред. С.А. Лебедева: Учебное пособие для вузов.- М., 2006.-779 с
13. Шаповалов В.Ф. Философия науки и техники: Учебное пособие. – М.: ФАИР-ПРЕСС, 2004. – 320 с.

Тема 9. Проблема социальной ответственности в философии науки и техники. Этика науки и техники.

В системе мировоззренческих отношений наряду с познавательными важнейшую роль играют ценностные отношения. В рамках классического этапа в развитии наук, особенно в XVII –XVIII вв. ученые считали, что познавательное отношение нейтрально в ценностном отношении. Внимание ученых должно быть направлено исключительно на характеристики объекта. Все другое необходимо вывести за скобки научного поиска.

Вопрос о соотношении истинности и ценности в научном знании был в центре внимания философов-неокантианцев Баденской школы в конце XIX в при обсуждении вопроса о специфике гуманитарного знания. Русские народники (П.Л. Лавров, Н. К.Михайловский и др.) активно отстаивали так называемый субъективный метод в социологии. По мере утверждения представления о науке не только как системе знаний и деятельности по их получению, но и как о социальном институте, социокультурные и ценностные факторы в развитии науки, проблемы социальной, нравственной, эстетической ценностных ориентаций в научном исследовании стали занимать все большее место в размышлениях о природе и сущности науки.

Ценностно-оценочный компонент всегда входил и входит в структуру познавательного образа, выражает его связь с системой общественных отношений, с культурой. Ценностные установки и ориентации образуют предпосылки, на которых строится познавательная деятельность. Важнейшим является и вопрос о смысле и целях научного познания. Все выше сказанное делает актуальным анализ аксиологической (ценностной) проблематики в науке и технике. Среди вопросов, которые представляются наиболее актуальными, можно выделить следующие:

- Соотношение истинности и ценности, истинности и добра, истинности и красоты;
- Соотношение свободы научного поиска и социальной ответственности ученого;
- Соотношения науки и власти, возможности и границ управления наукой; характер последствий развития научно-технической деятельности;

- Гуманистическая сущность науки и др.

Идеал научности, сложившийся в рамках классической науки, предполагал полную элиминацию социально-культурных, ценностных факторов, всего, относящегося к субъекту научного познания, из картины объекта исследования. На рубеже XVII-XVIII вв. это было оправдано, поскольку позволял игнорировать теологические догмы и все, что мешало научному познанию развиваться без всяких предубеждей («идолов или «призраков».- Ф. Бекон). И. Кант, рассматривая вопрос о соотношении истинности и ценности, развел сферу сущего (область теоретического разума) и сферу должного (царство ценностей), отведя при этом последней ведущую роль. Он указал на границы теоретического разума и его действий в рамках морального образа мыслей, нравственных требований. В наше время это становится все более актуальным. Сейчас уже ни у кого не вызывает сомнения, что наука и техника не могут развиваться в отрыве от своих мировоззренческих основ, не анализируя смысл своей деятельности и ее возможные последствия. Однако включение целей и ценностей в структуру научной рациональности не у всех ученых и специалистов по философии науки нашло понимание. Слишком сильны были убеждения в непоколебимости классического идеала научности. Однако научные и мировоззренческие результаты научной революции конца XIX – начала XX вв., итоги развития науки в XX столетии лишили фундаментальную науку представлений о ее ценностной нейтральности и обособленности. Появление оружия массового уничтожения, генной инженерии показали, что представления о ценностно-нейтральной науке не просто не верны, а и опасны для будущего человечества.

В современной постнеклассической науке формируется тип рациональности, который характеризуется соотнесенностью знания не только со средствами познания, но и со всей системой человеческих ценностей, ценностно-целевыми структурами деятельности. В настоящее время растет интерес к социальным, человеческим, гуманистическим аспекта науки и техники. Наука, как и философия, религия, искусство, будучи формой постижения мира, содержит интересы и ценности, отражающие содержание человеческой культуры, человеческие чувства и устремления. Объективность в современном понимании не только допускает, но и предполагает включение ценностных факторов в состав объяснительных схем. Исследование «человекоразмерных» объектов современной науки непосредственно затрагивает гуманистические ценности. В наше время происходит изменение нравственно ориентации представителей естествознания в направлении гуманизации естественнонаучного и технического знания. Сегодня большое значение имеет органическое соединение ценностей научно-технического мышления с социальными ценностями, которые представлены нравственностью, искусством, философией. Вместе с тем главной когнитивной ценностью в науке остается истина как объективное, доказанное знание. Процесс получение такого

знания регулируется внутренними нормами науки. Широко известны слова Аристотеля :«Платон мне друг, но истина дороже».

Этика – один из древнейших разделов философии. Она регулирует взаимные обязанности людей по отношению друг к другу.

Профессионально-прикладное определение этики применительно к инженерной профессии означает, что ничто человеческое инженеру не чуждо.

Профессиональная этика в сфере научной деятельности называется **научным этосом**. Понятие научного этоса ввел Р.Мертон для обозначения комплекса норм, предписаний и ценностей этического порядка, господствующих в общественном мнении научного сообщества. В переводе с греческого «этос» означает, нрав, характер, привычку, обычай. Р.Мертон связывал научный этос с внутринаучными императивами морального порядка, регулирующих профессиональную деятельность ученых. С точки зрения Р.Мертона, научный этос связан с профессиональной ответственностью, т.е. ответственностью ученых перед научным сообществом за качество проводимых ими исследований и получаемых результатов, за добросовестное выполнение других профессиональных ролей, за сохранение ценностей сообщества.

Профессиональная ответственность ученых базируется на определенной ценностной системе. Среди ценностных ориентации ученых доминирующее положение занимают *самоценность истины* и *ценность новизны*. С точки зрения научного сообщества главной целью деятельности ученых должно было служение Истине. С нравственной точки зрения это означает, что истина не должна быть средством для достижения личных выгод, а только средством достижения общественно значимых целей. В когнитивном плане эта установка воплощается в целом ряде идеалов и нормативов научного познания, выражающих его специфику: в определенных идеалах организации знания (например, требовании логической непротиворечивости теории и ее опытной подтверждаемости), в поиске объяснения явлений, исходя из законов и принципов, отражающих сущностные связи исследуемых объектов, и т.д.

Не менее важную роль в научном исследовании играет установка на постоянный рост знания и особую ценность новизны в науке. Эта установка выражена в системе идеалов и нормативных принципов научного творчества (например, запрете на плагиат, допустимости критического пересмотра оснований научного поиска как условия освоения все новых типов объектов и т.д.).

С этими ценностными установками связана ценность *универсализма и коллективизма***.** Ценностная установка универсализма связана с признанием внеличностного, объективного характера научного знания. Надежность нового знания определяется только соответствием его наблюдениям и ранее удостоверенным научным знанием.

Эта неписанная установка является основой моральной нормы этики ученых о *научной честности и недопустимости фальсификаций* результатов

исследования. Широко известны случаи, когда ученые, стремясь подтвердить выдвинутые или оригинальные гипотезы, влекущие за собой крупное научное открытие, идут на подтасовку результатов проводимых ими экспериментов. После того, как научная общественность приходит к выводу о невозможности проведения таких экспериментов, эти ученые были публично осуждены и с позором выведены за пределы научного сообщества (южнокорейский генетик – клонирование).

Универсализм обусловливает демократический характер науки. С точки зрения универсализма равны, никакие титулы, звания, прошлые заслуги не принимаются во внимание в качестве научного доказательства. Ценностная *установка коллективизма* ориентирует ученых, что плоды научного познания принадлежат всему научному сообществу и обществу в целом. Они всегда являются результатом коллективного научного сотворчества, так как любой ученый всегда опирается на какие-то идеи (знания) своих предшественников и современников. Права частной собственности на знания в науке не должно существовать, хотя ученые, которые вносят наиболее существенный личный вклад вправе ждать от коллег и общества признания их заслуг, а также справедливого материального и морального поощрения. Такое признание является важнейшим стимулом научной деятельности.

Однако, научное исследование не может носить полностью обезличенный характер. Нормы научной этики требуют необходимости ссылок на авторство той или иной идеи при оформлении результатов научных исследований. *Требование ссылок*, как обязательное условие оформления диссертации, научной монографии и статьи, призвано не только зафиксировать авторство тех или иных идей и научных текстов. Оно обеспечивает четкую селекцию уже известного в науке и новых результатов. Вне этой селекции не было бы стимула к напряженным поискам нового, в науке возникли бы бесконечные повторы пройденного, и, в конечном счете, было бы подорвано ее главное качество — постоянно генерировать рост нового знания, выходя за рамки привычных и уже известных представлений о мире.

Ценности, нормы и образцы поведения научного этоса создают особый этический фон в межличностных отношениях ученых. Без моральной оценки, особого нравственного климата невозможна никакая коллективная деятельность. Научный этос объединяет ученых в научное сообщество.

Этика прошла долгий путь развития. Мыслители всех времен старались расширить ее содержание. Например, И. Кант делал акцент на понятиях долга, долженствования, ответственности. По мере проникновения человека в тайны природы его ответственность за обладание этими тайнами возрастает. При синергетическом взаимодействии многих лиц затрудняется персонификация ответственности в случае, когда развитие техники перехлестывает порог ответственности. Например, кто ответит за кислотные дожди? За потепление климата? Таяние полярных ледников, повышения уровня мирового океана и связанные с этим наводнения? Когда все ответственны за все, когда каждый отдельный человек ответственен за целый

мир, тогда никто ни за что не отвечает.

Что значит «быть ответственным»? Это означает – быть готовым или быть обязанным давать ответ кому-нибудь и за что-нибудь. В исследованиях по философии права отмечается причинная ответственность за действия в силу обязанности, согласно которой кто-то ответственен за нежелательное или наносящее ущерб положение дел. Существует ответственность за способность выполнять задачу или роль, способность решать вопрос, понимать, планировать, осуществлять, оценивать события, обладать соответствующими познавательными и управленческими качествами, квалификацией и, наконец, подотчетная ответственность перед надлежащими инстанциями.

Моральная ответственность всегда индивидуальна, она не поставлена в строгие рамки, не управляется внешними нормами. Носителем моральной ответственности может быть индивид, она не может осмысленно приписываться объединениям и формальным организациям, хотя не обособлена от коллектива. В этике обычно указывается на совесть, перед которой человек держит ответ, – последняя инстанция для ответственности.

Ответственность имеет этические измерения. Это нечто большее, чем голос совести как «факт морального разума» (И. Кант). Применительно к профессиональной культуре характерными являются такие качества, как справедливость, патриотизм, способность признать приоритет общего над личным и вытекающая из него идея служения, милосердие, способность к сопереживанию, терпимость к другим людям, народам, культурам, приоритет духовно-нравственного начала над материально-прагматическим. Эти ценности могут быть сопоставлены с утвердившимися ценностями мировой цивилизации, к которым относятся гуманизм и антропоцентризм, свобода совести, индивидуальная свобода, права человека, уважение к собственности, материальное благополучие и т.п. Речь идет о том, что в глобальном аспекте могут и должны быть востребованы не только ценности западной цивилизации, но и ценности русской культуры и культур других народов России. В новой системе ценностей приоритетами должны стать устойчивое развитие, здоровый образ жизни, интеллект, природная одаренность, профессионализм, компромисс и социальное партнерство, честность и обязательность, взаимное доверие, толерантность и плюрализм, законопослушание и др.

Между этикой гражданской и этикой инженерной нет непроходимой грани, так как инженеры рекрутируются из числа граждан. В настоящее время влияние технического развития на человека и его образ жизни менее заметно, чем влияние на природу. Тем не менее, оно существенно. Неконтролируемые изменения природы вошли в разряд самых пристально изучаемых предметов, когда выяснилось, что человек и природа не успевают адаптироваться к стремительному развитию технической цивилизации. Неожиданно для многих оказалось, что инженерная деятельность, естественнонаучные знания и техника существенно влияют на природу и человека, изменяя их. Современное мышление человека стало воспринимать

природу иначе, чем, скажем, двести лет назад. Современный человек уже мыслит природу как технику. Поэтому очень важно сменить традиционную научно-инженерную картину мира, заменив ее новыми представлениями о природе, технике, способах решения задач, достойном существовании человека. Чтобы техника не уничтожала, не искалечила человечество, люди должны осознать как природу техники, так и последствия технического развития. Однако без комплексного гуманитарного и юридического образования решить эту проблему невозможно.

Техника проявляет гуманитарный облик инженера, обнаруживает потаенное бытие человека в мире образов, схем, ритмов и смыслов, поэтому так важно ориентироваться не только на познавательные процедуры, но и на аксиологический аспект техники, где высшие человеческие возможности и модели поведения – образец преданности истине. Обогащение технического знания содержанием философии, психологии, экономики, технической эстетики, эргономики расширяет воздействие технической деятельности на социальную и духовную жизнь. Вместе с тем технический прогресс порождает немало проблем, требующих нового применения этики для избежания ситуации риска. Обобщенно эти тенденции Г. Ленк, вице-президент Европейской академии наук, рассматривает в такой последовательности.

1. Увеличивается число людей, получивших побочные эффекты от технических мероприятий.

2. Масштабы разрушения природной системы под влиянием человеческой деятельности продолжают расти, приобретая глобальный размах.

3. Ухудшение медико-биологической и экологической ситуации актуализирует проблему ответственности за нерожденные поколения.

4. Человек все больше испытывает на себе манипуляции социального и медико-фармакологического типа. Как следствие подобного рода экспериментов над человеком обостряются этические проблемы таких исследований.

5. В результате вмешательства в генетический код человеку грозит превращение в «объект техники».

В жизни всегда существовали противоречия между должным и сущим. Этот недуг обыденного бытия затрагивает и проблему ответственности, связанной с функционированием техники, с решением вопроса о пользе и вреде. Немецкий физик, лауреат Нобелевской премии Макс Борн (1882-1970) подчеркивал, что в реальной науке и ее этике произошли изменения, которые делают невозможным сохранение старого идеала служения знанию ради него самого. Мы были убеждены, что это никогда не может обернуться злом, поскольку поиск истины есть добро само по себе. Это был прекрасный сон, от которого нас разбудили мировые события. Американский физик Роберт Оппенгеймер (1904-1967), создатель атомной бомбы, был еще более нетерпим, заявив, что физики после американских атомных бомбардировок японских городов в 1945 г. потеряли свою невинность и впервые познали

грех. Чувство вины принудило его отказаться от идеи создания водородной бомбы.

О необходимости упреждающей оценки всевозможных последствий технического развития говорил немецкий социолог и экономист Вернер Зомбарт (1883-1941). В своей книге «Немецкий социализм» в разделе «Обуздание техники» он выдвинул идею, согласно которой внедрение новой техники всегда будет сопровождаться или даже предваряться ценностным анализом ее возможных последствий. Данное положение, поддержанное многими его последователями, стало одним из важнейших тезисов философии техники, а осознание жизненной важности его практической реализации привело к созданию в 1972 г. при американском конгрессе первой официальной структуры по оценке техники «Office of Technology Asseement» («Бюро по оценке техники»). Позже подобные организации появились в Швеции (1973), в Канаде (1975) и в ряде других развитых стран.

«Отец» кибернетики Норберт Винер (1894-1964) в своей научной деятельности не ограничивался лишь отказом от всякого рода сотрудничества с военно-промышленным комплексом США, но и призывал своих коллег последовать его примеру, он, полностью осознавая тот факт, что эта новая наука ведет к техническим достижениям, создающим огромные возможности для добра и зла, призывал своих коллег отказаться от исследований по кибернетике. Винер выдвинул принцип, согласно которому было необходимо (а) позаботиться о том, чтобы широкая публика понимала общее направление и значение указанных исследований и (б) «ограничиться в своей собственной деятельности такими далекими от войны областями, как физиология и психология».

В 1957 г. III Пагуошская конференция в Вене обратилась с декларацией с призывом к ученым внести свой вклад в образование людей и распространить среди них понимание тех опасностей, которые таит в себе дальнейшее развитие науки и техники. В 1974 г. «Маунт-Кармельская декларация по технике и моральной ответственности, поддержанная учеными мира, констатировала морально-этическую несостоятельность использования энергии атома в военных целях. В 1970-х гг. группа ученых-генетиков и микробиологов ввела мораторий на проведение некоторых экспериментов и исследований, когда выяснилось, что полученные ими гибридные молекулы могут быть использованы для вмешательства в гены живого организма человека. В 1975 г. группа ученых во главе с Паулем Бергом организовала в г. Азиломаре (США) международную конференцию с участием 150 ученых-генетиков со всего мира. Была выработана система мер предосторожности, гарантирующая безопасность этого направления исследований для жизни человека.

Подобные активные инициативы стали возможны благодаря тому, что времена ученого-одиночки уже прошли. Научные открытия и внедрения стали результатом коллективного поиска знаний. Фундаментальные научные исследования требуют сосредоточения усилий смежных областей научного поиска. На фоне этих процессов философия техники не могла ограничиваться

сторонними наблюдениями.

Одним из важнейших механизмов учета названных изменений, по мнению Н. Г. Багдасарьян, является *этический кодекс*, который формируется в профессиональном сообществе. До определенного времени этические нормы могут существовать в виде «неписанных правил», но по мере расширения сферы социальных последствий инженерной деятельности, ее усложнения и разбалансировки появляется необходимость в специально разработанных и четко сформулированных этических кодексах. Они, как правило, коррелируют как с юридическим законодательством, так и с административными нормативными актами, но в значительной мере отражают специфику того или иного инженерного сообщества, являясь общественными регуляторами взаимодействия его членов. Хорошо разработанные этические кодексы существуют в Германии, Франции, США. Так, в США действуют многочисленные инженерные и научные профессиональные союзы и общества, которые объединяются под эгидой трех основных организаций: «Американская ассоциация инженерных обществ», имеющая в своем составе около 800000 членов (инженеров, проектировщиков, строителей и т. п.), «Американское общество инженерного образования» (10000 индивидуальных членов и более 300000 институциональных) и «Американская ассоциация содействия науке» — 300 самостоятельных организационных объединений. Их главные функции: создание условий для становления и поддержания профессиональной компетентности (на основе информирования, общения, обсуждения проектов и проблем, выработки критериев оценок профессиональной деятельности и др.), координация работы профессиональных союзов и обществ, поддержание связей с общественностью и правительством, содействие улучшению технического образования, выработка системы профессиональных ценностей, сочетающих свободу творчества и профессиональную ответственность, создание социально благоприятного этического климата (через образование, издание этических кодексов, анализ и оценку конфликтных ситуаций), издание профессиональной литературы — газет, журналов. Они участвуют в аккредитации университетов, их представители входят в комиссии, утверждающие учебные программы и курсы, а также в попечительские советы. Кодекс инженерной этики, разработанный Комитетом по инжинирингу и технологиям включает следующие каноны:

1. Инженеры при исполнении своих профессиональных обязанностей превыше всего ставят безопасность, здоровье и благосостояние общества.
2. Инженеры должны выполнять работы только в пределах своей компетенции.
3. Инженеры должны отвечать на запросы общественности только объективным и правдивым образом.
4. Инженеры в своей профессиональной области действуют в качестве преданных представителей или доверенных лиц для каждого работодателя или заказчика и должны избегать конфликтов интереса.

5. Инженеры должны строить свою профессиональную репутацию на достоинствах своего обслуживания, им не следует соревноваться нечестными методами с другими.
6. Инженеры должны действовать таким образом, чтобы поддерживать и развивать чистоту, честь и достоинство инженерной профессии.
7. Инженеры должны поддерживать свое профессиональное развитие и предоставлять возможности для профессионального развития инженерам, находящимся под их наблюдением.

Как видно из приведенного выше текста, этический кодекс фиксирует правила взаимодействия инженеров, направленного вовне (общество в целом, общественность, работодатель, заказчик) и внутрь профессиональной группы (патронаж, профессиональное соревнование).

Динамика изменений норм и ценностей в современной российской жизни, смена приоритетов в инженерной деятельности, связанная с переструктурированием промышленности, ряд других факторов, обусловленных интенсивными процессами компьютеризации, настоятельно требуют артикуляции этически значимых проблем инженерной деятельности, введения их в структуру инженерного образования. При этом следует подчеркнуть, что этические проблемы имеют междисциплинарный статус, а значит, и обучение в данном направлении должно вестись не только в рамках социально-научного и гуманитарного блока (что чаще всего и происходит), но этические модули должны иметь свое место в системе общетехнических и специальных дисциплин.

Таким образом, в настоящее время сущность изменения в профессиональной инженерной культуре заключена в установлении соответствия профессиональной деятельности, ее потенциала, результатов и последствий критериям социальной эффективности и приемлемости. Лишь на этом пути могут быть выработаны корректные ценностные ориентиры и задан новый тип индивидуальной и групповой профессиональной деятельности, соответствующие задаче конструктивного решения социальных проблем (См.: Багдасарян Н.Г. Профессиональная культура инженера. Механизмы освоения. М., 1998).

Социальная оценка техники – это определение качественных изменений в ее развитии, захватывающих всю техносферу. Социально-экологическая экспертиза научно-технических и хозяйственных проектов связана с экспертной оценкой процессов и явлений, не поддающихся непосредственному измерению. Она основывается на суждениях специалистов и опосредована проблемой ответственности ученого, науки перед обществом. Обозримое прошлое показывает, что в системе среды обитания человека, ориентированной на поддержание его активного долголетия, ситуация не изменилась в лучшую сторону. Социальные перемены последних лет породили, или углубили ранее существовавшие негативные тенденций. Так, ухудшение экологической среды привело к возрастанию факторов риска для жизнедеятельности человека. Было утрачено внимание к экологическим моментам, негативно влияющим на

здоровье человека.

На протяжении веков научная и техническая деятельность считались морально нейтральными (в силу непредсказуемости последствий того или иного открытия, изобретения). Соответственно вопрос об ответственности ученого или инженера вообще не ставился. В конце XX в. и, особенно, в будущем мы не можем себе позволить пренебрегать этическим контекстом деятельности ученого и инженера. Этические нормы не только регулируют применение научных результатов, но и содержатся в самой научной деятельности.

Норвежский философ Г. Скирбекк отмечает, что, будучи деятельностью, направленной на поиск истины, наука регулируется нормами: «ищи истину», «избегай бессмыслицы», «выражайся ясно», «старайся проверять свои гипотезы как можно более основательно» - примерно так выглядят формулировки этих внутренних норм науки. В этом смысле этика содержится в самой науке, и отношения между наукой и этикой не ограничиваются вопросом о хорошем или плохом применении научных результатов.

Наличие определенных ценностей и норм, воспроизводящихся от поколения к поколению ученых и являющихся обязательными для человека науки, т.е. определенного этоса науки, очень важно для самоорганизации научного сообщества (при этом нормативно-ценностная структура науки не является жесткой). Отдельные нарушения этических норм науки в общем скорее чреваты большими неприятностями для самого нарушителя, чем для науки в целом. Однако если такие нарушения приобретают массовый характер, под угрозой уже оказывается сама наука.

В условиях, когда социальные функции науки быстро умножаются и разнообразятся, дать суммарную этическую оценку науке как целому оказывается недостаточно и неконструктивно вне зависимости от того, положительной или отрицательной будет эта оценка.

Этическая оценка науки сейчас должна быть дифференцированной, относящейся не к науке в целом, а к отдельным направлениям и областям научного знания. Такие морально-этические суждения играют очень конструктивную роль.

Современная наука включает в себя человеческие и социальные взаимодействия, в которые вступают люди по поводу научных знаний.

«Чистое» изучение наукой познаваемого объекта - это методологическая абстракция, благодаря которой можно получить упрощенную картину науки. На самом деле объективная логика развития науки реализуется не вне ученого, а в его деятельности. В последнее время социальная ответственность ученого является неотъемлемым компонентом научной деятельности. Эта ответственность оказывается одним из факторов, определяющих тенденции развития науки, отдельных дисциплин и исследовательских направлений.

В 70-е годы XX века ученые впервые объявили мораторий на опасные исследования. В связи с результатами и перспективами биомедицинских и

генетических исследований группа молекулярных биологов и генетиков во главе с П. Бергом (США) добровольно объявили мораторий на такие эксперименты в области генной инженерии, которые могут представлять опасность для генетической конституции живущих ныне организмов. Тогда впервые ученые по собственной инициативе решили приостановить исследования, сулившие им большие успехи. Социальная ответственность ученых стала органической составляющей научной деятельности, ощутимо влияющей на проблематику и направления исследований.

Прогресс науки расширяет диапазон проблемных ситуаций, для решения которых недостаточен весь накопленный человечеством нравственный опыт. Большое число таких ситуаций возникает в медицине. Например, в связи с успехами экспериментов по пересадке сердца и других органов остро встал вопрос об определении момента смерти донора. Этот же вопрос возникает и тогда, когда у необратимо коматозного пациента с помощью технических средств поддерживается дыхание и сердцебиение. В США такими вопросами занимается специальная Президентская комиссия по изучению этических проблем в медицине, биомедицинских и поведенческих исследованиях. Под воздействием экспериментов с человеческими эмбрионами острым становится вопрос о том, с какого момента развития существо следует считать ребенком со всеми вытекающими отсюда последствиями.

Нельзя считать, что этические проблемы являются достоянием лишь некоторых областей науки. Ценностные и этические основания всегда были необходимы для научной деятельности. В современной науке они становятся весьма заметной и неотъемлемой стороной деятельности, что является следствием развития науки как социального института и роста ее роли в жизни общества. Ученые и инженеры должны осознавать свою ответственность перед человеческой цивилизацией. Человечество все больше оказывается зависимым от последствий технического развития. В этой связи управление техническим прогрессом, его сдерживание, регулирование, осуществление его целей, оценка результатов оказываются не только инженерной, управленческой, государственной, но и этико-философской проблемой.

Никогда еще прежде в истории на человека не возлагалась столь большая ответственность, как сегодня, ибо еще никогда он не обладал столь большой - многократно возросшей благодаря технике, властью над другими природными существами и видами, над своей окружающей средой и даже над всем живым на Земле. Сегодня человек в региональном или даже в глобальном масштабе может уничтожить свой собственный вид и все высшие формы жизни или, по меньшей мере, причинить огромный ущерб. Техника нашего времени больше не техника прошлых веков. Техническое развитие достигло такого уровня, что, в принципе, человек может осуществить любое свое намерение; все меньше и меньше невозможного остается для человека, оснащенного техникой. Это существенно обостряет проблему последствий технического развития. Человек так глубоко проникает в недра природы, что по сути своей техническая деятельность в

современном мире становится частью эволюционного процесса, а человек - "соучастником" эволюции. Мы не можем больше перекладывать ответственность за будущий мир на трансцендентного Бога или на внутреннюю эволюционную закономерность природы. Как соучастники, мы несем ответственность, и наша ответственность неизмеримо возросла.

Становясь соучастником эволюции, человек должен помогать ей. Нужно задуматься о том, должен ли человек делать все, что он может? Современная техника достигла такого уровня развития, обрела столь мощное влияние в мире, что можно говорить об определенной самостоятельности техники, о способности действовать, направлять развитие общества, формировать мировоззрение. Один из распространенных сюжетов научной фантастики связан с победой техники над человеческой цивилизацией, установлением власти компьютеров и т.п. И действительно, для такой фантазии есть основание. Сейчас уже трудно понять, техника ли служит человеку или человек технике. Усовершенствуя технику, человек попадает под ее власть. И чем совершеннее технические средства, тем больше нуждается в них человек и подчиняет им свое существование, что, в свою очередь, ограничивает свободу и достоинство человека. Подобное широкомасштабное развитие техники, охватившее почти все сферы человеческой жизнедеятельности, сродни экспансии. Стоит задуматься, нужно ли человеку делать все, что он может, на что способен его технический гений, нужно ли осуществлять все технические потенции?

Общество стоит перед проблемой выработки ясных ценностных и целевых представлений о достойной жизни в будущем. Поэтому дальнейшее развитие техники немыслимо без осознания социальной ответственности. Недостаточно говорить об ответственности какого-либо отдельного человека или оценивать возможные последствия какого-либо отдельного действия. В рамках философии техники этика должна быть ориентирована на все человечество.

Один из вопросов, порождающих общую тревогу и критику по поводу безудержного технологического развития, сводится к тому, что применение некоторых технологий может исказить само понятие ответственности и даже деморализовать человека. В адрес компьютерной техники выдвигаются обвинения в том, техника, постоянно отстраняя нас от ответственности, перепоручая все экспертам, воплощает в себе торжество зла, ибо если все делается за нас, если мы более ни за что не несем ответственности, то нас уже нельзя считать людьми. Таким образом, компьютерная техника, завладевая нашими полномочиями, тем самым трансформирует сам статус человека, лишая его ответственности.

Отвечая на такие обвинения, американский философ К. Митчем обращает внимание на то, что при всей своей определенности суждение такие обвинения не бесспорны. Совсем не очевидно, что компьютеры каким бы то ни было образом лишают людей ответственности, которую раньше те несли сами. Скорее, они сделали возможным осуществление некоторых особых видов ответственности, внедрение современных технологий привело

к расширению и трансформации всего поля ответственности. Проявлением этого было как отрицательное (реактивное), так и положительное (креативное) отношение к технике там, где ответственность уже была установлена и внимание заострилось на проблематике особых видов ответственности. Различные аспекты произошедших изменений нашли отражение в таких областях, как правовая ответственность, социальное сознание ученых, профессиональная этика инженеров, а также в теологических дискуссиях и философских исследованиях.

Согласно К. Митчему, мощь техники не только не уменьшила персональной ответственности человека, но и привела к расширению самого поля ответственности. Появляются такие составляющие этой нравственной сферы, как юридическая, социальная, профессиональная, религиозная ответственность, связанные с различными областями технической деятельности. Отмечая существенные изменения, происходящие в современных технологиях, политике государств по отношению к техническому развитию, выражающейся в создании специальных отраслей экономики по защите от последствий промышленного развития, следует увязать все это с осознанием человеком меры ответственности за последствия неограниченной технической экспансии и решением вопроса о лидерстве в тандеме «человек – техника».

ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

1. Каковы основные ценности науки?
2. Каковы перспективы и границы современной техногенной цивилизации?
3. В чем особенности этики науки?
4. Каковы основные этические принципы науки и инженерного отношения к миру?
5. Каковы экономические, социокультурные, социоэкологические последствия развития техники и технознания.
6. Научно-техническая политика и проблема управления научно- техническим прогрессом общества.
7. В чем состоит проблема комплексной оценки социальных, экономических, экологических и других последствий техники.
8. Назовите виды социальной ответственности ученого и проектировщика, моральные и юридические аспекты их реализации в обществе.
9. В чем суть «технократического» и «романтически-утопического» отношения к науке?
10. С какой целью философы обращаются к проблеме науки и гуманизма?

Список рекомендуемой литературы

Основная

1. История и философия науки: учебник /под ред. А.С. Мамзина, Е.Ю. Сиверцева. -2-е изд. перераб. и доп. М.: Юрайт, 2016.-360 с.

2. Канке, В. А. Философские проблемы науки и техники: учебник и практикум для магистратуры / В. А. Канке. — М. : Издательство Юрайт, 2016. — 288.

3. Лебедев С.А.Философия науки: Учебное пособие.-2-е изд.-М.; Изд.-тво Юрайт.-2015.- 296 с.

4. Фролов И.Т., Юдин Б.Г.Этика науки. -М.: Либрком,2016.-256 с.

Дополнительная

5. Агацци Э. Моральное измерение науки и техники. - М., 1988.-324 с.

6. Багдасарян Н.Г. Профессиональная культура инженера. Механизмы освоения. М., 1998.-356 с.

7. Воронин А.А. Техника и мораль // Вопросы философии. – 2004. – № 10. – С. 93–101

8. Лазар М. Г. Этика науки : Философско-социологические аспекты соотношения науки и морали / М. Г. Лазар ; под ред. И. А. Майзеля. Л. : Изд-во Ленингр. ун-та, 1985. 126 с.

9. Лэйси Х. Свободна ли наука от ценностей? Ценности и научное понимание.- М.:Логос,2001.-360 с.

10. Ленк Х. Размышления о современной технике. – М.: Аспект Пресс, 1996. – 183.-с.

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ ЦИФРОВАЯ ОБРАБОТКА ИЗОБРАЖЕНИЙ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

По данной дисциплине предусмотрены аудиторная и внеаудиторная виды самостоятельной работы.

Вид самостоятельной работы	***Оценочные средства***
Аудиторная самостоятельная работа	
1. Разработка алгоритмов решения поставленных задач при выполнении лабораторной работы (текущие консультации преподавателя).	Отчет, защита лабораторной работы
2. Программная реализация алгоритмов решения поставленных задач (текущие консультации преподавателя)	Отчет, защита лабораторной работы
Внеаудиторная самостоятельная работа	
1. Подготовка теоретических вопросов по теме лабораторной работы.	Защита лабораторной работы
2. Изучение лекционного материала, предусматривающее проработку конспекта лекций.	Конспект
3. Подготовка к промежуточной аттестации	зачет

Раздел 1. Введение в теорию РО и ОИ

Темы лекций:

- Введение. Цель и задачи дисциплины, ее роль и место в общей системе подготовки специалист.
- Представление образов и основные подходы к машинному распознаванию.
- Приложения методов распознавания образов: машинное зрение, распознавание рукописных символов, распознавание речи.
- Классификация на основе байесовской теории решений
- Байесовская дискриминантная функция.
- Принятие решение по максимуму правдоподобия.
- Ошибки классификации. Оптимальная дискриминантная функция для нормально распределенных образов.
- Обучение для статистических дискриминантных функций. Непараметрическое оценивание

Раздел 1. Введение в теорию РО и ОИ
Вопросы, выносимые на самостоятельное изучение:

1. Что такое «отношение правдоподобия»?

