математическая ЛОГИКА и теория алгоритмов для программистов

Д.В. ГРИНЧЕНКОВ, С.И. ПОТОЦКИЙ

УЧЕБНОЕ ПОСОБИЕ

КНОРУС

Д.В. Гринченков
С.И. Потоцкий

МАТЕМАТИЧЕСКАЯ ЛОГИКА И ТЕОРИЯ АЛГОРИТМОВ ДЛЯ ПРОГРАММИСТОВ

Допущено Министерством образования и науки
Российской Федерации
в качестве **учебного пособия**
для студентов высших учебных заведений,
обучающихся по специальности
«Программное обеспечение вычислительной техники
и автоматизированных систем» направления
подготовки «Информатика и вычислительная техника»

КноРус
МОСКВА
2010

УДК 510.5(075.8)
ББК 22.12я73
Г85

Рецензенты:

В.А. Волосухин, проректор по научной работе ФГОУ ВПО «Новочеркасская государственная мелиоративная академия», засл. деятель науки РФ, д-р техн. наук, проф.,

В.В. Изранцев, проректор по научной работе АНО «Международный банковский институт» (г. Санкт-Петербург), заведующий кафедрой прикладной информатики, д-р техн. наук, проф.

Гринченков Д.В.

Г85 Математическая логика и теория алгоритмов для программистов : учебное пособие / Д.В. Гринченков, С.И. Потоцкий. — М. : КНОРУС, 2010. — 208 с.

ISBN 978-5-406-00120-2

Пособие позволяет освоить основные положения и математические методы решения задач, представления знаний и построения доказательств в формальных системах, построения описания алгоритмов с использованием различных моделей, а также получить практические навыки по использованию методов математической логики и теории алгоритмов для решения практических задач и их программной реализации.

Для студентов вузов, обучающихся по специальностям 230105 «Программное обеспечение вычислительной техники и автоматизированных систем», 010503 «Математическое обеспечение и администрирование информационных систем» и специальностям направления «Информатика и вычислительная техника» дневной и заочной форм обучения.

УДК 510.5(075.8)
ББК 22.12я73

Гринченков Дмитрий Валерьевич
Потоцкий Сергей Иванович

МАТЕМАТИЧЕСКАЯ ЛОГИКА
И ТЕОРИЯ АЛГОРИТМОВ ДЛЯ ПРОГРАММИСТОВ

Санитарно-эпидемиологическое заключение
№ 77.99.60.953.Д.003365.04.09 от 01.04.2009 г.

Изд. № 1746. Подписано в печать 19.08.2009. Формат 60×90/16.
Гарнитура «NewtonC». Печать офсетная.
Усл. печ. л. 13,0. Уч.-изд. л. 8,3. Тираж 3000 экз. Заказ №
ООО «Издательство КноРус».
129110, Москва, ул. Большая Переяславская, 46, стр. 7.
Тел.: (495) 680-7254, 680-0671, 680-1278.
E-mail: office@knorus.ru http://www.knorus.ru

Отпечатано в ОАО «ИПК «Ульяновский Дом печати».
432980, г. Ульяновск, ул. Гончарова, 14.

ISBN 978-5-406-00120-2

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Математическая логика (ее называют также формальной логикой, теорией доказательств) изучает законы и формы корректных человеческих рассуждений.

Этот раздел математики имеет особое значение в изучении математических наук.

С одной стороны, предметом изучения математической логики является конкретная область знаний, связанная с расширением, развитием и формализацией положений и законов булевой алгебры. Положения этой теории лежат в основе таких направлений исследований, как функциональное и логическое программирование, системы искусственного интеллекта и др. С другой стороны, положения математической логики носят всеобщий характер, так как они определяют понятия и правила строгого выполнения логических доказательств. Строгое доказательство правильности тех или иных утверждений — это центральное звено любой математической теории.

Известно, что математика отличается от других наук тем, что использует доказательства, а не наблюдения. В физике одни законы выводятся из других, но окончательное подтверждение физического закона возможно только тогда, когда он подтвержден экспериментом. В математике же можно провести множество экспериментов, которые подтвердят некоторый факт, но математическим законом он будет признан только после того, как будет приведено строгое формальное доказательство. Поэтому можно сказать, что главная цель математической логики — дать точное и адекватное определение понятия «*математическое доказательство*».

Так как математика является наукой, в которой все утверждения доказываются с помощью умозаключений, то математическую логику можно рассматривать как инструмент для описания правил построения других математических теорий.

С точки зрения математической теории все знания в некоторой предметной области можно разделить на две составляющие: семантическую, которая непосредственно связана с изучаемым объектом и позволяет описывать его в терминах соответствующей области знаний, и синтаксическую, ее основу составляет набор правил, позволяющих осуществлять преобразование информации только в синтаксической форме, без учета содержания. Более того, одно и то же формальное описание может относиться к различным объектам реального мира.

Пример. Рассмотрим цепочку логических рассуждений:

Посылка 1: из *A* следует *B*.

Посылка 2: из *C* следует *A*.

Вывод: из C следует *B*.

Эта цепочка рассуждений может иметь практически любое содержание. Например:

1. Все слоны серые. Джамбо — слон. Следовательно, Джамбо серый.

2. Все студенты сдали сессию. Петров — студент. Следовательно, Петров сдал сессию и т. п.

Примером семантической теории является булева алгебра (алгебра высказываний). Одной из основных задач этой теории является установление значения истинности (или ложности) сложных (составных) высказываний и формирования в ее рамках средств для описания реальных логических устройств.

По отношению к булевой алгебре формальной синтаксической теорией является исчисление высказываний, с рассмотрения которого начинается настоящее пособие.

Математический подход к изучению какого-либо объекта или явления состоит в том, что вначале математик, изучая реальность, упорядочивает представление о ней в рамках семантической теории. Затем конструируется абстрактное представление об изучаемой предметной области на основе построения некоторой формальной системы. В рамках формальной системы производится доказательство теорем — истинных формул в данной теории. Далее происходит обратный переход к семантической части теории — возврат к реальной системе — и по отношению к ней осуществляется интерпретация теорем, полученных при формализации.

Считается, что первые работы по логике появились в V в. до н. э. Впервые как самостоятельную теорию ее оформил греческий философ Аристотель в своем труде «Аналитики», где систематизировал известные до него сведения, и эта система впоследствии стала называться *формальной логикой*. С этого времени формальная логика просуществовала без особых изменений почти двадцать столетий. Впоследствии возникла идея и о том, что, записав исходные рассуждения формулами, похожими на математические, можно заменить все цепочки логического вывода формальными «вычислениями». В Средние века даже делались попытки создания машин «логического вывода».

Развитие математики выявило недостатки логики, разработанной Аристотилем, и потребовало дальнейшего ее развития. Идея о по-

строении логики на математической основе была предложена немецким математиком Г. Лейбницем, который считал, что основные понятия логики возможно обозначить символами, соединяющимися по особым правилам, что позволит всякое рассуждение заменить вычислением.

Первая реализация идей Лейбница принадлежит английскому ученому Дж. Булю (середина XIX в.), создавшему алгебру, в которой буквами обозначены высказывания (повествовательные предложения, о которых можно сказать, что они истинны или ложны). Его алгебра получила название *алгебры высказываний*. Введение в логику символических обозначений послужило основой для создания новой науки — *математической логики*. Применение математики к логике позволило представить логические теории в новой удобной форме и использовать вычислительный аппарат в решении задач, ранее практически недоступных человеческому мышлению, что существенно расширило область логических исследований.

Особенности математического мышления объясняются особенностями математических абстракций и многообразием их взаимосвязей, которые отражаются в логической систематизации математики, в доказательстве математических теорем. Именно поэтому современную математическую логику определяют как *раздел математики, посвященный изучению математических доказательств и вопросов оснований математики*.

ГЛАВА 1

ТЕОРИЯ МНОЖЕСТВ

1.1. Основные понятия теории множеств

1.1.1. Множества, способы задания множеств

Под ***множеством*** понимают объединение в одно целое объектов, хорошо различимых человеческой интуицией или мыслью. Другими словами, ***множество*** — это совокупность объектов любой природы, рассматриваемая как единое целое. Обычно множества обозначают прописными латинскими буквами A, B, M, ...

Пример. $N = \{1, 2, 3, ...\}$ — множество натуральных чисел.

Объекты, образующие множество, называются ***элементами*** множества (обозначаются строчными буквами). Если элемент a входит в множество A, то обозначается это так: $a \in A$. Запись $a \notin A$ означает, что элемент a не принадлежит множеству A. Множество, содержащее конечное число элементов, называется ***конечным*** (в противном случае — ***бесконечным***). Если множество A конечно, то число его элементов называется ***мощностью*** множества и обозначается $|A|$. Если множество не содержит ни одного элемента, то оно называется ***пустым*** множеством и обозначается символом $\varnothing$. Множество A является подмножеством множества B, если любой элемент A принадлежит также множеству B. Это свойство обозначается $A \subseteq B$ (читается: B включает или равно A).

Равенство множеств. Множества A и B равны тогда и только тогда, когда их элементы совпадают. В этом случае пишут $A = B$. Так как при равенстве множеств A и B во множестве A нет элементов, не принадлежащих B, а в B нет элементов, не принадлежащих A, то признаком равенства множеств является одновременное выполнение двух условий: $A \subseteq B$ и $B \subseteq A$.

Если $A \subseteq B$ и $B \neq A$, то множество A называется ***собственным подмножеством*** множества B. Обозначается $A \subset B$ (строгое включение).

Одним из частных случаев является ситуация, когда элементами некоторого множества являются другие множества.

Например, пусть A — множество футболистов команды «Спартак», B — множество команд высшей лиги. Тогда $A \in B$.

Если в рамках некоторого класса задач рассматриваются различные множества, то полная совокупность всех элементов, из которых могут формироваться все множества и подмножества, образует универсальное множество — «универсум» или полное пространство. Обозначается универсальное множество символом U (генеральная совокупность).

Множество может быть задано:

1) перечислением всех его элементов. Например,
$A = \{a, b, c, d\}$, $B = \{0, 1, 3, 8, 9\}$;

2) порождающей процедурой. ***Порождающая процедура*** представляет собой правило получения элементов множества на основе уже имеющихся элементов либо из других объектов. Элементами множества считаются все объекты, которые получены с помощью этой процедуры;

3) описанием характеристик и свойств, которыми обладают все элементы множества. Например, $A = \{x | x \in N, x \le 100\}$.

1.1.2. Основные операции над множествами и их свойства

1. ***Объединение множеств*** $(A \cup B)$ — это множество, состоящее из тех элементов, которые принадлежат хотя бы одному из исходных множеств: $A \cup B = \{x | x \in A \text{ или } x \in B\}$.

Пример. $A = \{a, b, c\}$, $B = \{b, c, d, m\}$, $A \cup B = \{a, b, c, d, m\}$.

Операции над множествами удобно представлять с помощью диаграммы Венна — замкнутой линии, ограничивающей элементы одного множества. Диаграмма для операции объединения представлена на рис. 1.1.

2. ***Пересечение множеств*** $(A \cap B)$ — это множество тех и только тех элементов, которые принадлежат и множеству A, и множеству B (рис. 1.2): $A \cap B = \{x | x \in A \text{ и } x \in B\}$.

Пример. $A = \{a, b, c\}$, $B = \{b, c, d, m\}$, $A \cap B = \{b, c\}$.

3. ***Разность множеств*** $(A \setminus B)$ — это множество, состоящее из тех и только тех элементов множества A, которые не содержатся в множестве B (рис. 1.3): $A \setminus B = \{x | x \in A \text{ и } x \notin B\}$. Если $A \setminus B = \varnothing$, то $A \subseteq B$ (рис. 1.4).

4. ***Симметричная разность*** $(A \div B)$ — это множество элементов, принадлежащих множествам A или B за исключением их общих элементов (рис. 1.5): $A \div B = \{x \mid (x \in A \text{ и } x \notin B) \text{ или } (x \in B \text{ и } x \notin A)\}$.

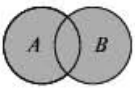

Рис. 1.1

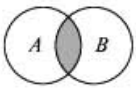

Рис. 1.2

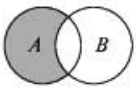

Рис. 1.3

Рис. 1.4

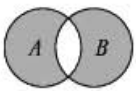

Рис. 1.5

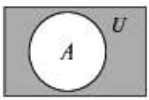

Рис. 1.6

5. ***Дополнением*** множества A до множества U (обозначается $\overline{A}$) называется множество всех элементов U, не принадлежащих множеству A (рис. 1.6).

Основные свойства операций над множествами. Для всех множеств A, B, C и универсального множества U справедливы следующие равенства:

1) $A \cup B = B \cup A$;
2) $A \cup (B \cup C) = (A \cup B) \cup C$;
3) $A \cap B = B \cap A$;
4) $A \cap (B \cap C) = (A \cap B) \cap C$;
5) $A \cap A = A$;
6) $A \cup A = A$;
7) $A \cup \varnothing = A$;
8) $A \cap \varnothing = \varnothing$;
9) $A \cup U = U$;
10) $A \cap U = A$;
11) $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$;
12) $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$;
13) $\overline{\overline{A}} = A$;
14) $\overline{\varnothing} = U$;
15) $\overline{U} = \varnothing$;
16) $A \cup \overline{A} = U$;
17) $A \cap \overline{A} = \varnothing$;
18) $\overline{A \cup B} = \overline{A} \cap \overline{B}$;
19) $\overline{A \cap B} = \overline{A} \cup \overline{B}$.

1.2. Прямое произведение множеств

Вектором (*кортежем*) называется упорядоченный набор элементов. Элементы, образующие вектор, называются ***координатами*** или ***компонентами***. Координаты нумеруются слева направо. Число координат называется ***длиной*** или ***размерностью*** вектора. В отличие

от элементов множества координаты вектора могут совпадать. Обозначение вектора: (a,b,c), где a,b,c — координаты вектора. ***Два вектора равны***, если они имеют одинаковую длину и равны их соответствующие координаты.

Прямым или ***декартовым произведением*** множеств A и B (обозначается $A\times B$) называется множество всех упорядоченных пар (a,b) таких, что $a\in A$ и $b\in B$, т. е. $A\times B=\{(a,b)|a\in A;b\in B\}$.

Частный случай: $A=B$. Тогда $A\times B=A\times A$. $A\times A$ обозначается A^2.

Прямым произведением $A_1\times A_2\times A_3\times\ldots\times A_n$ называется множество всех векторов $(a_1,a_2,a_3,\ldots,a_n)$ длины n таких, что $a_1\in A_1;a_2\in A_2;$ $\ldots;\ a_n\in A_n$.

Пример.

1) $A_1=\{0,1\}$, $A_2=\{x,y,z\}$;

$A_1\times A_2=\{(0,x),(0,y),(0,z),(1,x),(1,y),(1,z)\}$;

2) $A=\{a,b,c,d,e,f,g,h\}$; $B=\{1,2,3,4,5,6,7,8\}$;

$A\times B=\{(a,1),(a,2),\ldots,(h,7),(h,8)\}$.

Пусть имеется множество A, элементы которого являются символами (буквы, цифры, знаки). Такое множество называется ***алфавитом***. Элементы множества A^n — слова. Множество $R\times R=R^2$ — это множество точек (пар координат) плоскости (здесь R — множество всех действительных чисел).

Теорема 1.1. Пусть $A_1,A_2,\ldots,A_n$ — конечные множества и их мощности известны: $|A_1|=m_1,|A_2|=m_2,\ldots,|A_n|=m_n.$. Тогда

$|A_1\times A_2\times\ldots\times A_n|=m_1\cdot m_2\cdot\ldots\cdot m_n$. Частный случай: $|A^n|=|A|^n$.

Проекцией вектора $A=(a_1,a_2,\ldots,a_n)$ на ось i (обозначается пр_iA) называется его компонента a_i. ***Проекцией вектора*** A на оси $i_1\ldots i_k$ (пр_iA) называется вектор $(a_{i_1},a_{i_2},\ldots,a_{i_k})$ длины k.

Если V — множество векторов A_j одинаковой длины, то проекцией V на *i-ю* ось называется *множество проекций* всех A_j *на эту ось*:

$$\text{пр}_iV=\{\text{пр}_iA_j|A_j\in V\}.$$

ГЛАВА 2

ОСНОВНЫЕ ПОЛОЖЕНИЯ БУЛЕВОЙ АЛГЕБРЫ

2.1. Булева алгебра и ее применение

2.1.1. Определение булевой алгебры

Название этого раздела математики связано с именем его основателя — Джорджа Буля (1815—1864). Используя классическое понятие алгебры, булеву алгебру можно определить как систему $A = (B, \varphi_1, \varphi_2, ..., \varphi_n)$, в которой несущим множеством является двухэлементное множество двоичных чисел $B = \{0,1\}$, а $\Omega = \{\varphi_1, \varphi_2, ..., \varphi_n\}$ — заданные на этом множестве логические операции, сущность которых раскрывается ниже (см. разд. 2.2.2).

Основные логические операции — дизъюнкцию, конъюнкцию и отрицание — можно интерпретировать как операции, введенные в теорию множеств: свойства указанных операций аналогичны свойствам операций объединения, пересечения и дополнения множеств соответственно (см. разд. 1.1.2). Однако логические операции имеют несколько иной смысл; они позволяют формировать простые и сложные высказывания. Все множество логических операций обозначается E_2.

Как правило, существует логическая интерпретация элементов множества B: 1 — истинно; 0 — ложно. В ряде случаев в качестве элемента множества рассматривается двоичная переменная (ее называют также логической или булевой переменной) x, которая принимает значения $x = 0$ или $x = 1$.

2.1.2. Области применения булевой алгебры

Булева алгебра применяется как средство:

1) алгоритмического описания в языках программирования для определения логических условий;

2) формирования логических высказываний в математической логике, лингвистике, теории искусственного интеллекта;

3) разработки и описания дискретных технических систем.

Алгебра логики позволяет производить анализ и синтез логических устройств. *Анализ* — это поиск аналитического выражения, которое описывает работу системы. *Синтез* — обратная задача: создание технического устройства на основе математического описания средствами булевой алгебры.

2.1.3. Высказывания

Одним из базовых в булевой алгебре является понятие высказывания.

Высказывание — это любое повествовательное предложение, в отношении которого имеет смысл утверждение о его истинности или ложности. Обычно высказывания обозначаются буквами латинского алфавита: a, b, c, A, B, C. Для каждого высказывания вводится значение ***истинности***, которое может принимать одно из двух возможных значений: 1 — истина, 0 — ложь.

Пример. Рассмотрим справедливость утверждений:

a — число 4 — четное;

b — снег — красный;

c — $2 \times 2 = 5$.

Значения истинности данных высказываний следующие: $a = 1$, $b = 0$, $c = 0$.

Два высказывания A и B называются ***эквивалентными***, если их значения истинности совпадают. Значение истинности может быть постоянным либо изменяется в зависимости от обстоятельств. Изменяемое высказывание может рассматриваться как переменный параметр — двоичная переменная, принимающая одно из двух значений (обозначается x, y, z).

2.2. Функции алгебры логики

2.2.1. Понятие функции и способы ее задания

Пусть имеется n двоичных переменных $x_1, x_2, ..., x_n$. Каждая из них в некотором конкретном случае может принимать значение 0 или 1.

Полученный набор элементов есть двоичный вектор длины n. Каждому конкретному набору можно поставить в соответствие одно из значений — 0 или 1.

Функция f, задающая однозначное отображение множества всевозможных наборов значений двоичных переменных $x_1, x_2, ..., x_n$ во множество $\{0,1\}$, называется ***функцией алгебры логики*** (или логической функцией, булевой функцией, переключательной функцией):

$$f(x_1, x_2, ..., x_n) = y(y = 0 \text{ или } y = 1).$$

Таким образом, ***логическая функция*** — это зависимость, которая устанавливает связь между сочетанием значений входных двоичных переменных и двоичным значением этой функции.

Способы задания функции. Логическая функция может быть задана:

1) математическим выражением (формулой);
2) таблицей.

Таблица является наиболее общим и универсальным способом задания функции. В ее левой части перечисляют всевозможные наборы значений двоичных переменных, а в правой — значения функции на этих наборах.

Такие таблицы, описывающие функции, называют ***таблицами истинности***. В табл. 2.1 и 2.2 приведены примеры таблиц истинности.

Таблица 2.1

x_1	x_2	...	x_n	$f(x_1, x_2, ..., x_n)$
0	0		0	$f(0, 0, ..., 0)$
1	0		1	$f(1, 0, ..., 1)$
...	...	...	...	...
1	1		1	$f(1, 1, ..., 1)$

Таблица 2.2

x_1	x_2	x_3	$y = f(x_1, x_2, x_3)$
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Оценим число возможных наборов (число строк входных переменных).

Конкретный набор — это вектор значений $(b_1, b_2, ..., b_n)$. Количество наборов — это мощность прямого произведения n двухэлементных множеств B: $a = |B^n| = |B|^n = 2^n$, где n — число входных элементов.

Оценим возможное количество вариантов логических функций от n переменных. Множество вариантов логической функции можно представить как прямое произведение $M = B_1 \times B_2 \times ... \times B_i \times ... \times B_a = B^a$, где B_i — значение функции на наборе i.

Таким образом, общее количество функций от n переменных $m = |B^a| = |B|^a = 2^a$, где $a = 2^n$.

Наборы, на которых функция равна единице, называют *единичными* наборами, а наборы, на которых функция равна нулю, называют *нулевыми*.

Тождественно истинные, тождественно ложные, частично определенные функции. Если функция при любых значениях аргументов принимает значение 0, то такую функцию называют *нулевой* или *константой 0* (тождественно ложная функция).

Функция, которая на всех наборах равна 1, называется *единичной* или *константой 1* (тождественно истинная функция).

Если функция определена не на всех наборах аргументов, то она называется *неполностью определенной* или *частично определенной*.

Две булевы функции $f(x_1, x_2, ..., x_n)$ и $\varphi(x_1, x_2, ..., x_n)$ называют равными, если для всех возможных наборов значений аргументов они принимают одинаковые значения и, таким образом, имеют одну и ту же таблицу истинности. Равные булевы функции записывают: $f(x_1, x_2, ..., x_n) = \varphi(x_1, x_2, ..., x_n)$.

Говорят, что булева функция $f(x_1, x_2, ..., x_n)$ существенно зависит от аргумента x_i, если

$$f(x_1, x_2, ..., x_{i-1}, 0, x_{i+1}, ..., x_n) \neq f(x_1, x_2, ..., x_{i-1}, 1, x_{i+1}, ..., x_n)$$

хотя бы для одного набора остальных аргументов. В противном случае x_i называется *несущественной* или *фиктивной* переменной (ее можно исключить).

2.2.2. Элементарные логические операции

Из множества логических функций выделяется ряд наиболее простых операций, которые имеют ясную логическую интерпретацию:

1) ***отрицание*** (инверсия)

$$f_1(x) = \bar{x} \text{ (читается: не } x\text{)}.$$

Отрицание — это функция, выражающая высказывание, которое истинно, если высказывание x ложно, и ложно, если x истинно (табл. 2.3);

2) ***дизъюнкция*** (логическое сложение)

$$f_2(x,y)=x\vee y \text{ (читается: } x \text{ или } y\text{).}$$

Дизъюнкция — это функция, выражающая высказывание, которое истинно тогда и только тогда, когда по крайней мере одно из высказываний — x или y — является истинным (табл. 2.4);

3) ***конъюнкция*** (логическое умножение)

$$f_3(x,y)=x\wedge y \text{ (читается: } x \text{ и } y\text{).}$$

Для этой операции применяется также форма записи

$$f_3(x,y)=x\,\&\,y=x\cdot y=xy.$$

Конъюнкция — это функция, выражающая высказывание, которое истинно только в том случае, если истинны оба высказывания — x и y (табл. 2.5);

4) ***импликация***

$$f_4(x,y)=x\to y \text{ (читается: если } x\text{, то } y\text{).}$$

Функция f_4 принимает значение ложно только тогда, когда x истинно, а y ложно (табл. 2.6);

Таблица 2.3

x	$\bar{x}$
0	1
1	0

Таблица 2.4

x_1	x_2	$x_1\vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

Таблица 2.5

x_1	x_2	$x_1\wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

Таблица 2.6

x_1	x_2	$x_1\to x_2$
0	0	1
0	1	1
1	0	0
1	1	1

5) ***эквивалентность*** (равнозначность)

$$f_5(x,y)=x\sim y \text{ (читается: } x \text{ равно } y\text{).}$$

Функция $f_5=1$ тогда и только тогда, когда значения обоих аргументов совпадают (табл. 2.7);

6) ***сложение по модулю два*** (неравнозначность)

$$f_6(x,y)=x\oplus y.$$

Функция f_6 истинна тогда и только тогда, когда значения аргументов различны (табл. 2.8). Как следует из табл. 2.7 и 2.8, функция f_6 является обратной к f_5;

7) ***штрих Шеффера***

$$f_7(x,y) = x|y.$$

Операция, обратная по отношению к конъюнкции (функция ложна, только если оба аргумента истинны) табл. 2.9;

8) ***стрелка Пирса***

$$f_8(x,y) = x \downarrow y.$$

Функция, обратная дизъюнкции (f_8 истинно, только когда x и y ложны) табл. 2.10;

Таблица 2.7

x_1	x_2	$x_1 \sim x_2$
0	0	1
0	1	0
1	0	0
1	1	1

Таблица 2.8

x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Таблица 2.9

x_1	x_2	$x_1 \mid x_2$
0	0	1
0	1	1
1	0	1
1	1	0

Таблица 2.10

x_1	x_2	$x_1 \downarrow x_2$
0	0	1
0	1	0
1	0	0
1	1	0

Наиболее важными функциями являются первые три. Остальные могут быть выражены через эти три функции. С использованием трех основных функций (дизъюнкции, конъюнкции и отрицания) могут образовываться более сложные функции. Поэтому можно дать еще одно определение булевой алгебры. ***Булевой алгеброй*** называется алгебра типа $\left(B, \vee, \wedge, \bar{}\right)$, несущим множеством которой является множество двоичных чисел, а операциями — конъюнкция, дизъюнкция и отрицание.

2.2.3. Свойства основных логических функций

Основные логические функции имеют следующие свойства:

1) *коммутативность:* $a \vee b = b \vee a$; $a \cdot b = b \cdot a$;
2) *ассоциативность:* $a \vee (b \vee c) = (a \vee b) \vee c$; $a \cdot (b \cdot c) = (a \cdot b) \cdot c$;
3) *идемпотентность конъюнкции и дизъюнкции:* $a \vee a = a$; $a \cdot a = a$;
4) *дистрибутивность:*
   а) *конъюнкция относительно дизъюнкции:* $a \cdot (b \vee c) = (a \cdot b) \vee \vee (a \cdot c)$;
   б) *дизъюнкция относительно конъюнкции:* $a \vee (b \cdot c) = (a \vee b) \cdot \cdot (a \vee c)$;

5) *двойное отрицание:* $\overline{\overline{a}} = a$;

6) *правило де Моргана:* $\overline{a \cdot b} = \overline{a} \vee \overline{b}$; $\overline{a \vee b} = \overline{a} \cdot \overline{b}$;

7) *правило склеивания:* $a \cdot b \vee a \cdot \overline{b} = a$; $(a \vee b) \cdot (a \vee \overline{b}) = a$;

8) *правило поглощения:* $a \vee ab = a$; $a \cdot (a \vee b) = a$;
$a \vee \overline{a}\, b = a \vee b$; $a \cdot (\overline{a} \vee b) = a \cdot b$;

9) *действия с константами*:
$a \vee 0 = a$; $a \cdot 0 = 0$;
$a \vee 1 = 1$; $a \cdot 1 = a$;
$a \vee \overline{a} = 1$; $a \cdot \overline{a} = 0$.

Свойства основных булевых функций доказываются либо путем преобразования выражений, либо на основе сопоставления таблиц истинности правой и левой частей равенства.

Пример. Доказать, что $\overline{a \vee b} = \overline{a} \cdot \overline{b}$.

С учетом таблиц истинности элементарных логических операций определяем последовательно значения функций, указанных в верхней строке, для всех возможных значений аргументов a и b, т. е. строим для них соответствующие таблицы истинности (табл. 2.11).

Таблица 2.11

a	b	$a \vee b$	$\overline{a \vee b}$	$\overline{a}$	$\overline{b}$	$\overline{a} \cdot \overline{b}$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

Так как значения функций $\overline{a \vee b}$ и $\overline{a} \cdot \overline{b}$ на всех наборах совпадают, то эти функции равны.

2.2.4. Задание функции формулой. Эквивалентные преобразования логических выражений

Понятие формулы вводится для формализации представления и записи простого или сложного высказывания. Формула рассматривается как некоторый способ реализации функции и вводится индуктивно в соответствии со следующим правилом: если A и B — высказывания (простые или сложные, постоянные или переменные), то запись значения истинности каждого из этих высказываний есть формула; если A и B — формулы, то выражения $A * B$ и $\overline{A}$ (где символ * обозначает

знак одной из рассмотренных выше элементарных логических операций) — тоже формулы. Таким образом, рассмотренные выше выражения, которыми описывались элементарные логические операции и свойства основных логических операций, суть формулы. Применение по отношению к ним указанного правила позволяет получить новые формулы, соответствующие более сложным высказываниям.

Новые формулы могут быть получены на основе использования понятия суперпозиции функций. Суперпозицией функций $f_1, f_2, ..., f_n$ называется функция f, полученная путем подстановки функций $f_1, f_2, ..., f_n$ друг в друга и переименования переменных.

Пусть имеется множество логических функций, заданных формулами $E = \{f_1, f_2, ..., f_m\}$; при этом говорят, что имеет место множество формул над E. Тогда новую формулу можно получить, используя следующее правило.

Пусть $f_0(x_1, x_2, ..., x_n)$ — некоторая формула над E. Если $A_1, A_2, ..., A_n$ — либо символы переменных, либо другие формулы над E, то $f_0(A_1, A_2, ..., A_n)$ — тоже формула над E.

Пример. Пусть функция задана формулой $f_0(z_1, z_2) = z_1 \to z_2$, и при этом имеет место равенство $z_1 = A_1 = x_1 \vee x_2$, $z_2 = A_2 = x_3$. Тогда новую формулу над E можно получить путем подстановки A_1 и A_2 в исходную формулу: $f_0(x_1, x_2, x_3) = (x_1 \vee x_2) \to x_3$.

Полученную формулу вновь представим как исходную и, полагая далее $A_1 = f_0(x_1, x_2, x_3)$, $A_2 = x_3$, $A_3 = x_1 \cdot x_2$, делаем вновь подстановку.

Тогда новая формула над E

$$f_0(x_1, x_2, x_3) = (((x_1 \vee x_2) \to x_3) \vee x_3) \to x_1 \cdot x_2.$$

Логические операции обладают различным приоритетом с точки зрения порядка выполнения их в выражении. Принят следующий порядок выполнения операций в булевой алгебре: в первую очередь вычисляются выражения, над которыми стоит знак отрицания, далее выполняются операции конъюнкции $(\wedge)$, а затем дизъюнкции $(\vee)$.

Если выражение, заключенное в скобках, представляет собой конъюнкцию или имеет общий знак отрицания, то скобки опускаются.

Сопоставляя введенные понятия логической функции и формулы, следует иметь в виду, что логическая функция — это зависимость между логическими переменными, однозначно определяемая таблицей истинности, а формула — это выражение, которое используется для описания логической функции, причем одна и та же логическая функция может описываться несколькими формулами.

Пример. Рассмотрим две формулы:

$f(x_1, x_2) = x_1 | x_2$ и $f(x_1, x_2) = \overline{x_1 \cdot x_2}$.

Несложно показать, что обе формулы представляют одну и ту же функцию, так как таблицы истинности у них одинаковы.

Формулы, соответствующие одной и той же функции, называются ***эквивалентными*** или ***равносильными.***

Две формулы U и B называются эквивалентными (равносильными), если они реализуют одну и ту же функцию. При этом записывают $U = B$.

Эквивалентные преобразования логических выражений. *Эквивалентные преобразования* — это такие преобразования формул, при которых сохраняется их эквивалентность. Преобразование называется эквивалентным, если исходная формула $U = (x_1, x_2, ..., x_n)$ и полученная в результате преобразования формула $B = (x_1, x_2, ..., x_n)$ принимают одинаковые значения на каждом наборе значений аргументов.

Эквивалентное преобразование осуществляется на основе сопоставления таблиц истинности либо на основе применения свойств основных логических операций.

Покажем примеры эквивалентных преобразований, которые позволяют получить новые формулы для описания функций f_4 — f_8 (см. п. 1.2.2), используя только знаки операций конъюнкции, дизъюнкции и отрицания.

Примеры.

1. Преобразование формулы, описывающей функцию f_4: $a \to b = \bar{a} \vee b$. Справедливость преобразования доказывается соответствующей таблицей истинности (табл. 2.12).

Таблица 2.12

a	b	$\bar{a}$	$a \to b$	$\bar{a} \vee b$
0	0	1	1	1
0	1	1	1	1
1	0	0	0	0
1	1	0	1	1

2. Преобразование формулы, описывающей функцию f_5: $a \sim b = ab \vee \overline{a}\overline{b} = (a \to b)(b \to a)$. Справедливость преобразования доказывается соответствующей таблицей истинности (табл. 2.13).

Таблица 2.13

a	b	$a \sim b$	ab	$\overline{\overline{a}\,\overline{b}}$	$ab \vee \overline{\overline{a}\,\overline{b}}$
0	0	1	0	1	1
0	1	0	0	0	0
1	0	0	0	0	0
0	0	1	1	0	1

3. Функция f_6 : $a \oplus b = \overline{a \sim b} = \overline{ab \vee \overline{a}\,\overline{b}} = \overline{ab} \cdot \overline{\overline{a}\,\overline{b}} = (\overline{a} \vee \overline{b}) \cdot (a \vee b)$.
4. Функция f_7 : $a|b = \overline{a \cdot b}$.
5. Функция f_8 : $a \downarrow b = \overline{a \vee b}$.

Формулы, из которых построена некоторая исходная формула, называются ***подформулами***.

Чаще всего эквивалентные преобразования основаны на замене подформул на эквивалентные им подформулы. Если в формуле U заменить подформулу B на эквивалентную ей подформулу B', то формула U перейдет в эквивалентную ей формулу U'.

2.2.5. Двойственные функции

Логическая функция $f^*(x_1, x_2, ..., x_n)$, равная $\overline{f}(\overline{x}_1, \overline{x}_2, ..., \overline{x}_n)$, называется ***двойственной*** по отношению к функции $f(x_1, x_2, ..., x_n)$.

Очевидно свойство взаимности двойственных функций:

$$f^{**}(x_1, x_2, ..., x_n) = f(x_1, x_2, ..., x_n).$$

Теорема 2.1. Если некоторая формула Φ реализует функцию f, то формула Φ^*, реализующая двойственную функцию f^*, может быть получена из Φ путем замены всех подформул на двойственные им. То есть если

$$\Phi(x_1, x_2, ..., x_n) =$$
$$= f\left[f_1(x_1, x_2, ..., x_n), f_2(x_1, x_2, ..., x_n), ..., f_k(x_1, x_2, ..., x_n)\right], \text{ то}$$
$$\Phi^*(x_1, x_2, ..., x_n) =$$
$$= f^*\left[f_1^*(x_1, x_2, ..., x_n), f_2^*(x_1, x_2, ..., x_n), ..., f_k^*(x_1, x_2, ..., x_n)\right].$$

В частном случае из теоремы 2.1 вытекает следующее правило.

Если формула Φ содержит только операции $\wedge, \vee, \overline{\ }$ и константы $0, 1$, то для получения формулы Φ^*, двойственной к Φ, необходимо, сохраняя порядок выполнения операций, везде заменить 0 на 1 (1 на 0), операцию $\vee$ на & (операцию & на $\vee$).

Можно сформулировать следующие принципы двойственности для формул:

1) если две формулы эквивалентны, т. е. $\Phi_1(x_1, x_2, ..., x_n) = \Phi_2(x_1, x_2, ..., x_n)$,

то эквивалентны и отрицания этих формул $\overline{\Phi}_1(x_1, x_2, ..., x_n) = \overline{\Phi}_2(x_1, x_2, ..., x_n)$;

2) если формулы Φ_1 и Φ_2 эквивалентны, то и двойственные им функции тоже эквивалентны $\Phi_1^*(x_1, x_2, ..., x_n) = \Phi_2^*(x_1, x_2, ..., x_n)$;

3) если две формулы эквивалентны, то будут эквивалентны и формулы, полученные из исходных путем замены аргументов на их отрицания

$$\Phi_1(\bar{x}_1, \bar{x}_2, ..., \bar{x}_n) = \Phi_2(\bar{x}_1, \bar{x}_2, ..., \bar{x}_n).$$

2.3. Специальные разложения логических функций

2.3.1. Конъюнктивная и дизъюнктивная нормальные формы

Любая функция булевой алгебры может быть представлена некоторыми формулами специального вида. ***Нормальные формы*** — это некоторое стандартное представление функции с помощью элементарных конъюнкций и элементарных дизъюнкций.