2. Перечислите наиболее известные статистические критерии принятия решений. В чем их сходство и чем они отличаются?

3. Как выбрать наиболее подходящее решающее правило?

4. Как определяются вероятности ошибок первого и второго рода в задаче классического обнаружения?

5. Назовите показатели качества многоальтернативного параметрического Распознавания.

Литература

1. Потапова, М. Н. Основы обработки изображений в полиграфии [Электронный ресурс] : учебное пособие / Потапова М. Н., Сахабутдинова Г. Ф. Кемерово : КемГУ, 2020. 112 с. ISBN 978-5-8353-2711-9.

2. Гук, А. П. Методы и технологии распознавания объектов по их изображению [Электронный ресурс] : учебно-методическое пособие / Гук А. П. Новосибирск : СГУГиТ, 2019. 138 с. ISBN 978-5-907052-39-0.

Самостоятельная работа по разделу 1

Внимательно изучить лекционный материал

Цель работы: – изучить методику построения решающего правила с использованием критериев максимального правдоподобия и максимума апостериорной вероятности, получить навыки оценивания показателей качества двух альтернативного непараметрического распознавания.

Порядок выполнения работы

1. Для заданных (согласно варианту) значений параметров нор- мальных законов распределения (m1, σ1) и (m2, σ2), характеризую- щих два класса объектов наблюдения a1 и a2, определить условные по классу плотности вероятности результатов наблюдений f ( ) x a1 = ( ) 1 1 f x,m ,σ и f ( ) x a2 = ( ) 2 2 f x,m ,σ .

2. Построить решающее правило по критерию максимального правдоподобия (1.5).

3. Рассчитать теоретические величины вероятностей ошибок распознавания первого и второго рода по критерию (1.5).

4. Для заданных (согласно варианту) значений априорных вероятностей p1 и p2 появления классов a1 и a2 определить условные плотности полной вероятности результатов наблюдений и апостериорные вероятности классов a1 и a2.

5. Построить решающее правило по критерию максимальной апостериорной вероятности (1.3).

6. Рассчитать теоретические величины вероятностей ошибок распознавания первого и второго рода по критерию (1.3).

7. Сравнить эффективности решающих правил, построенных по критериям максимального правдоподобия и максимальной апостериорной вероятности.
8. Оформить отчет о работе, который должен содержать краткие теоретические сведения, результаты расчетов, графики исследуемых статистических характеристик и выводы.

Варианты заданий к лабораторной работе

№ 1 Номер варианта m1 σ1 m2 σ2 p1 p2

Номер варианта m1 σ1 m2 σ2 p1 p2 1 2 0.5 4 1 0.3 0.7 5 -1 0.3 1 0.9 0.6 0.4 2 0 0.4 2 1 0.4 0.6 6 -3 1 -1 0.5 0.9 0.1 3 0 1 2 0.8 0.1 0.9 7 -4 0.5 -1 1.2 0.7 0.3 4 -2 0.7 0 0.4 0.2 0.8 8 3 0.8 5 1 0.8 0.2

Для всех вариантов число точек для построения графиков N = 200. - 11 –

Решите задания.

Задание № 1.

Разработать алгоритм и реализовать программу «Распознавание простых геометрических фигур» (круг, квадрат, треугольник, 27 прямоугольник, эллипс) методом сравнения с эталоном. Написать программу и определить время решения задачи.

Задание № 2.

Разработать алгоритм и реализовать программу «Распознавание простых геометрических фигур» (круг, квадрат, треугольник, прямоугольник, эллипс) структурным методом. Представление фигур осуществить в виде дерева. Написать программу и определить время решения задачи.
Задание № 3.

Разработать алгоритм и реализовать программу «Распознавание простых геометрических фигур» (круг, квадрат, треуголь ник, прямоугольник, эллипс) структурным методом. Представление фигур осуществить в виде кода Фримена. Измерить время выполнения алгоритма.
Задание № 4.
Сравнить результаты выполнения программы заданий 1 и 2. 2. Сравнить результаты выполнения программы задания №1 и задания № 2. Сделать выводы.
Задание № 5.

Разработать алгоритм и реализовать программу «Распознавание сложных изображений (кораблей)» структурным методом. Представление изображений осуществить в виде кода Фримена.

Раздел 2 Классификаторы

- **Темы лекций**: Линейный и нелинейный классификаторы.
- Линейная дискриминантная функция.
- Алгоритм однослойного перцептрона.
- Схема Кеслера. Построение оптимальной разделяющей поверхности. Алгоритм Гаусса-Зейделя.
- Нелинейный классификатор.
- Многослойный перцептрон Сущность и отличия комитетных методов решения задач распознавания.
- Теоретико-множественная постановка задачи выбора алгоритма. Комитеты. Комитеты линейных функционалов.
- Функция Шеннона

Вопросы, выносимые на самостоятельное изучение:

1. Что такое «достаточная статистика»?

2. Что называют контрольной выборкой?

3. Что общего и в чем различие задачи принятия статистических гипотез и задачи статистического распознавания?

4. Чем отличаются постановки задач обнаружения и распознавания объектов?

Литература

1. Потапова, М. Н. Основы обработки изображений в полиграфии [Электронный ресурс] : учебное пособие / Потапова М. Н., Сахабутдинова Г. Ф. Кемерово : КемГУ, 2020. 112 с. ISBN 978-5-8353-2711-9.

2. Гук, А. П. Методы и технологии распознавания объектов по их изображению [Электронный ресурс] : учебно-методическое пособие / Гук А. П. Новосибирск : СГУГиТ, 2019. 138 с. ISBN 978-5-907052-39-0.

Самостоятельная работа по разделу 2

Внимательно изучить лекционный материал

Цель работы:

– закрепить знания о параметрических методах распознавания;

– исследовать метод распознавания одномерных нормальных совокупностей с точки зрения оптимизации временных характеристик распознающей системы;

– получить навыки реализации статистического распознавания на ЭВМ с использованием системы MathCAD

Порядок выполнения работы

1. Для заданных (согласно варианту) значений параметров нор- мальных законов распределения m1, m2 и σ, характеризующих два класса объектов наблюдения a1 и a2, определить условные плотности вероятности результатов наблюдений x: f (x a1) = f ( ) x,m1,σ , f ( ) x a2 = f ( ) x,m2 ,σ .

2. Вычислить порог принятия решения (2.8) и формализовать решающее правило.

3. С помощью алгоритма генерации нормально распределенной случайной величины смоделировать данные наблюдений классов a1 и a2; размер каждого массива данных N = 100 элементов.

4. Для исследования эффективности алгоритма распознавания с накоплением данных принять диапазон варьирования объема контрольной выборки n = 1, 2, …, 10.

5. Для текущего объема контрольной выборки вычислить N раз (N = 100) достаточные статистики для двух классов объектов.

6. Для текущего объема контрольной выборки рассчитать теоретические вероятности ошибок распознавания первого и второго рода и найти их эмпирические оценки. В качестве показателей эффективности алгоритма распознавания принять среднее значение вероятностей ошибок и среднее значение оценок этих вероятностей.

7. Построить графики зависимостей экспериментальной и теоретической вероятностей ошибок от объема накопления данных n.

8. Оформить отчет о самостоятельной работе, который должен со- держать краткие теоретические сведения, алгоритмы моделирования, результаты расчетов, графики исследуемых статистических характеристик и полученных зависимостей, выводы.

Решите задания

Задание № 1.

Разработать алгоритм и реализовать программу «Распознавание сложных изображений (кораблей)» структурным методом. Представление изображений осуществить в виде матрицы.

Задание № 2.
Сравнить результаты выполнения программы заданий 4 и 5. Сделать выводы.

Задание № 3.

Разработать алгоритм и реализовать программу на языке реляционной алгебры и на языке SQL.

Рассмотрим набор отношений, которые моделируют сдачу сессии студентами некоторого учебного заведения.

R1 = <ФИО, Дисциплина, Оценка>; R2 = <ФИО, Группа>; R3 = < Группа, Дисциплина>,

где ***R1*** — информация о попытках (как успешных, так и неуспешных) сдачи экзаменов студентами;

R2 — состав групп;

R3 — список дисциплин, которые надо сдавать каждой группе.

Найти список фамилий студентов имеющих отличные оценки по всем экзаменам.

Раздел 3. Комитетные методы распознавания

1. Сущность и отличия комитетных методов решения задач распознавания. Теоретико-множественная постановка задачи выбора алгоритма. Комитеты. Комитеты линейных функционалов. Функция Шеннона.
2. Распознавание объектов методом потенциалов.
3. Постановка задачи. Байесовский классификатор. Модель Марковской цепи. Алгоритм Витерби. Скрытые Марковские модели.
4. Постановка задачи. Байесовский классификатор. Модель Марковской цепи. Алгоритм Витерби. Скрытые Марковские модели.
5. Особенности методов селекции признаков. Постановка задачи селекции признаков. Общность классификатора. Предобработка векторов признаков
6. Особенности методов селекции признаков. Постановка задачи селекции признаков. Общность классификатора. Предобработка векторов признаков.
7. Селекция на основе проверки статистических гипотез. Векторная селекция признаков. Мера отделимости классов. Оптимальная селекция признаков. Оптимальная селекция на основе нейронной сети.
8. Сущность и отличия комитетных методов решения задач распознавания. Теоретико-множественная постановка задачи выбора алгоритма. Комитеты. Комитеты линейных функционалов. Функция Шеннона.

Вопросы для самостоятельного изучения:

1. В чем различие процессов «обучения с учителем» и «обучения без учителя»?

2. Как измерять сходство между данными наблюдений?

3. Дайте определения различных мер внутриклассового расстояния и мер расстояния между классами.

4. Какие методики группировки данных Вам известны?

5. Как оценить качество группировки любой части данных? При- ведите примеры функций критериев группировки.

6. Проанализируйте недостатки и преимущества алгоритмов ближайшего соседа и дальнего соседа. Как выбрать наиболее подходящий алгоритм для группировки данных?

Литература:

1. Потапова, М. Н. Основы обработки изображений в полиграфии [Электронный ресурс] : учебное пособие / Потапова М. Н., Сахабутдинова Г. Ф. Кемерово : КемГУ, 2020. 112 с. ISBN 978-5-8353-2711-9.

2. Гук, А. П. Методы и технологии распознавания объектов по их изображению [Электронный ресурс] : учебно-методическое пособие / Гук А. П. Новосибирск : СГУГиТ, 2019. 138 с. ISBN 978-5-907052-39-0.

Самостоятельная работа по разделу 3

Цель работы: – изучить основные принципы «обучения без учителя» и методики группировки данных в условиях полной апостериорной неопределенности и получить навыки иерархической группировки данных с применением различных мер внутриклассового расстояния.

Порядок выполнения работы.

1. Для заданных (согласно варианту) двумерных данных наблюдений $\xi i = (xi, yi)$, $i = 1, 2, \ldots, 10$, в условиях полной априорной неопределенности установить меры подобия – вычислить расстояния от - 48 - каждой точки из заданного множества ξi до всех других точек ξj, $i \neq j$.

2. Построить полигон распределения расстояний dij и найти центры рассеяния.

3. Установить критерии объединения данных dmin и dmax.

4. Выполнить группировки данных по алгоритму ближайшего соседа и по алгоритму дальнего соседа.

5. Определить статистические оценки характеристик разброса внутри классов: минимальное, максимальное и среднее расстояния между парами точек, составляющих одну группу (один класс).

6. Выполнить сравнительный анализ результатов группировок по алгоритмам ближайшего соседа и дальнего соседа.

7. Оформить отчет о самостоятельной работе , который должен со- держать алгоритмы группировки, результаты расчетов, графические представления исходных данных и полученных групп, выводы.

Варианты заданий

Номер варианта i 1 2 3 4 5 6 7 8 9 10

xi 0 1 2 3 4 4 6 6 8 8 1

yi 3 0 3 1 2 5 2 5 2 6

xi 0 1 2 2 3 3 4 5 6 6 2

yi 2 3 0 3 1 4 5 2 2 3

xi 0 1 2 2 4 4 4 5 6 6 3

yi 2 4 0 4 1 4 5 3 3 4

xi 1 2 3 3 4 4 5 5 7 8 4

yi 3 3 0 4 1 6 3 7 3 4

xi 0 1 2 3 3 5 5 7 7 8 5

yi 1 3 -1 -1 2 0 3 0 3 -1

xi 0 2 2 4 4 6 6 8 8 9 6

yi 2 -1 4 -1 3 0 3 -1 4 1

xi 0 1 1 2 2 3 3 4 5 6 7

yi 0 -1 3 0 2 0 3 1 3 1

xi 1 2 3 4 5 5 6 6 7 8 8

yi 1 4 0 5 1 4 0 1 4 3 - 48

Решите задания:

Задание №1 Разработать алгоритм и реализовать программу «Распознавание простых геометрических фигур» (круг, квадрат, треугольник, прямоугольник, эллипс) методом сравнения с эталоном. Измерить время выполнения алгоритма.

Задание №2 Разработать алгоритм и реализовать программу «Распознавание простых геометрических фигур» (круг, квадрат, треугольник, прямоугольник, эллипс) структурным методом. Представление фигур осуществить в виде дерева. Измерить время выполнения алгоритма.

Задание №3 Разработать алгоритм и реализовать программу «Распознавание простых геометрических фигур» (круг, квадрат, треугольник, прямоугольник, эллипс) структурным методом. Представление фигур осуществить в виде кода Фримена. Измерить время выполнения алгоритма.

Задание №4. Сравнить результаты выполнения программы заданий 1 и 2. 2. Сравнить результаты выполнения программы лабораторных работ 1 и 2. 3. Сделать выводы.

Задание №5 Разработать алгоритм и реализовать программу «Распознавание сложных изображений (домов)» структурным методом. Представление изображений осуществить в виде кода Фримена.

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ ЦИФРОВАЯ ОБРАБОТКА СИГНАЛОВ

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

Лабораторная работа № 1 Быстрое проебразование Фурье.

Цель работы: изучить основные принципы преобразования Фурье дискретных сигналов.

Основные приемы выполнения работы.

Дискретное преобразование Фурье (ДПФ) Основные функции в системе Mathcad.

1. Классические формулы ДПФ для периодической последовательности

$$x(nT) = x(nT + mNT) \qquad (1)$$

$$X(k) = \sum_{n=0}^{N-1} x(nT) e^{-j\frac{2kn}{N}\pi} \qquad (2)$$

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k\Omega) e^{j\frac{2kn}{N}\pi} \qquad (3)$$

$k = 0, 1, \ldots N-1$, $n = 0, 1, \ldots N-1$, $\Omega = 2\pi/(NT)$ - осн. частота.

2. Функции Mathcad fft и ifft образуют преобразования Фурье, отличные от (2) и (3).

$$X(k) = \frac{1}{\sqrt{N}} \cdot \sum_{n=0}^{N-1} x(nT)\, e^{j\frac{2kn}{N}\pi} \qquad k = 0, 1, \ldots N/2 \qquad (3)$$

$$x(n) = \frac{1}{\sqrt{N}} \cdot \sum_{k=0}^{N-1} X(k\Omega) e^{-j\frac{2kn}{N}\pi} \qquad (4)$$

Примеры: X=fft(x), x=ifft(X), где x – это массив чисел $x_n, n = 0, 1, \ldots N-1$, число n должно быть равно степени 2, числа x_n должны быть действительные. Функция fft возвращает $k = 0, 1, \ldots N/2$ значений.

Функции X=cfft(x) и x=icfft(X) работают также по формулам (4) и (5), но им необязательно, чтобы число n должно быть равно степени 2, а также значения x_n могут не быть действительными. Кроме того icfft(X) возвращает $N-1$ значение.

3. Функции Mathcad FFT и IFFT образуют преобразования Фурье, отличные от (2) - (5).

$$X(k) = \frac{1}{N} \cdot \sum_{n=0}^{N-1} x(nT)\, e^{-j\frac{2kn}{N}\pi} \qquad k = 0, 1, \ldots N/2 \qquad (4)$$

$$x(n) = \sum_{k=0}^{N-1} X(k\Omega) e^{j\frac{2kn}{N}\pi} \qquad (5)$$

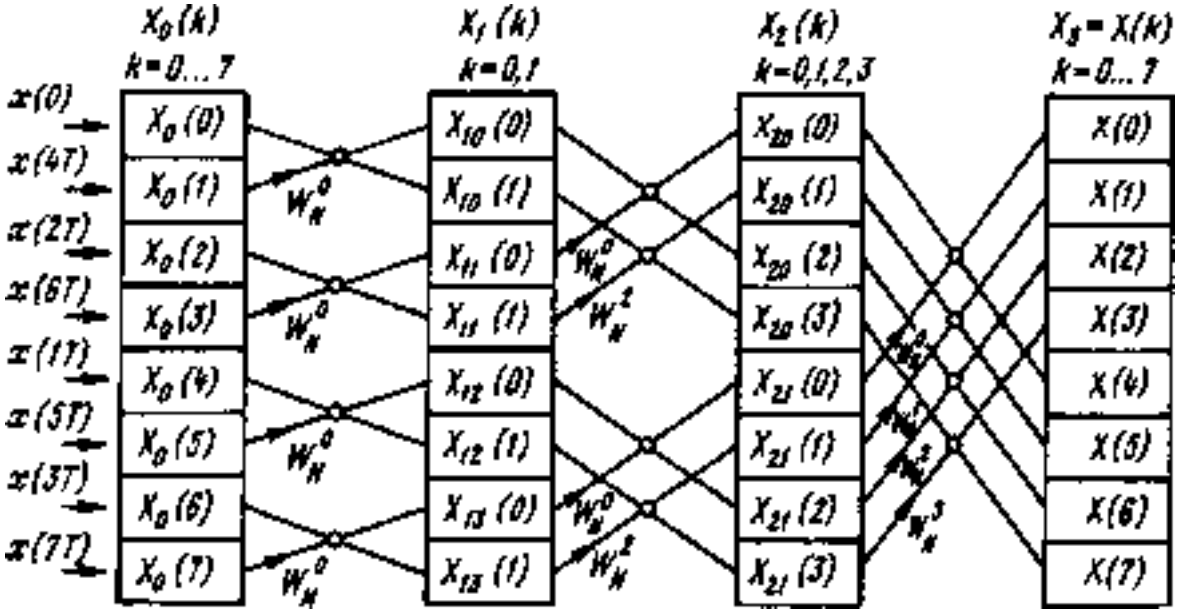
Примеры: X= FFT(x), x= IFFT(X), также, как и для fft(x) число n должно быть равно степени 2, числа x_n должны быть действительные.

Алгоритмы быстрого преобразования Фурье.

Использование БПФ основано на операции «бабочка». Для алгоритма с прореживанием по времени эта операция приведена на следующем рисунке.

При этом $c = a + b \cdot W_N^k, \quad d = a - b \cdot W_N^k, \; W_N = e^{-j \cdot \frac{2\pi}{N}}$.

Алгоритм БПФ для N=8 приведен ниже



Порядок выполнения работы

1. Выполнить дискретное преобразование Фурье (прямое и обратное) по классическим формулам (2) и (3) для следующих периодических последовательностей:

а) {0, 0, 1, 1};

б) {1, 1, 0, 0};

в) {1, 1, 1, 1, 0, 0, 0, 0};

г) {1, 1, 0, 0, 1, 1, 0, 0};

д) { 0, 1, 2, 3, 4, 3, 2, 1, 0, -1, -2, -3, -4, -3, -2, -1};

е) {-1, -2, -3, -4, -4, -3, -2, -1, 1, 2, 3, 4, 4, 3, 2, 1};

Результаты представить в виде таблиц и графиков.

2. Дискретное преобразование Фурье последовательности с ярко выраженной гармонической составляющей.

2.1. Выполнить дискретное преобразование Фурье (прямое и обратное) для последовательности x_n (n=0, 1, … N-1), содержащей ярко выраженную гармоническую составляющую.

$x_n = a_1 \cdot \sin(b_1 \cdot \Omega \cdot n) + a_2$, где $\Omega = \dfrac{2 \cdot \pi}{N \cdot T}$, T=1, a_1,= b_1=1, a_2=0, N=16.

В отчет здесь и далее включать реализацию и спектр на интервале 0 … N-1.

2.2. Задавая b_1=2, 3, 4, 6 выполнить дискретные преобразования Фурье для каждого значения b_1. Сделать выводы. Задавая b_1=7, 8, 9, 10 выполнить дискретные преобразования Фурье для каждого значения b_1. Сделать выводы.

2.3. Задавая b_1=2.1, 2.3, 2.5, выполнить дискретные преобразования Фурье для каждого значения b_1. Сделать выводы.

2.4. Изменяя a_1 и b_1 наблюдать изменение спектра. Сделать выводы. Увеличить N в 4 раза, сделать выводы.

3. Дискретное преобразование Фурье последовательности с двумя ярко выраженными гармоническими и случайной составляющими.

3.1. Выполнить дискретное преобразование Фурье для последовательности, содержащей две ярко выраженные гармонические составляющие и случайную составляющие.

$$x_n = a_1 \cdot \sin(b_1 \cdot \Omega \cdot n) + a_2 \cdot \sin(b_2 \cdot \Omega \cdot n) + rnd(b_3) + a_3$$

a_3=b_3=0, N=64

Таблица

Вар.	1	2	3	4	5	6	7	8	9	10	11	12
a_1	1	1.2	1.5	1	1.2	1.5	1	1.2	1.5	1	1.2	1.5
a_2	0.4	0.4	0.5	0.5	0.8	0.4	0.2	0.5	0.8	0.4	0.3	0.5
b_1	1	2	1	3	1	2	1	2	1	2	1	2
b_1	5	9	4	11	5	11	6	10	5	10	7	12

3.2. Изменяя a_1, a_2, b_1, и b_2 наблюдать изменение сигнала и его спектра. Сделать выводы.

3.3. Изменяя b_3 наблюдать изменение сигнала и его спектра. Сделать выводы.

3.4. Изменяя a_3 наблюдать изменение постоянной составляющей сигнала и его спектра. Сделать выводы.

4. Повторить п.п. 2 и 3 с использованием функций Mathcad fft и ifft. Сравнить полученные результаты, сделать выводы.

5. Повторить п.п. 2 и 3 с использованием функций Mathcad FFT и IFFT. Сравнить полученные результаты, сделать выводы.

6. В системе Mathcad выполнить быстрое преобразование Фурье (прямое и обратное) для следующих периодических последовательностей:

а) {0, 0, 1, 1};

б) {1, 1, 0, 0};

в) {1, 1, 1, 1, 0, 0, 0, 0};

г) {1, 1, 0, 0, 1, 1, 0, 0};

д) { 0, 1, 2, 3, 4, 3, 2, 1, 0, -1, -2, -3, -4, -3, -2, -1};

е) {-1, -2, -3, -4, -4, -3, -2, -1, 1, 2, 3, 4, 4, 3, 2, 1};

Результаты представить в виде таблиц и графиков.

Сравнить с результатами, полученными в предыдущей работе.

7. На языке С составить программу вычисления быстрого преобразования Фурье (прямого и обратного) для произвольного N. Выполнить контрольные просчеты для последовательностей, заданных в п. 1.

Вычислить БПФ для заданного сигнала.

8. (Выполнять по согласованию с преподавателем)

На языке С составить программу вычисления дискретного преобразования Фурье (прямого и обратного) для произвольного N. Выполнить контрольные просчеты для последовательностей, заданных в п. 1.

Сравнить время решения программ, составленных в п.2 и п.3 для N=8, N=16, N=256, N=2048, .

Лабораторная работа №2.

Разработка и изучение свойств цифровых фильтров**.**

Цель работы: изучение основных приемов построения цифровых фильтров и их реакции на типовые сигналы.

Основные приемы выполнения работы.

Сигнал $y(nT)$ на выходе цифрового фильтра определяется по следующей формуле

$$y(nT) = -\sum_{m=1}^{M-1} a_m\, y(nT - mT) + \sum_{k=0}^{N-1} b_k\, x(nT - kT) \quad (1)$$

Для нерекурсивного фильтра сигнал $y(nT)$ на выходе определяется по формуле

$$y(nT) = \sum_{k=0}^{N-1} b_k\, x(nT - kT) \quad (2)$$

В системе MathCad формула (2) может быть представлена следующим образом

$$y_n := \sum_{k=0}^{N-1} b_k x_{n-k}, \quad \text{где} \quad n := 0, 1, \ldots N-1 \quad k := 0, 1, \ldots K-1 \quad (3)$$

В системе MathCad в формуле (3) может возникнуть ситуация, когда $n < k$., Исключить эту ситуацию можно различными способами.

Один из способов заключается в том, что при вычисления по формуле (5) проводятся для $n := K-1, K, \ldots N-1$, а для $n < K - 1$ вычисления проводятся по упрощенным формулам. Например, пусть $K = 4$, в этом случае

$$y_n := b_0 x_n + b_1 x_{n-1} + b_2 x_{n-2} + b_3 x_{n-3}, \text{ где } n := K-1, K, \ldots N-1$$

$$y_0 := b_0 x_0$$

$$y_1 := b_0 x_1 + b_1 x_0$$

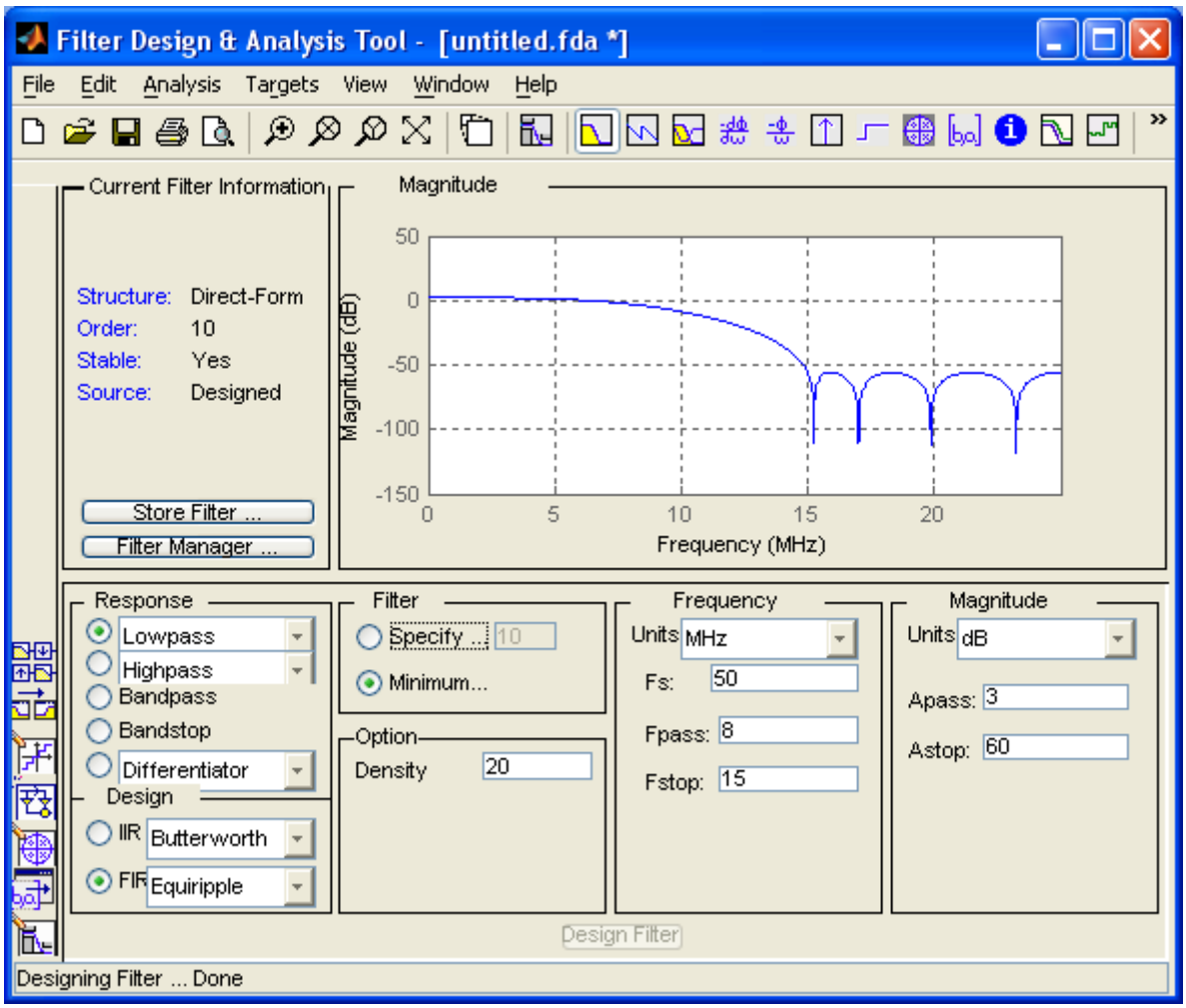
$$y_1 := b_0 x_2 + b_1 x_1 + b_2 x_0$$

Частотные характеристики фильтра задаются в fdatool, это позволяет определить коэффииенты фильтра.

Если задана частота дискретизации Fs=50Мгц, частота пропускания Fpass=8Мгц, частота подавления Fstop=8Мгц, то нерекурсивный фильтр нижни частот имеет 11 коэффициентов.

```
-0.01403751492090684
-0.035364181368855156
-0.0046746716790871442
 0.13760379002098341
 0.3371256672289798
 0.43396325771758426
 0.3371256672289798
 0.13760379002098341
-0.0046746716790871442
-0.035364181368855156
-0.01403751492090684
```

Частотная характеристика имеет следующий вид



Лабораторная работа проводится в системе Mathcad или в другой подобной системе.

Порядок выполнения работы

1. Разработать нерекурсивный фильтр. Определить реакцию на единичный импульс и единичную последовательность. Определить реакцию фильтра на заданный сигнал.

№ вар.	b_0	b_1	b_2	b_3	b_4
1,7	0.1	0.8	0.5	0.3	0.2
2,8	1	0.1	0.33	0.1	0.1
3,9	0.7	1	0.5	0.5	0.5
4,10	0.6	1.5	0.2	0.25	0.25
5,11	1	0	1.5	0.5	0.5
6,12	2.5	1.5	0.2	2	0.2

2. Разработать рекурсивный фильтр первого порядка. Определить реакцию на единичный импульс и единичную последовательность. Определить реакцию фильтра на заданный сигнал.

№ вар.	a_1	b_0	b_1
1,7	0.25	0.3	0.8
2,8	0.4	0.75	0.3
3,9	0.3	0.5	1
4,10	0.3	0.6	1.5
5,11	0.2	1	0.3
6,12	0.3	0.5	0.3

Изменяя a_1 и b_0 сделать выводы о влиянии этих коэффициентов на работу фильтра.

Задать a_1 = -0.5, b_0 = 0.5, b_1 = 0.0. Определить реакцию на единичный импульс, единичную последовательность и прямоугольный импульс различной длительности.

3. Разработать рекурсивный фильтр второго порядка. Определить реакцию на единичный импульс. Определить реакцию фильтра на заданный сигнал.

№ вар.	A_1	a_2	b_0	b_1	b_2
1	0.25	1	0	0.8	0.5
2	0.4	0.5	0.75	0	1
3	0.3	0.375	0	1	0.5
4	0	0.5	0.6	1.5	0.25
5	0.2	0.1	1	0	0.5
6	0	0.2	0.5	0.3	1

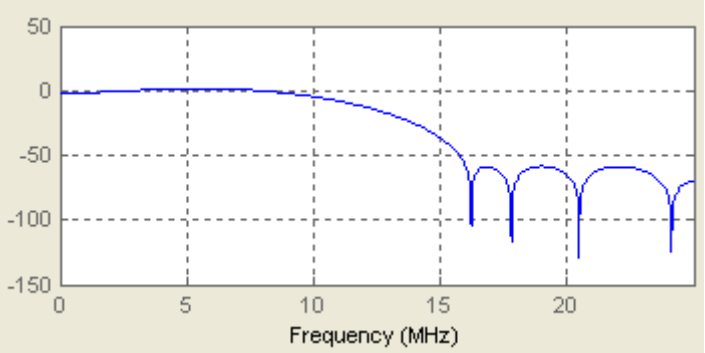
Изменяя a_1 и a_2, сделать выводы о влиянии этих коэффициентов на работу фильтра.

4. Разработать фильтр нижних частот по заданным коэффициентам. Определить реакцию на единичный импульс. Проверить частотные характеристики фильтра.

Вариант 1

Частота дискретизации Fs=50 Мгц, частота пропускания Fpass=6 Мгц, частота подавления Fstop=15 Мгц, фильтр имеет 10 коэффициентов.