Элементарной дизъюнкцией (ЭД) называют дизъюнкцию нескольких переменных или их отрицаний, например $d_i = a_1 \vee a_2 \vee \bar{a}_3$.

Элементарной конъюнкцией (ЭК) называют конъюнкцию нескольких переменных или их отрицаний, например $k_i = a_1 \cdot \bar{a}_2 \cdot a_3$.

Дизъюнктивной нормальной формой (ДНФ) некоторой заданной функции называется формула, которая имеет вид дизъюнкции элементарных конъюнкций, например $a_1 \bar{a}_2 a_3 \vee \bar{a}_1 \bar{a}_2 a_3$.

Конъюнктивной нормальной формой (КНФ) некоторой заданной функции называется формула, которая имеет вид конъюнкции элементарных дизъюнкций.

Если заданная функция уже представлена некоторой формулой, то путем эквивалентных преобразований эта формула может быть

приведена к равносильной ей формуле в соответствующей нормальной форме (КНФ или ДНФ).

Порядок действий при приведении исходной формулы к нормальной форме следующий:

1) все функции выражаются через дизъюнкцию, конъюнкцию и отрицание;

2) если в выражении над несколькими элементами имеются знаки отрицания, то следует, используя формулы де Моргана, изменить выражение так, чтобы отрицание относилось только к одной переменной;

3) дальнейшее преобразование производится с использованием свойств элементарных операций.

Пример. $A\cdot\overline{\overline{B}\cdot C}\cdot(M\vee\overline{N})=A(B\vee\overline{C})(M\vee\overline{N})=(AB\vee A\overline{C})(M\vee\overline{N})=$
$=ABM\vee AB\overline{N}\vee A\overline{C}M\vee A\overline{C}\,\overline{N}.$

Упрощение логических формул на основе применения понятий тождественно истинных и тождественно ложных форм. После приведения к нормальной форме можно существенно упрощать выражения логических функций.

Утверждение 1. Для того чтобы элементарная дизъюнкция была тождественно истинной, необходимо и достаточно, чтобы среди ее слагаемых нашлась хотя бы одна переменная и ее отрицание ($x_i\vee\overline{x_i}=1$).

Утверждение 2. Для того чтобы элементарная конъюнкция была тождественно ложной, необходимо и достаточно, чтобы среди ее сомножителей оказалась хотя бы одна переменная и ее отрицание ($x_i\,\&\,\overline{x_i}=0$).

Указанные два утверждения используются для упрощения выражений следующим образом.

В случае принадлежности логической формулы к КНФ рассматривается каждый ее сомножитель, и если в какой-то из них входят вместе x_i и $\overline{x_i}$, то он равен единице и его можно исключить. Если все сомножители равны единице, то такая функция тождественно истинна.

В случае принадлежности формулы к ДНФ рассматривается каждое слагаемое. Если в каком-либо слагаемом встречается произведение $x_i\cdot\overline{x_i}$, то это слагаемое равно нулю и его можно исключить.

Пример. Установить характер истинности формулы

$(P\cdot(P\to Q))\to Q=\overline{P\cdot(\overline{P}\vee Q)}\vee Q=\overline{P}\vee(\overline{\overline{P}\vee Q})\vee Q=\overline{P}\vee(P\cdot\overline{Q})\vee Q=$
$=\overline{P}\vee\overline{Q}\vee Q=1.$

2.3.2. Совершенно нормальные конъюнктивная и дизъюнктивная формы

Пусть имеется n переменных: $x_1, x_2, ..., x_n$. Будем формировать из них строки так, чтобы выполнялись три условия:

1) в каждую строку входят все n переменных;
2) каждая переменная входит в строку только один раз;
3) в строку входит либо сама переменная, либо ее отрицание, но не одновременно то и другое.

Число таких строк 2^n.

Элементарная дизъюнкция, сформированная в соответствии с указанными требованиями, образует логическую конструкцию вида $d_i = x_1 \vee \overline{x_2} \vee \overline{x_3} \vee ... \vee x_n$ и называется ***макстермом***.

Элементарная конъюнкция образует логическую конструкцию вида $k_i = x_1 \cdot x_2 \cdot \overline{x_3} \cdot ... \cdot x_n$ и называется ***минтермом***.

Совершенной дизъюнктивной нормальной формой (СДНФ) логической функции $f(x_1, x_2, ..., x_n)$ называют представление ее в виде дизъюнкции минтермов, построенных из аргументов $x_1, x_2, ..., x_n$:

$$f(x_1, x_2, ..., x_n) = x_1 x_2 \cdot ... \cdot x_n \vee \overline{x_1} x_2 \cdot ... \cdot x_n \vee x_1 \overline{x_2} ... x_n \vee ... \vee \overline{x_1} \overline{x_2} ... \overline{x_n}.$$

Совершенной конъюнктивной нормальной формой (СКНФ) логической функции $f(x_1, x_2, ..., x_n)$ называют представление ее в виде конъюнкции макстермов, построенных из аргументов рассматриваемой функции:

$$f(x_1, x_2, ..., x_n) = (x_1 \vee x_2 \vee ... \vee x_n) \cdot (\overline{x_1} \vee x_2 \vee ... \vee x_n) \cdot ... \cdot (\overline{x_1} \vee \overline{x_2} \vee ... \vee \overline{x_n}).$$

Каждая логическая функция может быть представлена единственным образом в совершенной дизъюнктивной нормальной форме и совершенной конъюнктивной нормальной форме. В полученной формуле заданной функции могут присутствовать или все термы, которые можно построить из n переменных, или часть из них, но дважды один терм входить в выражение не может.

Приведение функции к совершенно нормальной форме называют ее ***разложением по всем переменным или по термам***.

Если имеется формула, описывающая некоторую заданную функцию, то с применением эквивалентных преобразований ее можно привести к совершенной дизъюнктивной нормальной форме, применяя следующую последовательность действий.

1. Логическая формула $f(x_1, x_2, ..., x_n)$ приводится к дизъюнктивной нормальной форме

$$f(x_1, x_2, ..., x_n) = \varphi_1(x_1, x_2, ..., x_n) \vee \varphi_2(x_1, x_2, ..., x_n) \vee ... .$$

При этом каждое слагаемое представляет собой конъюнкцию некоторого числа переменных, но не обязательно всех.

2. Пусть слагаемое $\varphi_j(x_1, x_2, ..., x_n)$ не содержит ни x_i, ни $\overline{x_i}$. В этом случае к нему конъюнктивно присоединяют тождественно-истинную форму $\left(x_i \vee \overline{x_i}\right)$:

$$\varphi_j(x_1, x_2, ..., x_n) \cdot \left(x_i \vee \overline{x_i}\right) = \varphi_j(x_1, x_2, ..., x_n) x_i \vee \varphi_j(x_1, x_2, ..., x_n) \overline{x_i}.$$

Если отсутствуют несколько переменных, то операцию нужно повторить для всех отсутствующих сомножителей. По окончании слагаемое будет удовлетворять первому условию (см. с. 25).

3. Если в каком-либо слагаемом имеется несколько одинаковых сомножителей, то из них оставляется один.

4. Если слагаемое содержит $x_i \cdot \overline{x_i}$, то это слагаемое можно исключить.

5. Если среди всех слагаемых окажется два или несколько одинаковых, то они заменяются одним.

В результате указанных операций формула будет приведена к СДНФ.

При приведении к совершенной конъюнктивной нормальной форме последовательность действий остается той же, но все действия заменяются на двойственные.

Существует еще один метод приведения функции к совершенной нормальной форме.

Выражение в форме СКНФ включает столько сомножителей, сколько в таблице истинности содержится наборов, на которых функция равна нулю. Каждый сомножитель содержит дизъюнкцию всех входных переменных.

При этом если $a_1, a_2, ..., a_n$ — набор, где функция равна нулю, то в элементарную дизъюнкцию включают x_i, если $a_i = 0$, и $\overline{x_i}$, если $a_i = 1$.

Правила для построения совершенной дизъюнктивной нормальной формы аналогичны, только значения заменяются на двойственные.

Пример. Рассмотрим пример приведения заданной логической функции к форме СДНФ с использованием обоих известных способов (табл. 2.14).

$$f(x,y,z)=(x\oplus y)\to\bar{y}\,\&\,\bar{x}\vee x\,\&\,z\,\&\,(y\vee\bar{x})=\overline{(\bar{x}\vee\bar{y})\,\&\,(x\vee y)}\vee$$
$$\vee\bar{x}\,\&\,\bar{y}\vee x\,\&\,y\,\&\,z\vee x\,\&\,z\,\&\,\bar{x}=\overline{\bar{x}\vee\bar{y}}\vee\overline{x\vee y}\vee\bar{x}\,\&\,\bar{y}\vee x\,\&\,y\,\&\,z=$$
$$=x\,\&\,y\vee\bar{x}\,\&\,\bar{y}\vee\bar{x}\,\&\,\bar{y}\vee x\,\&\,y\,\&\,z=x\,\&\,y\vee\bar{x}\,\&\,\bar{y}\vee x\,\&\,y\,\&\,z=$$
$$=x\,\&\,y\,\&\,(z\vee\bar{z})\vee\bar{x}\,\&\,\bar{y}\,\&\,(z\vee\bar{z})\vee x\,\&\,y\,\&\,z=x\,\&\,y\,\&\,z\vee x\,\&\,y\,\&\,\bar{z}\vee$$
$$\vee\bar{x}\,\&\,\bar{y}\,\&\,z\vee\bar{x}\,\&\,\bar{y}\,\&\,\bar{y}\vee x\,\&\,y\,\&\,z=x\,\&\,y\,\&\,z\vee x\,\&\,y\,\&\,\bar{z}\vee$$
$$\vee\bar{x}\,\&\,\bar{y}\,\&\,z\vee\bar{x}\,\&\,\bar{y}\,\&\,\bar{z}.$$

Таблица 2.14

x	y	z	$x\oplus y$	$\bar{x}\,\&\,\bar{y}$	$x\,\&\,z\,\&\,(y\vee\bar{x})$	$f(x,y,z)$
0	0	0	0	1	0	1
0	0	1	0	1	0	1
0	1	0	1	0	0	0
1	0	0	1	0	0	0
0	1	1	1	0	0	0
1	0	1	1	0	0	0
1	1	0	0	0	0	1
1	1	1	0	0	1	1

$$f(x,y,z)=\bar{x}\bar{y}\bar{z}\vee\bar{x}\bar{y}z\vee x\bar{y}\bar{z}\vee xyz.$$

Пример. Рассмотрим пример приведения заданной логической функции к форме СКНФ с использованием обоих известных способов (табл. 2.15).

Таблица 2.15

x	y	z	$z\oplus\bar{x}$	$\bar{y}\vee\bar{z}\vee x$	$\bar{x}\to zy$	$f(x,y,z)$
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	1	0	1	1	0	0
1	0	0	0	0	1	0
0	1	1	0	1	1	0
1	0	1	1	1	1	1
1	1	0	0	1	1	0
1	1	1	1	1	1	1

$$f(x,y,z)=(x\vee y\vee\bar{z})(x\vee\bar{y}\vee\bar{z})(\bar{x}\vee\bar{y}\vee z)(\bar{x}\vee y\vee z)(x\vee z\vee y)(x\vee\bar{y}\vee z);$$

$$f(x,y,z)=(z\oplus\bar{x})(\bar{y}\vee\bar{z}\vee x)(\bar{x}\to zy)=(z\vee\bar{x})(\bar{z}\vee\bar{\bar{x}})(x\vee\bar{y}\vee\bar{z})(\bar{\bar{x}}\vee zy)=$$
$$=(z\vee\bar{x})(\bar{z}\vee x)(x\vee\bar{y}\vee\bar{z})(x\vee z)(x\vee y)=(z\vee\bar{x}\vee y\bar{y})(\bar{z}\vee x\vee y\bar{y})(x\vee\bar{y}\vee\bar{z})\&$$
$$\&(x\vee z\vee y\bar{y})(x\vee y\vee z\bar{z})=(z\vee\bar{x}\vee\bar{y})(z\vee\bar{x}\vee y)(\bar{z}\vee x\vee\bar{y})(\bar{z}\vee x\vee y)(x\vee\bar{y}\vee\bar{z})\&$$
$$\&(x\vee z\vee y)(x\vee z\vee\bar{y})(x\vee y\vee z)(x\vee y\vee\bar{z})=$$
$$=(x\vee y\vee\bar{z})(x\vee\bar{y}\vee\bar{z})(\bar{x}\vee\bar{y}\vee z)\&(\bar{x}\vee y\vee z)(x\vee z\vee y)(x\vee\bar{y}\vee z).$$

2.4. Минимизация булевых функций

2.4.1. Понятие минимизации

Задача минимизации булевой функции состоит в построении такой ДНФ (или КНФ) для некоторой заданной функции, которая реализует эту функцию и имеет наименьшее суммарное число операций и символов в формуле, т. е. минимальную сложность. Решение этой задачи важно, когда логические функции реализуются техническими устройствами. Одной из основных целей минимизации является упрощение логических устройств и достижение максимальной экономичности разрабатываемых систем.

Для минимизации булевых функций в целях получения самых простых выражений используется ряд методов, среди которых наибольшее применение находят:

1) метод неопределенных коэффициентов;
2) метод Квайна — Мак Класки;
3) метод карт Карно.

2.4.2. Метод неопределенных коэффициентов

Сущность метода заключается в представлении функции в самом общем виде в ДНФ и введении коэффициентов a_i, значения которых (0 или 1) подбираются таким образом, чтобы формула была минимальной. Пусть дана функция трех переменных. Представим формулу этой функции в виде ДНФ самого общего вида:

$$\begin{aligned}f(x_1,x_2,\ldots,x_n)=a_1x_1\vee a_2\bar{x}_1\vee a_3x_2\vee a_4\bar{x}_2\vee a_5x_3\vee a_6\bar{x}_3\vee a_7x_1x_2\vee\\ \vee a_8\bar{x}_1x_2\vee a_9x_1\bar{x}_2\vee a_{10}\bar{x}_1\bar{x}_2\vee a_{11}x_1x_3\vee a_{12}\bar{x}_1x_3\vee a_{13}x_1\bar{x}_3\vee a_{14}\bar{x}_1\bar{x}_3\vee\\ \vee a_{15}x_2x_3\vee a_{16}\bar{x}_2x_3\vee a_{17}x_2\bar{x}_3\vee a_{18}\bar{x}_2\bar{x}_3\vee a_{19}x_1x_2x_3\vee a_{20}\bar{x}_1x_2x_3\vee\\ \vee a_{21}\bar{x}_1x_2\bar{x}_3\vee a_{22}\bar{x}_1\bar{x}_2x_3\vee a_{23}x_1\bar{x}_2x_3\vee a_{24}\bar{x}_1x_2\bar{x}_3\vee a_{25}x_1x_2\bar{x}_3\vee a_{26}x_1x_2x_3.\end{aligned}\qquad(2.1)$$

Термы, содержащие k переменных, будем называть термами ранга k. В выражении (2.1) представлены все возможные формы термов.

Если записать уравнение (2.1) для всех возможных значений аргументов x_1, x_2, x_3, то получим систему 2^n уравнений (в рассматриваемом случае $n=3$):

$$
\begin{aligned}
&1)\ f(1,1,1) = a_1 \vee a_3 \vee a_5 \vee a_7 \vee a_{11} \vee a_{15} \vee a_{26};\\
&2)\ f(1,1,0) = a_1 \vee a_3 \vee a_6 \vee a_7 \vee a_{13} \vee a_{17} \vee a_{25};\\
&3)\ f(1,0,1) = a_1 \vee a_4 \vee a_5 \vee a_9 \vee a_{11} \vee a_{16} \vee a_{24};\\
&4)\ f(0,1,1) = a_2 \vee a_3 \vee a_5 \vee a_8 \vee a_{12} \vee a_{15} \vee a_{23};\\
&5)\ f(1,0,0) = a_1 \vee a_4 \vee a_6 \vee a_9 \vee a_{13} \vee a_{18} \vee a_{20};\\
&6)\ f(0,1,0) = a_2 \vee a_3 \vee a_6 \vee a_8 \vee a_{14} \vee a_{17} \vee a_{21};\\
&7)\ f(0,0,1) = a_2 \vee a_4 \vee a_5 \vee a_{10} \vee a_{12} \vee a_{16} \vee a_{22};\\
&8)\ f(0,0,0) = a_2 \vee a_4 \vee a_6 \vee a_{10} \vee a_{14} \vee a_{18} \vee a_{19}.
\end{aligned}
\qquad (2.2)
$$

Для заданной конкретной функции конкретные значения левой части (2.2) известны. Если набор x_1, x_2, x_3 такой, что функция на нем принимает значение 0, то все коэффициенты в правой части равны 0. Эти нулевые коэффициенты вычеркиваются и в остальных уравнениях. В каждом оставшемся уравнении приравниваем единице коэффициент, определяющий конъюнкцию наименьшего ранга (ранг — число переменных), а остальные коэффициенты в правой части уравнения приравниваем нулю с учетом ранее сделанных подстановок. После подстановки коэффициентов в уравнение (2.1) получаем результирующее выражение булевой функции.

Пример. Минимизировать заданное выражение

$f(x_1, x_2, x_3) = x_1x_2x_3 \vee x_1x_2\overline{x_3} \vee x_1\overline{x_2}x_3 \vee x_1\overline{x_2}\,\overline{x_3} \vee \overline{x_1}x_2x_3.$

После вычисления значений функции на всех наборах и подстановки в (2.2) получим:

1) $a_1 \vee a_3 \vee a_5 \vee a_7 \vee a_{11} \vee a_{15} \vee a_{26} = 1;$
2) $a_1 \vee a_3 \vee a_6 \vee a_7 \vee a_{13} \vee a_{17} \vee a_{25} = 1;$
3) $a_1 \vee a_4 \vee a_5 \vee a_9 \vee a_{11} \vee a_{16} \vee a_{24} = 1;$
~~4) $a_2 \vee a_3 \vee a_5 \vee a_8 \vee a_{12} \vee a_{15} \vee a_{23} = 0;$~~
5) $a_1 \vee a_4 \vee a_6 \vee a_9 \vee a_{13} \vee a_{18} \vee a_{20} = 1;$
~~6) $a_2 \vee a_3 \vee a_6 \vee a_8 \vee a_{14} \vee a_{17} \vee a_{21} = 0;$~~
~~7) $a_2 \vee a_4 \vee a_5 \vee a_{10} \vee a_{12} \vee a_{16} \vee a_{22} = 0;$~~
8) $a_2 \vee a_4 \vee a_6 \vee a_{10} \vee a_{14} \vee a_{18} \vee a_{19} = 1.$

Все коэффициенты, входящие в нулевые наборы, должны быть равны нулю, т. е.

$$a_2 = a_3 = a_4 = a_5 = a_6 = a_8 = a_{10} = a_{12} = a_{14} = a_{15} = \\ = a_{16} = a_{17} = a_{21} = a_{22} = a_{23} = 0.$$

Остается рассмотреть единичные наборы:

1) $a_1 \vee a_7 \vee a_{11} \vee a_{26} = 1$;
2) $a_1 \vee a_7 \vee a_{13} \vee a_{25} = 1$;
3) $a_1 \vee a_9 \vee a_{11} \vee a_{24} = 1$;
5) $a_1 \vee a_9 \vee a_{13} \vee a_{18} \vee a_{20} = 1$;
8) $a_{18} \vee a_{19} = 1$.

В первых четырех уравнениях конъюнкция наименьшего ранга — a_1, а в последнем — a_{18}. Таким образом, принимаем $a_1 = 1$, $a_{18} = 1$. Остальные коэффициенты принимаем равными нулю.

Подставляя коэффициенты в выражение (2.1), получаем минимальную ДНФ заданной функции $f(x, y, z) = x_1 \vee \overline{x_2}\,\overline{x_3}$.

2.4.3. Метод Квайна — Мак Класки

Метод Квайна. При минимизации методом Квайна исходная функция задается в СДНФ. Сущность метода состоит в поэтапном упрощении выражений на основе операций склеивания.

Шаг 1. *Нахождение первичных импликант.* Все термы сравниваются между собой попарно. Если два терма имеют вид $m_i = ax_i$, а $m_j = a\overline{x_i}$ (где a — конъюнкция нескольких переменных), тогда вместо m_i и m_j выписываются a, которые являются термом $(n-1)$ порядка. Исключаемые термы помечаются. Далее сравниваются все полученные термы $(n-1)$ ранга. В результате склеивания выписываются термы $(n-2)$ ранга. Процесс продолжается до тех пор, пока дальнейшее склеивание становится невозможным. Не исключенные в процессе выполнения указанной процедуры термы исходной функции, а также термы, которые были получены в результате склеивания, будем называть первичными или простыми импликантами.

Пример. Пусть задано следующее выражение в форме СДНФ:

$f(x_1, x_2, x_3) = \overline{x_1}\,\overline{x_2}x_3x_4 \vee \overline{x_1}x_2\overline{x_3}\,\overline{x_4} \vee \overline{x_1}x_2\overline{x_3}x_4 \vee \overline{x_1}x_2x_3x_4 \vee x_1\overline{x_2}\,\overline{x_3}\,\overline{x_4} \vee$
$\vee x_1\overline{x_2}x_3x_4 \vee x_1x_2\overline{x_3}x_4 \vee x_1x_2x_3\overline{x_4}$.

Пронумеруем все входящие в него минтермы:

$\overline{x_1}\,\overline{x_2}x_3x_4 - 1$; $\overline{x_1}x_2\overline{x_3}\,\overline{x_4} - 2$; $\overline{x_1}x_2\overline{x_3}x_4 - 3$; $\overline{x_1}x_2x_3x_4 - 4$; $x_1\overline{x_2}\,\overline{x_2}\,\overline{x_4} - 5$;
$x_1\overline{x_2}x_3x_4 - 6$; $x_1x_2\overline{x_3}\,\overline{x_4} - 7$; $x_1x_2\overline{x_3}x_4 - 8$.

Производим склеивания термов 4-го ранга и получаем термы 3-го ранга:

$1-4 \to (x_2)\ \overline{x_1}x_3x_4$ (1); $3-8 \to (x_2)\ x_2\overline{x_3}x_4$ (6);
$1-6 \to (x_1)\ \overline{x_2}x_3x_4$ (2); $5-6 \to (x_2)\ x_1\overline{x_2}x_4$ (7);
$2-3 \to (x_4)\ x_1\overline{x_2}\,\overline{x_3}$ (3); $5-8 \to (x_2)\ x_1x_3\overline{x_4}$ (8);
$2-7 \to (x_1)\ \overline{x_2}x_3x_4$ (4); $7-8 \to (x_2)\ x_1x_2\overline{x_3}$ (9).
$3-4 \to (x_3)\ \overline{x_1}x_2x_4$ (5);

Производим склеивание термов 3-го ранга и получаем термы 2-го ранга:

$3-9 \to (x_1)\ x_2\overline{x_3}$;
$4-6 \to (x_4)\ x_2\overline{x_3}$.

Дальнейшее склеивание невозможно, выписываем простые импликанты: $\overline{x_1}x_3x_4$, $\overline{x_2}x_3x_4$, $\overline{x_1}x_2x_4$, $x_1\overline{x_2}x_4$, $x_1\overline{x_3}x_4$, $x_2\overline{x_3}$.

Шаг 2. *Составление таблицы.* В колонках табл. 2.16 записываем термы исходного выражения (они обозначены порядковыми номерами), а в строках — простые импликанты.

Таблица 2.16

	1	2	3	4	5	6	7	8
$\overline{x_1}x_3x_4$	v			v				
$\overline{x_2}x_3x_4$	v					v		
$\overline{x_1}x_2x_4$			v	v				
$x_1\overline{x_2}x_4$					v	v		
$x_1\overline{x_3}x_4$					v			v
$x_2\overline{x_3}$		v	v				v	v

Если в исходный терм входит какая-либо импликанта, то на пересечении соответствующего столбца и строки ставится метка.

Шаг 3. *Нахождение существенных импликант.* Если в каком-либо из столбцов таблицы имеется одна метка, то импликанта, соответствующая этой строке, является существенной. Она обязательно входит в конечный результат. Из таблицы для дальнейшего анализа исключаются строки, соответствующие существенным импликантам, и покрываемые ими столбцы.

Шаг 4. *Вычеркивание лишних столбцов.* Из двух столбцов, имеющих метки в одинаковых строках, один вычеркивается (в рассматриваемом примере таких нет).

Шаг 5. *Вычеркивание лишних импликант.* Если после шага 4 в таблице появятся строки, в которых нет ни одной метки, то импликанты, соответствующие этим строкам, из рассмотрения исключаются (в нашем примере нет). Результаты выполнения описанных действий приведены в табл. 2.17.

Таблица 2.17

		1	4	5	6
1	$\overline{x_1}x_3x_4$		*v*		
2	$\overline{x_2}x_3x_4$	*v*			*v*
3	$\overline{x_1}x_2x_4$		*v*		
4	$x_1\overline{x_2}x_4$			*v*	*v*
5	$x_1\overline{x_3}x_4$			*v*	

Шаг 6. *Выбор конечного результата.* В результирующее выражение включается совокупность импликант, которые имеют метки во всех столбцах. Предпочтение отдается тому варианту, в котором минимально суммарное число литер (букв), входящих в конечный результат.

Выбирая 1-ю и 4-ю импликанты, которые в совокупности покрывают все столбцы, окончательно получаем

$$f(x_1, x_2, x_3, x_4) = x_2\overline{x_3} \vee \overline{x_1}x_3x_4 \vee x_1\overline{x_2}x_4.$$

Метод Мак Класки. Мак Класки предложил модернизировать метод Квайна следующим образом:

1) все термы кодируются в виде двоичных последовательностей: переменной соответствует 1; ее отрицанию — 0, например $x_1\overline{x_2}x_3 = 101$;

2) все последовательности разбиваются на группы по числу единиц в последовательности (в *i*-ю группу попадает терм, который имеет *i* единиц);

3) при сравнении сопоставляются только соседние группы, так как только они имеют отличие в одном разряде;

4) при исключении переменной вместо нее записывается прочерк.

Порядок формирования минимизированного выражения логической функции такой же, как в методе Квайна.

2.4.4. Метод карт Карно

Карта Карно — это графическое представление таблицы истинности. Она представляет собой совокупность ячеек, определяемых системой вертикальных и горизонтальных координат; каждой ячейке соот-

ветствует набор значений входных переменных. Запись в ячейке — это значение функции на соответствующем наборе.

Таким образом, имеется взаимно однозначное соответствие между ячейками Карно и строками таблицы истинности.

Пример. Функции $f(x, y, z) = \overline{x}(\overline{y} \vee \overline{z}) \vee xz$ соответствуют представленные ниже таблица истинности (табл 2.18) и карта Карно (рис. 2.1).

Карта Карно

x \ $y\,z$	00	01	11	10
0	1	1	0	1
1	0	1	1	0

Рис. 2.1

Таблица 2.18

Таблица истинности

x	y	z	$f(x,y,z)$
0	0	0	1
0	0	1	1
0	1	0	1
1	0	0	0
1	0	1	1
0	1	1	0
1	1	0	0
1	1	1	1

Сущность метода заключается в выделении в карте Карно прямоугольных ячеек (блоков), содержащих одно и то же значение функции. Любой блок может иметь размер $2^a \times 2^b$, где a, b — целые.

Правила формирования выражения минимальной булевой функции. Основные правила заключаются в следующем:

1) каждая ячейка, содержащая единицу, должна войти в какой-то из блоков;

2) результат записывается в виде логической суммы термов, соответствующих сформированным блокам;

3) терм, описывающий блок, записывается в форме произведения тех переменных, которые на координатной сетке не изменяют свое значение в пределах блока; если координата равна нулю, то соответствующая ей переменная входит в терм с отрицанием, а если — единице, то без отрицания.

Степень сложности булевой функции оценивается числом слагаемых и числом букв (литер). Выражение, которое получено с применением метода карт Карно, на основе термов, сформированных из групп с единичным значением логической функции, при минимальном количестве литер, называется ***минимальной суммой***.

При формировании групп для получения минимальных сумм необходимо руководствоваться двумя принципами:

1) группа должна быть как можно больше;

2) число групп должно быть как можно меньше.

Карту Карно следует рассматривать как трехмерную, представляя ее в виде тора (склеивая правую и левую, а также нижнюю и верхнюю границы).

Пример. Карте Карно, представленной на рис. 2.2, соответствует функция $f(w, x, y, z) = x\bar{z} \vee \bar{x}yz$.

Общая последовательность действий при минимизации:

1) в таблице Карно выбирается ячейка с единицей, не являющаяся подмножеством уже сформированного блока;

2) формируется наибольшая группа, содержащая эту ячейку;

3) выбирается следующая ячейка, не вошедшая в предшествующие группы, и для нее повторяются те же действия;

4) процесс повторяется, пока не останутся единицы, не включенные в какие-либо группы;

5) записывается выражение минимальной функции по правилу, которое было установлено выше.

Пример. Карте Карно, представленной на рис. 2.3, соответствует функция $f(w, x, y, z) = \overline{xy} \vee wy\bar{z} \vee \overline{xz}$.

Существует еще один способ записи минимальной формулы на основе метода минимальных произведений. При этом по тому же правилу объединяются ячейки с нулем, логическая формула записывается как конъюнкция элементарных дизъюнкций, а правила формирования переменных в дизъюнкции обратны описанным выше.

Минимизация частично определенных булевых функций. Если некоторые входные наборы невозможны либо значение функции на них несущественно, говорят о недоопределенных условиях. При этом особенность использования карт Карно следующая: недоопределенное условие на карте Карно обозначается прочерком, и его можно либо присоединить, либо не присоединить к любой группе.

Пример. Карте Карно, представленной на рис. 2.4, соответствует функция $f(w, x, y, z) = y\bar{z} \vee \bar{w}x\bar{y}$.

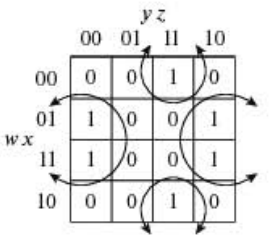

Рис. 2.2

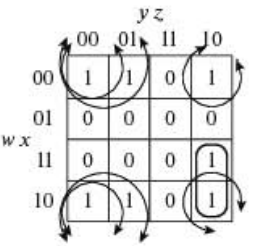

Рис. 2.3

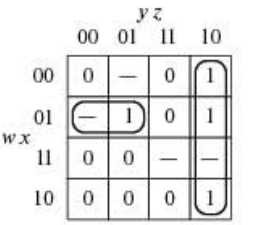

Рис. 2.4

2.5. Полнота и замкнутость множества булевых функций

2.5.1. Понятие функционально полной системы

Система функций $\{f_1, f_2, ..., f_n\}$ называется ***функционально полной***, если любая булева функция может быть записана в виде формул через функции этой системы. Одним из способов получения полных систем является использование некоторых существующих, ранее выявленных полных систем.

Теорема 2.1. Пусть даны две системы функций: $B = \{f_1, f_2, ..., f_n\}$ и $D = \{q_1, q_2, ..., q_m\}$. Относительно этих функций известно, что B — полная система и каждая ее функция может быть выражена в виде формул через функции второй системы. В этом случае система D — тоже полная.

Доказательство. Пусть h — это произвольная функция. В силу полноты B функция h может быть выражена в виде формул над B, т. е. $h = F(f_1, f_2, ..., f_n)$. Но любая из функций B по условию может быть выражена через функции D, т. е. $f_1 = \varphi_1(q_1, q_2, ...)$, $f_2 = \varphi_2(q_1, q_2, ...)$.

Следовательно, $h = F(\varphi_1(q_1, q_2, ...), \varphi_2(q_1, q_2, ...)) = F(q_1, q_2, ..., q_n)$.

Таким образом, произвольная функция h может быть выражена через функции системы D. Поэтому система D — полная.

Доказанную теорему можно использовать для доказательства полноты двух следующих систем: $S_1 = \{\wedge, \bar{}\}$ и $S_2 = \{\vee, \bar{}\}$. Известно, что $S_0 = \{\wedge, \vee, \bar{}\}$ — полная система (это вытекает из того факта, что любую булеву функцию можно представить в КНФ и ДНФ).

Для доказательства полноты S_1 и S_2 нужно установить, что недостающая относительно S_0 операция (в S_1 — дизъюнкция, в S_2 — конъюнкция) может быть выражена через две остальные. Такое подтверждение дают правила де Моргана и двойного отрицания: $x_1 \vee x_2 = \overline{\bar{x}_1 \cdot \bar{x}_2}$, $x_1 \wedge x_2 = \overline{\bar{x}_1 \vee \bar{x}_2}$. Следовательно, S_1 и S_2 — полные системы и их называют соответственно ***конъюнктивным и дизъюнктивным базисом Буля***.

Полной является также система $S_3 = \{\wedge, \oplus, 1\}$, которая сводится к конъюнктивному базису, так как отрицание в S_1 может быть выражено через операции S_3 следующим образом: $\bar{x} = x \oplus 1$ (проверяемся по таблице истинности функции f_6, см. табл. 2.8).

2.5.2. Алгебра Жегалкина

Алгебра над множеством логических функций с двумя бинарными операциями (конъюнкция и сложение по модулю два) называется ***алгеброй Жегалкина***. В этой алгебре выполняются следующие соотношения:

1) $x \oplus y = y \oplus x$;
2) $x(y \oplus z) = xy \oplus xz$;
3) $x \oplus x = 0$;
4) $x \oplus 0 = x$;
5) $\bar{\bar{x}} = x \oplus 1$.

Для подтверждения полноты алгебры Жегалкина представим дизъюнкцию через функции этой алгебры: $x \vee y = \overline{\bar{x} \cdot \bar{y}} = xy \oplus x \oplus y$.

Полученная формула, имеющая вид суммы произведений, называется ***полиномом по модулю два*** или ***полиномом Жегалкина***. Для каждой логической функции может быть получен полином Жегалкина, причем единственный для данной функции. Для этого следует привести заданное выражение к форме СДНФ, исключить операции дизъюнкции и отрицания и раскрыть скобки. В частном случае, если $x \cdot y = 0$, то $x \vee y = x \oplus y$.

Пример 1. $x_1x_2 \vee \overline{\bar{x}_1\bar{x}_2} = x_1x_2 \oplus \overline{\bar{x}_1\bar{x}_2} = x_1x_2 \oplus (x_1 \oplus 1)(x_2 \oplus 1) =$
$= x_1x_2 \oplus x_1x_2 \oplus x_1 \oplus x_2 \oplus 1 = x_1 \oplus x_2 \oplus 1.$

Существует еще один способ приведения функции к полиному Жегалкина. Запишем предполагаемый результат в виде выражения общего вида с неопределенными коэффициентами.

Пример 2. $x_1x_2 \vee \overline{\bar{x}_1\bar{x}_2} = ax_1x_2 \oplus bx_1 \oplus cx_2 \oplus d$.

Для определения значений коэффициентов вычислим левую и правую части для четырех возможных наборов входных переменных:

1) $x_1 = 0, x_2 = 0;\ 1 = d;\ d = 1$;
2) $x_1 = 0, x_2 = 1;\ 0 = c \oplus d = c \oplus 1;\ c = 1$;

3) $x_1 = 1,\ x_2 = 0;\ 0 = b \oplus d = b \oplus 1;\ b = 1;$
4) $x_1 = 1,\ x_2 = 1;\ 1 = a \oplus b \oplus c \oplus d = a \oplus 1 \oplus 1 \oplus 1;\ a = 0.$

Отсюда $x_1 x_2 \vee \overline{x_1}\,\overline{x_2} = x_1 \oplus x_2 \oplus 1$.

2.5.3. Замыкание и замкнутые классы

Замыканием множества функции M называют множество всех булевых функций, представляемых в виде формул через функции множества M. Замыкание множества M будем обозначать $[M]$. Очевидно, что $M \subseteq [M]$.

Множество M называют ***замкнутым классом***, если при замыкании M не происходит его дальнейшее расширение, т. е. $[M] = M$.

Ответ на вопрос, какую систему булевых функций можно считать функционально полной, дают две теоремы.