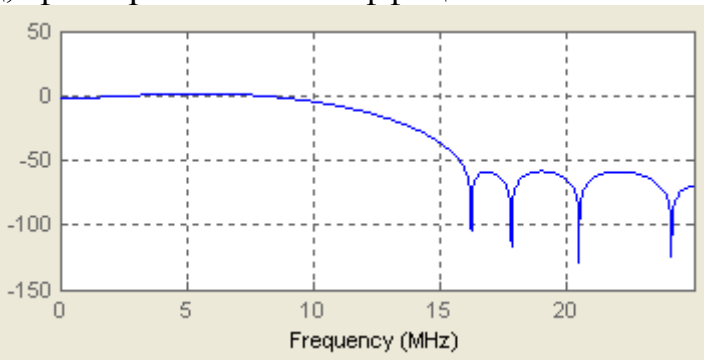
```
-0.012235833943837284
-0.01415399787471382
 0.053238444354665869
 0.20670417469779637
 0.34751888576536649
 0.34751888576536649
 0.20670417469779637
 0.053238444354665869
-0.01415399787471382
-0.012235833943837284
```



Вариант 2

Частота дискретизации Fs=50 Мгц, частота пропускания Fpass=9 Мгц, частота подавления Fstop=16 Мгц, фильтр имеет 11 коэффициентов.

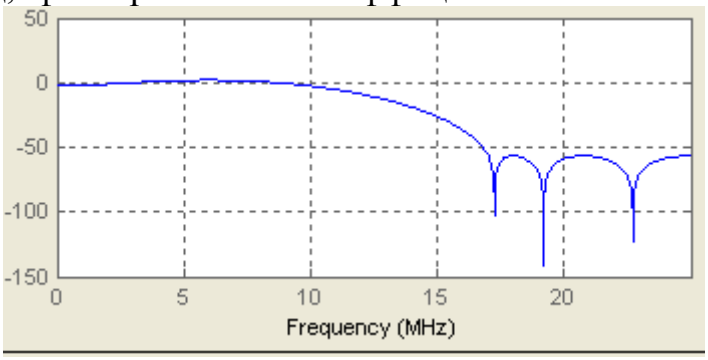
```
-0.021341618655808161
-0.075504404788717061
-0.087822853825447689
 0.054616807636556339
 0.30856450041518901
 0.44087057309082456
 0.30856450041518901
 0.054616807636556339
-0.087822853825447689
-0.075504404788717061
-0.021341618655808161
```



Вариант 3

Частота дискретизации Fs=50 Мгц, частота пропускания Fpass=10 Мгц, частота подавления Fstop=17 Мгц, фильтр имеет 11 коэффициентов.

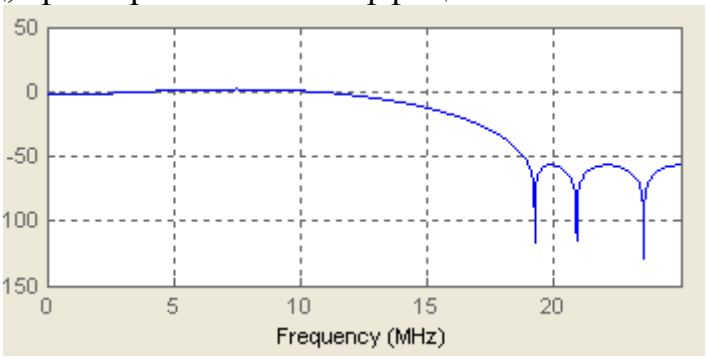
```
-0.015047699585687101
-0.074061388993217139
-0.1131287963326864
 0.021155036917353834
 0.31485181721566735
 0.47767333114390342
 0.31485181721566735
 0.021155036917353834
-0.1131287963326864
-0.074061388993217139
-0.015047699585687101
```



Вариант 4

Частота дискретизации Fs=50 Мгц, частота пропускания Fpass=12 Мгц, частота подавления Fstop=19 Мгц, фильтр имеет 11 коэффициентов.

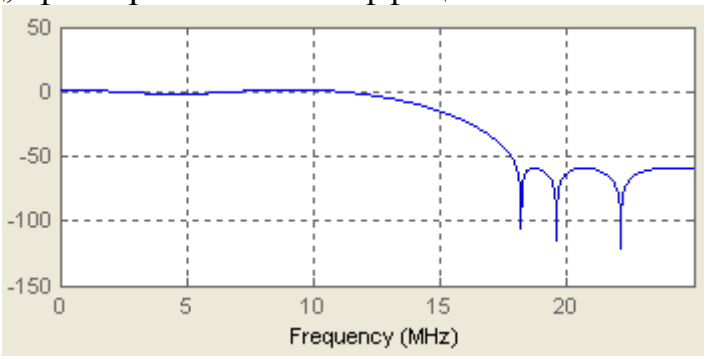
```
 0.010575582748755303
-0.023004458287932062
-0.12991847133240625
-0.070394066341376585
 0.30720026325852312
 0.56104959645895658
 0.30720026325852312
-0.070394066341376585
-0.12991847133240625
-0.023004458287932062
 0.010575582748755303
```



Вариант 5

Частота дискретизации Fs=50 Мгц, частота пропускания Fpass=12 Мгц, частота подавления Fstop=18 Мгц, фильтр имеет 13 коэффициентов.

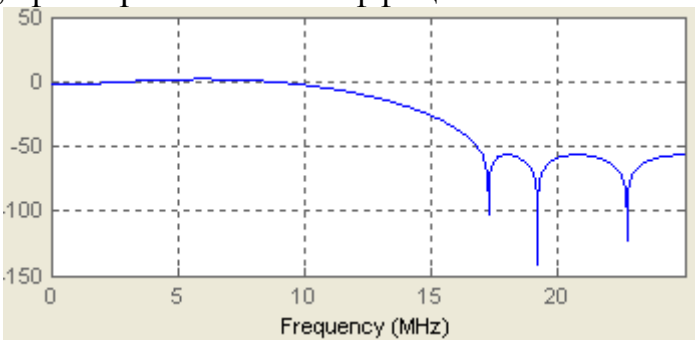
```
 0.02576679113357758
 0.078332741727956068
 0.045822453008765675
-0.089737835499959784
-0.045412711250419562
 0.31159703782721937
 0.5475647228387156
 0.31159703782721937
-0.045412711250419562
-0.089737835499959784
 0.045822453008765675
 0.078332741727956068
 0.02576679113357758
```



Вариант 6

Частота дискретизации Fs=50 Мгц, частота пропускания Fpass=10 Мгц, частота подавления Fstop=17 Мгц, фильтр имеет 11 коэффициентов.

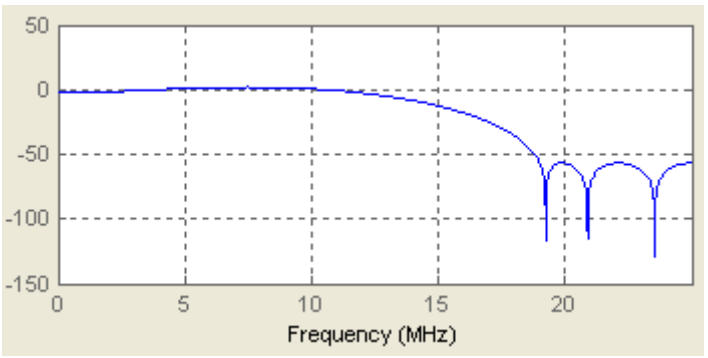
```
-0.016729038998476484
-0.077026546292626633
-0.11391441909396161
 0.022967295803103763
 0.31512920624712049
 0.47592671423840527
 0.31512920624712049
 0.022967295803103763
-0.11391441909396161
-0.077026546292626633
-0.016729038998476484
```



Вариант 7

Частота дискретизации Fs=50 Мгц, частота пропускания Fpass=12 Мгц, частота подавления Fstop=19 Мгц, фильтр имеет 11 коэффициентов.

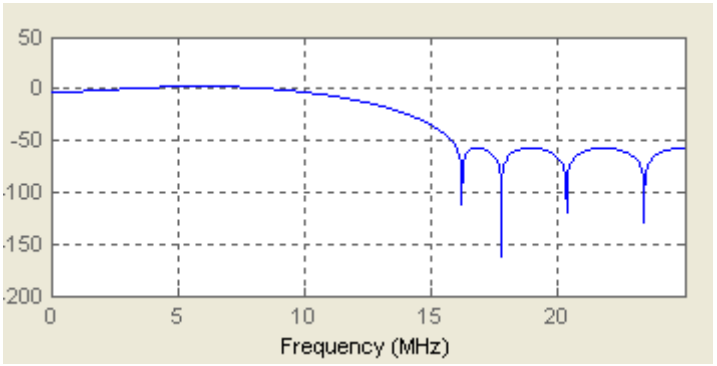
```
 0.0059226720571836141
-0.034874713727689979
-0.13732891493126465
-0.065844151438846865
 0.31074597857761171
 0.55886254515092315
 0.31074597857761171
-0.065844151438846865
-0.13732891493126465
-0.034874713727689979
 0.0059226720571836141
```



Вариант 8

Частота дискретизации Fs=50 Мгц, частота пропускания Fpass=10 Мгц, частота подавления Fstop=16 Мгц, фильтр имеет 13 коэффициентов.

```
 0.00058729941304685584
-0.02488601812927037 5
-0.09270779634637324 5
-0.11845189297549351
 0.032807846780783481
 0.31633952080514344
 0.4659836859309443
 0.31633952080514344
 0.032807846780783481
-0.11845189297549351
-0.09270779634637324 5
-0.02488601812927037 5
 0.00058729941304685584
```



Лабораторная работа №3.

Изучение архитектуры процессора цифровой обработки сигналов

Цель работы: изучение архитектуры процессора, порядка выполнения команд, а также способов адресации процессора цифровой обработки сигналов.

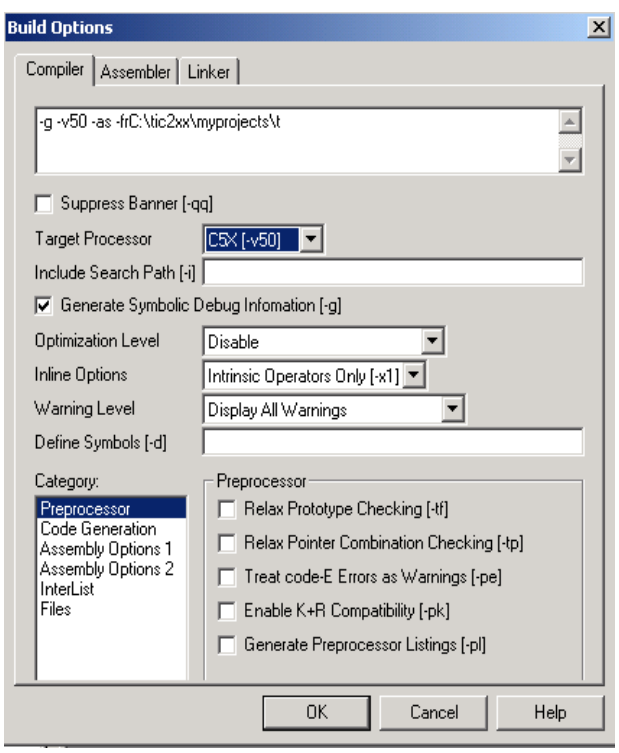
Изучение архитектуры сигнального процессора TMS320C5x проводится с использованием программы моделирования. При изучении системы команд следует особое внимание уделить специфике выполнения команд, специально предназначенных для цифровой обработки сигналов.

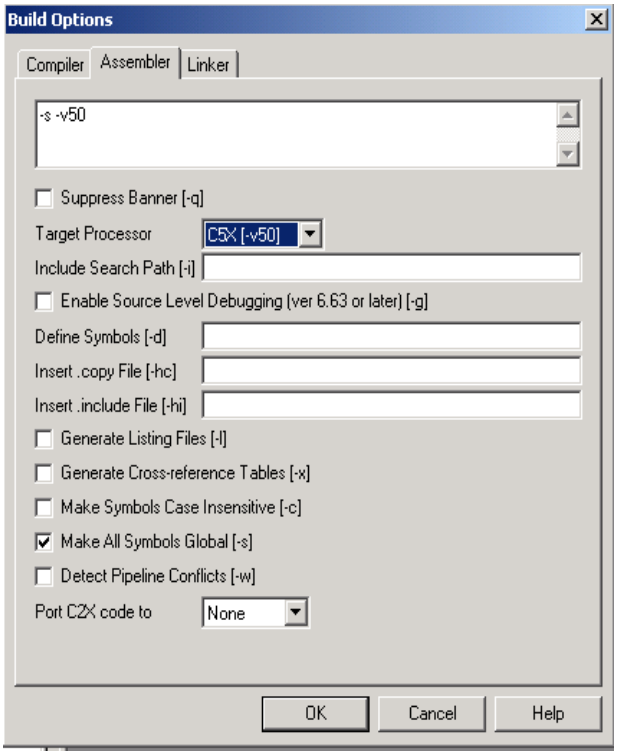
Пример простейшей программы:

```
    .text          ; начало программы в ПЗУ программ
    .global start
start:  ldp  #4h
        lacc #1234h,12
    .end
```

Для компилирования и выполнения программы на ассемблере необходимо выполнить следующие действия:

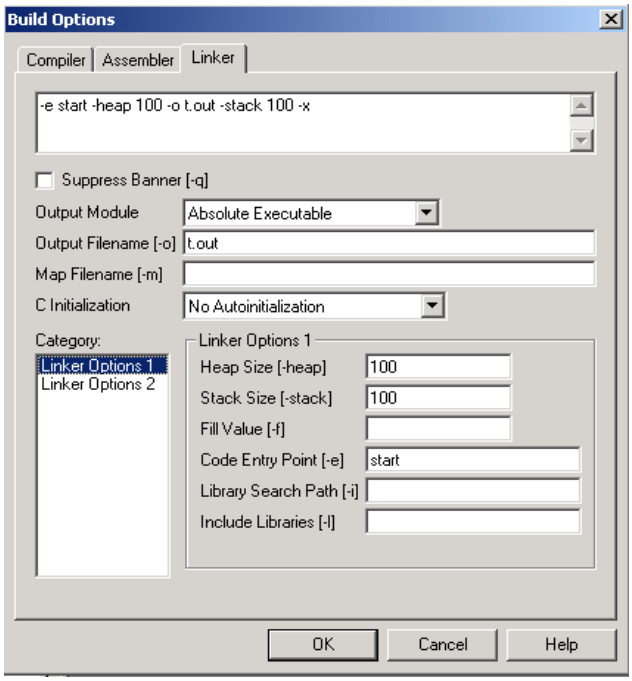
1. Создать новый проект, меню Project ⇒ New.
2. Добавить в проект файл с программой на ассемблере,
   меню Project ⇒ Add Files to Project.
3. Установить свойства проекта, меню Project ⇒ Options.
   В закладке Compiler установить Target Processor=C5X.





В закладке Assembler установить Target Processor=C5X, а также установить флажок Generate Listing Files.

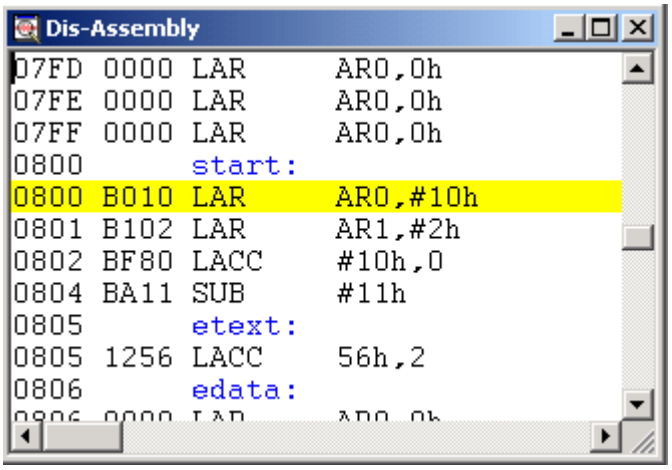
В закладке Linker установить Output Module=Absolute Executable, а также C Initialization=No Autoinitialization,



Там же установить нужное значение Heap Size, Stack Size и в поле Code Entry Point написать имя метки начала программы.

4. Для компиляции и запуска программы выбрать меню Project ⇒ Rebuild All или аналогичную кнопку на панели инструментов. Для запуска программы после компиляции в менюOptions ⇒ Program Load установить флаг Load Program After Build.

5. После компиляции появится окно Dis-Assembly.



Для пошагового выполнения программы служат кнопки Single Step или Step Over, клавиши F8 и F10. Чтобы вывести регистры, надо выбрать меню View ⇒ CPU Registers ⇒ CPU Registers. Для просмотра памяти выбрать View ⇒ Memory, для редактирования Edit ⇒ Memory ⇒ Edit, или выполнить двойной щелчок мышью на нужном элементе.

Порядок выполнения работы.

1. Изучить выполнение команд пересылок lacc, ldp, zap, sach, sacl, lt, lar, mar и других. Для этого надо последовательно составить ряд программ, в каждую из которых включить от 1 до 4 вышеперечисленных команд.

Команды lacc, sach и sacl выполнить со сдвигом данных на n разрядов. (n = 0, 1, 2, 4, 8, 16).

В отчет необходимо включить программу и протокол пошагового выполнения команды с указанием содержимого требуемых регистров процессора.

Данные, используемые в программе, располагать в четвертой странице ОЗУ (начиная с адреса 200h).

2. Изучить выполнение арифметических команд add, abs, mac, macd, apac, mpy. Особое внимание обратить на выполнение команд mac и macd в сочетании с командой rpt. В отчет необходимо включить программу и протокол пошагового выполнения команды с указанием содержимого

требуемых регистров процессора. Данные в ОЗУ располагать в четвертой странице (начиная с адреса 200h).

3. Изучить способы адресации по указанию преподавателя.

Реализоватьпо указанию преподавателя способы, приведенные в [2].

4. Составить программу по заданию преподавателя.

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ ЧИСЛЕННЫЕ МЕТОДЫ АЛГЕБРЫ И БЕЗУСЛОВНОЙ ОПТИМИЗАЦИИ В СИСТЕМАХ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Направление подготовки магистратуры
09.04.01 Информатика и вычислительная техника

Магистерская программа
«Прикладной искусственный интеллект»

Форма обучения очная

2021

СОДЕРЖАНИЕ

1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ЧИСЛЕННЫХ МЕТОДОВ

1.1. Математическое моделирование и вычислительный эксперимент

В современных науке и технике важную роль играет *математическое моделирование* [1 – 2], заменяющее эксперименты с реальными объектами экспериментами с их математическими моделями. Возник даже термин "*вычислительный эксперимент*". Вычислительный эксперимент имеет ряд преимуществ по сравнению с натурным экспериментом:

– экономичность, так как не тратятся ресурсы реальной системы;

– возможность моделирования гипотетических, т.е. не реализованных в природе объектов (прежде всего на разных этапах проектирования);

– возможность реализации режимов, опасных или трудновоспроизводимых в натуре (критический режим ядерного реактора, работа системы противоракетной обороны, природные и техногенные катастрофы);

– возможность изменения масштаба времени;

– простота многоаспектного анализа;

– большая прогностическая сила вследствие возможности выявления общих закономерностей.

Для практических задач довольно редко удается найти аналитическое решение уравнений, составляющих математическую модель явления. Поэтому приходится применять численные методы. Сущность применения численных методов рассмотрим на схеме вычислительного эксперимента [3], показанной на рис. 1.1.

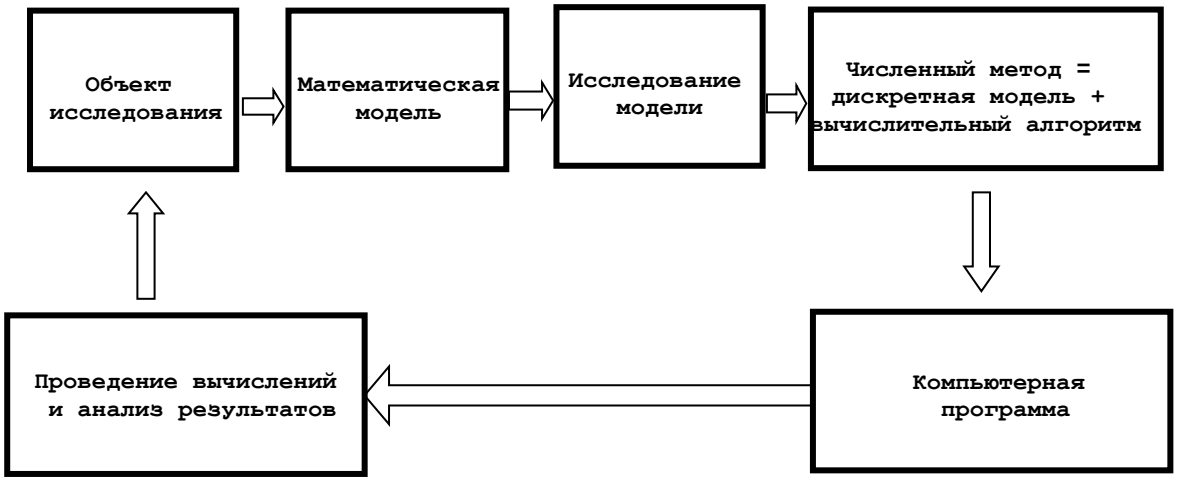


Рис. 1.1. Схема вычислительного эксперимента

Основу вычислительного эксперимента составляет триада: *модель – метод (алгоритм) – программа*. Сначала строится с некоторыми допущениями *математическая модель объекта.* Первоначально строится относительно простая, но достаточно полная с точки зрения экспериментальных данных модель. В ходе вычислительного эксперимента модель уточняется и дополняется. Поэтому можно говорить о *наборе математических моделей*, каждая из которых с различной точностью описывает объект или различные свойства объекта. Построение математических моделей выходит за рамки нашего курса. Более подробную информацию о построении математических моделей можно найти в [2].

После выбора или построения математической модели средствами прикладной математики проводится *предварительное качественное исследование модели*. Качественное исследование начинается с *приведения задачи к безразмерному виду* [4]. Суть приведения задачи к безразмерному виду состоит в выделении характерных значений (величин) и масштабировании всех величин задачи. Приведение задачи к безразмерному виду сокращает общее число параметров математической модели, что упрощает анализ. Кроме того, в безразмерной задаче можно сравнивать параметры задачи и упрощать задачу, отбрасывая малые параметры. Наконец, приведение к безразмерному виду уменьшает влияние ошибок округления при вычислениях на компьютере, т. к. все безразмерные величины изменяются примерно от - 1 до + 1. Приведение задачи к безразмерному виду представляет достаточно сложную задачу, так как основано на теории подобия [4] и требует знания особенностей процессов, протекающих в объекте исследования. В частности, масштабы разных величин оказываются взаимосвязанными. Математическая модель часто оказывается очень сложной для качественного исследования. В этом случае для качественного исследования строят *модельные (упрощённые) задачи*. После приведения к безразмерному виду рассматриваются *существование* и *единственность* решения, основные *свойства решения*, влияние различных параметров задачи на решение, устойчивость решения относительно малых возмущений входных данных и другие вопросы. Качественное исследование модели зачастую представляет собой сложную самостоятельную задачу. Однако для многих прикладных задач, например, для обыкновенных дифференциальных уравнений качественный анализ уже проведен.

Затем необходимо решить систему уравнений, представляющую собой математическую модель объекта. Как уже говорилось, обычно приходится применять численные методы. Под *численным методом* понимается

совокупность *дискретной модели*, реализуемой на компьютере, и *вычислительного алгоритма*, позволяющего решить дискретизированную задачу. Например, дискретной моделью вычисления определенного интеграла является вычисление суммы площадей прямоугольников, аппроксимирующих площадь криволинейной трапеции, являющейся геометрической интерпретацией задачи. Для реализации численного метода необходимо разработать программу на одном из языков программирования или применить готовый пакет прикладных программ. В настоящее время существуют пакеты прикладных программ, например, NumPy на языке Python, системы компьютерной математики, такие как MathCAD, MATLAB, Maple, Mathematica и другие, позволяющие решить большинство практически встречающихся задач. Однако грамотная постановка задачи, рациональный выбор метода решения, оценка погрешности и правильная интерпретация результатов требуют серьезных знаний численных методов. После отладки программы производятся вычисления на компьютере (обычно требуется провести много вариантов вычислений, для чего необходимо планировать вычислительный эксперимент) и анализ результатов. После получения результатов исследуется соответствие результатов вычислительного эксперимента процессу функционирования реального объекта (проверяется *адекватность модели*), и при необходимости уточняются компоненты схемы вычислительного эксперимента (рис. 1.1) до получения удовлетворительных результатов.

Можно выделить три основных типа вычислительного эксперимента:

- *поисковый*, заключающийся в исследовании на математических моделях различных объектов и процессов;
- *оптимизационный*, направленный, например, на поиск оптимальных характеристик конструкций, технологических процессов;
- *диагностический*, когда по результатам натурных экспериментов определяются свойства объектов или явлений (при этом решаются *обратные задачи*).

Как уже было сказано, численные методы используют дискретные модели. Огромное число вычислительных алгоритмов работают с матрицами и решают системы линейных алгебраических уравнений, т. е. основаны на *вычислительной линейной алгебре*. Например, дифференциальные уравнения в частных производных, описывающие различные физические процессы, путем дискретизации заменяются системами линейных алгебраических уравнений, содержащими тысячи и миллионы уравнений [5]. Вычислительная линейная алгебра наряду с математической статистикой и методами оптимизации является основой машинного обучения и науки о данных [6–7].

Одной и той же математической модели можно поставить в соответствие множество дискретных моделей и вычислительных алгоритмов, т. е. численных методов. Для выбора численных методов надо знать их основные свойства. Рассмотрение свойств численных методов начнем с анализа погрешностей вычислений.

1.2. Погрешности вычислений

1.2.1. Источники погрешностей вычислений

Исследование реального объекта методом вычислительного эксперимента носит приближенный характер. На каждом этапе вычислительного эксперимента (рис. 1.1) возникают погрешности. Можно выделить 5 источников погрешностей [3, 8–9].

1. Неустранимые по отношению к численному методу *погрешности математической моделей*. Эти погрешности связаны с недостаточным знанием или упрощением исходных физических явлений. То есть причиной этого вида погрешности является неадекватность математической модели. Мера адекватности математической модели может быть оценена путем сравнения реального физического процесса и реализации на компьютере модели процесса. Это очень сложная задача, т. к. реализация модели на компьютере включает много составляющих погрешностей, исследование реального процесса также происходит с погрешностью. Оценка погрешности математической модели выходит за рамки данного курса.

2. *Погрешность дискретизации*.

Это погрешность от замены непрерывной математической модели дискретной моделью. Примером дискретизации является замена дифференциального уравнения системой алгебраических уравнений. Решение системы алгебраических уравнений отличается от решения дифференциального уравнения, что и составляет погрешность дискретизации. Обычно дискретная модель зависит от некоторых параметров, изменением которых *теоретически* можно свести погрешность дискретизации к нулю. В рассмотренном примере, уменьшая шаг дискретизации (и увеличивая порядок системы алгебраических уравнений) можно уменьшить погрешность дискретизации.

3. *Трансформированная погрешность (погрешность искажения)* [8] – это погрешность, возникающая за счет *погрешности исходных данных*. Исходные данные, как правило, являются результатом измерений некоторых физических величин, естественно, эти измерения производятся с некоторой погрешностью. Кроме того, из-за ограниченной разрядности компьютеров

возникает *погрешность представления исходных данных* (погрешность округления исходных данных). Многие задачи являются весьма чувствительными к погрешностям исходных данных и погрешностям, возникающим в ходе реализации алгоритма. В этом случае говорят, что *задача является плохо обусловленной*. Для плохо обусловленной задачи погрешности исходных данных, даже при отсутствии погрешностей вычислений могут преобразоваться (трансформироваться) в значительные погрешности результата.

4. *Методические погрешности* возникают из-за применения приближенных алгоритмов. Например, вычисляя сумму рядя теоретически можно получить точное значение для бесконечного числа членов ряда. Практически всегда число членов приходится ограничивать, что приводит к погрешности метода.

5. *Погрешности округления* возникают из-за того, что все вычисления выполняются с ограниченным числом цифр, т. е. производится округление чисел. Погрешности округления могут накапливаться и при плохой обусловленности задачи могут привести к большим погрешностям результата.

Как выбирать допустимые погрешности? Известны различные рекомендации. В [3] предлагается обеспечивать одинаковый порядок всех погрешностей. То есть не следует очень точно решать задачу с неточными исходными данными. Точнее говоря, погрешность решения *всей* задачи (всей последовательности действий вычислительного эксперимента) должна быть соизмерима с погрешностью исходных данных. В [10] рекомендуется обеспечивать методическую погрешность в $2-10$ раз меньшую, чем погрешность модели, а погрешность округления – на порядок меньше погрешности метода.

1.2.2. Приближенные числа. Абсолютная и относительная погрешность

Мерой точности вычислений являются погрешности. Пусть a — приближенное значение точного числа A. *Погрешностью*, или *ошибкой* Δa приближенного числа a называется разность

$$\Delta a = a - A.$$

В качестве меры погрешности используют абсолютную и относительную погрешность. *Абсолютная погрешность* приближенного числа a определяется формулой

$$\Delta(a) = |a - A|.$$

Так как точное число A в большинстве случаев неизвестно, то используется оценка *предельной абсолютной погрешности* Δ_a, называемая также *границей абсолютной погрешности*:

$$\Delta(a) = |a - A| \le \Delta_a.$$

Относительная погрешность определяется формулой

$$\delta(a) = \frac{|a - A|}{|A|} = \frac{\Delta(a)}{|A|}.$$

Относительная погрешность часто выражается в процентах.

Предельной относительной погрешностью δ_a приближенного числа a называется число, не меньшее относительной погрешности этого числа

$$\delta(a) \le \delta_a.$$

Так как значение точного числа A неизвестно, то часто пользуются приближенными оценкам предельных погрешностей

$$\delta_a \approx \frac{\Delta_a}{|a|}, \quad \Delta_a \approx |a|\delta_a.$$

Очень часто используется термин "точность решения". Хотя точность решения противоположна по смыслу погрешности решения, для измерения точности используются те же характеристики, что и для измерения погрешности. Когда говорят, что точность решения равна ε, то это означает, что принятая мера погрешности решения не превышает ε.

Абсолютная и относительная погрешности тесно связаны с понятием *верных значащих цифр*. Значащими цифрами числа называют все цифры в его записи, начиная с первой ненулевой цифры слева. Например, число 0,00012900 имеет пять значащих цифр (значащие цифры подчеркнуты). Значащая цифра называется *верной*, если абсолютная погрешность числа не превышает вес разряда, соответствующего этой цифре. Значащая цифра называется *верной в узком смысле слова*, если абсолютная погрешность числа не превышает половины веса разряда, соответствующего этой цифре. Например, число равно $a = 9348$, абсолютная погрешность числа равна $\Delta(a) = 15$. Записывая число в виде

$$9348 = 9 \cdot 10^3 + 3 \cdot 10^2 + 4 \cdot 10^1 + 8 \cdot 10^0,$$

имеем $0{,}5 \cdot 10^1 < \Delta(a) < 0{,}5 \cdot 10^2$, следовательно, число имеет две верных в узком смысле значащих цифр (9 и 3).

Верная значащая цифра может не совпадать с соответствующей цифрой в записи точного числа. Например, $A = 1.000, \quad a = 0.999, \quad \Delta(a) = 0.001$, тогда у приближенного числа a все значащие цифры верные, но не совпадающие с цифрами точного числа A.

Количество верных значащих цифр числа связано со значением его относительной погрешности [10]. Если десятичное приближенное число a содержит n верных значащих цифр, то для относительной погрешности справедлива оценка

$$\delta(a) \le \left(10^{n-1} - 1\right)^{-1} \approx 10^{-n+1}.$$

Чтобы число a содержало n верных значащих цифр, достаточно выполнения неравенства

$$\delta(a) \le \left(10^{n} + 1\right)^{-1} \approx 10^{-n}.$$

Поэтому, если необходимо вычислить приближенное число a с *точностью* 10^{-n}, то необходимо сохранить верной значащую цифру, стоящую в n-разряде после десятичной запятой.

Тот факт, что число a является приближенным значением числа A с предельной абсолютной погрешностью Δ_a, записывают в виде

$$A = a \pm \Delta_a,$$

причем числа a и Δ_a записываются с одинаковым количеством цифр после запятой, например, $A = 2{,}347 \pm 0{,}002$ или $A = 2{,}347 \pm 2 \cdot 10^{-3}$.

Запись вида

$$A = a\left(1 \pm \delta_a\right)$$

означает, что число a является приближенным значением числа A с предельной относительной погрешностью δ_a. Предельные абсолютные и относительные погрешности принято записывать с одной или двумя значащими цифрами. Большая точность не имеет смысла, т. к. предельные погрешности представляют собой достаточно грубые оценки.

1.2.3. Особенности машинной арифметики

Одним из основных источников вычислительных погрешностей является приближенное представление чисел в компьютере, обусловленное конечностью разрядной сетки. При решении вычислительных задач обычно используют представление чисел в *форме с плавающей точкой (запятой)*. Число a в форме с плавающей точкой представляется в форме

$$a = Mr^{p},$$

где r – основание системы счисления (как правило, $r = 2$), p – порядок числа a, M – мантисса числа a, причем должно выполняться условие нормировки $r^{-1} \le M < 1$, означающее, что в компьютере хранятся только значащие цифры.