Теорема о функциональной полноте. Для того чтобы система функций B была полной, необходимо и достаточно, чтобы она не содержалась полностью ни в одном из пяти замкнутых классов: T_0, T_1, S, M, L, где T_0 — класс замкнутых булевых функций, сохраняющих константу 0, т. е. функций, для которых $f(0, 0, ..., 0) = 0;$; T_1 — замкнутый класс булевых функций, сохраняющих константу 1, т. е. таких функций, что $f(1, 1, ..., 1) = 1;$ S — замкнутый класс всех самодвойственных функций, т. е. функций, для которых $\overline{f}(\overline{x_1}, \overline{x_2}, ..., \overline{x_n}) = f(x_1, x_2, ..., x_n)$; M — замкнутый класс всех монотонных функций; L — замкнутый класс всех линейных функций.

Функция $f(x_1, x_2, ..., x_n)$ называется *монотонной*, если из условия $\alpha \le \beta$ следует $f(\alpha) \le f(\beta)$, где α и β — наборы переменных $\alpha = (\alpha_1, \alpha_2, ..., \alpha_n)$, $\beta = (\beta_1, \beta_2, ..., \beta_n)$ и введено отношение порядка $\le$, означающее $\alpha \le \beta$, если $\alpha_1 \le \beta_1, \alpha_2 \le \beta_2, ..., \alpha_n \le \beta_n$.

Линейной функцией называется функция, у которой полином Жегалкина имеет вид $c_0 \oplus c_1 x_1 \oplus c_2 x_2 \oplus ..... \oplus c_i x_i \oplus ..... \oplus c_n x_n$, где c_0 и c_i — коэффициенты (равны 1 или 0). В линейной функции отсутствует произведение переменных.

Существует еще одно, близкое по содержанию определение теоремы о полноте.

Теорема Поста. Система $B = \{f_1, f_2, ..., f_n\}$ является *полной* тогда, когда выполняются следующие условия:

1) $f_i \notin T_0$;
2) $f_j \notin T_1$;
3) $f_k \notin S$;
4) $f_m \notin M$;
5) $f_l \notin L$.

То есть хотя бы одна функция не должна принадлежать какому-либо замкнутому классу, так как из функций замкнутого класса нельзя построить функцию, не входящую в данный класс.

Примеры функционально полных базисов. Класс N функций из E_2 называется *предполным*, если он неполный, а для любой функции f, такой, что $f \in E_2$, но $f \notin N$, класс $N \cup \{f\}$ — полный.

Система B называется *полной*, если любая функция $f \in E_2$ может быть представлена в виде суперпозиций функций этой системы, и она называется *базисом*, если теряется полнота при удалении хотя бы одной функции.

Состав систем, которые относятся к базису, можно определить на основе табл. 2.19.

Таблица 2.19

N	Функция	T_0	T_1	L	S	M
1	$x_1 \wedge x_2$	—	—	+	+	—
2	$x_1 \oplus x_2$	—	+	—	+	+
3	$x_1 \vee x_2$	—	—	+	+	—
4	$x_1 \sim x_2$	+	—	—	+	+
5	$\bar{x}$	+	+	—	—	+
6	$x_1 \to x_2$	+	—	+	+	+
7	$x_1 \mid x_2$	+	+	+	+	+
8	1	+	—	—	+	—

Символом «+» обозначается факт непринадлежности функции к замкнутому классу.

Согласно теореме Поста к базису можно отнести минимальный набор функций, строки которого будут содержать хотя бы один символ «+» в каждом столбце (T_0, T_1, S, M, L).

Из табл. 2.19 видно, что к функционально полным системам можно отнести, например, системы $\{\wedge, \bar{}\}, \{|\}, \{\oplus, \to\}$.

ГЛАВА 3

МАТЕМАТИЧЕСКАЯ ЛОГИКА

3.1. Общие сведения о формальных и аксиоматических системах

Во введении уже говорилось, что в основе любой логики лежит определенная система, задающая правила формирования логических выражений, используемые механизмы построения рассуждений и т. п. Системы такого рода называются формальными. Рассмотрим общие положения, которые лежат в основе построения таких систем.

Определение. ***Формальная система*** представляет собой совокупность чисто абстрактных объектов, не связанных с внешним миром, в которой представлены правила оперирования множеством символов только в синтаксической трактовке без учета смыслового содержания.

Всякая формальная система строится на основе формализованного языка (как средства формирования и изложения выражений, имеющих смысл в данной теории) и совокупности теорем. Так же как в естественном языке, средством выражения мысли является предложение, построенное по определенным правилам, в математике средством выражения является формула, построенная из заданного набора символов. В формальной теории все формулы доказываются.

Под *теоремой* в формальной системе понимают высказывание, истинное в данной системе, это некоторое обоснованное и строгое утверждение, которое строится на основе определенных логических правил и доказательства. *Доказательство* — это способ получения одних выражений из других с помощью операций над символами и построения обоснованной аргументации, следствием которой и является теорема.

При построении любой формальной теории (системы) в качестве исходных посылок всегда используются некоторые неопределяемые термины и аксиомы. Неопределяемые термины — это те термины и понятия, смысл и содержание которых считается уже известным, через

них вводятся все новые понятия и термины. Совершенно аналогично вводится некоторая часть постулатов (формул), которые, как считается в данной теории, не требуют доказательства. Обычно это утверждения, правильность которых не вызывает сомнения, и они принимаются как очевидные истины. Такие выражения (формулы) называют *аксиомами*, а системы, в основе построения которых лежит использование аксиом, называются *аксиоматическими системами*.

Формирование строгой формальной теории осуществляется в следующем порядке:

1. Задается конечное множество символов, которые образуют алфавит формальной системы.

2. Устанавливаются процедуры построения формул формальной системы.

3. Устанавливается множество аксиом, т. е. формул, истинность которых не требует доказательства. Обычно к ним относят те утверждения, которые полагаются очевидными по самой природе рассматриваемых понятий.

4. Устанавливается конечное множество правил вывода, которые позволяют получать новые формулы из некоторого множества известных формул. В общем случае эти правила могут быть представлены в виде $H_1 \& H_2 \& \ldots \& H_m \to M$, что означает: из множества истинных формул $H_1, H_2, \ldots, H_m$, указанных в левой части выражения, следует истинность формул правой части выражения. Говорят, что формула M, полученная в результате применения правила вывода, выводима в данной теории. Таким образом, правила вывода позволяют, последовательно применяя их, получать новые истинные формулы или теоремы из аксиом и ранее выведенных формул.

Теперь с учетом понятия правила вывода уточним понятия формального доказательства и теоремы.

Определение. ***Формальным доказательством*** или просто ***доказательством*** называется последовательность формул $M_1, M_2, \ldots, M_n$ такая, что каждая формула M_i является либо аксиомой, либо выводима из предшествующих ей формул.

Определение. Формула t называется ***теоремой***, если существует доказательство, в котором она является последней.

Задаваемые при описании формальной системы правила вывода называют также ***правилами вывода заключений***, т. е. они позволяют определить, является ли данная формула теоремой данной формальной системы или нет.

Различают два типа правил вывода.

1. Правила, применяемые к формулам, рассматриваемым как единое целое, в этом случае их называют ***продукционными правилами***.

Пример. $x < y$ и $y < z \to x < z$ — это продукция с двумя посылками.

2. Правила, которые могут применяться к любой отдельной части формулы, причем сами эти части являются формулами, входящими в состав формальной системы. В этом случае их называют ***правилами переписывания***.

Пример. $x - x = 0$, это выражение имеет смысл при любом значении входящей в него в качестве подвыражения переменной x.

Определение. ***Правило подстановки*** заключается в замещении всех вхождений какой-либо переменной на формулу из формальной системы, которая не содержит этой переменной.

Пример. Рассмотрим формальную систему следующего вида.

1. Алфавит = $\{a, b, w\}$.

2. Формулы — символ или последовательность символов a, b или w.

3. Аксиома awa.

4. Правило вывода: $c_1\, w\, c_2 \to\ b\, c_1\, w\, c_2\, b$ (продукция).

Символы c_1 и c_2 не принадлежат алфавиту формальной системы (ФС), они служат посредниками для формализации правил вывода. То есть c_1 и c_2 обозначают какие-либо последовательности символов a или b формальной системы и могут быть замещены любыми последовательностями символов a или b.

Учитывая способ образования формул, можно сказать, что незамещаемые символы a и b называются константами, а символ w — оператором.

Из определения ФС вытекает и способ получения допустимых формул, т. е. формул, выводимых согласно правилу вывода путем его последовательного применения к аксиоме:

awa
$bawab$
$bbawabb$
$bbbawabbb$ и т. д.

Определение. Формальная система называется ***разрешимой***, если существует хорошо определенный способ действия, который за конечное число шагов для любой формулы формальной системы позволяет определить, выводима она в данной системе или нет. Сам способ действий, если он существует, называют ***процедурой разрешения***.

Так как формальные системы всегда представляют собой модель какой-то реальности, то возникает вопрос об установлении взаи-

мосвязи между объектами формальной системы и этой реальностью, для этого вводится понятие интерпретации.

Определение. ***Интерпретация*** представляет собой распространение исходных положений какой-либо формальной системы на реальный мир. Интерпретация придает смысл каждому символу формальной системы и устанавливает взаимно однозначное соответствие между символами формальной системы и реальными объектами. Теоремы формальной системы, будучи интерпретированы, становятся после этого утверждениями в обычном смысле слова, и в этом случае уже можно делать выводы об их истинности или ложности.

Следует отметить, что при интерпретации речь идет о замыкании или логическом завершении математического подхода, который в общем случае можно описать в виде следующей последовательности действий:

— вначале изучается реальность и конструируется некоторое абстрактное представление о ней, т. е. некоторая формальная система;

— затем строится доказательство теорем формальной системы. Вся польза и удобства формальных систем заключаются в их абстрагировании от конкретной реальности. Благодаря этому одна и та же формальная система может служить моделью многочисленных конкретных ситуаций;

— происходит возвращение к начальной точке всего построения и осуществляется интерпретация теорем, полученных при формализации.

Замечание. Изучение аксиом и теорем как абстрактных выражений, представленных в некоторой форме, называется *синтаксическим изучением* ***аксиоматических систем*** (АС); изучение и рассмотрение смысла этих выражений называется *семантическим изучением* АС. Отметим, что с синтаксическим аспектом АС и связано понятие формальной системы.

Формальную теорию часто называют исчислением. Под исчислением понимают формальное представление теории, которое позволяет оперировать с объектами без учета формального смысла выражений.

В рамках создания формальной системы, как правило, решаются (должны быть решены) следующие проблемы.

1. Проблема противоречивости. Логическое исчисление называется *непротиворечивым*, если в нем недоказуемы никакие две формулы, из которых одна является отрицанием другой.

2. Проблема полноты. Система исчисления считается полной, если любая теорема теории может быть доказана или опровергнута.

3. Проблема независимости аксиом. Для начала введем понятие независимой аксиомы. Аксиома называется *независимой*, если она не может быть выведена из других аксиом. Система аксиом исчисления называется *независимой*, если каждая аксиома в ней независима.

4. Проблема разрешимости. Система исчисления называется разрешимой, если существует алгоритм, позволяющий по каждому утверждению выяснить, является оно истинным или ложным.

3.2. Исчисление высказываний

Определение. ***Исчисление высказываний*** (ИВ), т. е. логика высказываний,— это формальная система, интерпретацией которой является алгебра высказываний. Под **высказыванием** понимается **повествовательное** предложение, о котором можно сказать, что оно истинно или ложно. Основной задачей исчисления высказываний является порождение общелогических законов — тождественно истинных высказываний, т. е. высказываний (в том числе составных), которые всегда истинны независимо от входящих в них элементарных высказываний.

Как и любая формальная система, исчисление высказываний строится на основе четырех основных процедур: задания алфавита, установления правил построения формул, аксиом и правил вывода. Определим все эти четыре компонента.

1. ***Алфавит*** состоит из символов трех категорий:

а) бесконечного счетного множества высказываний (или переменных высказываний), которые обычно обозначаются буквами: x, y, z, a, b, c, x_1, x_2 и т. д.;

б) логических операторов (или логических связок), которые обозначают символы логических операций ($\vee$, &, $\rightarrow$ и т. д.);

в) открывающихся и закрывающихся скобок: ().

Других символов в ИВ нет.

2. ***Правила построения формул.*** Для обозначения формул обычно используют заглавные буквы латинского алфавита. Эти буквы не являются символами исчисления и служат для условного обозначения формул. Формулы в ИВ определяются следующим образом.

Формулы в исчислении высказываний однозначно получаются с помощью правил, которые описываются базисом и индуктивным шагом:

базис: всякое высказывание есть формула;

индуктивный шаг: если X и Y — формулы, то $\overline{X}$, $(X \vee Y)$, $(X \rightarrow Y)$, $(X \,\&\, Y)$ и т. д. — также формулы.

Никакая другая последовательность символов не является формулой.

Пример. Если x, y, z — формулы в соответствии с правилом базиса, то $(x \to y)$, $(x \,\&\, z)$ — формулы в соответствии с правилом индуктивного шага. Очевидно, что не будут формулами: $\&x$, $(x \vee z$, так как они не удовлетворяют указанным правилам (в первом случае в бинарной операции используется один операнд, а во втором — отсутствует закрывающая скобка).

С введением понятия формулы вводится и понятие *подформулы* или части формулы, делается это следующим образом.

1. Подформулой элементарной формулы является только она сама.

2. Если $\overline{X}$ — формула, то ее подформулами будут: она сама, X и все подформулы X.

3. Обозначим w — любую из логических операций $(\vee, \&, \to)$, кроме отрицания. Тогда если (XwY) — формула, то ее подформулы: она сама, формулы X и Y и все подформулы X и Y.

Пример. Пусть задана формула $\left(x \vee \overline{y}\right) \to \left(\overline{\overline{z} \,\&\, y}\right)$, определим ее подформулы и глубину их вложенности.

Представим решение в табличной форме (табл 3.1)

Таблица 3.1

Подформула	Глубина
$\left(x \vee \overline{y}\right) \to \left(\overline{\overline{z} \,\&\, y}\right)$	0
$\left(x \vee \overline{y}\right), \left(\overline{\overline{z} \,\&\, y}\right)$	1
$x, \overline{y}, \left(\overline{z} \,\&\, y\right)$	2
$y, \overline{z}$	3
z	4

Кроме табличной формы каждая правильная формула может быть представлена в виде дерева, ветви которого — исходные и промежуточные формулы (рис. 3.1).

Из примера видно, что на самом низком уровне находятся только элементарные формулы, однако они могут быть и на более высоких уровнях.

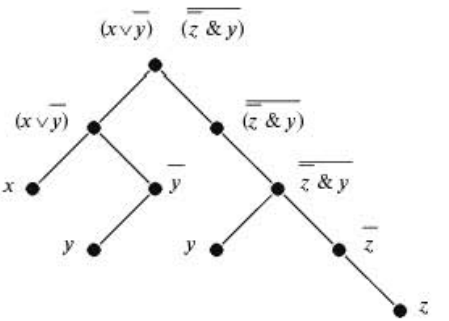

Рис. 3.1

Для упрощения записи формул ИВ используются те же соглашения, что и в алгебре логики, которые позволяют учитывать приоритеты операций и избавиться от лишних скобок.

Пример. $\big((X \,\&\, Y) \vee C\big) = X \,\&\, Y \vee C$;

$\overline{(X \vee Y)} = \overline{X} \vee \overline{Y}$.

3. ***Аксиомы.*** Существует несколько вариантов подбора аксиом как исходных тождественно истинных формул. Эти наборы эквивалентны в том смысле, что они определяют один и тот же класс выводимых формул. Один из наиболее часто применяемых вариантов включает 11 аксиом:

1) $x \to \big(y \to x\big)$;
2) $\big(x \to \big(y \to z\big)\big) \to \big(\big(x \to y\big) \to \big(x \to z\big)\big)$;
3) $x \,\&\, y \to x$;
4) $x \,\&\, y \to y$;
5) $\big(z \to x\big) \to \big(\big(z \to y\big) \to \big(z \to x \,\&\, y\big)\big)$;
6) $x \to x \vee y$;
7) $y \to x \vee y$;
8) $\big(x \to z\big) \to \big(\big(y \to z\big) \to \big(x \vee y \to z\big)\big)$;
9) $\big(x \to y\big) \to \big(\overline{y} \to \overline{x}\big)$;
10) $x \to \overline{\overline{x}}$;
11) $\overline{\overline{x}} \to x$.

Тождественную истинность аксиом можно проверить либо прямым вычислением значения формулы на каждом наборе, либо приведением их к константе 1 путем эквивалентных преобразований, применяемых в булевой алгебре. Например, докажем методом эквивалентных преобразований истинность аксиомы (2):

$$\big(x\to(y\to z)\big)\to\big((x\to y)\to(x\to z)\big)=\overline{\overline{x}\vee\overline{y}\vee\overline{z}}\vee\overline{\overline{x}\vee y\vee\overline{x}\vee z}=$$
$$=xy\overline{z}\vee x\overline{y}\vee\overline{x}\vee z=xy\overline{z}\vee\overline{y}\vee\overline{x}\vee z=x\overline{z}\vee\overline{y}\vee\overline{x}\vee z=\overline{z}\vee\overline{y}\vee\overline{x}\vee z=$$
$$=\overline{y}\vee\overline{x}\vee 1=1.$$

4. *Правила вывода.* Правила вывода устанавливают отношения на множестве формул исчисления высказываний. Правила вывода обычно представляются как отношения на множестве формул исчисления высказываний. Над чертой записываются формулы, которые играют роль посылки (уже известные истинные выражения), а под чертой — выводимая формула, истинность которой утверждается данным правилом. Она называется следствием или заключением.

В исчислении высказываний используются два правила вывода:

1) *правило заключения* (*modus ponens*). Если A и $A\to B$ — это выводимые формулы, то B также является выводимой формулой. Записывается это правило так:

$$\frac{A,\ A\to B}{B};$$

2) *правило подстановки*. Его формальная запись имеет вид

$$\frac{A}{A\big(x_1,x_2,\ldots,x_n\,\|B_1,B_2,\ldots,B_n\big)},$$

где $A, B_1, B_2,\ldots,B_n$ — это формулы, $x_1,x_2,\ldots,x_n$ — попарно различные переменные высказывания; через запись $A\big(x_1,x_2,\ldots,x_n\,\|B_1,B_2,\ldots,B_n\big)$ обозначен результат одновременной замены всех вхождений переменных $x_1,x_2,\ldots,x_n$ в A на формулы $B_1,B_2,\ldots,B_n$.

Справедливость правил вывода исчисления высказываний подтверждается применением методов булевой алгебры.

В частности, тождественную истинность выводимой формулы B можно установить следующим образом: если A и $A\to B$ — тождественно истинные формулы, т. е. $A=1$ и $A\to B=1$, следовательно $\overline{A}\vee B=1$. Так как в последнем выражении $\overline{A}=0$, а значение логической суммы равно 1, то B должно быть равно 1 (т. е. быть истинным).

Правило подстановки сохраняет тождественную истинность выражения, так как любая подстановка в тождественно истинную формулу также дает тождественно истинную формулу.

Используя два приведенных правила, можно формально порождать новые формулы без учета семантического смысла используемых выражений.

Кроме двух приведенных правил вывода, можно получить и другие правила, позволяющие строить новые доказуемые формулы. Но так как они реализуются с помощью правила подстановки и заключения, то получили название *производных правил* вывода. Наиболее распространенные из них следующие.

Правило сложного заключения. Если $A_1, A_2, \ldots, A_n$ — формулы и $A_1 \to \left(A_2 \to \left(A_3 \to \left(\ldots \left(A_m \to B\right)\ldots\right)\right)\right)$ — теорема, то формула B — также теорема.

Правило двойного отрицания. Если $A \to \overline{\overline{B}}$ и $\overline{\overline{A}} \to B$ — теоремы, то будет теоремой и формула $A \to B$, иначе $\dfrac{A \to \overline{\overline{B}}}{A \to B}$ и $\dfrac{\overline{\overline{A}} \to B}{A \to B}$.

Правило силлогизма (*замыкания*). Если $A \to B$ и $B \to C$ — теоремы, то $A \to C$ — также теорема, иначе $\dfrac{A \to B, B \to C}{A \to C}$.

Правило композиции. Если $A \to B$ — теорема, то $\overline{B} \to \overline{A}$ — также теорема, иначе $\dfrac{A \to B}{\overline{\overline{B}} \to \overline{\overline{A}}}$.

Правила вывода можно рассматривать и как результат логического анализа некоторых человеческих рассуждений. Рассмотрим примеры для приведенных выше правил.

1. Правилу заключения соответствует следующая схема рассуждений.

Исходные посылки. Если данный многоугольник правильный ($A = 1$), то в него можно вписать окружность ($A \to B$). Возьмем правильный многоугольник ($A = 1$).

Вывод. В данный многоугольник можно вписать окружность ($B = 1$).

2. Интерпретацией правила силлогизма может быть следующая схема рассуждений.

Исходные посылки. Если треугольник равнобедренный ($A = 1$), то две его стороны равны ($B = 1$). Если две стороны треугольника равны ($B = 1$), то два его угла равны ($C = 1$).

Вывод. Если треугольник равнобедренный, то два его угла равны ($A \to C$).

3. Правило композиции иллюстрируется следующей схемой рассуждений.

Исходная посылка. Если число рациональное ($A = 1$), то оно может быть представлено в виде отношения двух целых чисел ($B = 1$).

Вывод. Если число не может быть представлено в виде отношения двух целых чисел, то оно нерациональное ($\overline{B} \to \overline{A}$).

Как уже отмечалось, формулы исчисления высказываний можно интерпретировать как формулы алгебры высказываний (АВ). Для этого следует переменные ИВ трактовать как переменные АВ, т. е. переменные, которые могут принимать только значения 0 и 1 (ложь и истина). Все логические операции ($\vee, \&, \to$ и т. д.), которые используются в ИВ, интерпретируются так же, как и в АВ, т. е. на основе тех же самых таблиц истинности. Очевидно, что между формулами ИВ и АВ существует взаимнооднозначное соответствие. Однако формализма, реализованного в АВ, не всегда достаточно для построения доказательств в ИВ, поэтому существует и множество других методов.

Перед рассмотрением методов установления факта общезначимости формул рассмотрим используемые в исчислении высказываний следующие термины и определения:

1) формула *выполнима*, если она может принимать значение «истина» (например, $p \& q$, $\overline{p}$, $p \vee q$);

2) формула *невыполнима*, если ни при каких значениях, составляющих ее высказываний, она не может быть истинной (например, $p \& \overline{p}$);

3) формула *общезначима*, если она принимает значение «истина» независимо от истинности ее составляющих (например, $p \vee \overline{p}$);

4) формула *нейтральна*, если она не общезначима и не является невыполнимой.

5. ***Тавтологиями*** называются общезначимые формулы. Если формула $A \equiv 1$, т. е. A — тавтология, то это можно записать $\models A$.

Логический вывод на основе множества гипотез можно определить следующим образом.

Пусть E — это множество формул, тогда запись $E \models A$ означает, что если все формулы из E истинны, то будет истинной формула A. В этом случае A называется логическим следствием из E.

Если $E = \{H_1, H_2, \ldots, H_n\}$ и $E \models A$, то можно записать $\{H_1, H_2, \ldots, H_n\} \models A$. Формулы H_i называются *гипотезами*, а формула A — *заключением*.

Определение. ***Принцип дедукции*** состоит в следующем. Формула A является логическим следствием конечного множества E тогда и только тогда, когда $E \cup \{\overline{A}\}$ содержит невыполнимые формулы.

В силу того что для высказываний справедливы все свойства логических операций, которые были определены в алгебре логики, то, ис-

пользуя законы де Моргана, можно ввести понятие ***прямой и обратной дедукции.***

Действительно, если A есть логическое следствие гипотез $H_1, \ldots, H_n$, то, учитывая сформулированный принцип дедукции, можно считать справедливой следующую запись: H_1 & H_2 & … & H_n & $\overline{A} = 0$. Это правило называется *правилом прямой дедукции*. Возьмем отрицание от этого выражения, тогда по правилу де Моргана получим $\overline{H}_1 \vee \overline{H}_2 \vee \ldots \vee \overline{H}_n \vee A = 1$. Это правило называется *правилом обратной дедукции.*

Так как для распознавания выполнимости и общезначимости формул существует множество различных методов, рассмотрим некоторые из них.

Замечание. Следует учитывать, что не всегда алгоритм дедукции оказывается более эффективным по сравнению с тривиальным подходом, который основывается на алгебраическом (использующем эквивалентные преобразования из АВ) представлении вывода.

3.3. Методы, используемые для определения общезначимости формул исчисления высказываний

3.3.1. Алгоритм редукции

Алгоритм редукции позволяет доказывать общезначимость формул исчисления высказываний путем приведения к абсурду. Рассмотрим на примере. Пусть требуется доказать общезначимость следующей формулы:

$$((p \& q) \to r) \to (p \to (q \to r)).$$

Предположим, что при некоторой интерпретации эта формула принимает значение «ложь». Из определения функции импликации известно, что значение «ложь» она принимает только в том случае, когда посылка истинна, а заключение ложно. Учитывая указанное свойство, представляем исходную формулу в виде двух интерпретаций следующего вида:

$$p \to (q \to r) = 0; \qquad (3.1)$$

$$(p \& q) \to r = 1 . \qquad (3.2)$$

Применив ранее использованные рассуждения к (3.1), получим следующие значения переменных: $p = 1$, $q = 1$, $r = 0$. Если подставить полученные значения в (3.2), то получится противоречие. Значит, предположение о том, что существует некоторая интерпретация, при которой исходная формула принимает значение «ложь», неверно, и это означает общезначимость исходной формулы.

Пример. Используя алгоритм редукции, требуется доказать общезначимость формулы $(a \to b) \to ((c \to a) \to (c \to b))$.

Пусть при некоторой интерпретации $(a \to b) \to ((c \to a) \to (c \to b)) = 0$.

Это возможно только, если

$$a \to b = 1; \qquad (3.3)$$
$$(c \to a) \to (c \to b) = 0. \qquad (3.4)$$

Тогда из (3.3) следует, что возможна одна из следующих комбинаций значений переменных a и b:

$$a = 0, \quad \longleftrightarrow \quad b = 0;$$
$$a = 1, \quad \longleftrightarrow \quad b = 1;$$
$$a = 0, \quad \longleftrightarrow \quad b = 1.$$

Из (3.4) следует $(c \to a) = 1$, это означает, что возможны следующие значения c и a:

$$c = 0, \quad \longleftrightarrow \quad a = 0;$$
$$c = 1, \quad \longleftrightarrow \quad a = 1;$$
$$c = 0, \quad \longleftrightarrow \quad a = 1.$$

Из $c \to b = 0$ имеем $c = 1, b = 0$. Это единственно допустимые для c и b значения, при которых формула принимает значение «ложь».

Сопоставляем полученные результаты с ранее рассмотренными возможными значениями переменных. Оказывается, что при $c = 1$ единственное допустимое значение для a — это $a = 1$, а при $b = 0$ единственное допустимое значение $a = 0$. То есть переменная a должна принимать взаимно исключающие значения, что невозможно. Следовательно, предположение о существовании интерпретации, при которой формула $(a \to b) \to ((c \to a) \to (c \to b))$ принимает значение «ложь», неверно, и это означает ее общезначимость.

3.3.2. Метод резолюций

Для порождения логических следствий будет использована очень простая схема рассуждений. Пусть A, B и X — формулы. Предположим,

что две формулы $(A \vee X)$ и $(B \vee \bar{X})$ истинны. Тогда если X — истина, то следует, что B — тоже истина. Наоборот, если X — ложь, то следует, что A — истина. Получаем правило $\{A \vee X, B \vee \bar{X}\} \models (A \vee B)$, которое можно записать в виде $\{\bar{X} \to A, X \to B\} \models (A \vee B)$.

В том частном случае, когда X — высказывание, а A и B — дизъюнкты, это правило называется ***правилом резолюций.***

Рассмотрим, каким образом это правило может быть использовано для построения доказательства.

В методе резолюций применяется также приведенное выше правило прямой дедукции: для того чтобы доказать, что формула C является логическим следствием из гипотез H_1, H_2, ..., H_n, следует доказать, что $H_1 \& H_2 \& ... \& H_n \& \bar{C} = 0$. Так как левая часть последнего равенства представляет собой конъюнкцию, для его выполнения достаточно доказать равенство нулю любой входящей в уравнение формулы. Таким образом, для доказательства выводимости исходной формулы C необходимо доказать, что в множестве $\{H_1, H_2, ..., H_n, \bar{C}\}$ имеется хотя бы одна невыполнимая формула. Для этого каждый элемент указанного множества рассматривается как элементарная дизъюнкция (дизъюнкт). Применение метода резолюций предусматривает порождение новых дизъюнктов на основе следующей леммы, в основе которой лежит правило резолюций.

Лемма. Пусть S_1 и S_2 — дизъюнкты нормальной формы S, а l — литера. Если $l \in S_1$ и $\bar{l} \in S_2$, то дизъюнкт $r = (S_1 \setminus \{l\}) \cup (S_2 \setminus \{\bar{l}\})$ является логическим следствием нормальной формы S.

Следствие. Нормальные формы S и $S \cup \{r\}$ эквивалентны.

Замечание. Дизъюнкт r называется *резольвентой* дизъюнктов S_1 и S_2.

Для доказательства приведенных утверждений о выполнимости формулы C необходимо, как отмечалось выше, доказать, что хотя бы один дизъюнкт из рассматриваемого множества исходных и порожденных дизъюнктов тождественно равен нулю. В этом случае говорят, что получен пустой дизъюнкт.

Таким образом, принцип резолюций заключается в использовании факта, что множество дизъюнктов S не выполнимо, если пустой дизъюнкт является логическим следствием из него. Для доказательства невыполнимости множества S необходимо строить логические следствия из него до тех пор, пока не будет получен пустой дизъюнкт или дальнейшее построение логических следствий окажется невозможным.

Метод резолюций выгодно отличается от других методов тем, что он дает возможность использовать средства автоматического доказательства, применяемые в логическом программировании.

Прежде чем перейти к описанию алгоритма, напомним несколько терминов, известных из курса дискретной математики, которые будут использованы при описании алгоритма метода резолюций.

Определение. ***Литера*** — это элементарное высказывание или его отрицание, например $x, \bar{y}, z$.

Определение. ***Дизъюнкт*** или ***элементарная дизъюнкция*** — это совокупность различных литер, связанных дизъюнктивной связью, например $x \vee \bar{y}, \bar{x} \vee y \vee \bar{z}, x$.

Определение. ***Конъюнктивной нормальной формой*** (КНФ) некоторой формулы называется равносильная ей формула, представляющая конъюнкции элементарных дизъюнкций, например $\left(x \vee \bar{y}\right) \& \\ \& \left(\bar{x} \vee z \vee y\right) \& \left(x \vee z\right)$.

Так как для того чтобы выражение в форме КНФ было тождественно истинным, необходимо и достаточно, чтобы был истиной каждый дизъюнкт, в него входящий, то при построении логического вывода можно перейти от КНФ к множеству дизъюнктов, в котором каждый элемент множества имеет значение «истина».

Определение. ***Пустой дизъюнкт*** — это такой дизъюнкт, значение которого тождественно ложно.

Итак, невыполнимость формул, из которых формируется конечное множество дизъюнктов S, доказывается с помощью следующего алгоритма.

Шаг 1. *Проверка множества S на невыполнимость.* Если пустой дизъюнкт принадлежит множеству S (он либо может присутствовать изначально, либо получается из-за того, что в множестве одновременно присутствует некоторая литера и ее отрицание), это означает, что множество S невыполнимо и алгоритм свою работу закончил. Иначе переходим на шаг 2.

Шаг 2. *Построение резольвенты.* Выбираем l, S_1, S_2 такие, что $l \in S_1$ и $\bar{l} \in S_2$, и вычисляем резольвенту r. Если невозможно выбрать l, S_1, S_2, соответствующие указанным условиям, то алгоритм свою работу закончил и можно сказать, что исходное множество выполнимо. Иначе переходим на шаг 3.

Шаг 3. *Обновление множества дизъюнктов.* Заменяем множество дизъюнктов S на $S \cup \{r\}$, т. е. добавляем к существующим дизъюнктам

новый дизъюнкт — резольвенту, полученную на предыдущем шаге. После чего переходим на шаг 1.

Пример. Доказать, используя метод резолюций, невыполнимость множества дизъюнктов $S = \{p \vee q, p \vee r, \bar{q} \vee \bar{r}, \bar{p}\}$. Пронумеруем дизъюнкты. Это делается для того, чтобы при построении резольвенты можно было сослаться на дизъюнкты, которые для этого использовались:

1) $p \vee q$;
2) $p \vee r$;
3) $\bar{q} \vee \bar{r}$;
4) $\bar{p}$.

Порождаем логические следствия, при порождении следствия будем указывать номера участвовавших в построении дизъюнктов:

5) $p \vee \bar{r}$ (1,3);
6) q (1,4);
7) $p \vee \bar{q}$ (2,3);
8) r (2,4);
9) p (2,5);
10) $\bar{r}$ (3,6);
11) $\bar{q}$ (3,8);
12) $\bar{r}$ (4,5);
13) $\bar{q}$ (4,7);
14) пустой дизъюнкт (4,9).

Замечание. Алгоритм проверки невыполнимости недетерминирован. Вообще говоря, возможен не один вариант выбора значений для l, S_1 и S_2. В приведенном примере дизъюнкты выбирались в лексико-графическом порядке номеров. Такая стратегия не оптимальна, так как некоторые резольвенты оказались не нужны, а также вычислялись более одного раза. Этот же пример с минимумом резолюций будет выглядеть так:

5) q (1,4);
6) r (2,4);
7) $\bar{q}$ (3,6);
8) пустой дизъюнкт (5,7).

Ясно, что принятая стратегия может существенно влиять на процесс выполнения алгоритма. Тем не менее существуют два свойства, не зависящие от используемой стратегии.

Свойство 1. Если множество S не содержит ни одной пары дизъюнктов, допускающих резольвенту, то оно выполнимо (за исключением случая, когда оно содержит пустой дизъюнкт).

Свойство 2. Если выполнение этого алгоритма закончилось нормально после порождения пустого дизъюнкта, то установлена невыполнимость исходного множества S.

В заключение рассмотрим несколько примеров применения метода резолюций в логике высказываний.

Пример. Доказать, используя метод резолюций, что S является логическим следствием множества гипотез H, где $H = \left\{\bar{a} \vee (b \to c), \overline{c \,\&\, d} \vee e, f \vee \left(\overline{\bar{d} \vee e}\right)\right\}$, а $S = \bar{a} \vee \bar{b} \vee f$. Сначала преобразуем множество гипотез в множество дизъюнктов:

1) $\bar{a} \vee (b \to c) = \bar{a} \vee \bar{b} \vee c$;
2) $\overline{c \,\&\, d} \vee e = \bar{c} \vee \bar{d} \vee e$;
3) $f \vee \left(\overline{\bar{d} \vee e}\right) = f \vee d\bar{e} = (f \vee d)\left(f \vee \bar{e}\right)$;
4) $\bar{S} = \overline{\bar{a} \vee \bar{b} \vee f} = a \,\&\, b \,\&\, \bar{f}$.

Для доказательства $H \mid = S$ необходимо и достаточно доказать невыполнимость следующего множества дизъюнктов:

$H = \{\bar{a} \vee \bar{b} \vee c, \bar{c} \vee \bar{d} \vee e, f \vee d, f \vee \bar{e}, a, b, \bar{f}\}$:

1) $\bar{a} \vee \bar{b} \vee c$;
2) $\bar{c} \vee \bar{d} \vee e$;
3) $f \vee d$;
4) $f \vee \bar{e}$;
5) a;
6) b;
7) $\bar{f}$;
8) d (3 – 7);
9) $\bar{b} \vee c$ (1 – 5);
10) c (6 – 9);
11) $\bar{e}$ (4 – 7);
12) $\bar{c} \vee \bar{d}$ (2 – 11);
13) $\bar{d}$ (10 – 12);
14) пустой дизъюнкт (8 – 13).

Пример. Пусть дано множество утверждений, сформулированных на естественном языке, и некоторое заключение, которое из них следует. Требуется превратить их в множество высказываний и доказать справедливость рассуждений, используя метод прямой дедукции. Набор рассуждений следующий:

1) если пойти на первую пару, то надо встать рано, а если играть в преферанс, то лечь придется поздно;

2) если лечь поздно и рано встать, то спать придется мало;

3) мало спать нельзя.

Заключение: надо или не играть в преферанс, или не идти на первую пару.