Диапазон изменения чисел в компьютере ограничен. Для всех представимых в компьютере нормализованных чисел x (за исключением нуля) справедливо неравенство

$$0 < X_0 \le |x| < X_\infty,$$

где X_0 – минимальное представимое в компьютере нормализованное число (*машинный нуль*)

$$X_0 = 2^{-(p_{\max}+1)},$$

$p_{\max} = 2^{l-1} - 1$ – максимальное по абсолютной величине значение порядка, l – разрядность порядка (полагаем, что один бит отведен под знак порядка);

$X_\infty = M_{\max} 2^{p_{\max}} = \left(1 - 2^{-t}\right) 2^{p_{\max}} \approx 2^{p_{\max}}$ – максимальное представимое в компьютере нормализованное число (*машинная бесконечность*), $M_{\max}$ – максимальное по абсолютной величине значение мантиссы, t – разрядность мантиссы

В большинстве языков программирования получение машинной бесконечности приводит к аварийному завершению программы по *переполнению*. Все числа по модулю меньшие X_0 представляются как *нуль (исчезновение порядка*).

Большинство современных компьютеров поддерживают *стандарт двоичной арифметики IEEE 754-1985* (ANSI 754) Binary floating-point arithmetic (1985 г.)[1] [11]. Представление чисел с плавающей точкой в этом стандарте несколько отличается от рассмотренного "классического" представления. Под знак числа отводится один бит: 0 соответствует положительному числу, 1 — отрицательному. Мантисса нормализованных двоичных чисел удовлетворяет условию $1 \le M < 2$, то есть мантисса нормализованного числа всегда содержит единицу в целой части. Так как целая часть всех нормализованных чисел рана единице, то эта единица не хранится, хранится только дробная часть мантиссы. Число получается прибавлением единицы к дробной части. Сэкономленный разряд используется

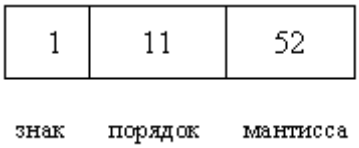
[1] IEEE — The Institute of Electrical and Electronics Engineers, Inc., (произносится "ай-трипл-и") Институт инженеров по электротехнике и радиоэлектронике (www.ieee.org/) — международная некоммерческая ассоциация специалистов в области техники, мировой лидер в области разработки стандартов по радиоэлектронике, электротехнике и аппаратному обеспечению вычислительных систем и сетей, была основана в 1963 году. IEEE, объединяя более 400000 индивидуальных членов из 170 стран (в том числе более 100 000 студентов), издаёт третью часть мировой технической литературы, касающейся применения радиоэлектроники, компьютеров, систем управления, электротехники, проводит в год более 300 крупных конференций. Ассоциация принимала участие в разработке около 900 действующих стандартов.

для хранения еще одного двоичного разряда. Математически представление мантиссы, принятое в стандарте эквивалентно "классическому" представлению, но увеличивает разрядность мантиссы и диапазон представления чисел. Порядок может быть как положительным, так и отрицательным. Чтобы не вводить бит знака порядка, используют смещенный порядок, прибавляя к порядку смещение, равное $2^{l-1}-1$, где l – число разрядов, отведенное под смещенный показатель. Например, если под порядок отведен один байт, то смещение равно $2^7-1=127$. Если порядок равен +4, то смещенный порядок равен $4+127=131$. Если порядок равен -4, то смещенный порядок равен $-4+127=123$. Число с нулевым порядком и нулевой мантиссой считается нулем. При этом формально возможны $+0$ и -0.

Стандарт предусматривает два основных типа чисел с плавающей точкой: числа одинарной и двойной точности. В стандарте предусмотрены также расширенный одинарный и расширенный двойной форматы, но они не имеют точного описания и практически мало применимы.

Числа одинарной точности (представляются 32 битами – 4 байтами)

числа двойной точности (представляются 64 битами – 8 байтами)



IEEE-арифметика включает *субнормальные числа* — очень малые *ненормализованные* числа, расположенные между 0 и наименьшим по абсолютной величине нормализованным числом X_0. При таких условиях нормализация чисел не производится. Субнормальные числа имеют нулевой смещенный порядок. Наличие субнормальных чисел обеспечивает то, что малые числа не превращаются в машинный нуль. Например, равенство $x-y=0$ возможно только при $x=y$.

IEEE-арифметика поддерживает специальные символы $\pm\infty$ (в MATLAB обозначается inf) и *NaN* . Символ $\pm\infty$ генерируются при переполнении. Бесконечность содержит единицы во всех разрядах смещенного порядка и нули в разрядах мантиссы. Правила для символа $\pm\infty$: $x/\pm\infty=0,\quad x/0=\pm\infty,\quad \infty+\infty=+\infty$ и т. д. Любая операция, результат которой (конечный или бесконечный) не определен корректно, генерирует

символ *NaN* (*Not a Number* – НеЧисло). Например, символ *NaN* генерируется при операциях: $\infty - \infty$, $\frac{\infty}{\infty}$, $\frac{0}{0}$, $NaN \square x$, где $\square$ – любая операция.

В случае, если в результате арифметической операции получается неопределенность (*NaN*), машинная бесконечность, деление на нуль, машинный нуль, процессор выставляет флаг исключительного состояния (exception flag). Большинство компиляторов по умолчанию не маскируют исключения с плавающей точкой. Это приводит к завершению программ, когда происходит исключение с плавающей точкой. Программист должен маскировать и обрабатывать исключения с плавающей точкой, что может сильно усложнить программу. Система *MATLAB* маскирует и обрабатывает исключительные ситуации, что существенно упрощает программирование.

В некоторых случаях оказывается недостаточно двойной точности стандарта *IEEE 754-1985*. Поэтому многие компиляторы поддерживают 80 битный формат Intel двойной точности. Некоторые компиляторы программно поддерживают увеличенную разрядность вычислений (при этом существенно увеличивается время вычислений). В новой редакции стандарта *IEEE 754-2008* предусмотрены числа учетверенной точности, мантисса которых содержит 112 бит, а порядок — 16 бит.

Для уменьшения погрешностей вычислений иногда используется *рациональная арифметика* — числа представляются в виде рациональных дробей и операции производятся как над обычными дробями. Рациональная арифметика реализована, например, в пакете *Mathematica*.

Существует также подход, известный как *интервальные вычисления* [12–13]. В интервальных вычислениях при выполнении каждой арифметической операции вычисляются границы ошибки. Однако алгоритмы интервальных вычислений оказываются сложными и медленными.

Получим некоторые оценки точности представления чисел и выполнения арифметических операций. Для простоты используем простейшее представление чисел с плавающей точкой, т. е. будем использовать условие нормировки мантиссы $2^{-1} \le M < 1$. Полученные результаты справедливы и для представления чисел в стандарте *IEEE 754*, если учесть, что в стандарте разрядность мантиссы фактически больше на единицу физически выделенной под мантиссу разрядности. Из-за конечной разрядной сетки в компьютере можно представить не все числа. Число a, не представимое в компьютере, подвергается округлению, т. е. заменяется близким числом $\tilde{a}$, представимым в компьютере точно.

Известно несколько способов округления числа до n значащих цифр. Простейший способ — *усечение*. При усечении отбрасываются все цифры,

расположенные справа от n-й значащей цифры. Наибольшее распространение получило *округление по дополнению*. В простейшем варианте этого способа анализируется первая отбрасываемая цифра. Если эта цифра равна 1 (в двоичной системе счисления), то к младшей сохраняемой цифре прибавляется единица.

Найдем границу *относительной погрешности представления числа с плавающей точкой* [3]. Допустим, что применяется простейшее округление – усечение. Система счисления — двоичная. Пусть надо записать число, представляющее бесконечную двоичную дробь

$$a = \pm \underbrace{2^{p}}_{\text{порядок}} \underbrace{\left(\frac{a_1}{2} + \frac{a_2}{2^2} + \cdots + \frac{a_t}{2^t} + \frac{a_{t+1}}{2^{t+1}} + \cdots \right)}_{\text{мантисса}},$$

где $a_j = \begin{cases} 0, \\ 1, \end{cases} \quad (j = 1,2,...)$ – цифры мантиссы.

Пусть под запись мантиссы отводится t двоичных разрядов. Отбрасывая лишние разряды, получим округленное число

$$\tilde{a} = \pm 2^{p} \left(\frac{a_1}{2} + \frac{a_2}{2^2} + \cdots + \frac{a_t}{2^t} \right).$$

Абсолютная погрешность округления в этом случае равна

$$\left| \tilde{a} - a \right| = 2^{p} \left(\frac{a_{t+1}}{2^{t+1}} + \frac{a_{t+2}}{2^{t+2}} + \cdots \right).$$

Наибольшая погрешность будет в случае $a_{t+1} = 1, \quad a_{t+2} = 1, \ldots$, тогда

$$\left| \tilde{a} - a \right| \le 2^{p} \frac{1}{2^{t+1}} \underbrace{\left(1 + \frac{1}{2} + \frac{1}{2^2} + \cdots \right)}_{=2} = 2^{p-t}.$$

Из условия нормировки для мантиссы всегда $a_1 = 1$, поэтому выполняется неравенство $M \ge 0{,}5$. Тогда $\left| a \right| \ge 2^{p} \cdot 2^{-1} = 2^{p-1}$ и относительную погрешность можно оценить следующим образом: $\delta(a) = \dfrac{\left| \tilde{a} - a \right|}{\left| a \right|} \le 2^{-t+1}$. Практически применяют более точные методы округления и для погрешности представления чисел справедлива оценка

$$\delta(a) = \frac{\left| \tilde{a} - a \right|}{\left| a \right|} \le 2^{-t}, \qquad (1.1)$$

т.е. точность представления чисел определяется разрядностью мантиссы t.
Тогда приближенно представленное в компьютере число можно записать в виде $\tilde{a} = a(1 \pm \varepsilon)$, где $\varepsilon = 2^{-t}$ – так называемый машинный эпсилон, или

машинная точность (часто обозначается *macheps*) – относительная погрешность представления чисел (1.1). В системе MATLAB переменная `eps` обозначает машинную точность `eps=2^-52`, что примерно равно `2.2204e-016`.

При вычислениях с плавающей точкой операция округления может потребоваться после выполнения любой арифметической операции. Так умножение или деление двух чисел сводится к умножению или делению мантисс и к сложению или вычитанию порядков. Так как в общем случае количество разрядов мантисс произведений и частных больше допустимой разрядности мантиссы, то требуется округление мантиссы результатов. При сложении или вычитании чисел с плавающей точкой операнды должны быть предварительно приведены к одному порядку, что осуществляется сдвигом вправо мантиссы числа, имеющего меньший порядок, и увеличением в соответствующее число раз порядка этого числа. Сдвиг мантиссы вправо может привести к потере младших разрядов мантиссы, т. е. появляется погрешность округления.

Обозначим округленное в системе с плавающей точкой число, соответствующее точному числу x, через $\mathrm{fl}(x)$(от англ. *floating* – плавающий). Известно [3], что *выполнение каждой арифметической операции вносит относительную погрешность, не большую, чем погрешность представления чисел с плавающей точкой* (1.1). Тогда можно записать

$$\mathrm{fl}(a \square b) = a \square b(1 \pm \delta),$$

где $\square$ – любая из арифметических операций, $\delta \le 2^{-t}$.

Таким образом, погрешность арифметической операции над числами, представленными в форме с плавающей точкой, не превышает машинный эпсилон.

1.3. Свойства вычислительных задач и алгоритмов

1.3.1. Обусловленность вычислительной задачи

Рассмотрим корректную вычислительную задачу. Корректность задачи теоретически позволяет найти ее решение со сколь угодно малой погрешностью при условии малых погрешностей исходных данных. Практически погрешность исходных данных всегда конечна и не может быть сколь угодно малой. Поэтому важно знать, как малые погрешности исходных данных трансформируются в погрешность результата. Погрешности исходных данных не могут быть устранены никакими методами решения. Практически

важно знать как при изменении (вариации) исходных данных изменится решение задачи.

Чувствительность решения вычислительной задачи к малым погрешностям исходных данных называется *обусловленностью задачи*. Задача называется *хорошо обусловленной*, если малым погрешностям исходных данных соответствуют малые погрешности решения. В противном случае задача называется *плохо обусловленной*.

Степень обусловленности задачи определяется *числом обусловленности*. Выделяют *абсолютное число обусловленности* ν_Δ, определяемое неравенством

$$\Delta(y) \le \nu_\Delta \Delta(x),$$

и *относительное число обусловленности* ν_δ, определяемое неравенством

$$\delta(y) \le \nu_\delta \delta(x).$$

Чаще всего используется относительное число обусловленности. Если $\nu \gg 1$, то задача плохо обусловлена. Практически можно считать [10], если $\nu_\delta \approx 10^N$, то N показывает число верных цифр, которое может быть потеряно по сравнению с числом верных цифр исходных данных.

В практике решения вычислительных задач большое внимание уделяется *обусловленности систем линейных алгебраических уравнений (СЛАУ)*. Рассмотрим, как влияют погрешности исходных данных на погрешность решения СЛАУ. Корректно поставленные задачи решения СЛАУ (т. е. задачи, решение которых существует, единственно и непрерывно зависит от входных данных) в зависимости от чувствительности решения к погрешности исходных данных делятся на хорошо и плохо обусловленные. Рассмотрим пример плохо обусловленной СЛАУ [14]. Система

$$1{,}1x + y = 1{,}1;$$
$$x + y = 1$$

имеет определитель

$$\begin{vmatrix} 1{,}1 & 1 \\ 1 & 1 \end{vmatrix} = 0{,}1$$

и решение $x = 1; \quad y = 0$. С учетом вариации одного из коэффициентов система примет вид

$$1{,}1x + y = 1{,}1;$$
$$(1+\varepsilon)x + y = 1$$

и имеет решение $x=\dfrac{1}{1-10\varepsilon}$; $y=\dfrac{-11\varepsilon}{1-10\varepsilon}$. Решение очень сильно зависит от вариации коэффициента. Например, если $|\varepsilon|\le 0,001$, то $0,990\le x\le 1,010$; $-0,011\le y\le 0,011$. Если $|\varepsilon|\le 0,010$, то $0,909\le x\le 1,110$; $-0,122\le y\le 0,100$. Если $|\varepsilon|\le 0,100$, то $0,500\le x\le\infty$; $-\infty\le y\le 0,550$. Таким образом, уже при $|\varepsilon|=0,1$ вариация решения может быть сколь угодно большой.

Для СЛАУ из двух уравнений явление плохой обусловленности имеет наглядный геометрический смысл. В этом случае каждое уравнение системы описывает прямую, а координаты точки пересечения прямых являются решением. Рассмотрим СЛАУ [15]

$$1000x_1+999x_2=b_1,$$
$$999x_1+998x_2=b_2.$$

Наклоны двух прямых, описываемых этими уравнениями соответственно равны: $k_1=-1000/999=-1,001001$, $k_2=-999/998=-1,001002$. То есть прямые практически параллельны. Найти точку пересечения этих прямых трудно. Небольшое изменение правой части одного из уравнений приводит к небольшому параллельному сдвигу соответствующей прямой. При этом точка пересечения прямых (решение СЛАУ) резко изменится.

Оценим число обусловленности СЛАУ. Рассмотрим систему линейных алгебраических уравнений, записанную в матричном виде

$$\mathbf{Ax}=\mathbf{b}, \qquad (1.2)$$

где $\mathbf{A}$ – невырожденная квадратная матрица коэффициентов, $\mathbf{b}$ – ненулевой вектор правой части, $\mathbf{x}$ – вектор искомого решения.

Пусть элементы матрицы $\mathbf{A}$ *заданы точно, а правая часть СЛАУ задана неточно*, т. е. используется возмущенный вектор $\mathbf{b}+\Delta\mathbf{b}$. Тогда изменится и решение системы:

$$\mathbf{A}(\mathbf{x}+\Delta\mathbf{x})=\mathbf{b}+\Delta\mathbf{b}. \qquad (1.3)$$

Если $\det\mathbf{A}\neq 0$, то из (1.2) и (1.3), получим[2]

$$\mathbf{Ax}+\mathbf{A}\Delta\mathbf{x}=\mathbf{b}+\Delta\mathbf{b},$$
$$\mathbf{A}\Delta\mathbf{x}=\Delta\mathbf{b}. \qquad (1.4)$$

Решение системы (1.4) запишем как

$$\Delta\mathbf{x}=\mathbf{A}^{-1}\Delta\mathbf{b}.$$

Откуда

$$\|\Delta\mathbf{x}\|=\|\mathbf{A}^{-1}\Delta\mathbf{b}\|,$$

[2] Определение и свойства норм векторов и согласованных норм матриц см. в разделе 2.1

$$\|\Delta\mathbf{x}\| \le \|\mathbf{A}^{-1}\| \cdot \|\Delta\mathbf{b}\|, \qquad (1.5)$$

где под $\|\Delta\mathbf{x}\|$ и $\|\Delta\mathbf{b}\|$ будем понимать *абсолютные погрешности векторов*.

Перемножая (1.5) и очевидное неравенство $\|\mathbf{b}\| \le \|\mathbf{A}\| \cdot \|\mathbf{x}\|$, с учетом того, что $\|\mathbf{b}\| \ne 0$ и $\|\mathbf{x}\| \ne 0$, получим

$$\frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \le \|\mathbf{A}^{-1}\| \cdot \|\mathbf{A}\| \frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|}.$$

В вычислительной линейной алгебра существует понятие — число обусловленности (от англ. *conditionality*) матрицы $\mathbf{A}$

$$\mathrm{cond}(\mathbf{A}) = \|\mathbf{A}^{-1}\| \cdot \|\mathbf{A}\|$$

Значение числа обусловленности зависит от используемой нормы.

Тогда можно записать

$$\delta(\mathbf{x}) \le \mathrm{cond}(\mathbf{A})\delta(\mathbf{b}), \qquad (1.6)$$

где $\delta(\mathbf{x})$ и $\delta(\mathbf{b})$ — *относительные погрешности решения и правой части,*

Матрицы с большим числом обусловленности называются *плохо обусловленными*. Обычно считается [8], что плохую обусловленность имеют СЛАУ с числом обусловленности 10^4 - 10^5 и более. При решении СЛАУ с плохо обусловленными матрицами возможно сильное накопление погрешностей.

Числа обусловленности обладают следующими свойствами:

1. $\mathrm{cond}(\mathbf{A}) \ge 1$.

Действительно, $\mathrm{cond}(\mathbf{A}) = \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\| \ge \|\mathbf{A}\mathbf{A}^{-1}\| = \|\mathbf{E}\| = 1$, где $\mathbf{E}$ – единичная матрица.

2. $\mathrm{cond}(\mathbf{A}) \ge \dfrac{|\lambda(\mathbf{A})|_{\max}}{|\lambda(\mathbf{A})|_{\min}}$ – большой разброс абсолютных величин собственных чисел матрицы характеризует ее плохую обусловленность.

Для доказательства рассмотрим спектральный радиус матрицы $\rho(\mathbf{A}) = |\lambda(\mathbf{A})|_{\max}$. Известно, что $\rho(\mathbf{A}) \le \|\mathbf{A}\|$, $\rho(\mathbf{A}^{-1}) = 1/|\lambda(\mathbf{A})|_{\min}$, тогда

$$\mathrm{cond}(\mathbf{A}) = \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\| \ge \rho(\mathbf{A}) \cdot \rho(\mathbf{A}^{-1}) = |\lambda(\mathbf{A})|_{\max} / |\lambda(\mathbf{A})|_{\min}.$$

3. $\mathrm{cond}(\mathbf{AB}) \le \mathrm{cond}(\mathbf{A}) \cdot \mathrm{cond}(\mathbf{B})$, что непосредственно следует из свойства норм матриц $\|\mathbf{AB}\| \le \|\mathbf{A}\| \, \|\mathbf{B}\|$.

4. $\mathrm{cond}(\alpha\mathbf{A}) = \mathrm{cond}(\mathbf{A})$, где $\alpha \ne 0$ произвольное число.

Действительно, $\mathrm{cond}(\alpha\mathbf{A}) = \|\alpha\mathbf{A}\| \cdot \|(\alpha\mathbf{A})^{-1}\| = \|\mathbf{A}\| \cdot \|\mathbf{A}^{-1}\| = \mathrm{cond}(\mathbf{A})$

Замечено также, что большой разброс величин элементов в пределах строки матрицы $\mathbf{A}$ свидетельствует о плохой обусловленности матрицы [16].

Пусть теперь *возмущены как правая часть, так и элементы матрицы*, тогда $\tilde{\mathbf{A}}\tilde{\mathbf{x}} = \tilde{\mathbf{b}}$, где $\Delta\mathbf{A} = \tilde{\mathbf{A}} - \mathbf{A}, \Delta\mathbf{x} = \tilde{\mathbf{x}} - \mathbf{x}, \Delta\mathbf{b} = \tilde{\mathbf{b}} - \mathbf{b}$. Пусть матрица $\mathbf{A}$ имеет обратную матрицу и пусть выполняется условие $\|\Delta\mathbf{A}\| < \left\|\mathbf{A}^{-1}\right\|^{-1}$, тогда матрица $\tilde{\mathbf{A}}$ имеет обратную матрицу и справедлива оценка [3, 9, 15]

$$\delta(\mathbf{x}) \le \frac{\mathrm{cond}(\mathbf{A})}{1 - \mathrm{cond}(\mathbf{A})\delta(\mathbf{A})}\left(\delta(\mathbf{A}) + \delta(\mathbf{b})\right) \qquad (1.7)$$

где $\delta(\mathbf{A}) = \dfrac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|}$, $\delta(\mathbf{b}) = \dfrac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|}$.

Казалось бы, вычислить число обусловленности матрицы и оценить погрешность решения СЛАУ несложно. Но как будет показано в разделе 2, вычисление элементов обратных матриц представляет собой задачу гораздо более сложную, чем решение СЛАУ, поэтому при решении СЛАУ элементы обратной матрицы $\mathbf{A}^{-1}$ не вычисляют. Однако, имея решение системы (1.2), полученное каким-то методом, например, методом Гаусса, можно получить приближенную апостериорную оценку числа обусловленности. Действительно, решение СЛАУ (1.2) любым методом формально можно записать в виде $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$, тогда $\left\|\mathbf{A}^{-1}\right\| \ge \|\mathbf{x}\|/\|\mathbf{b}\|$ и оценка нижней границы для числа обусловленности имеет вид

$$\mathrm{cond}(\mathbf{A}) \ge \frac{\|\mathbf{A}\|\|\mathbf{x}\|}{\|\mathbf{b}\|}. \qquad (1.8)$$

Известно [17], что оценка (1.8) за исключением очень редких случаев может быть принята за оценку числа обусловленности. Оценка (1.8) основана на решении СЛАУ, поэтому является оценкой обусловленности матрицы, возмущенной погрешностями решения СЛАУ. В той же книге [17] доказывается, что практически число обусловленности возмущенной матрицы мало отличается от числа обусловленности исходной матрицы. Известны [15, 17] более сложные и точные оценки числа обусловленности матрицы.

1.3.3. Требования, предъявляемые к численному методу

Одной и той же математической модели можно поставить в соответствие множество дискретных моделей и вычислительных алгоритмов, т. е. численных методов. При выборе численного метода необходимо учитывать две группы требований:

– дискретная модель должна быть *адекватна* математической модели;

– численный метод должен быть реализуемым на компьютере.

Для обеспечения адекватности численный метод должен обладать свойствами *корректности*, *хорошей обусловленности*, *сходимости, выполнения дискретных аналогов законов сохранения и качественно правильного поведения решения* [3, 10].

Численный метод называется *корректным*, если выполняются три условия:

1) метод позволяет получить решение за конечное число элементарных операций;

2) метод является устойчивым по входным данным;

3) метод обладает вычислительной устойчивостью.

Первое требование очевидно. Например, метод вычисления суммы членов бесконечного ряда является некорректным, если не задано условие окончания вычислений. *Устойчивость по входным данным* означает непрерывную зависимость решения от входных данных при условии отсутствия погрешностей вычислений. *Вычислительная устойчивость* означает устойчивость к погрешностям представления данных и погрешностям округления. Метод называется вычислительно устойчивым, если погрешность результата стремится к нулю при стремлении к нулю машинной погрешности.

Вычислительно устойчивый алгоритм называется *хорошо обусловленным*, если малые погрешности округления приводят к малым погрешностям результата. Следует учесть, что, решая хорошо обусловленную задачу плохо обусловленным методом, можно не получить решение.

Сходимость численного метода связана с его корректностью и, например, означает, что при уменьшении шага разбиения интервала интегрирования результат численного интегрирования стремится к точному значению.

Различные математические модели являются выражением физических законов сохранения, поэтому для дискретной модели *законы сохранения* также должны выполняться. Например, дифференциальные уравнения заменяются разностными с помощью конечно-разностной аппроксимации. Если для разностных схем не выполняются аналоги законов сохранения (разностные схемы не являются *консервативными*), то дискретная модель будет неправильно отражать поведение решения исходной задачи.

Качественно правильное поведение дискретной модели означает, что из-за дискретного характера поведения модели не теряются важные детали поведения реальной системы.

Реализуемость численного метода на компьютере ограничена объемом памяти и быстродействием компьютера. Вычислительный алгоритм должен быть предъявлять разумные требования к ресурсам компьютера. Например, математически корректный метод Крамера решения систем линейных алгебраических уравнений в известной классической реализации с использованием алгебраических дополнений абсолютно неприменим для решения реальных задач: если принять, что каждая арифметическая операция выполняется за 10^{-6} с, то для решения системы с 20 неизвестными методом Крамера потребуется более миллиона лет [18]. В то же время простейшим методом Гаусса эта система будет решена за доли секунды[3].

Говоря о реализуемости численных методов, следует отметить все возрастающую сложность решаемых задач и повышение требований к компьютерам, что приводит к всё более широкому применению *параллельных методов* вычислений. Рассмотрение параллельных методов вычислений выходит за рамки нашего курса.

1.4. Вопросы и задания для самопроверки

1. Перечислите основные этапы вычислительного эксперимента.
2. Что такое дискретная модель?
3. Что дает приведение задачи к безразмерному виду?
4. Перечислите основные источники погрешностей вычислительного эксперимента.
5. На компьютере вычисляется сумма 1000 первых членов бесконечного ряда. Какие виды погрешностей при этом возникают?
6. Какие цифры являются значащими в числе 0.0012345000?
7. Определить верные значащие цифры приближенного числа -3.1234567, если абсолютная погрешность равна $0.6 \cdot 10^{-5}$?
8. Определить верные значащие цифры приближенного числа 1234, если абсолютная погрешность равна 0.5?
9. Чему равна относительная погрешность представления числа с плавающей точкой, если разрядность порядка равна 11, а разрядность мантиссы — 52?
10. Чему равна погрешность округления чисел с плавающей точкой, если разрядность порядка равна 8, а разрядность мантиссы равна 23?
11. Что произойдет в MATLAB при делении числа на ноль?
12. Перечислите особенности стандарта двоичной арифметики IEEE 754.
13. Что дает применение субнормальных чисел?

[3] В последние годы предложен вариант метода Крамера с трудоемкостью, соизмеримой с методом Гаусса, и хорошо распараллеливаемый (Habgood K., Arel I. A condensation-based application of Cramer's rule for solving large-scale linear systems // Journal of Discrete Algorithms. — 2012. No 10. — P. 98–109).

14. Что такое обусловленность вычислительной задачи?
15. По каким признакам можно судить о плохой обусловленности матрицы системы линейных алгебраических уравнений?
16. Какие основные требования предъявляются к численному методу?
17. Что такое вычислительная устойчивость?

Библиографический список к разделу 1

1. Самарский А. А.Б., Михайлов А. П. Математическое моделирование: Идеи. Методы. Примеры. — М.: ФИЗМАТЛИТ, 2002. — 320 с.

2. Введение в математическое моделирование / В. Н. Ашихмин, М. Б. Гитман, И. Э. Келлер, О. Б. Наймарк, В. Ю. Столбов, П. В. Трусов, П. Г. Фрик; Под ред. П. В. Трусова. — М.: Логос, 2004. — 440 с.

3. Самарский А. А., Гулин А. В. Численные методы математической физики. — М.: "Научный мир", 2003. — 316 с.

4. Гухман А. А. Введение в теорию подобия. — М.: Издательство ЛКИ, 2010. — 296 с.

5. Галанин М. П., Савенков Е. Б. Методы численного анализа математических моделей. — М.: Изд-во МГТУ им. Н. Э. Баумана, 2010. — 591 с.

6. Olver P. J., Shakiban C. Applied Linear Algebra. — Cham: Springer, 2018. — 679 p.

7. Линейная алгебра в действии. [Электронный ресурс].
URL: https://ichi.pro/ru/linejnaa-algebra-v-dejstvii-198396710787594 (дата обращения: 17.02.2021).

8. Калиткин Н. Н. Численные методы. — СПб.: БХВ-Петербург, 2014. — 592 с.

9. Михлин С. Г. Некоторые вопросы теории погрешностей. — Л.: Изд-во ЛГУ, 1988. — 334 с.

10. Амосов А. А., Дубинский Ю. А., Кончёнова Н. В. Вычислительные методы. — М.: Издательский дом МЭИ, 2008. — 672 с.

11. Закляков П. В. Информатика. — М.: ДМК-Пресс, 2021. — 750 с.

12. Алефельд Г., Херцбергер Ю. Введение в интервальные вычисления. — М.: Мир, 1987. — 360 с.

13. Шарый С. П. Конечномерный интервальный анализ. — Новосибирск: Изд-во XYZ, 2020. — 646 с.

14. Петров Ю. П., Петров Л. Ю. Неожиданное в математике и его связь с авариями и катастрофами. — СПб.: БХВ-Петербург, 2005. — 217 с.

15. Уоткинс Д. Основы матричных вычислений. — М.: БИНОМ. Лаборатория знаний, 2006. — 664 с.

16. Жидков Н. П. Несколько замечаний по поводу обусловленности систем линейных алгебраических уравнений //Журнал вычислительной математики и математической физики, 1963, т.3. — №5. — С. 803–811.

17. Каханер Д., Моулер К., Нэш С. Численные методы и программное обеспечение. — М.: Мир, 2001. — 575 с.

18. Ортега Дж., Пул У. Введение в численные методы решения дифференциальных уравнений. — М.: Наука, 1986. — 288 с.

ГЛАВА 2. ПРЯМЫЕ МЕТОДЫ РЕШЕНИЯ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ

2.1. Системы линейных алгебраических уравнений. Матрицы и их свойства

Рассмотрим самые необходимые сведения из теории матриц и *систем линейных алгебраических уравнений* (СЛАУ), которые нам потребуются в дальнейшем. Для более подробного ознакомления можно обратиться к литературе [1–14]. Несколько электронных курсов по линейной алгебре, например, "Введение в линейную алгебру" представлена на сайте Интернет-университета информационных технологий (www.intut.ru).

Мы будем рассматривать решение СЛАУ вида

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n = b_2, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots + a_{nn}x_n = b_n. \end{cases} \qquad (2.1)$$

или в матричной форме записи

$$\mathbf{Ax} = \mathbf{b} \qquad (2.2)$$

где $\mathbf{A}$ — матрица коэффициентов, $\mathbf{x}$ — вектор неизвестных, $\mathbf{b}$ — вектор правой части (свободных членов).

Пусть матрица $\mathbf{A}$ системы (2.1) является невырожденной. Мы будем рассматривать лишь *действительные матрицы*, т. е. матрицы, элементы a_{ij} которых являются действительными числами. Матрица, состоящая из одной строки, называется *вектором-строкой*, матрица, состоящая из одного столбца — *вектором-столбцом*. Квадратная матрица вида

$$A = \begin{bmatrix} a_{11} & 0 & 0 & \ldots & 0 \\ 0 & a_{22} & 0 & \ldots & 0 \\ 0 & 0 & a_{33} & \ldots & 0 \\ \ldots\ldots\ldots\ldots\ldots & & & & \\ 0 & 0 & 0 & \ldots & a_{nn} \end{bmatrix} \qquad (2.3)$$

называется *диагональной матрицей*. Если в (2.3) все $a_{ii} = 1 \quad (i = 1, 2, \ldots, n)$, то матрица называется *единичной* и обозначается $\mathbf{E}$.

Если в матрице переставить строки и столбцы, то получится *транспонированная* матрица $\mathbf{A}^{\mathrm{T}}$. Матрица называется *симметричной*, если $\mathbf{A}^{\mathrm{T}} = \mathbf{A}$, т.е. $a_{ij} = a_{ji}$.