Введем следующие обозначения для высказываний:

g — встать рано;

d — играть в преферанс;

c — идти на первую пару;

s — лечь поздно спать;

e — мало спать.

Используя введенные обозначения, перейдем от утверждений к следующему набору гипотез: $H_1 = (c \to g) \& (d \to s)$, $H_2 = s \& g \to e$, $H_3 = \bar{e}$. Следствие примет вид $A = \bar{c} \vee \bar{d}$. При построении доказательства по дедукции в качестве механизма воспользуемся методом эквивалентных преобразований и методом резолюций. Требуется доказать невыполнимость следующего множества: $\{H_1, H_2, H_3, \bar{A}\}$, где $\bar{A} = \overline{\bar{c} \vee \bar{d}} = cd$.

Используя метод эквивалентных преобразований, требуется доказать, что имеет место равенство $cd(c \to g)(d \to s)(sg \to e)\bar{e} = 0$. Докажем, что $cd(c \to g) \& (d \to s)(sg \to e)\bar{e} = cd(\bar{c} \vee g)(\bar{d} \vee s)(\bar{s} \vee \bar{g} \vee e)\bar{e} =$ $= (cd\bar{c} \vee cdg)(\bar{d} \vee s)(\bar{e}\bar{s} \vee \bar{g}\bar{e} \vee e\bar{e}) = (cd\bar{d}g \vee cdgs)(\bar{e}\bar{s} \vee \bar{e}\bar{g}) = cdgs\bar{e}\bar{s} \vee cdgs\bar{e}\bar{g} =$ $= 0$.

Теперь построим доказательство, используя метод резолюций. Для этого приведем имеющиеся гипотезы к форме дизъюнктов:

$H_1 = (c \to g) \& (d \to s) = (\bar{c} \vee g)(\bar{d} \vee s)$;

$H_2 = s \& g \to e = \overline{s \& g} \vee e = \bar{s} \vee \bar{g} \vee e$;

$H_3 = \bar{e}$.

Отрицание следствия будет иметь вид $\bar{A} = \overline{\bar{c} \vee \bar{d}} = cd$, а цепочка доказательства:

1) $\bar{c} \vee g$;

2) $\bar{d} \vee s$;

3) $\bar{s} \vee \bar{g} \vee e$;

4) $\bar{e}$;

5) c;

6) d;
7) $g(1-5)$;
8) $s(2-6)$;
9) $\bar{s} \vee e$ (3–7);
10) e;
11) пустой дизъюнкт (4–10).

3.4. Логика предикатов

3.4.1. Основные понятия логики предикатов

Высказывания в алгебре логики рассматриваются как единый объект с точки зрения истинности или ложности. Структура и содержание высказываний не рассматриваются. Однако на практике для построения полноценного логического вывода важно иметь представление о структуре и содержании используемых в выводе высказываний. Поэтому логика предикатов является, по сути, расширением логики высказывания, которую включает в себя в качестве составной части.

Определение. ***Одноместным предикатом P(x)*** называется произвольная функция переменной x, определенная на множестве M и принимающая значение из множества $\{0,1\}$.

Обобщим это определение.

Определение. ***Предикатом P*** называется n-местная функция, определенная на производном множестве M и принимающая в качестве значений элементы из двухэлементного множества $\{0,1\}$, где 0 и 1 интерпретируются как ложь и истина соответственно. Выражение вида $P(x_1, x_2, \ldots, x_n)$ можно трактовать, что переменные $x_1, x_2, \ldots, x_n$ связаны отношением P.

Используя функциональную форму записи для предикатов, можно сказать, что предикатом $P(x_1, x_2, \ldots, x_n)$ называется функция $P: M^n \to B$, где B — двоичное множество, а M — произвольное множество.

Таким образом, n-местный предикат — это двузначная функция от n аргументов, определенная на произвольном множестве M, принимающая значение 0 или 1 (0 и 1 интерпретируются как ложь и истина соответственно). Область определения M называется предметной областью предиката, а $x_1, x_2, \ldots, x_n$ — предметными переменными.

Предикат может выражать высказывание либо о свойствах объекта, либо об отношении между объектами. Возможность описывать с по-

мощью предикатов не только функции, но и отношения определяется следующим:

а) если a_1, a_2, ..., a_n — элементы множества M, то каждому n-местному отношению R соответствует предикат P такой, что $P(a_1, a_2, ..., a_n) = 1$ тогда и только тогда, когда $(a_1, a_2, ..., a_n) \in R$;

б) всякий предикат $P(x_1, x_2, ..., x_n)$ определяет отношение R такое, что $(a_1, a_2 ..., a_n) \in R$, если и только если $P(a_1, a_2, ... a_n) = 1$. При этом R задает область истинности предиката P.

Предикат можно поставить в соответствие и непрерывной функции типа F: $M^n \to M$. Такой функции можно поставить в соответствие $(n + 1)$-местный предикат P такой, что $P(a_1, a_2, ..., a_n, a_{n+1}) = 1$, если и только если $f(a_1, a_2, ..., a_n) = a_{n+1}$.

Таким образом, в общем случае предикат P — двоичная переменная, т. е. переменное высказывание, истинность которого определяется значениями аргументов $(x_1, x_2, ..., x_n)$, а аргументы x_i — чаще нелогические переменные. После подстановки вместо x_i конкретных элементов множества M предикат $P(a_1, a_2, ..., a_n)$ перестает быть переменной и принимает одно из двух возможных значений (0 или 1).

Пример.

1. Рассмотрим утверждение: x — целое число. Введем предикат I, обозначающий отношение «быть целым числом», тогда в виде предикатного выражения утверждение может быть записано так: $I(x)$.

2. Рассмотрим утверждение $x < y$. Введем предикат S с двумя аргументами, первый из которых меньше второго, тогда $S(x, y)$ соответствует введенному утверждению.

3. Элементы x_i множества M — города. Предикат $P(x)$ устанавливается таким образом: x — это столица Украины. Тогда $P(Воронеж) = 0$, а $P(Киев) = 1$.

4. Задана функция $z = x+y$, где x, y, z — действительные числа. Пусть предикат $P(x, y, z)$ соответствует этой функции. Тогда $P(2, 3, 5) = 1$.

Если вместо переменных в предикат подставить объекты $a_1, a_2, \ldots, a_m \in M$, где M — множество, на котором определено P, то значение $P(a_1, a_2, \ldots, a_m)$ можно рассматривать как истинное или ложное.

Пример (для предикатов, определенных в предыдущем примере). Пусть в обоих случаях предикаты определены на множестве R действительных чисел. Тогда:

1) если $x = 5$, то предикат $I(5) = 1$;
   если $x = 7{,}3$; то $I(7{,}3) = 0$;

2) если $x = 5$; $y = 10{,}5$, то $S(5;\ 10{,}5) = 1$;
если $x = 27{,}1$; $y = 4{,}3$, то $S(27{,}1;\ 4{,}3) = 0$.

Определение. Для предиката $P(x_1, x_2, \ldots, x_n)$ можно выделить множество наборов $b_i = (a_{i1}, a_{i2}, \ldots, a_{in})$ таких, что $a_{i1}, a_{i2}, \ldots, a_{in} \in M$, которые, будучи подставленными в предикат P, приводят его к значению «истина». Объединение всех этих наборов называется ***множеством истинных наборов*** предиката P, обозначим его I_P.

Пример. Для предиката $S(x, y)$, определенного ранее, множество I_P может быть определено следующим образом:

$I_P = \{(x, y) \mid x \in R, y \in R, x < y\}$.

Определение. Предикат $P(x_1, x_2, \ldots, x_n)$ называется ***тождественно истинным***, если его множество I_P образуют все возможные наборы, которые можно определить на множестве M.

Определение. Предикат $P(x_1, x_2, \ldots, x_n)$ называется ***тождественно ложным***, если его множество $I_P = \varnothing$.

Так как предикаты, как и высказывания, могут принимать значения «истина» или «ложь» (1 или 0), то к ним применимы все операции логики высказываний.

Пример. Рассмотрим предикаты $I(x)$ и $S(x, y)$ из предыдущих примеров. Пусть I и S определены на множестве R, тогда:

1) при $x = 3$, $y = 10$: $I(3) = 1$, $S(3,\ 10) = 1$ имеем $I(3)\ \&S(3,\ 10) = 1\&$ $\&1 = 1$, $S(3,\ 10) \to I(3) = 1 \to 1 = 1$;

2) при $x = 7{,}5$, $y = 11$: $I(7{,}5) = 0$, $S(7{,}5;\ 11) = 1$ имеем $I(7{,}5)\&$ $\&S(7{,}5;\ 11) = 0\&1 = 0$, $I(7{,}5) \to S(7{,}5;11) = 0 \to 1 = 1$;

3) задана вычислительная процедура: повторять цикл, пока переменные x и y не станут равными, либо прекратить вычисления после 100 повторений. Область определения: x и y — действительные числа; i — целое число, которое в каждом шаге увеличивается на единицу. Предикат P задается условием: $\overline{(x = y) \vee (i > 100)}$. При этом процедура может задаваться условием: повторять цикл, если $P = 1$.

Кроме логических операций в логике предикатов вводятся *кванторные операции* (или просто *кванторы*). Наличие кванторов в выражении позволяет существенно повысить выразительную мощность предикатных предложений. Кванторов всего два.

1. Квантор всеобщности ($\forall$). Пусть $P(x)$ это предикат, определенный на множестве M. Тогда под выражением $\forall x P(x)$ понимается высказывание, которое истинно для любого элемента $x \in M$. Соответствующее этому высказыванию предложение можно сформулировать так: для любого x выражение $P(x)$ истинно.

2. Квантор существования ($\exists$). Пусть $P(x)$ — это предикат, определенный на множестве M. Тогда под выражением $\exists x P(x)$ понимается высказывание, которое истинно, если существует элемент $x \in M$, для которого $P(x)$ истинно, и ложно в противном случае. Соответствующее ему предложение: существует x, при котором значение $P(x)$ истинно.

Пример. Рассмотрим предикат $I(x)$, означающий x — целое число. Пусть предикат I определен на множестве R действительных чисел. Тогда выражение $\forall x I(x)$ соответствует утверждению: все действительные числа — целые. Очевидно, что это утверждение ложно. Выражение $\exists x I(x)$ соответствует утверждению: существуют действительные числа, являющееся целыми. Это истинное утверждение. Теперь предположим, что предикат I определен на множестве N натуральных чисел. Тогда оба выражения — $\forall x I(x)$ и $\exists x I(x)$ — будут истинными.

Кванторы могут применяться и к предикатам, определенным относительно n переменных, при этом к одному предикату может быть применено несколько кванторов, относящихся к разным переменным. Порядок следования кванторов существенно влияет на истинность предиката.

Пример. Пусть предикат $P(x,y)$ — «$x \le y$» определен на множестве N.

Рассмотрим предикатные выражения, приведенные в табл. 3.2

Таблица 3.2

Предикатная запись	Предложение	Значение истинности
$\exists x \forall y P(x,y)$	Существует x, который меньше любого y	1
$\exists x \exists y P(x,y)$	Существуют такие x и y, что $x \le y$	1
$\forall x \forall y P(x,y)$	Для любого x и любого y имеет место $x \le y$	0

Входящие в предикатное выражение переменные могут быть *связанными* или *свободными*, и это зависит от того, каким образом определена *область действия квантора*, входящего в формулу. Например, для выражения вида $\forall x \exists y (P(x,y) \to \forall z R(x,z))$ область действия кванторов определена следующим образом: для квантора $\forall x$ — $\exists y (P(x,y) \to \forall z R(x,z))$, для квантора $\exists y$ — $P(x,y) \to \forall z R(x,z)$, а для квантора $\forall z$ — $R(x,z)$.

Определение. ***Вхождение*** переменной x в формулу называется ***связанным*** тогда и только тогда, когда оно совпадает с вхождением в квантор $(\forall x)$ или $(\exists y)$ и находится в области действия квантора. Вхождение переменной в формулу свободно тогда и только тогда, когда оно не является связанным.

Определение. ***Переменная свободна*** в формуле, если хотя бы одно ее вхождение в эту формулу свободно. Отметим, что переменная в формуле может быть свободной и связанной одновременно.

Определение. ***Переменная*** называется ***квантифицированной***, если она используема в кванторных операциях, т. е. $\forall x$ или $\exists x$.

Пример.

$P(x)$ — x свободная;

$\forall x P(x)$, $\exists x P(x)$ — x связанная;

$\exists x P(x, y)$ — x связанная, y свободная;

$\exists x \forall y P(x, y)$ — x и y связанные.

3.4.2. Логика предикатов как формальная система

Как и в случае логики высказываний, логику предикатов можно рассматривать как формальную систему. Для этого необходимо определить четыре основные компоненты, лежащие в основе любой формальной системы.

1. Алфавит. В логике предикатов обычно используют следующие категории символов:

— предметные переменные — x, y, z и т. п., которые принимают значения из множества M;

— индивидные константы — a, b, c и т. п., т. е. нульместные функциональные константы или константы предметной области;

— функциональные константы — f, g, h и т. п., используются для обозначения функций;

— высказывания — p, q, r и т. п.;

— предикатные константы — P, R, H, Q и т. п.;

— символы логических операций — $\&$, $\vee$, $\rightarrow$ и т. д.;

— символы кванторных операций — $\forall, \exists$;

— вспомогательные символы — (,).

2. Правила построения формул. Для начала введем несколько определений, которыми будем оперировать при определении правил построения формул.

- ***Термом*** является всякая переменная и всякая функциональная форма.

- ***Функциональная форма*** — это функциональная константа, соединенная с некоторым числом термов. Если f — функциональная константа (n-местная) и $t_1,\ldots,t_n$ — термы, то соответствующая форма записывается $f(t_1,t_2,\ldots,t_n)$. Если $n=0$, то f превращается в индивидную константу.
- ***Предикатная форма*** — это предикатная константа, соединенная с соответствующим числом термов. Если P — предикатная m-местная константа и $t_1,\ldots,t_m$ — термы, то соответствующая форма записывается $P(t_1,t_2,\ldots,t_m)$. Если $m=0$, то пишут P.

Замечание. Следует помнить, что функциональная форма без аргументов — это просто индивидуальная константа, а предикатная форма без аргументов — просто высказывание.

- ***Атом*** — это предикатная форма или некоторое равенство (выражение вида $s=t$, где s и t — термы). Для равенства характерно то же, что и для предикатной формы, т. е. о нем можно сказать, что оно истинно или ложно.

Теперь перейдем к определению понятия формулы, которое делается рекурсивно по следующим правилам:

1. Атом есть формула.
2. Если A — формула, то $\overline{A}$ — формула.
3. Если A и B — формулы, то $(A\vee B),(A\to B)$, $(A\&B)$, $(A\sim B)$ — формулы.
4. Если A — формула и x — переменная, то $\forall xA,\exists xA$ — формулы.

Замечание. Можно опускать лишние скобки так же, как и в логике высказываний, если это допускает контекст.

3. Определение аксиом. Выбор системы аксиом в исчислении предикатов может осуществляться по-разному. Один из наиболее простых способов состоит в том, что берутся уже ранее определенные аксиомы из исчисления высказываний и дополняются еще двумя, связанными с использованием кванторов:

$\forall xF(x)\to F(y)$;

$F(t)\to\exists x\,F(x)$, где t — не содержит переменной x.

4. Правила вывода. Правила вывода также полностью заимствуются из логики высказываний, кроме того, к ним добавляются еще два правила следующего вида:

— правило введения квантора существования: $\dfrac{P(x)\to F}{\exists x\,P(x)\to F}$;

— правило введения квантора общности: $\dfrac{F\to P(x)}{F\to\forall x\,P(x)}$.

При этом в обоих случаях считается, что F не зависит от x.

Замечание 1. Понятия вывода и доказуемой формулы базируются на введенных ранее этих понятиях для произвольной ФС.

Замечание 2. Любые формальные системы, в качестве аксиом имеющие аксиомы, совпадающие с исчислением предикатов, называется *формальными системами первого порядка*. Обычно они отличаются только правилами словообразования и могут содержать дополнительные аксиомы. Термин «система первого порядка» означает, что в их формулах действие квантора может распространяться только на предикатные переменные и внутренние символы в формулах. К системам второго порядка относятся те, в которых действие квантора распространяется и на предикаты. В системах более высокого порядка действие квантора может распространяться и на предикаты от предикатов и т. д.

В заключение приведем несколько примеров построения предикатных выражений.

Примеры.

1. Утверждение: все слоны серые.

Вводимые предикаты: *слон* (x) — x — слон; *цвет* (x, y) — x цвета y.

Предикатное выражение: $\forall x(\textit{слон}\,(x) \to \textit{цвет}\,(x, \textit{серый}))$.

2. Утверждение: для любого множества x существует множество y такое, что мощность y больше, чем мощность x.

Вводимые предикаты: *множество* (x) — x — множество; *мощность* (x, u) — мощность множества x равна u; *больше* (x, u) — значение x больше значения u.

Предикатное выражение:

$$\forall x(\textit{множество}(x) \to (\exists y)(\exists u)(\exists v)(\textit{множество}(y)\,\&$$

$$\&\,\textit{мощность}(x, u)\,\&\,\textit{мощность}(y, v)\,\&\,\textit{больше}(v, u)).$$

3. Утверждение: все кубики, находящиеся на кубиках, которые были сдвинуты или скреплены с кубиками, которые сдвигались, тоже будут сдвинуты.

Вводимые предикаты: $\textit{куб}(x)$ — x — куб; $\textit{сверху}(x, y)$ — x расположен сверху y; $\textit{скреплен}(x, y)$ — x скреплен с y; $\textit{сдвинут}(x)$ — x сдвинут.

Предикатное выражение:

$$(\forall x)(\forall y)(\textit{куб}\,(x)\,\&\,\textit{куб}\,(y)\,\&\,(\textit{сверху}(x, y) \vee \textit{скреплен}(x, y))\,\&$$

$$\&\,\textit{сдвинут}(y)) \to \textit{сдвинут}(x).$$

3.4.3. Определение значения истинности предикатных формул

О логическом значении формулы логики предикатов можно говорить лишь тогда, когда задано множество M, на котором определены

входящие в формулу предикаты. При задании аргументам предикатов конкретных значений предикаты становятся высказываниями и можно говорить об их истинности или ложности.

Как и в алгебре высказываний, в логике предикатов имеет место понятие равносильности предикатных выражений.