Матрицы $\mathbf{A}$ и $\mathbf{B}$ называются *равными* $\mathbf{A} = \mathbf{B}$ или $\mathbf{B} = \mathbf{A}$, если они имеют одинаковое число строк и столбцов и соответствующие их элементы равны, т. е. $a_{ij} = b_{ij}$. Элементы суммы матриц $\mathbf{C} = \mathbf{A} + \mathbf{B}$ с одинаковыми размерами $m \times n$ вычисляются по правилам $c_{ij} = a_{ij} + b_{ij}$, $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$. Сумма матриц обладает следующими свойствами:

$$\mathbf{A} + (\mathbf{B} + \mathbf{C}) = (\mathbf{A} + \mathbf{B}) + \mathbf{C}, \quad \mathbf{A} + \mathbf{B} = \mathbf{B} + \mathbf{A}, \quad \mathbf{A} + \mathbf{0} = \mathbf{A}$$

Произведение матрицы $\mathbf{A}$ на число α определяется следующим образом: $\mathbf{C} = \alpha\mathbf{A} = \mathbf{A}\alpha$, $c_{ij} = \alpha a_{ij}$, $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$ и обладает свойствами:

$$1 \cdot \mathbf{A} = \mathbf{A}, \quad 0 \cdot \mathbf{A} = \mathbf{0}, \quad \alpha(\beta\mathbf{A}) = (\alpha\beta)\mathbf{A}, \quad (\alpha + \beta)\mathbf{A} = \alpha\mathbf{A} + \beta\mathbf{A}, \quad \alpha(\mathbf{A} + \mathbf{B}) = \alpha\mathbf{A} + \alpha\mathbf{B}$$

Произведение $\mathbf{AB}$ — произведение матрицы $\mathbf{A}$ на матрицу $\mathbf{B}$ — определено только в том случае, когда число столбцов матрицы $\mathbf{A}$ равно числу строк матрицы $\mathbf{B}$. Пусть $\mathbf{A}$ имеет размеры $m \times n$, а $\mathbf{B}$ — $n \times q$, тогда $\mathbf{C} = \mathbf{AB}$ имеет размеры $m \times q$ и элементы матрицы $\mathbf{C}$ определяются выражением

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}, \; i = 1, 2, ..., m, \quad j = 1, 2, ..., q$$

Умножение матриц не подчиняется перестановочному закону, т. е. в общем виде $\mathbf{AB} \neq \mathbf{BA}$. Если $\mathbf{AB} = \mathbf{BA}$, то матрицы $\mathbf{A}$ и $\mathbf{B}$ называются *перестановочными*. Если $\mathbf{AA}^{\mathrm{T}} = \mathbf{A}^{\mathrm{T}}\mathbf{A}$, то матрица $\mathbf{A}$ называется *нормальной*. Произведение матриц обладает свойствами:

$$\mathbf{A}(\mathbf{BC}) = (\mathbf{AB})\mathbf{C}, \quad \alpha(\mathbf{AB}) = (\alpha\mathbf{A})\mathbf{B}, \quad (\mathbf{A} + \mathbf{B})\mathbf{C} = \mathbf{AC} + \mathbf{BC},$$

$$\mathbf{C}(\mathbf{A} + \mathbf{B}) = \mathbf{CA} + \mathbf{CB}, \quad \mathbf{AE} = \mathbf{EA} = \mathbf{A}, \quad (\mathbf{AB})^{\mathrm{T}} = \mathbf{B}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}.$$

Квадратная матрица называется *неособенной* (*невырожденной*), если ее определитель отличен от нуля: $\det \mathbf{A} \neq 0$. Для неособенной матрицы существует *обратная матрица* $\mathbf{A}^{-1}$, причем $\mathbf{AA}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{E}$, если $\mathbf{C} = \mathbf{AB}$, то $\mathbf{C}^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$. Формально решение *любым методом* СЛАУ (2.2) можно трактовать как умножение (2.2) слева на $\mathbf{A}^{-1}$

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} \qquad (2.4)$$

Запись (2.4) удобна для математических исследований, но *при решении СЛАУ практически крайне редко находят* $\mathbf{A}^{-1}$.

Если обратная матрица равна транспонированной $\mathbf{A}^{-1} = \mathbf{A}^{\mathrm{T}}$, т. е. $\mathbf{A}^{\mathrm{T}}\mathbf{A} = \mathbf{E}$, то матрица $\mathbf{A}$ называется *ортогональной*.

Собственным вектором матрицы $\mathbf{A}$, отвечающим собственному значению λ, называется ненулевой вектор $\mathbf{v}$, удовлетворяющий уравнению

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}.$$

Матрица $\mathbf{A}$ порядка n имеет n собственных чисел (возможно кратных).

Скалярным произведением вещественных векторов $\mathbf{x} = (x_1, x_2, ..., x_n)^{\mathrm{T}}$ и $\mathbf{y} = (y_1, y_2, ..., y_n)^{\mathrm{T}}$ называется число

$$(\mathbf{x}, \mathbf{y}) = \mathbf{x}^{\mathrm{T}}\mathbf{y} = \mathbf{y}^{\mathrm{T}}\mathbf{x} = \sum_{i=1}^{n} x_i y_i = x_1 y_1 + x_2 y_2 + \ldots + x_n y_n.$$

Скалярное произведение удовлетворяет следующим аксиомам:

$$(\mathbf{x}, \mathbf{y}) = (\mathbf{y}, \mathbf{x}),$$
$$(c\mathbf{x}, \mathbf{y}) = c(\mathbf{x}, \mathbf{y}),$$
$$(\mathbf{x} + \mathbf{y}, \mathbf{z}) = (\mathbf{x}, \mathbf{z}) + (\mathbf{y}, \mathbf{z}),$$
$$(\mathbf{x}, \mathbf{x}) > 0, \text{ если } \mathbf{x} \neq 0, \quad (\mathbf{x}, \mathbf{x}) = 0, \text{ если } \mathbf{x} = 0.$$

Симметричная матрица $\mathbf{A}$ называется положительно *определенной*, если для любого вектора $\mathbf{x} \neq 0$ положительно скалярное произведение $(\mathbf{A}\mathbf{x}, \mathbf{x}) > 0$. Известно, что все собственные значения положительно определенной матрицы положительны. Матрицы многих СЛАУ положительно определены. Например, положительно определена матрица системы, описывающей линейную цепь постоянного тока, положительно определены матрицы систем, аппроксимирующих во многих методах дифференциальные уравнения в частных производных.

Матрица $\mathbf{A}$ называется *положительно полуопределенной*, если $(\mathbf{A}\mathbf{x}, \mathbf{x}) \geq 0$, при $\mathbf{x} \neq 0$ все собственные значения такой матрицы неотрицательны.

Квадратная матрица называется нижней (верхней) *треугольной*, если элементы, стоящие выше (ниже) главной диагонали, равны нулю. Определитель треугольной матрицы равен произведению диагональных элементов $\det \mathbf{A} = a_{11}a_{22}a_{33}\ldots a_{nn}$.

Матрица называется *ленточной*, если все ее ненулевые элементы заключены внутри ленты, образованной диагоналями, параллельными главной диагонали [15]. То есть $a_{ij} = 0$, если $|i - j| > \beta$, здесь β — полуширина, а $2\beta + 1$ — ширина ленты матрицы. Ленточные матрицы получаются, например, при конечно-разностной аппроксимации дифференциальных уравнений в частных производных. Ленточные матрицы являются частным случаем *разреженных матриц*, т. е. матриц, число ненулевых элементов которых много меньше общего числа элементов. В противном случае матрица называется плотной

(заполненной). Если плотные матрицы обычно хранятся в виде двумерных массивов, то для хранения разреженных матриц применяют сложные методы хранения, исключающие хранение ненулевых элементов [15–17]

Элементами рассмотренных матриц были числа. В общем виде элементами матрицы могут быть *подматрицы***,** или *блоки* (*блочные матрицы*). Многие алгоритмы, сконструированные для числовых матриц, формально переносятся и на блочные матрицы.

Чтобы как-то оценивать "величину" векторов, вводится понятие *нормы вектора*. Рассмотрим множество $\mathbb{R}_n$ всех *n*-мерных вещественных векторов $\mathbf{x} = [x_1, x_2, ..., x_n]^{\mathrm{T}}$. Это множество образует линейное пространство[4]. Норма вектора определяется аксиоматически. Норма вектора — это действительное число, обозначаемое $\|\mathbf{x}\|$, удовлетворяющего условиям:

1) $\|\mathbf{x}\| > 0$ при $\mathbf{x} \neq \mathbf{0}$, $\|\mathbf{x}\| = 0$ при $\mathbf{x} = 0$,

2) $\|c\mathbf{x}\| = |c|\|\mathbf{x}\|$ при любом скалярном *c* (аксиома однородности),

3) $\|\mathbf{x}+\mathbf{y}\| \le \|\mathbf{x}\| + \|\mathbf{y}\|$ (неравенство треугольника).

(2.5)

Существует множество различных способов введения нормы вектора, удовлетворяющей условиям (2.5). Следуя [4, 18], введем нормы

$\|\mathbf{x}\|_1 = |x_1| + |x_2| + \ldots + |x_n|$ — первая (октаэдрическая) норма;

$\|\mathbf{x}\|_2 = \sqrt{|x_1|^2 + |x_2|^2 + \ldots + |x_n|^2}$ — вторая (евклидова или сферическая) норма;

$\|\mathbf{x}\|_\infty = \max_i |x_i|$ — кубическая норма.

Эти нормы являются частными случаями более общей нормы [18]

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}.$$

Первая норма получается при $p = 1$, вторая — при $p = 2$, кубическая — при $p \to \infty$. Для норм векторов справедливы неравенства

$$\|\mathbf{x}\|_\infty \le \|\mathbf{x}\|_2 \le \|\mathbf{x}\|_1 \le n\|\mathbf{x}\|_\infty$$

Используются и другие обозначения приведенных норм (см., например, [11]).

Рассмотрим множество квадратных $n \times n$-матриц с вещественными элементами, образующее линейное пространство $\mathbb{R}_{n\times n}$. *Нормой квадратной матрицы* **A** называется действительное число, удовлетворяющее условиям

[4] Подробнее можно ознакомиться, например, в [11].

1) $\|\mathbf{A}\| > 0$ при $\mathbf{A} \neq \mathbf{0}$, $\|\mathbf{A}\| = 0$ при $\mathbf{A} = \mathbf{0}$,

2) $\|c\mathbf{A}\| = |c|\|\mathbf{A}\|$,

3) $\|\mathbf{A}+\mathbf{B}\| \le \|\mathbf{A}\| + \|\mathbf{B}\|$,

4) $\|\mathbf{AB}\| \le \|\mathbf{A}\|\,\|\mathbf{B}\|$,

где $\mathbf{B}$ — произвольная $n \times n$-матрица.

В анализе обычно участвуют как нормы векторов, так и матриц, поэтому используют *нормы матриц, согласованные с нормами векторов*, что означает выполнение неравенства $\|\mathbf{Ax}\| \le \|\mathbf{A}\|\|\mathbf{x}\|$. Еще более сильным требованием к норме матрицы является *условие подчиненности*. Норма матрицы $\mathbf{A}$ называется подчиненной норме вектора $\mathbf{x}$, если

$$\|\mathbf{A}\| = \max_{\|\mathbf{x}\|=1} \|\mathbf{Ax}\|.$$

То есть при заданной векторной норме за подчиненную матричную норму принимается максимум нормы вектора $\mathbf{Ax}$, где $\mathbf{x}$ — множество векторов, норма которых равна единице. Среди всех норм, согласованных с заданными векторными нормами, подчиненная норма является минимальной. Норма матрицы $\|\mathbf{A}\|_1 = \max_k \sum_{i=1}^{n} |a_{ik}|$ подчинена первой норме вектора. Норма $\|\mathbf{A}\|_2 = \|\mathbf{A}\| = \sqrt{\lambda_1}$, (где λ_1 — наибольшее собственное число матрицы $\mathbf{A}^{\mathrm{T}}\mathbf{A}$) подчинена евклидовой норме вектора $\|\mathbf{x}\|_2^2$. Норма $\|\mathbf{A}\|$ называется *спектральной нормой*. Норма $\|\mathbf{A}\|_\infty = \max_i \sum_{k=1}^{n} |a_{ik}|$ подчинена кубической норме вектора. Используется также *евклидова норма матрицы* $\|\mathbf{A}\|_E = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} |a_{ij}|^2}$.

Введенные определения норм распространяются на векторы и матрицы с комплексными коэффициентами (поэтому в определении второй нормы использован модуль), а также на прямоугольные матрицы.

2.2. Метод Гаусса

Метод Гаусса (*Gauss Carl Friderich*) является одним из самых распространенных прямых методов решения СЛАУ, т. е. методов, в которых решение системы находится в результате конечного числа арифметических действий. Пусть решается система (2.1) порядка n, имеющая единственное решение. Метод состоит из прямого и обратного хода. Прямой ход состоит из

n шагов и заключается в последовательном исключении неизвестных из системы. В результате система (2.1) преобразуется в СЛАУ с верхней треугольной матрицей. На этапе обратного хода решается система с треугольной матрицей, и находятся неизвестные.

Рассмотрим простейший вариант метода Гаусса, называемый *схемой единственного деления.*

Рассмотрим прямой ход. На *первом шаге* производится исключение неизвестного x_1 из второго, третьего и т. д., n-го уравнений. Пусть $a_{11} \neq 0$. Этот элемент называется *ведущим* (*главным*) *элементом* первого шага. Разделив первое уравнение на a_{11}, получим

$$x_1 + \frac{a_{12}}{a_{11}} x_2 + \frac{a_{13}}{a_{11}} x_3 + \ldots + \frac{a_{1n}}{a_{11}} x_n = \frac{b_1}{a_{11}}. \qquad (2.6)$$

Умножая (2.6) на a_{21} и вычитая результат из второго уравнения, получим преобразованное второе уравнение, из которого исключено x_1.

$$\left(a_{22} - a_{21}\frac{a_{12}}{a_{11}}\right)x_2 + \left(a_{23} - a_{21}\frac{a_{13}}{a_{11}}\right)x_3 + \ldots + \left(a_{2n} - a_{21}\frac{a_{1n}}{a_{11}}\right)x_n = b_2 - a_{21}\frac{b_1}{a_{11}}. \qquad (2.7)$$

Далее исключается x_1 из третьего, четвертого, и т. д., из n-го уравнения. Обозначим (2.6)

$$x_1 + a_{12}^{(1)} x_2 + a_{13}^{(1)} x_3 + \ldots + a_{1n}^{(1)} x_n = b_1^{(1)},$$

где
$$a_{sk}^{(s)} = \frac{a_{sk}^{(s-1)}}{a_{ss}^{(s-1)}}, \qquad b_s^{(s)} = \frac{b_s^{(s-1)}}{a_{ss}^{(s-1)}}. \qquad (2.8)$$

Для (2.6) $s = 1$ — номер шага, $a_{sk}^{(0)} = a_{sk}$, $b_s^{(0)} = b_s$, $k = s, s+1, ..., n$.

Обозначим (2.7):

$$a_{22}^{(1)} x_2 + a_{23}^{(1)} x_3 + \ldots + a_{2n}^{(1)} x_n = b_2^{(1)}, \qquad (2.9)$$

где
$$a_{ik}^{(s)} = a_{ik}^{(s-1)} - a_{sk}^{(s)} a_{is}^{(s-1)}, \qquad (2.10)$$

$b_i^{(s)} = b_i^{(s-1)} - b_s^{(s)} a_{is}^{(s-1)}$, $i = s+1, s+2, \ldots, n;$ $k = s, s+1, \ldots, n$, $a_{sk}^{(s)}$ и $b_s^{(s)}$ определяются из (2.8).

На *втором шаге* $(s = 2)$ прямого хода производится исключение неизвестного x_2 из третьего и т. д., n-го уравнений, преобразованных на первом шаге. Если $a_{22}^{(1)} \neq 0$, делим на $a_{22}^{(1)}$ второе уравнение преобразованной на первом шаге системы и исключаем x_2 из третьего и т. д. уравнений. Преобразования проводятся по формулам (2.8)–(2.10) при $s = 2$. После n шагов преобразований получаем СЛАУ с верхней треугольной матрицей (шаг n состоит из преобразования последнего уравнения по (2.8))

$$
\begin{aligned}
x_1 + a_{12}^{(1)} x_2 + a_{13}^{(1)} x_3 + \ldots + a_{1n}^{(1)} x_n &= b_1^{(1)}, \\
x_2 + a_{23}^{(2)} x_3 + \ldots + a_{2n}^{(2)} x_n &= b_2^{(2)}, \\
\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
x_n &= b_n^{(n)}
\end{aligned}
\qquad (2.11)
$$

Далее в процессе *обратного хода* метода Гаусса решается система (2.11) и определяются неизвестные

$$
x_n = b_n^{(n)}, \quad x_i = b_i^{(i)} - \sum_{k=i+1}^{n} a_{ik}^{(i)} x_k, \quad i = n-1, n-2, \ldots, 1.
$$

При реализации прямого хода на компьютере нет необходимости действовать с переменными $x_1, x_2, \ldots, x_n$. Прямой ход сводится к преобразованию матрицы $\mathbf{A}$ и вектора $\mathbf{b}$ по формулам (2.8)–(2.10). Для этого к матрице $\mathbf{A}$ справа присоединяется столбец $\mathbf{b}$ и преобразуется эта расширенная матрица.

Широко используется вариант метода Гаусса (см. например, [18]), когда на *s*-ом шаге *s*-е уравнение не преобразуется по формуле (2.8). В результате в преобразованной системе, в отличие от (2.11) диагональные элементы не равны единице.

Известна модификация метода Гаусса, называемая *методом Гаусса-Жордана*[5]. Если в методе Гаусса на s-ом шаге прямого хода преобразуются $s+1,\ s+2, \ldots,\ n$-е уравнения, то в методе Гаусса-Жордана на s-ом шаге прямого хода по тем же формулам (2.8), (2.10) преобразуются 1, 2,..., $s-1,\ s+1,\ s+2, \ldots,\ n$-е уравнения. После завершения прямого хода метода Гаусса-Жордана матрица $\mathbf{A}$ приводится к диагональному виду.

В методе Гаусса необходимо, чтобы все *ведущие элементы* $a_{ss}^{(s-1)}$, $s = 1, 2, \ldots, n$ (2.8) были отличны от нуля. Кроме того, близость ведущих элементов к нулю может привести к потере точности вычислений, т. к. будет происходить деление на очень малые числа. Этот недостаток устраняется в *методе Гаусса с выбором главного элемента.* Чаще всего используется *выбор главного элемента по столбцу* (*схема частичного выбора*). При этом при исключении x_s просматривается *s*-й столбец для *еще не преобразованных уравнений* и находится максимальный по абсолютной величине коэффициент

---

[5] Метод назван в честь К. Ф. Гаусса и немецкого геодезиста и математика Вильгельма Йо́рдана (*Wilhelm Jordan* — https://ru.wikipedia.org/wiki/Йордан,_Вильгельм_(геодезист)). Правильное название: метод Гаусса-Йордана. Транскрипция фамилии Йордан как "Жордан" является ошибочной, но она общепринята и встречается в большинстве русскоязычных источников. В истории математики был французский математик Мари́ Энмо́н Ками́ль Жорда́н (*Marie Ennemond Camille Jordan* — https://ru.wikipedia.org/wiki/Жордан,_Мари_Энмон_Камиль), в честь которого названа, в частности, жорданова форма матрицы.

a_{ks}, затем уравнения *s* и *k* меняются местами. После этого по обычной схеме исключается x_s. Если все элементы столбца для непреобразованных уравнений равны нулю, то *матрица* $\mathbf{A}$ *вырождена*. Для плохо обусловленных систем из-за погрешности округлений ведущие элементы вырожденной матрицы могут отличаться от нуля, что не позволяет обнаружить вырожденность матрицы. Лучше сравнивать указанные элементы с малым числом $\varepsilon > 0$. Величина ε зависит от погрешностей вычислений.

В методе Гаусса с *выбором главного элемента по всей матрице* (*схема полного выбора*) на *s*-ом шаге среди коэффициентов $a_{ij}^{(s-1)}$ в уравнениях s, $s+1$,..., n выбирают максимальный по модулю коэффициент $a_{kl}^{(s-1)}$. Затем s-е уравнение и *k*-е уравнение меняются местами и исключается x_l. На этапе обратного хода неизвестные исключаются в порядке, обратном выбору исключаемых неизвестных на этапе прямого хода. Метод Гаусса с выбором главного элемента по всей матрице достаточно сложен и требует много операций сравнения элементов матрицы, поэтому на практике наибольшее распространение получил метод Гаусса с выбором главного элемента по столбцу.

Рассмотрим формальное описание перестановки строк и столбцов матриц. Перестановки описываются с помощью матриц перестановок [4]. Матрица перестановок — это квадратная матрица, у которой в каждой строке и каждом столбце только один элемент отличен от нуля и равен единице. Матрица перестановок $\mathbf{P}$ является ортогональной, то есть $\mathbf{P}^{\mathrm{T}}\mathbf{P} = \mathbf{P}\mathbf{P}^{\mathrm{T}} = \mathbf{E}$, $\mathbf{P}^{\mathrm{T}} = \mathbf{P}^{-1}$. Произведение матриц перестановок одного порядка является матрицей перестановок. Рассмотрим построение матрицы $\mathbf{P}_R$ перестановок строк матрицы $\mathbf{A}$. Сначала полагаем $\mathbf{P}_R = \mathbf{E}$, где $\mathbf{E}$ — единичная матрица. При каждой перестановке строк в преобразуемой матрице $\mathbf{A}$ переставляются строки с теми же номерами в текущем значении матрицы перестановок $\mathbf{P}_R$. Так как переставляются непреобразованные строки матрицы $\mathbf{A}$, то после перестановок получим тот же результат, если бы переставили строки в исходной матрице $\mathbf{A}$. Перестановка строк не меняет порядок неизвестных. Рассмотрим решение системы (2.2). Известно [4], что при умножении матрицы $\mathbf{A}$ слева на матрицу перестановок $\mathbf{P}_R$ переставляются строки матрицы $\mathbf{A}$. При перестановке строк также переставляются компоненты вектора $\mathbf{b}$. Поэтому метод Гаусса с выбором главного элемента по столбцу формально описывается уравнением

$$\mathbf{P}_R\mathbf{A}\mathbf{x}_1 = \mathbf{P}_R\mathbf{b},$$

откуда видно, что $\mathbf{x}_1 = \mathbf{x}$ и нет смысла хранить матрицу $\mathbf{P}_R$.

Матрица перестановок столбцов $\mathbf{P}_C$ строится аналогично: при каждой перестановке столбцов преобразуемой матрицы переставляются столбцы в матрице $\mathbf{P}_C$. Так как переставляются столбцы непреобразованной части матрицы, то результат эквивалентен перестановке столбцов в исходной матрице $\mathbf{A}$. Перестановка столбцов меняет порядок исключения неизвестных, но не изменяет порядок уравнений. Известно [4], что при умножении матрицы $\mathbf{A}$ справа на матрицу перестановок $\mathbf{P}_C$ переставляются столбцы матрицы $\mathbf{A}$. При перестановке столбцов компоненты вектора $\mathbf{b}$ не переставляются. Метод Гаусса с выбором главного элемента по всей матрице описывается уравнением

$$\mathbf{P}_R\mathbf{A}\mathbf{P}_C\mathbf{x}_1 = \mathbf{P}_R\mathbf{b},$$

откуда

$$\mathbf{P}_C\mathbf{x}_1 = \mathbf{A}^{-1}\mathbf{b} = \mathbf{x}$$

и решение исходной системы равно

$$\mathbf{x} = \mathbf{P}_C\mathbf{x}_1. \qquad (2.12)$$

Причем выражение (2.12) справедливо для всех методов решения систем линейных алгебраических уравнений с перестановкой столбцов исходной матрицы в процессе решения. Действительно, пусть в результате перестановок исходная матрица $\mathbf{A}$ преобразована в $\mathbf{A}_1 = \mathbf{P}_R\mathbf{A}\mathbf{P}_C$. Любым методом, например, рассматриваемыми далее методами разложения матрицы решается система $\mathbf{A}_1\mathbf{x}_1 = \mathbf{P}_R\mathbf{b}$. Решение исходной системы находится по выражению (2.12).

Рекомендуется перед решением СЛАУ *уравновесить ее матрицу* умножением строк и столбцов на подходящие масштабные множители. Масштабные множители подбирают так, чтобы элементы матрицы были величинами одного порядка. В случае *масштабирования столбцов* после получения решения необходимо произвести его демасштабирование. На практике часто делят каждое уравнение на наибольший по модулю коэффициент.

Отметим, что для матриц со свойством *диагонального преобладания*

$$|a_{ii}| > \sum_{j=1,\, j\neq i}^{n} |a_{ij}|, \quad i = 1, 2, \ldots, n$$

метод исключения Гаусса без выбора главного элемента всегда осуществим. Матрицами с диагональным преобладанием является, например, матрицы СЛАУ при расчете электрических цепей, при конечно-разностной аппроксимации дифференциальных уравнений в частных производных.

2.3 Метод прогонки

Метод прогонки — это метод Гаусса, примененный к СЛАУ с ленточными трех- и пятидиагональными матрицами, возникающими, в частности, при разностной аппроксимации краевых задач для обыкновенных дифференциальных уравнений и дифференциальных уравнений в частных производных [19]. Рассмотрим простейший вариант *правой прогонки для СЛАУ с трехдиагональной матрицей* [18]:

$$
\begin{aligned}
&b_1x_1 + c_1x_2 &&= d_1,\\
&a_2x_1 + b_2x_2 + c_2x_3 &&= d_2,\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots &&\\
&\qquad a_ix_{i-1} + b_ix_i + c_ix_{i+1} &&= d_i,\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots &&\\
&\qquad a_{n-1}x_{n-2} + b_{n-1}x_{n-1} + c_{n-1}x_n &&= d_{n-1},\\
&\qquad\qquad a_nx_{n-1} + b_nx_n &&= d_n.
\end{aligned}
\tag{2.13}
$$

Прямой ход прогонки состоит в вычислении *прогоночных коэффициентов*, а обратный ход заключается в вычислении значений неизвестных. Преобразуем первое уравнение системы (2.13):

$$x_1 = \alpha_1x_2 + \beta_1, \tag{2.14}$$

$$\alpha_1 = -c_1/b_1,\ \ \beta_1 = d_1/b_1. \tag{2.15}$$

Подставим (2.14) во второе уравнение системы (2.13) и после преобразования получим

$$x_2 = \alpha_2x_3 + \beta_2, \tag{2.16}$$

$$\alpha_2 = -\frac{c_2}{b_2 + a_2\alpha_1},\quad \beta_2 = \frac{d_2 - a_2\beta_1}{b_2 + a_2\alpha_1}. \tag{2.17}$$

Выражения (2.16) и (2.17) подставим в третье уравнение и т. д. На i-ом шаге получим

$$
\begin{aligned}
&x_i = \alpha_ix_{i+1} + \beta_i,\\
&\alpha_i = -\frac{c_i}{b_i + a_i\alpha_{i-1}}\ ,\quad \beta_i = \frac{d_i - a_i\beta_{i-1}}{b_i + a_i\alpha_{i-1}}\ .
\end{aligned}
\tag{2.18}
$$

На n-ом шаге имеем

$$x_n = \beta_n = \frac{d_n - a_n\beta_{n-1}}{b_n + a_n\alpha_{n-1}},\quad \alpha_n = 0. \tag{2.19}$$

Таким образом, прогоночные коэффициенты при $i = 1$ вычисляются по (2.15), при $2 \le i \le n-1$ по (2.18), а при $i = n$ по (2.19).

В процессе обратного *хода* прогонки вычисляется x_i по формулам

$$x_n = \beta_n \ ,$$
$$x_i = \alpha_i x_{i+1} + \beta_i \ , \quad i = n-1, n-2, \ldots, 1.$$

Если матрица системы (2.13) обладает свойством диагонального преобладания, то метод прогонки является корректным и устойчивым.

Метод прогонки требует гораздо меньше арифметических операций, чем метод Гаусса ($\approx 8n$ вместо $\approx (2/3)n^3$ [18]). Практически часто приходится решать СЛАУ с одной и той же матрицей и разными правыми частями. В этом случае коэффициенты α_i могут быть рассчитаны один раз, а для каждой задачи необходимо определять только коэффициенты β_i и решение x_i.

Известны различные варианты метода прогонки [19], например, *матричная прогонка*, когда в качестве коэффициентов системы (2.13) выступают матрицы. Матричная прогонка используется при численном решении задач математической физики, описываемых дифференциальными уравнениями в частных производных.

2.4. Метод LU-разложения

Известна теорема о *LU*-разложении [18, 20]: если все угловые миноры матрицы $\mathbf{A}$ отличны от нуля, то матрицу $\mathbf{A}$ можно представить в виде

$$\mathbf{A} = \mathbf{LU}, \qquad (2.20)$$

где $\mathbf{L}$ — нижняя треугольная матрица, $\mathbf{U}$ — верхняя треугольная матрица. Если элементы диагонали одной из матриц $\mathbf{L}$ или $\mathbf{U}$ ненулевые, то такое разложение единственно.

Часто строят разложение с матрицей $\mathbf{L}$, имеющей ненулевую диагональ и матрицей $\mathbf{U}$, имеющей единичную диагональ. Мы рассмотрим такой вариант разложения. Во многих реализациях, например, в [18] матрица $\mathbf{L}$ имеет единичную диагональ, а $\mathbf{U}$ — ненулевую. Названия матриц являются начальными буквами английских слов *Lowe* — "нижний" и *Upper* — "верхний". Напомним, что угловые миноры это:

$$\Delta_1 = a_{11} \ , \ \Delta_2 = \det\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \ldots, \quad \Delta_n = \det(\mathbf{A}).$$

Как мы увидим, практически никогда не вычисляют миноры, а возможную вырожденность матрицы определяют в процессе разложения (2.20).

Пусть решается система (2.2)

$$\mathbf{Ax} = \mathbf{b}.$$

Представив матрицу $\mathbf{A}$ в виде разложения (2.20), получим

$$\mathbf{LUx} = \mathbf{b}.$$

Обозначив

$$\mathbf{Ux} = \mathbf{y}, \tag{2.21}$$

получим

$$\mathbf{Ly} = \mathbf{b} \tag{2.22}$$

Таким образом, вместо решения СЛАУ (2.2) необходимо последовательно решить систему (2.22) с нижней треугольной матрицей и систему (2.21) с верхней треугольной матрицей. Метод *LU*-разложения удобно использовать, когда многократно решаются СЛАУ с постоянной матрицей и различными правыми частями.

Формулы для элементов l_{ij} и u_{ij} можно получить, приравнивая соответствующие элементы в матричном равенстве

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 & \dots & 0 \\ l_{21} & l_{22} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & 0 \\ l_{n1} & l_{n2} & l_{n3} & \dots & l_{nn} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & u_{13} & \dots & u_{1n} \\ 0 & 1 & u_{23} & \dots & u_{2n} \\ \dots & \cdots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}.$$

Например, $a_{11} = l_{11}$, $a_{21} = l_{21}$, …, $a_{n1} = l_{n1}$ — это элементы первого столбца матрицы $\mathbf{L}$. $a_{12} = l_{11}u_{12}$, откуда $u_{12} = \dfrac{a_{12}}{l_{11}}$; $a_{13} = l_{11}u_{13}$, откуда $u_{13} = \dfrac{a_{13}}{l_{11}}$ и т. д. — так могут быть вычислены элементы первой строки матрицы $\mathbf{U}$. Аналогично находим $a_{22} = l_{21}u_{12} + l_{22}$, $a_{32} = l_{31}u_{12} + l_{32}$ и т. д., откуда получаем выражения для элементов второй столбца матрицы $\mathbf{L}$:

$$l_{22} = a_{22} - l_{21}u_{12}, \quad l_{32} = a_{32} - l_{31}u_{12} \quad \text{и т. д.}$$

Из $a_{23} = l_{21}u_{13} + l_{22}u_{23}$ получаем выражение для элемента второй строки матрицы $\mathbf{U}$: $u_{23} = \dfrac{1}{l_{22}}\left(a_{23} - l_{21}u_{13}\right)$.

В общем виде вычисление элементов матриц $\mathbf{L}$ и $\mathbf{U}$ состоит из *n* шагов. На *r*-ом шаге ($r = 1, 2, ..., n$) вычисления проводятся в следующем порядке:

1. Вначале вычисляются элементы *r*-го столбца матрицы $\mathbf{L}$

$$l_{ir} = a_{ir} - \sum_{k=1}^{r-1} l_{ik}u_{kr}, \qquad i = r, \dots, n.$$

В формуле принято, что при $r = 1$ имеем $\sum_{k=1}^{r-1} l_{ik}u_{kr} = 0$.

2. Затем вычисляются элементы *r*-й строки матрицы $\mathbf{U}$

$$u_{ri} = \frac{1}{l_{rr}}\left(a_{ri} - \sum_{k=1}^{r-1} l_{rk}u_{ki}\right), \quad i = r+1, \dots, n.$$

Заметим, что элементы матриц **L** и **U** можно формировать на местах соответствующих элементов матрицы **A** (единичную диагональ матрицы **U** можно не хранить).

Решения системы (2.22) находятся по формуле

$$y_i = \frac{1}{l_{ii}}\left(b_i - \sum_{k=1}^{i-1} l_{ik} y_k \right), \quad i = 1,2,\ldots,n.$$

Решения системы (2.21) определяются выражением

$$x_i = y_i - \sum_{k=i+1}^{n} u_{ik} x_k \ , \ i = n, n-1, \ldots, 1 \ .$$

LU-разложение может быть реализовано, если $l_{rr} \neq 0$. Кроме того, близость l_{rr} к нулю может привести к большим погрешностям. Чтобы избежать этого, можно использовать *LU*-разложение *с выбором главного элемента по столбцу* [18]. Треугольные матрицы **L** и **U** строятся на месте исходной матрицы **A** (единичные диагональные элементы матрицы **U** не хранятся). На каждом шаге r вычисляются элементы l_{ir} столбца r матрицы **L** и размещаются на местах элементов a_{ir}. В непреобразованный элементах столбца r (расположенных ниже диагонали) находится максимальный по абсолютной величине элемент. Пусть $\max\limits_{i \geq r} |l_{ir}| = |l_{pr}|$, тогда строки r и p меняются местами. Затем вычисляются элементы u_{ri} строки r матрицы **U** и размещаются на местах элементов a_{ri}. Вырожденность матрицы решаемой системы определяется аналогично методу Гаусса с выбором главного элемента по столбцу — матрица вырождена, если все элементы непреобразованной части столбца близки к нулю. Так строки переставляются в непреобразованной части матрицы, то перестановка строк эквивалентна умножению исходной системы слева на матрицу перестановок: $\mathbf{PAx} = \mathbf{Pb}$. В результате разложения получаем $\mathbf{PA} = \mathbf{LU}$. Тогда вместо системы (2.22) должна решаться система $\mathbf{Ly} = \mathbf{Pb}$. В практически часто встречающемся случае диагонального преобладания матрицы **A** метод *LU*-разложения всегда осуществим.

Мы рассмотрели простейший и наиболее популярный вариант метода LU-разложения. Известны и другие варианты алгоритма [18].

2.5. Метод Холецкого

Этот метод называют еще методом Холеского (André-Louis Cholesky), *методом квадратных корней, методом LL^T-разложения*. Пусть матрица **A**

системы (2.2) симметричная и положительно определенная. Из симметрии матрицы следует $\mathbf{A}=\mathbf{A}^{\mathrm{T}}$. Тогда из разложения $\mathbf{A}=\mathbf{LU}$ и $\mathbf{A}^{\mathrm{T}}=\mathbf{U}^{\mathrm{T}}\mathbf{L}^{\mathrm{T}}$ следует, что $\mathbf{U}=\mathbf{L}^{\mathrm{T}}$, $\mathbf{L}=\mathbf{U}^{\mathrm{T}}$ и матрицу $\mathbf{A}$ можно представить в виде

$$\mathbf{A}=\mathbf{L}\mathbf{L}^{\mathrm{T}} \tag{2.23}$$

или

$$\mathbf{A}=\mathbf{U}^{\mathrm{T}}\mathbf{U}, \tag{2.24}$$

где $\mathbf{L}$ — нижняя треугольная матрица, $\mathbf{U}$ — верхняя треугольная матрица. Будем предполагать, что все диагональные элементы матриц $\mathbf{L}$ и $\mathbf{U}$ положительны.

Матрица $\mathbf{A}$ при использовании представлений (2.23) и (2.24) должна быть положительно определена, так как известно [20], что для произвольной невырожденной квадратной матрицы $\mathbf{M}$ матрица $\mathbf{A}=\mathbf{M}^{\mathrm{T}}\mathbf{M}$ является положительно определенной. Действительно, матрица $\mathbf{A}$ является симметричной, так как $\mathbf{A}^{\mathrm{T}}=\left(\mathbf{M}^{\mathrm{T}}\mathbf{M}\right)^{\mathrm{T}}=\mathbf{M}^{\mathrm{T}}\mathbf{M}^{\mathrm{TT}}=\mathbf{M}^{\mathrm{T}}\mathbf{M}=\mathbf{A}$. Для доказательства положительной определенности матрицы $\mathbf{A}$ докажем, что $\left(\mathbf{Ax},\mathbf{x}\right)>0$ для любого $\mathbf{x}\neq 0$. Действительно, $\left(\mathbf{Ax},\mathbf{x}\right)=\mathbf{x}^{\mathrm{T}}\mathbf{Ax}=\mathbf{x}^{\mathrm{T}}\mathbf{M}^{\mathrm{T}}\mathbf{Mx}$. Обозначим $\mathbf{y}=\mathbf{Mx}$, тогда $\mathbf{x}^{\mathrm{T}}\mathbf{Ax}=\mathbf{y}^{\mathrm{T}}\mathbf{y}=\sum_{i=1}^{n}y_i^2>0$.

Рассмотрим разложение (2.23). Разложение вида (2.24) используется, например, в MATLAB. Элементы матрицы $\mathbf{L}$ можно получить из матричного равенства

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \dots\dots\dots\dots\dots & & & & \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 & 0 & \dots & 0 \\ l_{21} & l_{22} & 0 & 0 & \dots & 0 \\ l_{31} & l_{32} & l_{33} & 0 & \dots & 0 \\ \dots\dots\dots\dots\dots & & & & & \\ l_{n1} & l_{n2} & l_{n3} & l_{n4} & \dots & l_{nn} \end{bmatrix} \cdot \begin{bmatrix} l_{11} & l_{21} & l_{31} & l_{41} & \dots & l_{n1} \\ 0 & l_{22} & l_{32} & l_{42} & \dots & l_{n2} \\ 0 & 0 & l_{33} & l_{43} & \dots & l_{n3} \\ \dots\dots\dots\dots\dots & & & & & \\ 0 & 0 & 0 & 0 & \dots & l_{nn} \end{bmatrix}$$

Из $a_{11}=l_{11}^2$ получаем

$$l_{11}=\sqrt{a_{11}}\,. \tag{2.25}$$

В (2.25) берется положительное значение корня, т. к. диагональные элементы матрицы $\mathbf{L}$ должны быть положительными.

Из формулы для элементов первой строки матрицы $\mathbf{A}$

$$a_{12}=l_{11}l_{21},\quad a_{13}=l_{11}l_{31},\quad \dots,\quad a_{1i}=l_{11}l_{i1},$$

получаем выражение для элементов первого столбца матрицы $\mathbf{L}$

$$l_{i1}=\frac{a_{1i}}{l_{11}}\quad .\quad i=2,3,\dots,n\,. \tag{2.26}$$

Рассмотрим выражения для диагональных элементов матрицы $\mathbf{A}$:

$$a_{22} = l_{21}^2 + l_{22}^2, \quad a_{33} = l_{31}^2 + l_{32}^2 + l_{33}^2,$$

откуда

$$l_{22} = \sqrt{a_{22} - l_{21}^2}, \quad l_{33} = \sqrt{a_{33} - l_{31}^2 - l_{32}^2}$$

и в общем виде

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}, \ i = 2, 3, \ldots, n. \tag{2.27}$$

Из $a_{23} = l_{21}l_{31} + l_{22}l_{32}$, $a_{34} = l_{31}l_{41} + l_{32}l_{42} + l_{33}l_{43}$ получим

$$l_{32} = \frac{a_{23} - l_{21}l_{31}}{l_{22}}, \ l_{43} = \frac{a_{34} - \left(l_{31}l_{41} + l_{32}l_{42}\right)}{l_{33}},$$

в общем виде

$$l_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik}l_{jk}}{l_{jj}}, \ i = 3, 4, \ldots, n; \ \ j = 2, 3, \ldots, \ i-1. \tag{2.28}$$

В (2.28) учтена симметрия матрицы $\mathbf{A}$.

Расчет элементов матрицы $\mathbf{L}$ производится в следующем порядке: сначала вычисляется l_{11} по (2.25), затем по (2.26) вычисляются элементы l_{i1} первого столбца матрицы $\mathbf{L}$. Далее производится расчет строк матрицы $\mathbf{L}$ по формулам (2.28) и (2.27). В (2.27) подкоренное выражение всегда положительно (доказывается, что это является следствием положительной определенности матрицы [18]).

Вместо исходной СЛАУ решаются две системы с треугольными матрицами

$$\mathbf{L}\mathbf{y} = \mathbf{b},$$

$$\mathbf{L}^{\mathrm{T}}\mathbf{x} = \mathbf{y}.$$

Решение проводится по следующим очевидным формулам

$$y_1 = \frac{b_1}{l_{11}}, \quad y_i = \frac{1}{l_{ii}}\left(b_i - \sum_{k=1}^{i-1} l_{ik}y_k\right), \ i = 2,3,\ldots,n,$$

$$x_n = \frac{y_n}{l_{nn}}, \quad x_i = \frac{1}{l_{ii}}\left(y_i - \sum_{k=i+1}^{n} l_{ki}x_k\right), \ i = n-1, n-2, \ldots, 1.$$

Для положительно определенной матрицы LL^{T}-разложение всегда выполнимо и не требует никаких перестановок. Достаточно вычислять и хранить только матрицу $\mathbf{L}$, поэтому число арифметических операций и объем памяти в методе Холецкого примерно вдвое меньше, чем в методе Гаусса и LU-разложении. Учитывая симметрию матрицы $\mathbf{A}$, можно хранить только элементы a_{ij} при $i \geq j$, т. е. нижнюю треугольную матрицу. Элементы l_{ij}

можно помещать на место элементов a_{ij}, так как из (2.26)–(2.28) следует, что элементы матрицы $\mathbf{A}$ участвуют в расчете элементов матрицы $\mathbf{L}$ только один раз. Метод Холецкого удобен для решения СЛАУ с симметричными положительно определенными ленточными матрицами. Из (2.26)–(2.28) можно доказать, что за пределами ленты $l_{ij} = 0$, т. е. матрица $\mathbf{L}$ имеет тот же вид, что и нижняя половина ленточной матрицы $\mathbf{A}$. При этом нулевым элементам a_{ij} внутри ленты будут соответствовать, как правило, ненулевые элементы l_{ij}. Таким образом, в случае ленточных матриц достаточно хранить только ленты матриц $\mathbf{A}$ и $\mathbf{L}$.

Рассмотрим пример применения метода Холецкого для решения СЛАУ с симметричными трехдиагональными матрицами [21] вида:

$$\mathbf{A} = \begin{pmatrix} a_1 & b_1 & 0 & 0 & \dots & 0 \\ b_1 & a_2 & b_2 & 0 & \dots & 0 \\ \dots\dots\dots\dots\dots\dots\dots\dots & & & & & \\ 0 & \dots & b_{n-2} & a_{n-1} & b_{n-1} & \\ 0 & \dots\dots\dots\dots & & b_{n-1} & a_n & \end{pmatrix}$$

Матрица $\mathbf{L}$ имеет вид

$$\mathbf{L} = \begin{pmatrix} w_1 & 0 & 0 & \dots & 0 \\ q_1 & w_2 & 0 & \dots & 0 \\ \dots\dots\dots\dots\dots\dots\dots\dots & & & & \\ 0 & \dots & q_{i-1} & w_i \dots 0 & & \\ \dots\dots\dots\dots\dots\dots\dots\dots & & & & & \\ 0 & \dots\dots\dots\dots\dots & & q_{n-1} & w_n & \end{pmatrix},$$

где $w_1 = \sqrt{a_1}$, $q_i = \dfrac{b_i}{w_i}$, $w_{i+1} = \sqrt{a_{i+1} - q_i^2}$.

Аналогично можно решать СЛАУ с пятидиагональными матрицами. СЛАУ с трех- и пятидиагональными матрицами — типичный пример разреженных матриц, формируемых при решении дифференциальных уравнений в частных производных методом конечных разностей.

Метод Холецкого является хорошим тестом на положительную определенность матрицы. Для проверки положительной определенности матрицы $\mathbf{A}$ необходимо попытаться вычислить матрицу $\mathbf{L}$. Если на некотором шаге вычислений под знаком квадратного корня в (2.27) получается неположительное число, то матрица $\mathbf{A}$ не является положительно определенной.

2.6. Варианты LU-разложения

Известно несколько вариантов *LU*-разложения [20]. Прежде всего — это *LDV*-разложении: если $\mathbf{A}$ — $n \times n$ матрица, у которой все ведущие главные подматрицы невырожденны (все угловые миноры отличны от нуля), то $\mathbf{A}$ можно представить единственным образом в виде произведения матриц

$$\mathbf{A} = \mathbf{LDV},$$

где $\mathbf{L}$ — нижняя унитреугольная матрица, $\mathbf{D}$ — диагональная матрица, $\mathbf{V}$ — верхняя унитреугольная матрица (униитреугольная матрица — треугольная матрица, все элементы диагонали которой равны единице).

Если $\mathbf{A}$ — симметричная матрица, у которой все ведущие главные подматрицы невырожденны, то $\mathbf{A}$ единственным образом может быть представлена в виде произведения $\mathbf{A} = \mathbf{LDL}^{\mathrm{T}}$, где $\mathbf{L}$ — нижняя унитреугольная матрица, а $\mathbf{D}$ — диагональная матрица.

Если $\mathbf{A}$ — положительно определенная матрица, то $\mathbf{A}$ можно единственным образом представить в виде произведения

$$\mathbf{A} = \mathbf{LDL}^{\mathrm{T}}, \tag{1.29}$$

где $\mathbf{L}$ — нижняя унитреугольная матрица, а $\mathbf{D}$ — диагональная матрица с положительными диагональными элементами.

Для СЛАУ с симметричными, положительно определенными матрицами практически удобнее использовать не LL^{T}-разложение, а LDL^{T}-разложение (1.29) [15, 22]. В методе LDL^{T}-разложения при сохранении всех свойств LL^{T}-разложения исключена операция извлечения квадратного корня, поэтому метод LDL^{T}-разложения находит большое практическое применение.

Из (2.2) и (1.29) получаем

$$\mathbf{LDL}^{\mathrm{T}}\mathbf{x} = \mathbf{b}.$$

Обозначим

$$\mathbf{DL}^{\mathrm{T}}\mathbf{x} = \mathbf{y}, \tag{2.30}$$

Тогда

$$\mathbf{Ly} = \mathbf{b}. \tag{2.31}$$

Система (2.31) — это СЛАУ с нижней треугольной матрицей, решения которой находятся из выражения

$$y_i = b_i - \sum_{k=1}^{i-1} l_{ik} y_k, \quad i = 1, 2, \ldots, n.$$

Вектор $\mathbf{x}$ — решение СЛАУ (2.30) с верхней треугольной матрицей

$$x_n = \frac{y_n}{d_{nn}}, \quad x_i = \frac{y_i}{d_{ii}} - \sum_{k=i+1}^{n} l_{ki} x_k, \quad i = n-1, n-2, \ldots, 1$$

Теперь получим расчетные выражения для разложения (1.29). Известны общие подходы для факторизованного представления матриц [17, 22]. Мы получим разложение (1.29), исходя из известного LL^{T}-разложения (для удобства сменим только обозначения)

$$\mathbf{A} = \mathbf{W}\mathbf{W}^{\mathrm{T}} \tag{2.32}$$

Сравнивая(1.29) и (2.32), можно записать

$$\mathbf{W} = \mathbf{L}\mathbf{Q}\ ,\quad \mathbf{Q} = \mathbf{D}^{\frac{1}{2}}\ ,\quad \mathbf{D} = \mathbf{D}^{\frac{1}{2}}\mathbf{D}^{\frac{1}{2}}$$

или в развернутом виде

$$\begin{pmatrix} w_{11} & 0 & 0 & 0 & \dots & 0 \\ w_{21} & w_{22} & 0 & 0 & \dots & 0 \\ w_{31} & w_{32} & w_{33} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ w_{n1} & w_{n2} & w_{n3} & w_{n4} & \dots & w_{nn} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ l_{21} & 1 & 0 & 0 & \dots & 0 \\ l_{31} & l_{32} & 1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ l_{n1} & l_{n2} & l_{n3} & l_{n4} & \dots & 1 \end{pmatrix} *$$

$$* \begin{pmatrix} q_{11} & 0 & 0 & 0 & \dots & 0 \\ 0 & q_{22} & 0 & 0 & \dots & 0 \\ 0 & 0 & q_{33} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & q_{nn} \end{pmatrix}$$

Из правил умножения матриц получаем $w_{11} = q_{11}$, что с учетом (2.25) даёт

$$d_{11} = q_{11}^2 = w_{11}^2 = a_{11}\text{, т. е.}$$

$$d_{11} = a_{11}.$$

Отметим также, что

$$q_{ii} = w_{ii}\ ,\quad i = 1, 2, \dots, n. \tag{2.33}$$

$$w_{ij} = l_{ij} q_{jj}\ ,\quad i = 2, 3, \dots, n\ ;\quad j = 1, 2, \dots, n. \tag{2.34}$$

Из (2.33), (2.34) и (2.27) получаем

$$d_{ii} = q_{ii}^2 = w_{ii}^2 = a_{ii} - \sum_{k=1}^{i-1} w_{ik}^2 = a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2 q_{kk}^2 = a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2 d_{kk},\quad i = 2, 3, \dots, n. \tag{2.35}$$

Из (2.34), (2.33) и (2.26) получаем с учетом симметрии матрицы $\mathbf{A}$

$$l_{i1} = \frac{w_{i1}}{q_{11}} = \frac{a_{i1}}{w_{11} q_{11}} = \frac{a_{i1}}{d_{11}}\ ,\quad i = 2, 3, \dots, n \tag{2.36}$$

Из (2.34), (2.33) и (2.28) получаем

$$l_{ij} = \frac{1}{q_{jj}} w_{ij} = \frac{1}{q_{jj} w_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} q_{kk} l_{jk} q_{kk} \right) = \frac{1}{d_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} d_{kk} \right), \tag{2.37}$$

$$i = 3, 4, \dots, n;\quad j = 2, 3, \dots, i-2.$$

Таким образом, расчет элементов матриц $\mathbf{L}$ и $\mathbf{D}$ производится в следующем порядке:

1. Все диагональные элементы $l_{ii} = 1, \quad i = 1, 2, ..., n$ (эти элементы можно не хранить).

2. $d_{11} = a_{11}$.

3. Рассчитываются по (2.36) элементы первого столбца матрицы $\mathbf{L}$.

4. Далее по строкам рассчитываются d_{ii} (2.35) и l_{ij} (2.37).

2.7. Понятие QR-разложения

Известно, что произвольную матрицу $\mathbf{A}$ можно представить в виде *QR*-разложения

$$\mathbf{A} = \mathbf{QR}, \tag{1.38}$$

где $\mathbf{Q}$ — ортогональная матрица, $\mathbf{R}$ — верхняя треугольная матрица.

Вспомним, что в случае ортогональной матрицы обратная матрица равна транспонированной.

С учетом (1.38) для СЛАУ

$$\mathbf{Ax} = \mathbf{b} \tag{1.39}$$

получаем

$$\mathbf{QRx} = \mathbf{b}.$$

Обозначим $\mathbf{Rx} = \mathbf{y}$, тогда

$$\mathbf{Qy} = \mathbf{b}. \tag{1.40}$$

Решение СЛАУ (1.40) с учетом ортогональности матрицы $\mathbf{Q}$ равно $\mathbf{y} = \mathbf{Q}^{-1}\mathbf{b} = \mathbf{Q}^{\mathrm{T}}\mathbf{b}$. Тогда для нахождения решения системы (1.39) необходимо решить систему

$$\mathbf{Rx} = \mathbf{Q}^{\mathrm{T}}\mathbf{b}.$$

Формулы реализации QR-разложения (1.38) можно найти, например, в [20, 23].

В методе *QR*-разложения, в отличие от других прямых методов, не наблюдается роста элементов в процессе преобразования и *метод является численно устойчивым и лучше обусловленным*, чем метод Гаусса. Однако метод *QR*-разложения значительно более трудоемок, чем метод Гаусса.

Отметим, что метод QR-разложения применим к *любым матрицам*, в том числе вырожденным. Признаком вырожденности матрицы $\mathbf{A}$ будет являться наличие нулевых (точнее, очень малых) значений на диагонали матрицы $\mathbf{R}$. Действительно, из ортогональности матрицы $\mathbf{Q}$ следует, что $\mathbf{QQ}^{\mathrm{T}} = \mathbf{E}$. Так как транспонирование не изменяет определитель матрицы, то

$\det\left(\mathbf{QQ}^{\mathbf{T}}\right)=\det\mathbf{E}$ или $\left(\det\mathbf{Q}\right)^2=1$, откуда $\det\mathbf{Q}=\pm1$. Из разложения $\mathbf{A}=\mathbf{QR}$ и из равенства определителя треугольной матрицы произведению диагональных элементов следует

$$\det\mathbf{A}=\det\mathbf{Q}\cdot\det\mathbf{R}=\pm\det\mathbf{R}=\pm r_{11}r_{22}...r_{nn}\,.$$

Таким образом, равенство нулю определителя матрицы $\mathbf{A}$ возможно, если хотя бы один из диагональных элементов матрицы $\mathbf{R}$ равен нулю.

2.8. Вычисление определителей и обращение матриц

В современных численных методах вычисление определителей используется редко. Следует также отметить крайнюю неэффективность непосредственного вычисления определителей. Так, для вычисления определителя матрицы $n\times n$ потребуется $(n-1)n!$ умножений и $n!$ сложений. Если $n=20$ (практически это весьма простая задача) и на выполнение одной операции требуется $10^{-6}c$, то для вычисления определителя потребуется более миллиона лет, а для решения СЛАУ с такой матрицей при такой же производительности компьютера — примерно 0.005с [26]. По этой причине математически безукоризненный *классический метод Крамера совершенно не применим для решения реальных задач*.

Если все же необходимо вычислить *определитель матрицы*, то используют какое-либо разложение матрицы на треугольные, например, *LU*-разложение или преобразование к треугольному виду, например, в результате прямого хода метода Гаусса и известные свойства определителей [4]: определитель произведения квадратных матриц равен произведению определителей сомножителей; определитель треугольной матрицы равен произведению диагональных элементов; определитель меняет знак при перестановке строк матрицы. Тогда, используя, например, *LU*-разложение, получим

$$\det\mathbf{A}=(-1)^s\det\mathbf{L}\cdot\det\mathbf{U}=(-1)^s l_{11}l_{22}\ldots l_{nn}, \qquad (2.41)$$

где s — число перестановок строк в процессе LU-преобразования. В (2.41) учтено, что матрица $\mathbf{U}$ имеет единичную диагональ.

Вычисление обратных матриц также крайне редко используется в современном вычислительном эксперименте из-за большой трудоемкости решения такой задачи. Практически всегда целесообразно заменить умножение $\mathbf{x}=\mathbf{A}^{-1}\mathbf{b}$ решением системы $\mathbf{Ax}=\mathbf{b}$. Если требуется многократно решать системы с одной и той же матрицей $\mathbf{A}$ и различными правыми частями, то использование, например, *LU*-разложения потребует примерно в

три раза меньше времени, чем нахождение $\mathbf{A}^{-1}$. Обратную матрицу целесообразно вычислять, если требуется анализ элементов этой матрицы.

Исходя из свойств обратной матрицы видно, что нахождение элементов обратной матрицы эквивалентно решению матричного уравнения

$$\mathbf{AX} = \mathbf{E},$$

где $\mathbf{X}$ — искомая обратная матрица; $\mathbf{E}$ — единичная матрица.

Например, при $n = 2$ имеем

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

откуда получаем

$$\begin{bmatrix} a_{11}x_{11} + a_{12}x_{21} & a_{11}x_{12} + a_{12}x_{22} \\ a_{21}x_{11} + a_{22}x_{21} & a_{21}x_{12} + a_{22}x_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \qquad (2.42)$$

Из условия равенства матриц (2.42) получаем

$$\begin{cases} a_{11}x_{11} + a_{12}x_{21} = 1, \\ a_{21}x_{11} + a_{22}x_{21} = 0, \end{cases} \qquad \begin{cases} a_{11}x_{12} + a_{12}x_{22} = 0, \\ a_{21}x_{12} + a_{22}x_{22} = 1, \end{cases}$$

т. е. нахождение элементов обратной матрицы эквивалентно в рассматриваемом примере решению двух СЛАУ с одной и той же матрицей $\mathbf{A}$, но с различными правыми частями. В общем виде для матрицы ($n \times n$) требуется решить n СЛАУ вида

$$\mathbf{A}\mathbf{x}^{(j)} = \boldsymbol{\delta}^{(j)}, \quad j = 1, 2, \ldots, n, \qquad (2.43)$$

где $\mathbf{x}^{(j)} = \left(x_{1j}, x_{2j}, \ldots, x_{nj} \right)^{\mathrm{T}}$ — j-й столбец обратной матрицы,

$$\delta_i^{(j)} = \begin{cases} 1, & \text{если } i = j, \\ 0, & \text{если } i \neq j. \end{cases}$$

Систему (2.43) можно решить, например, методом LU-разложения. В процессе прямого хода получаются и запоминаются матрицы $\mathbf{L}$ и $\mathbf{U}$. В процессе обратного хода решаются системы

$$\mathbf{L}\mathbf{y}^{(j)} = \boldsymbol{\delta}^{(j)}, \ \mathbf{y}^{(j)} = \left(y_{1j}, y_{2j}, \ldots, y_{nj} \right)^{T}, \qquad (2.44)$$

$$\mathbf{U}\mathbf{x}^{(j)} = \mathbf{y}^{(j)} \qquad (2.45)$$

При решении систем (2.44) и (2.45) можно существенно сократить число действий, если учитывать вид векторов $\boldsymbol{\delta}^{(j)}$ и $\mathbf{y}^{(j)}$. Так, очевидно, для (2.44)

$$y_{ij} = 0, \text{ если } i < j,$$

$$y_{jj} = \frac{1}{l_{ii}}.$$

$$y_{ij} = -\frac{1}{l_{ii}}\left(\sum_{k=j}^{i-1} l_{ik} y_{kj}\right), \quad i = j+1, \quad j+2, \ldots, n.$$

Аналогичные формулы можно получить и для (2.45).

Элементы обратной матрицы можно вычислить, используя преобразования Гаусса-Жордана [27] Для этого исходная матрица дополняется единичной матрицей и расширенная таким образом матрица преобразуется по методу Гаусса-Жордана. В результате исходная матрица преобразуется в единичную, а дополняющая единичная — в обратную. Этот метод также основан на решении n систем уравнений.

2.9. Решение систем с прямоугольными матрицами

Пусть в результате эксперимента построена таблица (x_i, y_i) приближенных значений функции y. Из некоторых соображений, например, из физического смысла функции или внешнего вида экспериментальной зависимости выбран вид функции $y = \psi(x)$. Функция $\psi(x)$ содержит ряд числовых параметров. Требуется так выбрать эти параметры, чтобы функция $y = \psi(x)$ в некотором смысле наилучшим образом отображала экспериментальную зависимость.

Часто для аппроксимации результатов эксперимента используется линейная модель

$$y = \Phi_m(x) = a_1\varphi_1(x) + a_2\varphi_2(x) + \ldots + a_m\varphi_m(x), \qquad (2.46)$$

где $\varphi_1(x), \ldots, \varphi_m(x)$ — заданные *базисные функции*, $a_1, \ldots, a_m$ — искомые параметры (модель является *линейной*, так как искомые параметры входят в нее линейно).

Наиболее простой линейной моделью (2.46) является *полиномиальная модель*

$$P_m(x) = a_1 + a_2 x + \ldots + a_m x^{m-1}.$$

Если потребовать, чтобы $\Phi_m(x_i) = y_i$, $i = 1, 2, \ldots, n$, то искомые параметры $a_1, a_2, \ldots, a_m$ могут быть найдены из решения системы линейных уравнений

$$\mathbf{Pa} = \mathbf{y}, \qquad (2.47)$$

где

$$\mathbf{P}=\begin{bmatrix}\varphi_1(x_1) & \varphi_2(x_1) & \cdots & \varphi_m(x_1)\\ \varphi_1(x_2) & \varphi_2(x_2) & \cdots & \varphi_m(x_2)\\ \cdots & \cdots & \cdots & \cdots\\ \varphi_1(x_n) & \varphi_2(x_n) & \cdots & \varphi_m(x_n)\end{bmatrix},\quad \mathbf{a}=\begin{bmatrix}a_1\\ a_2\\ \vdots\\ a_m\end{bmatrix},\quad \mathbf{y}=\begin{bmatrix}y_1\\ y_2\\ \vdots\\ y_n\end{bmatrix}. \qquad (2.48)$$

Так как обычно $n>m$ (число значений данных больше степени полинома), то система (2.47)–(2.48) является *переопределенной*: число уравнений превосходит число переменных. Найти вектор $\mathbf{a}$, точно удовлетворяющий системе, невозможно. Более того, это нецелесообразно, так как тогда полином будет повторять ошибки эксперимента. Необходимо *сгладить* экспериментальные зависимости. Сглаживающий полином должен быть выбран таким образом, чтобы минимизировать некоторую меру расхождения между полиномом и экспериментальными данными. Обычно используют критерий *наименьших квадратов* [28], минимизирующий евклидову норму вектора невязки $r_i=\Phi_m(x_i)-y_i$. В методе наименьших квадратов необходимо найти вектор $\mathbf{a}=\left[\,a_1\ldots a_m\right]^{\mathrm{T}}$, который при фиксированном наборе функций $\varphi_1,\varphi_2,\ldots,\varphi_m$ обеспечивает минимальное значение функции

$$I=\|\mathbf{r}\|^2=\sum_{i=1}^{n}\left(\Phi_m(x_i)-y_i\right)^2. \qquad (2.49)$$

Так как минимизируется сумма квадратов, то метод назван методом наименьших квадратов.

Простейшим методом решения задачи наименьших квадратов является подход, основанный на решении *нормальной системы уравнений*. Используя (2.48), запишем (2.49) в виде

$$I=\sum_{i=1}^{n}\left(\sum_{j=1}^{m}a_j\varphi_j(x_i)-y_i\right)^2=\|\mathbf{Pa}-\mathbf{y}\|^2, \qquad (2.50)$$

Формирование нормальной системы уравнений основано на использовании необходимого условия экстремума функции I (2.50)

$$\frac{\partial I}{\partial a_k}=0,\quad k=1,2,...,m. \qquad (2.51)$$

Вычисляя частные производные (2.51) с учетом (2.50), получаем

$$\sum_{i=1}^{n}2\left(\sum_{j=1}^{m}a_j\varphi_j(x_i)-y_i\right)\frac{\partial}{\partial a_k}\left(\sum_{j=1}^{m}a_j\varphi_j(x_i)-y_i\right)=0,$$

$$\sum_{i=1}^{n}\left(\sum_{j=1}^{m}a_j\varphi_j(x_i)-y_i\right)\varphi_k(x_i)=0,$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m} a_j \varphi_j(x_i)\varphi_k(x_i) = \sum_{i=1}^{n} \varphi_k(x_i) y_i, \quad k = 1, 2, ..., m. \tag{2.52}$$

Изменяя порядок суммирования в (2.52), получаем систему линейных алгебраических уравнений — *нормальную систему метода наименьших квадратов*

$$\sum_{j=1}^{m}\left(\sum_{i=1}^{n} \varphi_j(x_i)\varphi_k(x_i) \right) a_j = \sum_{i=1}^{n} y_i \varphi_k(x_i), \quad k = 1, ..., m. \tag{2.53}$$

В матричной форме система (2.53) имеет вид

$$\mathbf{P}^{\mathrm{T}}\mathbf{P}\mathbf{a} = \mathbf{P}^{\mathrm{T}}\mathbf{y}. \tag{2.54}$$

Матрица $\mathbf{P}^{\mathrm{T}}\mathbf{P}$ системы (2.54) является квадратной с размерами $m \times m$, симметричной и положительно определенной (если матрица $\mathbf{P}$ невырожденная). Поэтому система (2.54) может быть решена известными методами. Однако нормальная система может быть плохо обусловленной, а полученное решение настолько искаженным ошибками в исходных данных и ошибками округления, что не может быть использовано. Плохая обусловленность матрицы $\mathbf{P}$ возникает потому, что очень часто система базисных функций $\varphi_1 \ \varphi_2 \ \cdots \ \varphi_m$ оказывается близкой к линейно зависимой. В случае линейной зависимости базисных функций матрица $\mathbf{P}^{\mathrm{T}}\mathbf{P}$ является вырожденной [18]. Поэтому нормальные системы применяют при $m < 5$ [18], что позволяет, однако, решить многие практические задачи сглаживания экспериментальных зависимостей.

2.10. Использование сингулярного разложения

Для любой вещественной $n \times m$ матрицы $\mathbf{A}$ произведения $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ и $\mathbf{A}\mathbf{A}^{\mathrm{T}}$ являются квадратными $m \times m$ и $n \times n$ матрицами. Собственные числа матриц $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ и $\mathbf{A}\mathbf{A}^{\mathrm{T}}$ вещественные и неотрицательные (некоторые собственные числа могут быть нулевыми). Известно [4], что ненулевые собственные числа матриц $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ и $\mathbf{A}\mathbf{A}^{\mathrm{T}}$ совпадают. *Сингулярными числами* [4] (от лат. Singularis — отдельный, особый) вещественной матрицы $n \times m$ $\mathbf{A}$ называются арифметические значения квадратных корней из общих собственных чисел матриц $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ и $\mathbf{A}\mathbf{A}^{\mathrm{T}}$.

Практически, если $n > m$, то сингулярными числами матрицы $\mathbf{A}$ определяются собственными числами матрицы $\mathbf{A}^{\mathrm{T}}\mathbf{A}$, если $n < m$, то — матрицы $\mathbf{A}\mathbf{A}^{\mathrm{T}}$. Так как мы рассматриваем, прежде всего, переопределенные матрицы, то будем определять сингулярные числа по матрице $\mathbf{A}^{\mathrm{T}}\mathbf{A}$. Если

матрица $\mathbf{A}$ является квадратной и симметричной, то $\mathbf{A}^{\mathrm{T}}\mathbf{A} = \mathbf{A}\mathbf{A}^{\mathrm{T}}$ и модули собственных чисел равны их сингулярным числам. Для положительно определенных матриц собственные и сингулярные числа совпадают. То есть понятие сингулярного числа является обобщением понятия собственного числа матрицы.

Будем обозначать сингулярные числа буквой σ и нумеровать в порядке убывания: $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_k$. Количество ненулевых сингулярных чисел матрицы $\mathbf{A}$ совпадает с ее рангом (ранги матриц $\mathbf{A}$, $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ и $\mathbf{A}\mathbf{A}^{\mathrm{T}}$ равны) [29]. Произвольная $n \times m$ матрица $\mathbf{A}$ может быть представлена в виде *сингулярного разложения*, или SVD-разложения (от англ. Singular Value Decomposition) [20, 29–30]

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}, \tag{2.55}$$

где $\mathbf{\Sigma}$ — $n \times m$ матрица, квадратная подматрица которой является диагональной с диагональю из невозрастающих сингулярных чисел $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_k$, $k = \operatorname{rank}\mathbf{A}$; $\mathbf{U}$ и $\mathbf{V}$ — ортогональные, соответственно, $n \times n$ и $m \times m$ матрицы.

Столбцы $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_m$ матрицы $\mathbf{U}$ называются *левыми сингулярными векторами* матрицы $\mathbf{A}$. Столбцы $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ матрицы $\mathbf{V}$ называются *правыми сингулярными векторами* матрицы $\mathbf{A}$. Сингулярные векторы являются ортонормированными [20, 29], то есть имеют единичную евклидову норму и попарно ортогональны [4]. Поясним структуру матрицы $\mathbf{\Sigma}$. Если $n > m$, матрица имеет следующую блочную структуру

$$\mathbf{\Sigma} = \begin{bmatrix} \mathbf{\Sigma}_m \\ \cdots\cdots \\ 0 \end{bmatrix},$$

где $\mathbf{\Sigma}_m$ — квадратная диагональная матрица $m \times m$ с элементами $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_m$.

Если $n < m$, то $\mathbf{\Sigma} = \left[\mathbf{\Sigma}_n \vdots 0\right]$, где $\mathbf{\Sigma}_n$ — квадратная диагональная матрица $n \times n$ с элементами $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n$.

Получение сингулярного разложения выходит за рамки нашего курса. Различные библиотеки численных методов включают функции сингулярного разложения. Предположим, что сингулярное разложение известно и рассмотрим некоторые *применения SVD* [13, 14, 20].

Как уже отмечалось, *ранг матрицы* совпадает с числом ее ненулевых сингулярных чисел. Если $\mathbf{A}$ — квадратная матрица $n \times n$, то $\left|\det\mathbf{A}\right| = \sigma_1\sigma_2\ldots\sigma_n$, так как определитель произведения матриц равен произведению определителей, определитель ортогональной матрицы по модулю равен

единице, а определитель диагональной матрицы равен произведению диагональных элементов. Если $\operatorname{rank}\mathbf{A} < n$, то $\det\mathbf{A} = 0$, так как на диагонали матрицы $\mathbf{\Sigma}$ будут нулевые элементы.

Число *обусловленности прямоугольной матрицы* $\mathbf{A}$ может быть определено отношением наибольшего σ_1 и наименьшего σ_k ненулевых сингулярных чисел: $\operatorname{cond}\mathbf{A} = \sigma_1/\sigma_k$. Таким образом, понятие числа обусловленности обобщается на прямоугольные матрицы.

Рассмотрим использование SVD-разложения для *решения переопределенной задачи наименьших квадратов* (2.47) в общем случае, когда $\operatorname{rank}\mathbf{A} = r < m$. Пусть имеется разложение (2.55).

$$\mathbf{P} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}.$$

При сделанных допущениях матрица $\mathbf{\Sigma}$ имеет вид

$$\mathbf{\Sigma} = \left[\begin{array}{c|c} \hat{\mathbf{\Sigma}} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} \end{array}\right],$$

где $\hat{\mathbf{\Sigma}} = \operatorname{diag}\{\sigma_1\ \sigma_2 \ldots \sigma_r\}$.

Так как умножение на ортогональную матрицу не изменяет евклидову норму вектора, то с учетом сингулярного разложения (2.55) представим евклидову норму невязки в виде

$$\|\mathbf{y} - \mathbf{P}\mathbf{a}\| = \left\|\mathbf{U}^{\mathrm{T}}\left(\mathbf{y} - \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}\mathbf{a}\right)\right\| = \left\|\mathbf{U}^{\mathrm{T}}\mathbf{y} - \mathbf{\Sigma}\left(\mathbf{V}^{\mathrm{T}}\mathbf{a}\right)\right\|.$$

Обозначим $\mathbf{c} = \mathbf{U}^{\mathrm{T}}\mathbf{y}$ и $\mathbf{z} = \mathbf{V}^{\mathrm{T}}\mathbf{a}$. Тогда

$$\|\mathbf{y} - \mathbf{P}\mathbf{a}\|^2 = \|\mathbf{c} - \mathbf{\Sigma}\mathbf{z}\|^2 = \sum_{i=1}^{r}\left(c_i - \sigma_i z_i\right)^2 + \sum_{i=r+1}^{m} c_i^{\,2}. \qquad (2.56)$$

В (2.56) учтено, что ненулевые подматрицы $\hat{\mathbf{\Sigma}}$ матрицы $\mathbf{\Sigma}$ имеет размер $r \times r$, а вектор $\mathbf{c}$ имеет размер $m \times 1$. Выражение (2.56) принимает минимальное значение при

$$z_i = \frac{c_i}{\sigma_i}, \quad i = 1, 2, \ldots, r. \qquad (2.57)$$

При этом компоненты решения z_i, $i = r+1, \ldots, m$ могут принимать произвольные значения. Выбор $z_i = 0$, $i = r+1, \ldots, m$ дает решение $\mathbf{a}$ с минимальной нормой, называемое *псевдорешением*. Действительно, решение системы (2.47) равно $\mathbf{a} = \mathbf{V}\mathbf{z}$, матрица $\mathbf{V}$ ортогональна, поэтому $\|\mathbf{a}\| = \|\mathbf{z}\|$. Таким образом, задача минимальных квадратов имеет единственное псевдорешение.

Как видно из (2.57), наличие малых сингулярных чисел приводит к большим погрешностям z_i. На практике все сингулярные числа, меньшие

некоторой малой величины, заменяются нулями, а соответствующие значения z_i полагаются равными нулю. Такой прием называется *TSVD — Truncated Singular Value Decomposition* [30–31]. В простейшем случае сингулярные числа, меньшие абсолютной величины погрешности задания элементов матрицы, считаются равными нулю. Поэтому SVD-разложение всегда позволяет оценить *численный ранг матрицы* в неполноранговых задачах наименьших квадратов. Например, в MATLAB функция **`ranc`** вычисляет ранг матрицы как количество сингулярных значений, превышающих машинную точность, умноженную на максимальную размерность матрицы.

Введя блочное представление векторов

$$\mathbf{c} = \begin{bmatrix} \hat{\mathbf{c}} \\ \hline \mathbf{d} \end{bmatrix}, \quad \mathbf{z} = \begin{bmatrix} \hat{\mathbf{z}} \\ \hline \mathbf{w} \end{bmatrix},$$

где $\mathbf{c}$ и $\mathbf{z}$ — векторы размером $m \times 1$, а $\hat{\mathbf{c}}$ и $\hat{\mathbf{z}}$ — векторы размером $r \times 1$, запишем алгоритм решения неполноранговой задачи наименьших квадратов с использованием SVD-разложения [20].

1. Вычисляем $\begin{bmatrix} \hat{\mathbf{c}} \\ \mathbf{d} \end{bmatrix} = \mathbf{U}^{\mathrm{T}}\mathbf{y}$.
2. Находим $\hat{\mathbf{z}} = \hat{\mathbf{\Sigma}}^{-1}\hat{\mathbf{c}}$.
3. Полагаем $\mathbf{w} = 0$, размерностью $(m - r) \times 1$.
4. Полагаем $\mathbf{z} = \begin{bmatrix} \hat{\mathbf{z}} \\ \mathbf{w} \end{bmatrix}$.
5. Полагаем $\mathbf{a} = \mathbf{V}\mathbf{z}$.

Понятие SVD-разложения позволило ввести обобщение обратной матрицы — *псевдообратную матрицу Мура-Пенроуза* (Moor-Penrose Pseudoinverse) [20, 25, 29]. Для прямоугольной матрицы $\mathbf{A}$, возможно, неполного ранга псевдообратная матрица имеет вид

$$\mathbf{A}^{+} = \mathbf{V}\mathbf{\Sigma}^{+}\mathbf{U}^{\mathrm{T}},$$

где $\mathbf{\Sigma}^{+} = \left[\begin{array}{c|c} \hat{\mathbf{\Sigma}}^{-1} & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} \end{array}\right]$.

Зачастую, например, в [20] используют следующее обозначение псевдообратной матрицы: $\mathbf{A}^{\dagger}$. С помощью псевдообратной матрицы решение произвольной СЛАУ $\mathbf{A}\mathbf{x} = \mathbf{b}$ запишется в виде

$$\mathbf{x} = \mathbf{A}^{+}\mathbf{b} = \mathbf{V}\mathbf{\Sigma}^{+}\mathbf{U}^{\mathrm{T}}\mathbf{b}. \qquad (2.58)$$

Выражение (2.58) приводит к тому же результату, что и описанный алгоритм решения неполноранговой задачи наименьших квадратов с использованием SVD-разложения. То есть $\mathbf{A}^{+}$ играет роль обратной матрицы.

С помощью псевдообратной матрицы можно определить число обусловленности прямоугольной матрицы $\mathbf{A}$ [4]: $\operatorname{cond}\mathbf{A} = \|\mathbf{A}\| \cdot \|\mathbf{A}^{+}\|$.

Псевдообратная матрица — это инструмент теоретического исследования. Так же как и в случае обратной матрицы, решение в виде (2.58) практически не применяется.

2.11. Вопросы и задания для самопроверки

1. Какие из рассмотренных прямых методов применимы для невырожденных матриц общего вида, а какие — только для положительно определенных?
2. Как при реализации метода Гаусса можно обнаружить вырожденность матрицы решаемой системы?
3. Как получить решение СЛАУ, если производились перестановки строк и столбцов?
4. Как формируются матрицы перестановки строк и столбцов?
5. Какие из рассмотренных методов не увеличивают ширину ленты матрицы?
6. Постройте алгоритмы, например, в виде псевдокода, блок-схемы или пошагового описания и структуры данных для следующих методов:
   - метод Гаусса с выбором главного элемента;
   - метод Жордана-Гаусса;
   - метод LU-разложения;
   - метод Холецкого;
   - метод прогонки;
   - вычисление определителей с использованием одного из треугольных разложений матрицы;
   - вычисления обратной матрицы с использованием LU-разложения;
   - решение переопределенных систем с использованием сингулярного разложения.
7. Получите расчетные формулы для метода Холецкого в случае симметричной положительно определенной пятидиагональной матрицы (за пределами ленты вычисления не проводить).
8. В чем преимущества использования сингулярного разложения при решении переопределенных систем?

Библиографический список к главе 2

1. Босс В. Лекции по математике: линейная алгебра. Т. 3. — М.: Либроком, 2014. — 230 с.

2. Воеводин В. В. Линейная алгебра. — СПб.: Лань, 2006. — 416 с.

3. Воеводин В. В., Воеводин Вл. В. Энциклопедия линейной алгебры. Электронная система ЛИНЕАЛ. — СПб.: БХВ-Петербург, 2006. — 544 с.

4. Воеводин В. В., Кузнецов Ю. А. Матрицы и вычисления. — М.: Наука, 1984. — 320 с.

5. Гантмахер Ф. Л. Теория матриц. — М.: Наука, 1967. — 576 с.

6. Голуб Дж., Ван Лоу Ч. Матричные вычисления. — М.: Мир, 1999. — 548 с.

7. Ильин В. А., Позняк Э. Г. Линейная алгебра. — М.: ФИЗМАТЛИТ, 2014. — 370 с.

8. Канатников А. Н., Крищенко А. П. Линейная алгебра. — М.: МГТУ им. Баумана, 2006. — 336 с.

9. Стренг Г. Линейная алгебра и ее приложения. — М.: Мир, 1980. — 454 с.

10. Тыртышников Е. Е. Матричный анализ и линейная алгебра. — М.: ФИЗМАТЛИТ, 2007. — 480 с.

11. Фаддеев Д. К., Фаддева В. Н. Вычислительные методы линейной алгебры. — СПб.: "Лань", 2002. — 736 с.

12. Хорн Р., Джонсон Ч. Матричный анализ. — М.: Мир, 1989. — 655 с.

13. Шевцов Г. С. Линейная алгебра: теория и прикладные аспекты. — М.: Финансы и статистика, 2003. — 576 с.

14. Шевцов Г. С., Крюкова О. Г., Мызникова Б. И. Численные методы линейной алгебры. — М.: Финансы и статистика: ИНФРА-М, 2008. — 480 с.

15. Писсанецки С. Технология разреженных матриц. — М: Мир, 1988. — 410 с.

16. Тьюарсон Р. Разреженные матрицы. — М.: Мир, 1977. — 190 с.

17. Джордж А., Лю Дж. Численное решение больших разреженных систем уравнений. — М.: Мир, 1984. — 333 с.

18. Амосов А. А., Дубинский Ю. А., Копчёнова Н. В. Вычислительные методы. — М.: Лань, 2014. — 672 с.

19. Самарский А. А., Николаев Е. С. Методы решения сеточных уравнений. — М.: Наука, 1978. — 592 с.

20. Уоткинс Д. С. Основы матричных вычислений. — М.: БИНОМ. Лаборатория знаний, 2006. — 664 с.

21. Азиз Х., Сеттари Э. Математическое моделирование пластовых систем. — Ижевск: РХД, 2004. — 416 с.

22. Икрамов Х. Д. Численные методы для симметричных линейных систем. — М.: Наука, 1988. — 160 с.

23. Горбаченко В. И. Вычислительная линейная алгебра с примерами на MATLAB. — СПб.: БХВ-Петербург, 2011. — 320 с.

24. Вержбицкий В. М. Численные методы (линейная алгебра и нелинейные уравнения). — М.: ООО "Издательский дом "ОНИКС 21 век", 2005. — 432 с.

25. Уилкинсон Дж. Алгебраическая проблема собственных значений. — М.: Наука, 1970. — 564 с.

26. Ортега Дж., Пул У. Введение в численные методы решения дифференциальных уравнений. — М.: Наука, 1986. — 288 с.

27. Шуп Т. Прикладные численные методы в физике и технике. — М.: Высшая школа, 1990. — 255 с.

28. Лоусон Ч., Хенсен Р. Численное решение задач метода наименьших квадратов. — М.: Наука, 1986. — 232 с.

29. Деммель Дж. Вычислительная линейная алгебра. Теория и приложения. — М.: Мир, 2001. — 430 с.

30. Каханер Д., Моулер К., Нэш С. Численные методы и программное обеспечение. — М.: Мир, 2001. — 575 с.

31. Леонов А. С. Решение некорректно поставленных обратных задач: Очерк теории, практические алгоритмы и демонстрации в MATLAB. — М.: Книжный дом "ЛИБРОКОМ", 2010. — 336 с.

ГЛАВА 3. КЛАССИЧЕСКИЕ ИТЕРАЦИОННЫЕ МЕТОДЫ РЕШЕНИЯ СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ

3.1. Основные теоретические положения итерационных методов

Итерациионные методы формируют решение СЛАУ путем последовательных приближений. Итерационные методы применяют прежде всего для решения СЛАУ с большими разреженными матрицами. Такие системы возникают, в частности, при решении большого количества задач моделирования различных физических процессов. Такие задачи описываются дифференциальными уравнениями в частных производных. После дискретизации уравнения в частных производных заменяются СЛАУ большой размерности с разреженными матрицами. Прямые методы для решения таких систем неудобны, так как при их использовании большое число нулевых элементов матриицы превращается в ненулевые, что затрудняет решение. При использованиии итерационных методов матрица не меняется и соохраняет разреженность.

Пусть решается СЛАУ $\mathbf{Ax}=\mathbf{b}$ с вещественной невырожденной квадратной матрицей $\mathbf{A}$. В итерационных методах исходя из некоторого начального приближения $\mathbf{x}^{(0)}$, строится последовательность векторов $\mathbf{x}^{(0)},\mathbf{x}^{(1)},\mathbf{x}^{(2)},\ldots,\mathbf{x}^{(k)},\ldots$, где k – номер *итерации* (приближения). Эта последовательность векторов должна сходиться к точному решению $\mathbf{x}^{*}=\mathbf{A}^{-1}\mathbf{b}$ СЛАУ при любом $\mathbf{x}^{(0)}$

$$\lim_{k\to\infty}\mathbf{x}^{(k)}=\mathbf{x}^{*}.$$

В этом случае итерационный процесс называется *сходящимся.*

С использованием матричных обозначений классические итерационные методы [1–5] могут быть описаны следующим образом. Представим квадратную невырожденную матрицу $\mathbf{A}$ в виде

$$\mathbf{A}=\mathbf{M}-\mathbf{N}, \qquad (3.1)$$

где $\mathbf{M}$ – невырожденная матрица, называемая *расщепляющей матрицей*, а представление (3.1) называется *расщеплением матрицы* $\mathbf{A}$.

С использованием понятия расщепления перепишем СЛАУ $\mathbf{Ax}=\mathbf{b}$ в виде $\mathbf{Mx}=\mathbf{Nx}+\mathbf{b}$ или $\mathbf{x}=\mathbf{M}^{-1}\mathbf{Nx}+\mathbf{M}^{-1}\mathbf{b}$, тогда одна *итерация итерационного метода* определяется следующим образом

$$\mathbf{M}\mathbf{x}^{(k+1)} = \mathbf{N}\mathbf{x}^{(k)} + \mathbf{b},$$

или

$$\mathbf{x}^{(k+1)} = \mathbf{M}^{-1}\mathbf{N}\mathbf{x}^{(k)} + \mathbf{M}^{-1}\mathbf{b} = \mathbf{C}\mathbf{x}^{(k)} + \mathbf{d}, \qquad (3.2)$$

где

$$\mathbf{C} = \mathbf{M}^{-1}\mathbf{N} = \mathbf{E} - \mathbf{M}^{-1}\mathbf{A} \qquad (3.3)$$

— *итерирующая матрица* (*матрица перехода*, *матрица коррекции*), $\mathbf{d} = \mathbf{M}^{-1}\mathbf{b}$, $\mathbf{E}$ – единичная матрица, $\mathbf{x}^{(k)}$ – приближение решения на *о*й итерации.

Выражение (3.2) означает, что на каждой итерации должна решаться СЛАУ

$$\mathbf{M}\mathbf{x}^{(k+1)} = \mathbf{N}\mathbf{x}^{(k)} + \mathbf{b}, \qquad (3.4)$$

поэтому матрица $\mathbf{M}$ должна быть такой, чтобы это система легко решалась. Для быстрой сходимости желательно брать $\mathbf{M} \approx \mathbf{A}$ и $\mathbf{N} \approx 0$. Если взять $\mathbf{M} = \mathbf{A}$, то решение будет получено за одну итерацию. Но тогда, как видно из (3.4), итерационный процесс формально сведется к решению исходной системы. Поэтому так никогда не делают. В простейших итерационных методах $\mathbf{M}$ является единичной или диагональной матрицей.

В итерационных методах широко используется понятие *невязки* решения (*residual*)

$$\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}. \qquad (3.5)$$

Невязка – это разность между правой и левой частями СЛАУ при подстановке в систему приближенного решения. Невязка характеризует степень близости приближения к решению и связана с погрешностью решения. Действительно, $\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)} = \mathbf{A}\left(\mathbf{x}^{*} - \mathbf{x}^{(k)}\right)$, где $\mathbf{x}^{*}$ – точное решение системы. То есть ошибка решения после k итераций $\Delta\left(\mathbf{x}^{(k)}\right) = \mathbf{x}^{*} - \mathbf{x}^{(k)}$ формально может быть рассмотрена как решение системы $\mathbf{A}\Delta\left(\mathbf{x}^{(k)}\right) = \mathbf{r}^{(k)}$. Но оценка погрешности таким методом практически неприменима, так как столь же трудоемка как решение исходной задачи.

С использованием понятия невязки итерация (3.2) может быть описана в виде

$$\mathbf{x}^{(k+1)} = \mathbf{C}\mathbf{x}^{(k)} + \mathbf{d} = \left(\mathbf{E} - \mathbf{M}^{-1}\mathbf{A}\right)\mathbf{x}^{(k)} + \mathbf{M}^{-1}\mathbf{b} = \mathbf{x}^{(k)} + \mathbf{M}^{-1}\mathbf{r}^{(k)} = \mathbf{x}^{(k)} + \Delta\mathbf{x}^{(k)}, \qquad (3.6)$$

где $\Delta\mathbf{x}^{(k)}$ находится из решения СЛАУ $\mathbf{M}\Delta\mathbf{x}^{(k)} = \mathbf{r}^{(k)}$.

Итерационный процесс строится следующим образом. Исходя из вектора *начального приближения* $\mathbf{x}^{(0)}$, на первой итерации по формуле (3.2) вычисляется первое приближение $\mathbf{x}^{(1)}$. На второй итерации, исходя из первого приближения, вычисляется второе приближение $\mathbf{x}^{(2)}$ и т. д. Очень часто

принимают $\mathbf{x}^{(0)} = 0$. Если, исходя из особенностей задачи, удается определить более точное значение $\mathbf{x}^{(0)}$, то для достижения решения потребуется меньше итераций.

Итерационный процесс называется *сходящимся*, если при любом начальном приближении $\lim\limits_{k\to\infty} \mathbf{x}^{(k)} = \mathbf{x}^*$, где $\mathbf{x}^*$ – точное решение системы (2.2). Это означает, что в сходящемся итерационном процессе точное решение достигается за бесконечно большое число итераций. Практически итерационный процесс останавливается после некоторого числа итераций, тем самым вносится *методическая погрешность*. Различные *критерии останова* итерационных процессов будут рассмотрены позже.

Рассмотрим *сходимость итерационного процесса*. После каждой итерации имеется ошибка $\mathbf{e}^{(k)} = \mathbf{x}^* - \mathbf{x}^{(k)}$ – разность между неизвестным точным решением $\mathbf{x}^*$ и приближенным решением после k итераций. В сходящемся итерационном процессе $\mathbf{e}^{(k)} \to 0$ при $k \to \infty$. Выведем условия сходимости. Очевидно, что точное решение, удовлетворяющее системе $\mathbf{Ax} = \mathbf{b}$, удовлетворяет и системе

$$\mathbf{Mx}^* = \mathbf{Nx}^* + \mathbf{b}. \tag{3.7}$$

Вычитая из (3.7) выражение (3.4), получаем $\mathbf{Me}^{(k+1)} = \mathbf{Ne}^{(k)}$. Следовательно,

$$\mathbf{e}^{(k+1)} = \mathbf{Ce}^{(k)}, \tag{3.8}$$

где, как и ранее, $\mathbf{C} = \mathbf{M}^{-1}\mathbf{N} = \mathbf{E} - \mathbf{M}^{-1}\mathbf{A}$.

Так как (3.8) справедливо для всех k, то $\mathbf{e}^{(1)} = \mathbf{Ce}^{(0)}$, $\mathbf{e}^{(2)} = \mathbf{C}^2\mathbf{e}^{(0)}$, …, $\mathbf{e}^{(k)} = \mathbf{C}^k\mathbf{e}^{(0)}$. Независимо от начального приближения $\mathbf{e}^{(k)} \to 0$ при $k \to \infty$, если $\mathbf{C}^k \to 0$. Для этого необходимо и достаточно, чтобы все собственные значения итерирующей матрицы были по модулю меньше единицы [6]. Максимальный из модулей собственных значений матрицы A называется *спектральным радиусом* $\rho(\mathbf{A})$ матрицы $\mathbf{A}$. Тогда *необходимое и достаточное условие сходимости* имеет вид

$$\rho(\mathbf{C}) < 1. \tag{3.9}$$

Часто на практике применяют *достаточное условие сходимости* [6]: для сходимости итерационного процесса достаточно, чтобы какая-либо норма матрицы $\mathbf{C}$ была меньше единицы

$$\|\mathbf{C}\| < 1 \tag{3.10}$$

Известно, что собственное значение матрицы по модулю не превосходит любую из ее норм [6]. Тогда из (3.10) вытекает условие (3.9).

Доказывается [4], что для достаточно больших k $\left\|\mathbf{e}^{(k+1)}\right\| / \left\|\mathbf{e}^{(k)}\right\| \approx \rho(\mathbf{C})$, то есть сходимость линейна с коэффициентом сжатия $\rho(\mathbf{C})$. Чем меньше $\rho(\mathbf{C})$, тем быстрее сходятся итерации. Рассмотренные соображения учитываются в *асимптотической скорости сходимости* итерационного процесса [4] $R(\mathbf{C}) = -\ln \rho$, где ρ – спектральный радиус итерирующей матрицы $\mathbf{C}$.

В случае сходящегося итерационного процесса при бесконечно большом числе итераций и при отсутствии погрешностей округлений приближенное решение $\mathbf{x}^{(k)}$ стремится к точному решению $\mathbf{x}^{*}$ системы $\mathbf{Ax} = \mathbf{b}$. Реально итерационный процесс всегда конечен, и нужно выбрать условие окончания итерационного процесса (*критерий останова*). Например, итерационный процесс можно закончить, если выполняется одно из условий

$$\left\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\right\| \le \varepsilon_1, \tag{3.11}$$

$$\left\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\right\| / \left\|\mathbf{x}^{(k)}\right\| \le \varepsilon_2, \tag{3.12}$$

$$\left\|\mathbf{r}^{(k)}\right\| \le \varepsilon_3, \tag{3.13}$$

$$\left\|\mathbf{r}^{(k)}\right\| / \left\|\mathbf{b}\right\| \le \varepsilon_4, \tag{3.14}$$

где ε_i, $i = 1, 2, 3, 4$ – заданное малое число, $\|\ldots\|$ – произвольная норма.

В условиях (3.11) – (3.14) удобно использовать *кубическую норму* векторов. Для проверки условия окончания итерационного процесса целесообразно применять условия (3.12) и (3.14), в которых используются относительные нормы, тем самым исключается зависимость от абсолютной величины решения. Однако малое значение ε в условиях (3.11) – (3.14), не гарантирует малой погрешности решения.

Итерационные методы предназначены для решения СЛАУ с разреженными матрицами. В библиотеках прикладных программ функции, реализующие итерационные методы, используют разреженный формат хранения матриц и производят вычисления только с ненулевыми элементами матриц. Одним из преимуществ *MATLAB* являются методы хранения и обработки разреженных матриц, позволяющие формально работать с разреженными матрицами как с обычными плотными, но сводящие к минимуму затраты памяти и времени на работу с нулевыми элементами матриц. Матричная форма упрощает запись и исследование итерационных методов. Поэтому будем использовать матричную форму записи итерационных методов, не учитывая форму хранения матриц.

Естественно, относительно несложные задачи можно решать итерационными методами, применяя полное хранение матриц. Более того,

итерационные методы можно применять для решения СЛАУ с плотными матрицами (если выполняются условия сходимости для этих матриц). Целесообразность применения итерационных методов в этих случаях должна быть обоснована.

Оценим погрешность, возникающую при замене бесконечного итерационного процесса конечным числом k итераций, т. е. рассмотрим *методическую погрешность итерационного процесса*. Погрешностями округления пренебрежем. Так как $\mathbf{A}\mathbf{x}^* = \mathbf{b}$, где $\mathbf{x}^*$ – точное решение СЛАУ (2.2), то получаем

$$\mathbf{A}\left(\mathbf{x}^* - \mathbf{x}^{(k)}\right) = \mathbf{A}\Delta\mathbf{x}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)} = \mathbf{r}^{(k)}, \tag{3.15}$$

где $\Delta\mathbf{x}^{(k)}$ – погрешность решения после k итераций.

Из (3.15) получаем

$$\left\|\Delta\mathbf{x}^{(k)}\right\| \le \left\|\mathbf{A}^{-1}\right\| \cdot \left\|\mathbf{r}^{(k)}\right\|, \tag{3.16}$$

то есть при использовании условия (3.13) и большой норме обратной матрицы абсолютная погрешность решения будет большой даже при малой величине критерия останова ε. Из (3.16), а также очевидного неравенства $\|\mathbf{b}\| \le \|\mathbf{A}\| \cdot \|\mathbf{x}^*\|$ и условия (3.14) и $\mathbf{b} \ne \mathbf{0}$, $\mathbf{x}^* \ne \mathbf{0}$, получаем

$$\frac{\left\|\Delta\mathbf{x}^{(k)}\right\|}{\|\mathbf{x}^*\|} \le \mathrm{cond}(\mathbf{A}) \frac{\left\|\mathbf{r}^{(k)}\right\|}{\|\mathbf{b}\|} \le \mathrm{cond}(\mathbf{A}) \cdot \varepsilon,$$

т. е. для плохо обусловленной матрицы относительная погрешность решения может быть значительно больше ε. Чтобы гарантировать решение с относительной погрешностью $\left\|\Delta\mathbf{x}^{(k)}\right\| / \|\mathbf{x}^*\| \le \varepsilon$ вместо условия (3.14) следует использовать условие

$$\frac{\left\|\mathbf{r}^{(k)}\right\|}{\|\mathbf{b}\|} \le \frac{\varepsilon}{\mathrm{cond}(\mathbf{A})}, \tag{3.17}$$

которое означает, что для многих практических задач итерационный процесс необходимо заканчивать при чрезвычайно малых значениях относительной нормы невязки. Условие (3.17) справедливо для любого метода решения. В [7] даны условия окончания некоторых популярных итерационных процессов, гарантирующих заданную погрешность решения.

Еще более сложные оценки получаются при использовании в условиях окончания итерационного процесса поправок решения (3.11), (3.12). Так, из (3.6) получаем

$$\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} = \mathbf{M}^{-1}\mathbf{r}^{(k)},\ \mathbf{r}^{(k)} = \mathbf{M}\left(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right). \qquad (3.18)$$

Из (3.15) и (3.18) получаем

$$\Delta\mathbf{x}^{(k)} = \mathbf{A}^{-1}\mathbf{M}\left(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right)$$

и с учетом условия (3.12) получаем

$$\frac{\left\|\Delta\mathbf{x}^{(k)}\right\|}{\left\|\mathbf{x}^{*}\right\|} \le \operatorname{cond}(\mathbf{A})\frac{\|\mathbf{M}\|}{\|\mathbf{b}\|}\varepsilon.$$

Погрешности округления в итерационных методах меньше сказываются на результате, чем в прямых методах. Сходящиеся итерационные процессы являются устойчивыми к погрешностям округления, т. е. не происходит их накопления. Однако это не означает, что, увеличивая число итераций можно достичь сколь угодно малой погрешности решения. Из-за погрешностей округления существует некоторая окрестность решения, после попадания в которую дальнейшего уточнения приближения решения $\mathbf{x}^{(k)}$ не происходит с ростом числа итераций.

Рассмотрение итерационных методов начнем с рассмотрения классических методов (методы Ричардсона, простой итерации, Зейделя, последовательной верхней релаксации). При рассмотрении будем придерживаться терминологии, принятой в [5].

3.2. Метод Ричардсона

В *методе Ричардсона* (*L. F. Richardson*) итерационный процесс решения СЛАУ (2.2) строится по формуле

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \mathbf{r}^{(k-1)} = \mathbf{x}^{(k-1)} + \left(\mathbf{b} - \mathbf{A}\mathbf{x}^{(k-1)}\right) = \left(\mathbf{E} - \mathbf{A}\right)\mathbf{x}^{(k-1)} + \mathbf{b}, \qquad (3.19)$$

где $\mathbf{r}^{(k-1)}$ – невязка (3.5).

Используя расщепление (3.1), видно, что расщепляющая матрица в методе Ричардсона равна $\mathbf{M} = \mathbf{E}$, а $\mathbf{N} = \mathbf{E} - \mathbf{A}$, где $\mathbf{E}$ – единичная матрица. Тогда итерирующая матрица равна

$$\mathbf{C} = \mathbf{E} - \mathbf{A}. \qquad (3.20)$$

В покомпонентной записи формула метода Ричардсона (3.19) имеет вид

$$x_i^{(k)} = x_i^{(k-1)} + r_i^{(k-1)}, \qquad i = 1, 2, \ldots, n, \qquad (3.21)$$

где невязка рассчитывается по формуле

$$r_i^{(k-1)} = b_i - \sum_{j=1}^{n} a_{ij} x_j^{(k-1)}, \qquad i = 1, 2, \ldots, n. \tag{3.22}$$

Метод Ричардсона реализуется на компьютере следующим образом. Исходными данными являются матрица $\mathbf{A}$, хранящаяся в виде двумерного массива или более сложной структуры данных в случае разреженной матрицы, и вектор правой части $\mathbf{b}$, хранящийся в виде одномерного массива. Необходимы также два массива для хранения векторов $\mathbf{x}^{(k-1)}$ и $\mathbf{x}^{(k)}$. В массиве для хранения $\mathbf{x}^{(k-1)}$ задается начальное приближение $\mathbf{x}^{(0)} = 0$. Можно задать $\mathbf{x}^{(0)} = \mathbf{b}$ — это экономит одну итерацию. Алгоритм решения состоит из ряда итераций. В начале каждой итерации рассчитывается невязка по формуле (3.22) и проверяется условие окончания итерационного процесса, например (3.14). Если условие окончания итерационного процесса выполняется, то процесс заканчивается, а $\mathbf{x}^{(k-1)}$ принимается за решение системы. В противном случае по формуле (3.21) рассчитывается вектор нового приближения $\mathbf{x}^{(k)}$. Вектор $\mathbf{x}^{(k)}$ записывается на место вектора $\mathbf{x}^{(k-1)}$ и происходит переход к следующей итерации, т. е. рассчитывается невязка и т. д.

Рассмотрим *условие сходимости* метода Ричардсона. Для расчета собственного значения итерирующей матрицы $\mathbf{C}$ используем *отношение Рэлея*

$$\lambda(\mathbf{C}) = \frac{(\mathbf{C}\mathbf{x}, \mathbf{x})}{(\mathbf{x}, \mathbf{x})},$$

тогда с учетом (3.20) получим

$$\lambda(\mathbf{C}) = \frac{((\mathbf{E} - \mathbf{A})\mathbf{x}, \mathbf{x})}{(\mathbf{x}, \mathbf{x})} = \frac{(\mathbf{x}, \mathbf{x}) - (\mathbf{A}\mathbf{x}, \mathbf{x})}{(\mathbf{x}, \mathbf{x})} = 1 - \lambda(\mathbf{A}).$$

Условие сходимости (3.9) примет вид

$$\rho(\mathbf{C}) = \max\left(\left|1 - \lambda_{\min}(\mathbf{A})\right|, \left|1 - \lambda_{\max}(\mathbf{A})\right|\right) < 1, \tag{3.23}$$

где $\lambda_{\min}(\mathbf{A})$ и $\lambda_{\max}(\mathbf{A})$ – минимальное и максимальное собственное значение матрицы $\mathbf{A}$.

Если $\mathbf{A}$ – симметричная и положительно определенная матрица, то условие сходимости имеет вид $\lambda_{\max}(\mathbf{A}) < 2$. Можно использовать достаточное условие сходимости $\|\mathbf{E} - \mathbf{A}\| < 1$, в случае положительно определенной матрицы $\mathbf{A}$ принимающее вид

$$\|\mathbf{A}\| < 2 \tag{3.24}$$

Очевидно, что условия (3.23) или (3.24) выполняются далеко не для всех матриц. Для обеспечения сходимости и повышения скорости сходимости

используют *итерации со сглаживанием*, включающие *итерационный параметр* (*множитель сглаживания*) τ

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \tau \mathbf{r}^{(k-1)}. \qquad (3.25)$$

Тогда расщепляющая матрица $\mathbf{M} = \frac{1}{\tau}\mathbf{E}$, а $\mathbf{N} = \frac{1}{\tau}\mathbf{E} - \mathbf{A}$. Итерирующая матрица имеет вид $\mathbf{C} = \mathbf{E} - \tau\mathbf{A}$. В общем виде расщепляющая матрица со сглаживанием равна $\mathbf{M} = \tau^{-1}\mathbf{M}$, а итерирующая матрица равна $\mathbf{C} = \mathbf{E} - \tau\mathbf{M}^{-1}\mathbf{A}$. Практически значение τ находят подбором. Для положительно определенных матриц оптимальное значение τ равно [5]

$$\tau = \frac{2}{\lambda_{\min}(\mathbf{A}) + \lambda_{\max}(\mathbf{A})}. \qquad (3.26)$$

Из-за сложности вычисления собственных чисел эта оценка имеет скорее теоретическое значение.

Метод Ричардсона обладает низкой скоростью сходимости, но его можно рассматривать как хорошую основу построения других методов.

3.3. Методы простой итерации и Якоби

В *методе простой итерации* [7] систему $\mathbf{Ax} = \mathbf{b}$ предварительно приводят к виду, удобному для итераций

$$\mathbf{x} = \mathbf{Bx} + \mathbf{c},$$

где $\mathbf{x}$ – искомый вектор, $\mathbf{B}$ и $\mathbf{c}$ – новые матрица и вектор.
Итерационный процесс строится по формуле

$$\mathbf{x}^{(k)} = \mathbf{Bx}^{(k-1)} + \mathbf{c}, \quad k = 1, 2, \ldots \quad . \qquad (3.27)$$

Для сходимости процесса (3.27) для итериирующей матрицы $\mathbf{B}$ должно выполняться условие (3.9) или (3.10).

Например, преобразуя СЛАУ $\mathbf{Ax} = \mathbf{b}$, получим $\mathbf{x} + \tau\mathbf{Ax} = \mathbf{x} + \tau\mathbf{b}$, откуда получаем метод Ричардсона с итерационным параметром $\mathbf{x}^{(k)} = \left(\mathbf{E} - \tau\mathbf{A}\right)\mathbf{x}^{(k-1)} + \tau\mathbf{b}$. То есть метод Ричардсона можно рассматривать как метод простой итерации с $\mathbf{B} = \mathbf{E} - \tau\mathbf{A}$, и $\mathbf{c} = \tau\mathbf{b}$, где $\mathbf{E}$ — единичная матрица.

Приведение системы к виду, удобному для итераций не является простой задачей. Самый простой прием состоит в следующем. Предполагая, что матрица $\mathbf{A}$ невырожденная и все $a_{ii} \neq 0$ (в противном случае можно переставить строки матрицы), преобразуем систему (2.1) к виду

$$
\begin{aligned}
x_1 &= -\frac{a_{12}}{a_{11}}x_2 - \frac{a_{13}}{a_{11}}x_3 - \ldots - \frac{a_{1n}}{a_{11}}x_n + \frac{b_1}{a_{11}},\\
x_2 &= -\frac{a_{21}}{a_{22}}x_1 - \frac{a_{23}}{a_{22}}x_3 - \ldots - \frac{a_{2n}}{a_{22}}x_n + \frac{b_2}{a_{22}},\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
x_n &= -\frac{a_{n1}}{a_{nn}}x_1 - \frac{a_{n2}}{a_{nn}}x_2 - \ldots - \frac{a_{n,n-1}}{a_{nn}}x_{n-1} + \frac{b_n}{a_{nn}},
\end{aligned}
\qquad (3.28)
$$

или

$$
x_i = -\sum_{\substack{j=1\\ j\neq i}}^{n}\frac{a_{ij}}{a_{ii}}x_j + \frac{b_i}{a_{ii}}, \quad i = 1,2,\ldots,n.
$$

В *методе Якоби* (*C. Jacobi*) компоненты вектора $\mathbf{x}^{(k)}$ находятся по формуле

$$
x_i^{(k)} = -\sum_{\substack{j=1\\ j\neq i}}^{n}\frac{a_{ij}}{a_{ii}}x_j^{(k-1)} + \frac{b_i}{a_{ii}} = \frac{1}{a_{ii}}\left(-\sum_{\substack{j=1\\ j\neq i}}^{n}a_{ij}x_j^{(k-1)} + b_i\right), \quad i = 1, 2, \ldots, n. \qquad (3.29)
$$

Часто (см. [6]) методы простой итерации и Якоби считаются синонимами.

Реализация метода Якоби очень похожа на реализацию метода Ричардсона. В методе Якоби также необходимо хранить векторы $\mathbf{x}^{(k-1)}$ и $\mathbf{x}^{(k)}$, однако метод Якоби в явном виде не требует расчета невязки и в качестве условия окончания итерационного процесса часто используют условия (3.11) или (3.12).

Для исследования сходимости метода Якоби используем расщепление, для чего представим матрицу $\mathbf{A}$ в виде

$$
\mathbf{A} = \mathbf{M} - \mathbf{N} = \mathbf{D} - (\mathbf{C}_L + \mathbf{C}_U), \qquad (3.30)
$$

где $\mathbf{D} = [a_{ii}]$ – диагональная матрица;

$$
\mathbf{C}_L = -\begin{bmatrix} 0 & 0 & 0 & \ldots & 0\\ a_{21} & 0 & 0 & \ldots & 0\\ a_{31} & a_{32} & 0 & \ldots & 0\\ \ldots & \ldots & \ldots & \ldots & \ldots\\ a_{n1} & a_{n2} & a_{n3} & \ldots & 0 \end{bmatrix}, \quad \mathbf{C}_U = -\begin{bmatrix} 0 & a_{12} & a_{13} & a_{14} & \ldots & a_{1n}\\ 0 & 0 & a_{23} & a_{24} & \ldots & a_{2n}\\ 0 & 0 & 0 & a_{34} & \ldots & a_{3n}\\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots\\ 0 & 0 & 0 & 0 & \ldots & 0 \end{bmatrix}. \qquad (3.31)
$$

То есть расщепляющая матрица равна $\mathbf{M} = \mathbf{D}$, а матрица $\mathbf{N}$ равна $\mathbf{N} = (\mathbf{C}_L + \mathbf{C}_U)$. Исходя из (3.30) получаем $\mathbf{N} = (\mathbf{C}_L + \mathbf{C}_U) = \mathbf{D} - \mathbf{A}$, а из (3.2) получаем запись метода Якоби в матричной форме

$$
\begin{aligned}
\mathbf{x}^{(k)} &= \mathbf{M}^{-1}\mathbf{N}\mathbf{x}^{(k-1)} + \mathbf{M}^{-1}\mathbf{b} = \mathbf{D}^{-1}\left(\mathbf{C}_L + \mathbf{C}_U\right)\mathbf{x}^{(k-1)} + \mathbf{D}^{-1}\mathbf{b} =\\
&= \left(\mathbf{E} - \mathbf{D}^{-1}\mathbf{A}\right)\mathbf{x}^{(k-1)} + \mathbf{D}^{-1}\mathbf{b},
\end{aligned}
\qquad (3.32)
$$

тогда итерирующая матрица равна

$$\mathbf{C} = \mathbf{D}^{-1}(\mathbf{C}_L + \mathbf{C}_U) = \mathbf{E} - \mathbf{D}^{-1}\mathbf{A}. \qquad (3.33)$$

Отметим, что запись (3.32) нужна только для теоретического исследования, реализуется метод Якоби с использованием покомпонентной формулы (3.29).

Для сходимости метода Якоби должны выполняться условия (3.9) – (3.10) для матрицы перехода (3.33). *Достаточным условием сходимости* метода Якоби является диагональное преобладание матрицы $\mathbf{A}$

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|, \qquad i = 1, 2, \ldots, n.$$

Из (3.32) видно, что метод Якоби совпадает с методом Ричардсона, примененным к преобразованной (*предобусловленной*) системе

$$\mathbf{D}^{-1}\mathbf{A}\mathbf{x} = \mathbf{D}^{-1}\mathbf{b}. \qquad (3.34)$$

Преобразование (3.34) заключается в делении каждого уравнения СЛАУ (2.2) на $a_{ii} \neq 0$. Для (3.34) формула метода Ричардсона имеет вид

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + (\mathbf{D}^{-1}\mathbf{b} - \mathbf{D}^{-1}\mathbf{A}\mathbf{x}^{(k-1)}) = \left(\mathbf{E} - \mathbf{D}^{-1}\mathbf{A}\right)\mathbf{x}^{(k-1)} + \mathbf{D}^{-1}\mathbf{b}, \qquad (3.35)$$

т.е. (3.35) совпадает с (3.32). Решая преобразованную систему (3.34) методом Ричардсона, можно ввести *итерационный параметр* τ, что повысит скорость сходимости.

3.4. Методы Зейделя и последовательной верхней релаксации

3.4.1. Метод Зейделя

Метод Зейделя (*P. L. von Seidel*), или *метод Гаусса-Зейделя*, как и метод Якоби, основан на преобразовании (3.28), однако при вычислении $x_i^{(k)}$ при $i > 1$ используются уже найденные приближения $x_1^{(k)}, x_2^{(k)}, \ldots, x_{i-1}^{(k)}$, т. е.

$$x_1^{(k)} = -\frac{a_{12}}{a_{11}} x_2^{(k-1)} - \frac{a_{13}}{a_{11}} x_3^{(k-1)} - \ldots - \frac{a_{1n}}{a_{11}} x_n^{(k-1)} + \frac{b_1}{a_{11}},$$

$$x_2^{(k)} = -\frac{a_{21}}{a_{22}} x_1^{(k)} - \frac{a_{23}}{a_{22}} x_3^{(k-1)} - \ldots - \frac{a_{2n}}{a_{22}} x_n^{(k-1)} + \frac{b_2}{a_{22}},$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$x_n^{(k)} = -\frac{a_{n1}}{a_{nn}} x_1^{(k)} - \frac{a_{n2}}{a_{nn}} x_2^{(k)} - \ldots - \frac{a_{n,n-1}}{a_{nn}} x_{n-1}^{(k)} + \frac{b_n}{a_{nn}}.$$

Метод Зейделя легко программируется. Для его реализации необходим один массив для хранения вектора $\mathbf{x}$. Вначале в этот массив помещаются

значения начального приближения. В каждой итерации вычисления сводятся к расчету x_i, $i = 1, 2, ..., n$ по формуле

$$x_i = \frac{1}{a_{ii}} \left(-\sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j + b_i \right) \tag{3.36}$$

и к записи x_i на старое место в массиве $\mathbf{x}$. В качестве условия окончания итерационного процесса в методе Зейделя обычно используют условия (3.11) – (3.12). Для вычисления нормы поправки к решению $\left\| \mathbf{x}^{(k)} - \mathbf{x}^{(k-1)} \right\|$ до записи компонента нового приближения к решению необходимо вычислить компонент вектора поправки решения $\Delta x_i^{(k)} = x_i^{(k)} - x_i^{(k-1)}$.

Для исследования сходимости запишем метод Зейделя в *матричной форме*, используя представление $\mathbf{A} = \mathbf{D} - (\mathbf{C}_L + \mathbf{C}_U)$, где $\mathbf{D} = [a_{ii}]$ – диагональная матрица; $\mathbf{C}_L$ и $\mathbf{C}_U$ определяются (3.31). Тогда

$$\begin{aligned} \mathbf{x}^{(k)} &= \mathbf{D}^{-1}\mathbf{C}_L\mathbf{x}^{(k)} + \mathbf{D}^{-1}\mathbf{C}_U\mathbf{x}^{(k-1)} + \mathbf{D}^{-1}\mathbf{b} = \\ &= \mathbf{D}^{-1}\left(\mathbf{C}_L\mathbf{x}^{(k)} + \mathbf{C}_U\mathbf{x}^{(k-1)} + \mathbf{b}\right). \end{aligned} \tag{3.37}$$

откуда

$$\left(\mathbf{D} - \mathbf{C}_L\right)\mathbf{x}^{(k)} = \mathbf{C}_U\mathbf{x}^{(k-1)} + \mathbf{b},$$

т. е. матрица расщепления равна $\mathbf{M} = \mathbf{D} - \mathbf{C}_L$, а итерационный процесс в матричной форме имеет вид

$$\mathbf{x}^{(k)} = \left(\mathbf{D} - \mathbf{C}_L\right)^{-1}\mathbf{C}_U\mathbf{x}^{(k-1)} + \left(\mathbf{D} - \mathbf{C}_L\right)^{-1}\mathbf{b}. \tag{3.38}$$

Из (3.38) видно, что *сходимость метода Зейделя* определяется спектральным радиусом матрицы перехода $\mathbf{C} = \left(\mathbf{D} - \mathbf{C}_L\right)^{-1}\mathbf{C}_U$. Известно [11–12], что *метод Зейделя сходится в случае симметричной положительно определенной матрицы* $\mathbf{A}$ решаемой системы (2.2). Отметим, что запись (3.38) метода Зейделя в матричной форме *используется для исследования метода*. Программная реализация метода Зейделя строится на основе покомпонентной записи метода (3.36).

3.4.2. Метод последовательной верхней релаксации

Если матрица $\mathbf{A}$ решаемой системы СЛАУ симметрична и положительно определена, то можно использовать *метод последовательной верхней релаксации* [8], или SOR-метод (от англ. *successive over relaxation*). Это один из самых распространенных итерационных методов, что объясняется его высокой скоростью сходимости и простотой реализации. Релаксацией

называют процесс коррекции уравнения путем изменения одного из входящих в него неизвестных. В *SOR*-методе при вычислении $x_i^{(k+1)}$ по той же формуле (3.36), как и в методе Зейделя, находится промежуточное значение, которое обозначим как $x_i^{(k+1/2)}$. Затем находится очередное приближение решения

$$x_i^{(k+1)} = x_i^{(k)} + \omega\left(x_i^{(k+1/2)} - x_i^{(k)} \right), \tag{3.39}$$

где ω – коэффициент релаксации.

В остальном реализация *SOR*-метода не отличается от реализации метода Зейделя.

Для исследования метода запишем (3.39) в виде

$$x_i^{(k+1)} = \left(1-\omega\right) x_i^{(k)} + \omega x_i^{(k+1/2)}. \tag{3.40}$$

Так как $x_i^{(k+1/2)}$ вычисляется как в методе Зейделя, то используя (3.37), запишем *SOR*-метод в матричной форме

$$\mathbf{x}^{(k+1)} = (1-\omega)\mathbf{x}^{(k)} + \omega\mathbf{D}^{-1}\left(\mathbf{C}_L\mathbf{x}^{(k+1)} + \mathbf{C}_U\mathbf{x}^{(k)} + \mathbf{b}\right)$$

или

$$\left(\mathbf{E} - \omega\mathbf{D}^{-1}\mathbf{C}_L\right)\mathbf{x}^{(k+1)} = \left[(1-\omega)\mathbf{E} + \omega\mathbf{D}^{-1}\mathbf{C}_U\right]\mathbf{x}^{(k)} + \mathbf{D}^{-1}\mathbf{b}.$$

Умножая последнее выражение слева на $\mathbf{D}$ и проведя элементарные преобразования, получаем

$$\left(\frac{1}{\omega}\mathbf{D} - \mathbf{C}_L\right)\mathbf{x}^{(k+1)} = \left(\frac{1-\omega}{\omega}\mathbf{D} + \mathbf{C}_U\right)\mathbf{x}^{(k)} + \mathbf{b},$$

следовательно, матрица расщепления *SOR*-метода равна $\mathbf{M} = \frac{1}{\omega}\mathbf{D} - \mathbf{C}_L$ и $\mathbf{N} = \frac{1-\omega}{\omega}\mathbf{D} + \mathbf{C}_U$. Тогда матрица перехода *SOR*-метода согласно (3.3) равна

$$\mathbf{C}_\omega = \mathbf{M}^{-1}\mathbf{N} = \left(\frac{1}{\omega}\mathbf{D} - \mathbf{C}_L\right)^{-1}\left(\frac{1-\omega}{\omega}\mathbf{D} + \mathbf{C}_U\right) \tag{3.41}$$

SOR-метод *сходится*, если спектральный радиус матрицы $\mathbf{C}$ (3.41) меньше единицы. Известно [3, 5], что неравенство для спектрального радиуса матрицы $\mathbf{C}_\omega$ выполняется неравенство: $\rho\left(\mathbf{C}_\omega\right) \geq \left|\omega - 1\right|$. Тогда для симметричной положительно определенной матрицы $\mathbf{A}$ *SOR*-метод сходится, если $0 < \omega < 2$. Как правило, коэффициент релаксации берется из диапазона $1 < \omega < 2$, что соответствует *верхней релаксации*. При $\omega = 1$ *SOR*-метод переходит в метод Зейделя. При $0 < \omega < 1$ метод называется *методом нижней релаксации.* Обычно значение коэффициента релаксации ω подбирают экспериментально, при этом следует учитывать, что для целых классов задач

оптимальные (т. е. обеспечивающие минимальное число итераций) значения ω мало отличаются друг от друга.

Как и в методе Зейделя, матричное представление *SOR*-метода используется только для теоретических исследований. Расчеты ведутся по формуле (3.39).

3.5. Методы спуска

Эти методы называют методами вариационного типа, потому что определение оптимальных итерационных параметров в этих методах эквивалентно минимизации некоторого квадратичного функционала [9].

Пусть решается система (2.2) с симметричной положительно определенной матрицей $\mathbf{A}$. Можно доказать, что решение этой системы может быть сведено к минимизации функционала

$$J(\mathbf{y}) = \frac{1}{2}\mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{y} - \mathbf{y}^{\mathrm{T}}\mathbf{b}, \qquad (3.42)$$

то есть вектор, минимизирующий (3.42), будет точным решением системы $\mathbf{A}\mathbf{x} = \mathbf{b}$. Для доказательства преобразуем (3.42) с учетом очевидных равенств: $\mathbf{A}\mathbf{x} = \mathbf{b}$, $\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{y} = \mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{x}$

$$\begin{aligned} J(\mathbf{y}) &= \frac{1}{2}\mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{y} - \mathbf{y}^{\mathrm{T}}\mathbf{b} = \frac{1}{2}\mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{y} - \mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{x} + \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} - \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} = \\ &= \left(\frac{1}{2}\mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{y} - \frac{1}{2}\mathbf{y}^{\mathrm{T}}\mathbf{A}\mathbf{x} - \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{y} + \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}\right) - \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} = \\ &= \frac{1}{2}(\mathbf{y}-\mathbf{x})^{\mathrm{T}}\mathbf{A}(\mathbf{y}-\mathbf{x}) - \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}. \end{aligned} \qquad (3.43)$$

Так как член $\frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}$ не зависит от $\mathbf{y}$, то значение $J(\mathbf{y})$ будет минимально, если минимален член $\frac{1}{2}(\mathbf{y}-\mathbf{x})^{\mathrm{T}}\mathbf{A}(\mathbf{y}-\mathbf{x})$. В силу положительной определенности матрицы $\mathbf{A}$ функционал $J(\mathbf{y})$ принимает минимальное значение при $\mathbf{y} = \mathbf{x}$.

В методах спуска $\mathbf{A}\mathbf{x} = \mathbf{b}$ решается путем минимизации функционала $J(\mathbf{y})$. Минимизация производится итерационным путем: строится последовательность приближений $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots$ так, чтобы $J\left(\mathbf{x}^{(k+1)}\right) < J\left(\mathbf{x}^{(k)}\right)$. Если на некоторой итерации получается малое значение нормы невязки $\left\|\mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}\right\|$, то итерации прекращаются, и $\mathbf{x}^{(k)}$ принимается за решение.

Переход от $\mathbf{x}^{(k)}$ к $\mathbf{x}^{(k+1)}$ выполняется в два этапа: сначала происходит выбор вектора $\mathbf{p}^{(k)}$ направления перехода от $\mathbf{x}^{(k)}$ к $\mathbf{x}^{(k+1)}$, а затем $\mathbf{x}^{(k+1)}$ выбирается как точка на прямой $\mathbf{x}^{(k)} + \alpha\mathbf{p}^{(k)}$, где α – вещественное число. Таким образом

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{p}^{(k)}. \qquad (3.44)$$

В *методе скорейшего* (*наискорейшего*) *спуска* направление движения выбирается противоположным направлению градиента ∇J функционала (3.42), то есть движение производится в направлении *антиградиента*. Это объясняется тем, что функционал растет при движении в направлении градиента. Вычислим градиент функционала $\nabla J = \left[\frac{\partial J}{\partial y_1} \quad \frac{\partial J}{\partial y_2} \quad \cdots \quad \frac{\partial J}{\partial y_n} \right]^{\mathrm{T}}$. Для этого запишем функционал (3.42) в компонентной форме

$$J\left(y_1, y_2, ..., y_n\right) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} y_i y_j - \sum_{i=1}^{n} b_i y_i ,$$

где a_{ij} – элементы матрицы $\mathbf{A}$.

Вычислим компоненты градиента функционал ∇J

$$\frac{\partial J}{\partial y_m} = \sum_{j=1}^{n} a_{mj} y_j - b_m, \qquad m = 1, 2, ..., n. \qquad (3.45)$$

При получении выражения (3.45) учтено, что ненулевые производные дают только слагаемые вида $a_{jm} y_j y_m$, $a_{mj} y_m y_j$, $b_m y_m$ $\quad j = 1, 2, ..., m, ..., n$. Учтена также симметрия матрицы $\mathbf{A}$: $a_{jm} = a_{mj}$.

Матричная запись градиента с учетом (3.45) имеет вид

$$\nabla J\left(\mathbf{y}\right) = \mathbf{A}\mathbf{y} - \mathbf{b},$$

т. е. градиент функционала равен невязке с противоположным знаком. Таким образом, *в качестве направления движения может быть взят вектор невязки.*

Задачу определения итерационного параметра α_k представим как задачу *одномерной минимизации*. Рассмотрим функционал

$$\varphi(\alpha) = J\left(\mathbf{x}^{(k)} + \alpha\mathbf{p}^{(k)}\right), \qquad (3.46)$$

который представляет собой функционал (3.42) при изменении вектора решения в направлении антиградиента. Очевидно, что в качестве α_k необходимо выбрать то значение, для которого выполняется равенство $\varphi(\alpha_k) = \min_{\alpha} \varphi(\alpha)$. Преобразуем (3.46) с учетом (3.42)

$$\varphi(\alpha) = J\left(\mathbf{x}^{(k)} + \alpha\mathbf{p}^{(k)}\right) = \frac{1}{2}\left(\mathbf{x}^{(k)} + \alpha\mathbf{p}^{(k)}\right)^{\mathrm{T}} \mathbf{A}\left(\mathbf{x}^{(k)} + \alpha\mathbf{p}^{(k)}\right) - \left(\mathbf{x}^{(k)} + \alpha\mathbf{p}^{(k)}\right)^{\mathrm{T}} \mathbf{b} =$$

$$= \frac{1}{2}\mathbf{x}^{(k)\mathrm{T}}\mathbf{A}\mathbf{x}^{(k)} + \frac{\alpha}{2}\mathbf{p}^{(k)\mathrm{T}}\mathbf{A}\mathbf{x}^{(k)} + \frac{\alpha}{2}\mathbf{x}^{(k)\mathrm{T}}\mathbf{A}\mathbf{p}^{(k)} + \frac{\alpha^2}{2}\mathbf{p}^{(k)\mathrm{T}}\mathbf{A}\mathbf{p}^{(k)} - \tag{3.47}$$

$$- \mathbf{x}^{(k)\mathrm{T}}\mathbf{b} - \alpha\mathbf{p}^{(k)\mathrm{T}}\mathbf{b} = J\left(\mathbf{x}^{(k)}\right) - \alpha\mathbf{p}^{(k)\mathrm{T}}\mathbf{r}^{(k)} + \frac{\alpha^2}{2}\mathbf{p}^{(k)\mathrm{T}}\mathbf{A}\mathbf{p}^{(k)},$$

где $\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}$ – невязка на k-ой итерации.

Полученное выражение представляет собой квадратный полином относительно α и его единственный минимум может быть найден из условия минимума функционала $\varphi(\alpha)$

$$\frac{d\varphi(\alpha)}{d\alpha} = -\mathbf{p}^{(k)\mathrm{T}}\mathbf{r}^{(k)} + \alpha\mathbf{p}^{(k)\mathrm{T}}\mathbf{A}\mathbf{p}^{(k)} = 0,$$

откуда получаем

$$\alpha = \frac{\mathbf{p}^{(k)\mathrm{T}}\mathbf{r}^{(k)}}{\mathbf{p}^{(k)\mathrm{T}}\mathbf{A}\mathbf{p}^{(k)}} = \frac{\left(\mathbf{r}^{(k)}, \mathbf{p}^{(k)}\right)}{\left(\mathbf{A}\mathbf{p}^{(k)}, \mathbf{p}^{(k)}\right)}. \tag{3.48}$$

Так как $\mathbf{p}^{(k)} = \mathbf{r}^{(k)}$, то оптимальное значение итерационного параметра равно

$$\alpha_k = \frac{\left(\mathbf{r}^{(k)}, \mathbf{r}^{(k)}\right)}{\left(\mathbf{A}\mathbf{r}^{(k)}, \mathbf{r}^{(k)}\right)}. \tag{3.49}$$

Таким образом, вместо (3.44) получаем

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k\mathbf{r}^{(k)}, \tag{3.50}$$

где α_k вычисляется по (3.49).

Вычисляя невязку с использованием (3.50), получаем

$$\mathbf{r}^{(k+1)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k+1)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)} - \alpha_k\mathbf{A}\mathbf{r}^{(k)},$$

или

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha_k\mathbf{A}\mathbf{r}^{(k)}. \tag{3.51}$$

Выражение (3.51) позволяет вычислить невязку на каждой итерации без вычисления произведения $\mathbf{A}\mathbf{x}^{(k)}$.

Скорость сходимости метода скорейшего спуска относительно невысокая – примерно такая же, как и у метода Ричардсона с оптимальным значением итерационного параметра [7]. Но метод скорейшего спуска не требует сложного вычисления или подбора оптимального итерационного параметра.

Скорость сходимости метода скорейшего спуска падает с ухудшением обусловленности решаемой системы.

Кроме метода скорейшего спуска известны и другие методы спуска [7, 9]. Так в *методе минимальных невязок* спуск также происходит по направлению невязки, а итерационный параметр α_k выбирается из условия минимума нормы $\left\|\mathbf{r}^{(k+1)}\right\|$ при заданной норме $\left\|\mathbf{r}^{(k)}\right\|$. Это означает, что при фиксированной норме невязки $\left\|\mathbf{r}^{(k)}\right\|$ итерационный параметр обеспечит минимальную норму невязки на следующей итерации. При этом будем использовать норму вектора $\|\mathbf{v}\| = \sqrt{(\mathbf{v},\mathbf{v})}$

Рассмотрим итерационный процесс (3.50) решения СЛАУ с симметричной положительно определенной матрицей $\mathbf{A}$. Умножая слева (3.50) на $\mathbf{A}$ и вычитая каждую часть от $\mathbf{b}$, получим уравнение для невязки

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha_k \mathbf{A}\mathbf{r}^{(k)}. \qquad (3.52)$$

Выберем параметр α_k из условия минимума нормы $\left\|\mathbf{r}^{(k+1)}\right\|$. Вычисляя квадрат нормы обеих частей равенства (3.52) и учитывая принятое определение нормы вектора, получим

$$\left\|\mathbf{r}^{(k+1)}\right\|^2 = \left(\mathbf{r}^{(k+1)},\mathbf{r}^{(k+1)}\right) = \left(\mathbf{r}^{(k)} - \alpha_k \mathbf{A}\mathbf{r}^{(k)}, \mathbf{r}^{(k)} - \alpha_k \mathbf{A}\mathbf{r}^{(k)}\right) =$$

$$= \left(\mathbf{r}^{(k)},\mathbf{r}^{(k)}\right) - 2\alpha_k \left(\mathbf{A}\mathbf{r}^{(k)},\mathbf{r}^{(k)}\right) + \alpha_k^2 \left(\mathbf{A}\mathbf{r}^{(k)},\mathbf{A}\mathbf{r}^{(k)}\right).$$

Норма невязки $\left\|\mathbf{r}^{(k+1)}\right\|$ достигает своего минимума в точке α_k, в которой $\partial\left\|\mathbf{r}^{(k+1)}\right\|^2 \Big/ \partial\alpha_k = 0$. Эта производная равно нулю при

$$\alpha_k = \frac{\left(\mathbf{A}\mathbf{r}^{(k)},\mathbf{r}^{(k)}\right)}{\left(\mathbf{A}\mathbf{r}^{(k)},\mathbf{A}\mathbf{r}^{(k)}\right)}. \qquad (3.53)$$

Метод минимальных невязок сходится с той же скоростью, что и метод скорейшего спуска. При плохой обусловленности матрицы СЛАУ методу свойственны те же проблемы, что и методу скорейшего спуска.

Классические итерационные методы можно рассматривать как методы спуска [6]. *Метод Ричардсона* в простейшей форме (3.19) представляет метод спуска с постоянным параметром и далеко не всегда обеспечивает достижение минимума функционала (3.42). Метод Ричардсона с подбираемым параметром (3.25) реализует неточный поиск минимума в направлении спуска. Так как *метод Якоби* совпадает с методом Ричардсона, примененным к преобразованной (предобусловленной) системе (3.34), то он также является методом спуска. В *методах Зейделя* и *последовательной верхней релаксации* производится *координатный спуск*. Один шаг

координатного спуска – это спуск в направлении одного из стандартных единичных векторов $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n$. Одна итерация этих методов представляет собой n шагов координатного спуска (n – порядок СЛАУ). Рассмотренная геометрическая интерпретация метода спуска объясняет низкую скорость сходимости класссических итерационных методов в случае плохой обусловленности СЛАУ.

Библиографический список к главе 3

1. Саад Ю. Итерационные методы для разреженных систем. — В 2-х томах. Том. 1. — М.: Издательство Московского университета, 2013. — 344 с.

2. Горбаченко В. И. Вычислительная линейная алгебра с примерами на MATLAB. — СПб.: БХВ-Петербург, 2011. — 320 с.

3. Деммель Дж. Вычислительная линейная алгебра. Теория и приложения. — М.: Мир, 2001. — 430 с.

4. Уоткинс Д. С. Основы матричных вычислений. — М.: БИНОМ. Лаборатория знаний, 2006. — 664 с.

5. Хейгеман Л., Янг Д. Прикладные итерационные методы. — М.: Мир, 1986. — 448 с.

6. Фаддеев Д. К., Фаддева В. Н. Вычислительные методы линейной алгебры. — Спб.: "Лань", 2002. — 736 с.

7. Амосов А. А., Дубинский Ю. А., Копченова Н. В. Вычислительные методы. — М.: Издательский дом МЭИ, 2008. — 672 с.

8. Самарский А. А., Гулин А. В. Численные методы математической физики. — М.: "Научный мир", 2003. — 316 с.

9. Самарский А. А., Николаев Е. С. Методы решения сеточных уравнений. — М.: Наука, 1978.— 592 с.