Определение. Две формулы логики предикатов считаются ***равносильными в области M***, если они принимают одинаковые логические значения при всех входящих в них переменных, отнесенных к области *M*.

Определение. Две формулы логики предикатов называются ***равносильными***, если они равносильны на всякой области.

На понятие равносильности формул опираются и правила эквивалентных преобразований предикатных выражений, которые позволяют получить из предикатного выражения одного вида равносильное ему выражение другого вида. Очевидно, что в тех случаях, когда преобразования не изменяют ограничений, накладываемых кванторами, можно использовать все известные правила из АВ, если заменить в них высказывания на предикаты. Кроме того, в исчислении предикатов существуют и собственные правила, регламентирующие преобразование выражений, содержащих кванторы.

Пусть $A(x)$, $A(x, y)$ и $B(x)$ — предикаты, p — высказывание, тогда правила имеют следующий вид:

1) $\overline{\forall x\, A(x)} = \exists x\, \overline{A}(x)$;
2) $\overline{\exists x\, A(x)} = \forall x\, \overline{A}(x)$;
3) $\forall x\, A(x) = \overline{\exists x\, \overline{A}(x)}$;
4) $\exists x\, A(x) = \overline{\forall x\, \overline{A}(x)}$;
5) $\forall x\big(A(x) \,\&\, B(x)\big) = \forall x\, A(x) \,\&\, \forall x\, B(x)$;
6) $\forall x\big(A(x) \vee B(x)\big) = \forall x\, A(x) \vee \forall x\, B(x)$;
7) $p \,\&\, \forall x\, A(x) = \forall x\big(p \,\&\, A(x)\big)$;
8) $p \vee \forall x\, A(x) = \forall x\big(p \vee A(x)\big)$;
9) $p \to \forall x\, B(x) = \forall x\big(p \to B(x)\big)$;
10) $\forall x\big(B(x) \to p\big) = \exists x\, B(x) \to p$;
11) $\exists x\big(A(x) \vee B(x)\big) = \exists x\, A(x) \vee \exists x\, B(x)$;
12) $\exists x\big(A(x) \,\&\, B(x)\big) = \exists x\, A(x) \,\&\, \exists x\, B(x)$;
13) $p \vee \exists x\, B(x) = \exists x\big(p \vee B(x)\big)$;
14) $p \,\&\, \exists x\, B(x) = \exists x\big(p \,\&\, B(x)\big)$;
15) $\exists x\, A(x) \,\&\, \exists y\, B(x) = \exists x\, \exists y\big(A(x) \,\&\, B(x)\big)$;

16) $\exists x\,\exists y\,A(x,y)=\exists y\,\exists x\,A(x,y)$;

17) $\forall x\,\forall y\,A(x,y)=\forall y\,\forall x\,A(x,y)$;

18) $\exists x\big(B(x)\to p\big)=\forall x\,B(x)\to p$.

Таким образом, в случае если не определены значения предметных переменных и нельзя установить значение истинности предикатов, считается, что невозможно установить значение истинности предикатных формул. Однако в некоторых частных случаях это возможно сделать путем эквивалентных преобразований, базирующихся на ранее определенных формулах эквивалентности.

Пример. Докажем тождественную истинность заданного предикатного выражения

$$\forall x\Big(P(x)\to\big(P(x)\vee P(y)\big)\Big)=\forall x\Big(\overline{P}(x)\vee P(x)\vee P(y)\Big)=\forall x\big(1\vee P(y)\big)=$$

$$=\forall x(1)=1.$$

Пример. Докажем тождественную ложность заданного предикатного выражения

$$\exists x\exists y\Big(\big(F(x)\to F(y)\big)\&\big(F(x)\to\overline{F}(y)\big)\&F(x)\Big)=$$

$$=\exists x\exists y\Big(\big(\overline{F}(x)\vee F(y)\big)\&\big(\overline{F}(x)\vee\overline{F}(y)\big)\&F(x)\Big)=$$

$$=\exists x\exists y\ \Big(\overline{F}(x)\vee F(y)\big)\&\big(\overline{F}(x)\&F(x)\vee\overline{F}(y)\&F(x)\Big)=$$

$$=\exists x\exists y\ \Big(\overline{F}(x)\vee F(y)\big)\&\big(0\vee\overline{F}(y)\&F(x)\Big)=$$

$$=\exists x\exists y\Big(\big(\overline{F}(x)\vee F(y)\big)\&\overline{F}(y)\&F(x)\Big)=$$

$$=\exists x\exists y\Big(\overline{F}(x)\&\overline{F}(y)\&F(x)\vee F(y)\&\overline{F}(y)\&F(x)\Big)=\exists x\exists y\big(0\vee 0\big)=0.$$

В большинстве случаев, особенно если в выражении присутствуют кванторы, найти универсальный метод определения выполнимости такого выражения невозможно, поэтому считается, что исчисление предикатов неразрешимо. Однако существует большое множество задач в логике предикатов, для которых существуют эффективные алгоритмы, позволяющие определить их общезначимость или вычислимость.

Одним из наиболее эффективных методов определения разрешимости множества предикатных формул является метод резолюций, который, по сути, является развитием уже известного из АВ метода применительно к исчислению предикатов.

3.4.4. Методы резолюций для логики предикатов

Как и ранее, под резолюцией будем понимать правило вывода, которое может быть применено к определенному классу формул — *дизъюнктам*. Принцип резолюции состоит в том, что, будучи примененным к двум формулам, позволяет получить новую формулу как их следствие.

Однако, прежде чем перейти к рассмотрению самого метода, следует ознакомиться с некоторыми понятиями, без которых невозможна его реализация.

Унификация. Так как анализ выражений в формальных системах осуществляется в чисто синтаксической форме, без учета семантики, то возникает необходимость приведения отдельных выражений к единой форме. Эти приведения базируются на использовании унификации.

Например, для создания $W_2(A)$ из $\forall x(W_1(x) \to W_2(x))$ и $W_1(A)$ необходимо найти подстановку «*A вместо x*», которая сделает идентичными $W_1(A)$ и $W_1(x)$. Поиск подстановок термов на место переменных называется ***унификацией***. Унификация основывается на другом понятии — понятии ***подстановки***.

Определение. ***Подстановочным частным случаем*** называется подстановка в некоторое выражение термов вместо переменных.

Пример. Для выражения $P(x, f(y), B)$ имеется четыре частных случая подстановки:

1) $P(z, f(\omega), B)$ — алфавитная, просто замена одних переменных на другие;

2) $P(x, f(A), B)$ — подставка константы в функцию f;

3) $P(g(z), f(A), B)$ — функциональное выражение вместо переменной x;

4) $P(C, f(A), B)$ — основной частный случай, так как везде подставлены константы вместо переменных.

Любую подстановку можно представить с помощью множества упорядоченных пар вида $S = \{t_1/v_1, t_2/v_2, \ldots, t_n/v_n\}$, где пара t_i/v_i означает, что переменная v_i заменяется на терм t_i. Следует отметить, что при выполнении подстановки должны соблюдаться два правила:

— каждое вхождение переменной заменяется на один и тот же терм;

— переменную нельзя заменить на терм, содержащий ту же самую переменную.

Для предыдущего примера подстановки, описанные с помощью введенного формализма, имеют следующий вид:

1) $S_1 = \{z/x, \omega/y\}$;
2) $S_2 = \{A/y\}$;
3) $S_3 = \{g(z)/x, A/y\}$;
4) $S_4 = \{C/x, A/y\}$.

Для обозначения подстановки часто используется следующая запись. Если S — подстановка, а E — выражение, к которому она применяется, то пишут E_S.

Если подстановка S применяется к каждому элементу некоторого множества $\{E_i\}$, то множество подстановочных примеров обозначается $\{E_i\}_S$.

Определение. Говорят, что ***множество*** $E = \{E_1, E_2, ..., E_n\}$ ***унифицируемо***, если существует такая подстановка S, что $E_{1S} = E_{2S} = ... = E_{nS}$. В этом случае подстановка S называется ***унификатором*** для множества E, так как после ее применения множество склеивается в один элемент.

Пример. Пусть $A = \{P(x, f(y), B), P(x, f(B), B)\}$, рассмотрим подстановку $S = \{C/x, B/y\}$. Унифицируем и получаем $A = \{P(A, f(B), B)\}$ (в подстановке C необходимости не было).

Пример. $P(x, f(y), B)_{S_1} = P(z, f(\omega), B)$.

Унификация производится при следующих условиях:

1. Если термы — константы, то они унифицируемы тогда и только тогда, когда они совпадают.

2. Если в первом дизъюнкте терм — переменная, а во втором — константа, то они унифицируемы, при этом вместо переменной подставляется константа.

3. Если терм в первом дизъюнкте — переменная и во втором дизъюнкте терм тоже переменная, то они унифицируемы.

4. Если в первом дизъюнкте терм — переменная, а во втором — употребление функции, то они унифицируемы, при этом вместо переменной подставляется употребление функции.

5. Унифицируются между собой термы, стоящие на одинаковых местах в одинаковых предикатах.

Пример. Рассмотрим дизъюнкты:

1) $Q(a, b, c)$ и $Q(a, d, l)$. Дизъюнкты не унифицируемы;

2) $Q(a, b, c)$ и $Q(x, y, z)$. Дизъюнкты унифицируемы. Унификатор — $Q(a, b, c)$.

Если необходимо последовательно выполнить несколько подстановок: S_1, S_2, то можно записать $E_{S_1S_2}$. На этом действии базируется такое понятие, как ***композиция подстановок.*** Если S и S' являются двумя множествами подстановок, то их композиция S и S' (пишется SS') получается после применения S' к элементам S и добавления резуль-

тата к S. Можно сказать, что композиция подстановок — это метод, с помощью которого объединяются подстановки унификации. Обобщая сказанное, можно ввести определение понятия композиции подстановок.

Определение. Композиция подстановок g и s есть функция gs, определяемая следующим образом: $(gs)\ [t] = g\ [s\ [t]]$, где t — терм, g и s — подстановки, а $s\ [t]$ — терм, который получается из t путем применения к нему подстановки s.

Пример. Пусть задана последовательность подстановок $\{x/y,\ w/z\}$, $\{v/x\}$, $\{A/v, f(B)/w\}$, они эквивалентны одной подстановке $\{A/y, f(B)/z\}$. Последняя подстановка была выведена путем компоновки $\{x/y,\ w/z\}$ и $\{v/x\}$ для получения $\{v/y,\ w/z\}$ и компоновки результата с $\{A/v, f(B)/w\}$ для получения $\{A/y, f(B)/z\}$.

Основное требование алгоритма унификации состоит в том, что унификатор должен быть максимально общим, т. е. для любых двух выражений должен быть найден ***наиболее общий унификатор*** (НОУ). Это важно, так как при потере общности в процессе решения уменьшается вероятность достижения окончательного решения или такая возможность исчезает полностью. Например, предикаты $P(x)$ и $P(y)$, определенные на множестве целых чисел, можно унифицировать любым константным выражением вида $\{4/x,\ 4/y\}$. Однако число 4 в данном случае не является НОУ. Используя в качестве унификатора любую переменную, можно получить более общее выражение: $\{z/x,\ z/y\}$. Решения, полученные при использовании первой подстановки, всегда будут ограничены содержащейся в них константой 4, лимитирующей логический вывод. Следовательно, константу 4 можно применить в качестве унификатора, но это снижает универсальность результата.

Определение. Если s — произвольный унификатор выражения E, а g — ***наиболее общий унификатор*** этого набора выражений, то в случае применения s к E будет существовать еще один унификатор s' такой, что $Es = Egs'$, где Es и Egs' — композиции унификации, примененные к выражению E.

Очевидно, что прежде чем приводить выражения к некоторому общему виду, необходимо выявить их различия, т. е. *рассогласования*.

Определение. ***Множество рассогласований*** непустого множества дизъюнктов $\{E_1, \ldots, E_n\}$ получаются путем выявления первой (слева) позиции, на которой не для всех дизъюнктов из E стоит один и тот же символ, и путем выписывания из каждого дизъюнкта терма, который начинается с символа, занимающего данную позицию. Множество термов и есть множество рассогласований в E.

Пример. Рассмотрим дизъюнкты $\{P(x, f(y, z)), P(x, a), P(x, g(h(k(x))))\}$.

Множество рассогласований состоит из термов, которые начинаются с пятой позиции, и представляет собой множество $\{f(x, y), a, g(h(k(x)))\}$.

Рассмотрим алгоритм унификации для нахождения наиболее общего унификатора. Пусть E — множество дизъюнктов, D — множество рассогласований, k — номер итерации, g_k — наиболее общий унификатор на k-й итерации.

Шаг 1. Пусть $k = 0$, $g_k = e$ (пустой унификатор), $E_k = E$.

Шаг 2. Если для E_k не существует множества рассогласований D_k, то алгоритм завершает работу и g_k — наиболее общий унификатор для E. Иначе найдем множество рассогласований D_k.

Шаг 3. Если существуют такие элементы v_k и t_k в D_k, что v_k — переменная, не входящая в терм t_k, то перейдем к шагу 4. В противном случае алгоритм завершает работу и E не унифицируемо.

Шаг 4. Пусть $g_{k+1} = g_k \{t_k/v_k\}$, заменим во всех дизъюнктах E_k терм t_k на v_k.

Шаг 5. $k = k+1$. Перейти к шагу 2.

Пример. Рассмотрим дизъюнкты $E = \{P(f(a), g(x)), P(y, y)\}$.

Итерация I

Шаг 1. $E_0 = E$, $k = 0$, $g_0 = e$.

Шаг 2. $D_0 = \{f(a), y\}$.

Шаг 3. $v_0 = y$, $t_0 = f(a)$.

Шаг 4. $g_1 = \{f(a)/y\}$, $E_1 = \{P(f(a), g(x)), P(f(a), f(a))\}$. Переход на шаг 2.

Итерация II

Шаг 2. $D_1 = \{g(x), f(a)\}$.

Шаг 3. Так как нет переменной в множестве рассогласований D_1, то, следовательно, алгоритм унификации завершается, множество E — не унифицируемо.

Обычно унификацию используют для приведения в соответствие различных выражений. Если в одном из выражений подставлены вместо переменных константы, а другие выражения приводятся в соответствие с ним, то это называется ***сравнением с образом.***

Как отмечалось ранее, метод резолюций применим к множеству дизъюнктов. Поэтому рассмотрим ***последовательность действий, которую необходимо реализовать для приведения любой формулы логики предикатов к множеству дизъюнктов.***

Проиллюстрируем процесс приведения примером. Пусть задано следующее предикатное выражение:

$$(\forall x)(P(x) \to (\forall y)(P(y) \to P(f(x,y))) \,\&\, \overline{\forall y(Q(x,y) \to P(y))}.$$

Его следует привести к множеству предложений. На каждом шаге будем приводить пример, который демонстрирует результат применения действий соответствующего шага к заданному выражению.

Шаг 1. Используя правила эквивалентных преобразований, в исходном выражении все логические операции записывают через операции дизъюнкции, конъюнкции и отрицания. Таким образом, результат будет записан как выражение, определенное в рамках следующей функционально полной системы логических функций $\{\vee, \&, \bar{}\}$.

Пример к шагу 1. $\forall x(\bar{P}(x) \vee \forall y(\bar{P}(y) \vee P(f(x,y))) \,\&\, \overline{\forall y(\bar{Q}(x,y) \vee P(y))}.$

Шаг 2. Используя правила де Моргана, выполняют преобразования, обеспечивающие применение операции отрицания только к литералам.

Пример к шагу 2. $\forall x(\bar{P}(x) \vee \forall y(\bar{P}(y) \vee P(f(x,y))) \,\&\, \exists y(Q(x,y) \,\&\, \bar{P}(y)).$

Шаг 3. Разделение переменных. Так как в пределах области действия квантора можно заменить любую переменную на другую и от этого значение истинности формулы не изменится, можно преобразовать формулу так, чтобы каждый квантор имел свою собственную переменную, например вместо $(\forall x)(P(x) \to \exists x Q(x))$ пишется $(\forall x)(P(x) \to \to \exists y Q(y))$.

Пример к шагу 3. $(\forall x)(\bar{P}(x) \vee \forall y(\bar{P}(y) \vee P(f(x,y))) \,\&\, \exists \omega(Q(x,\omega) \,\&\, \&\, \bar{P}(\omega))$.

Шаг 4. Исключение кванторов существования. Рассмотрим формулу $(\forall x)(\exists x P(x,y))$. В этой формуле получается, что все выражение выполняется для любого y и некоторого x, который, возможно, зависит от y. Эту зависимость можно обозначить явно с помощью некоторой функции $g(y)$, отображающей каждое значение y в то x, которое существует. Такая функция называется ***сколемовской функцией***. Если заменить на нее x, то квантор существования можно убрать и переписать выражение в новом виде $\forall y P(g(y), y)$. Пусть требуется привести к сколемовской форме следующую формулу: $(\exists x)(\forall y)(\forall z)(\exists u)(\forall v)(\exists w)(P(x, y, z, u, v, w)))$. В этой формуле левее $(\exists x)$ нет никаких кванторов всеобщности, левее $(\exists u)$ стоят $(\forall y)$ и $(\forall z)$, а левее $(\exists w)$ стоят $(\forall y)$, $(\forall z)$ u $(\forall v)$. Следовательно, можно заменить переменную x на константу a, переменную u — на двухместную $f(y, z)$, переменную w — на трехместную функцию $g(y, z, v)$. Таким образом, получаем следующую форму: $(\forall y)(\forall z)(\forall v)(P(a, y, z, f(y, z), g(y, z, v)))$. Общее правило исключения квантора существования состоит в замене каждого вхождения пере-

менной, относящейся к квантору существования, на сколемовскую функцию, аргументы которой представляют собой те переменные, которые связаны с кванторами общности, в область действия которых попал квантор существования. Функциональные символы, используемые в сколемовских функциях, должны быть новыми, т. е. они не должны встречаться ранее в формуле. Например, исключаем $\exists z$ из следующего выражения $\forall\omega Q(\omega)\to\forall x(\forall y(\exists z(P(x,y,z)\to\forall uR(x,y,u,z))))$, получим $\forall\omega Q(\omega)\to\forall x(\forall y(P(x,y,g(x,y))\to\forall uR(x,y,u,g(x,y))))$. Если квантор существования не входит в область действия никакого квантора общности, то применяется сколемовская функция без аргументов, т. е. константа.

Пример к шагу 4. $\forall x(\bar{P}(x)\vee(\forall y(\bar{P}(y)\vee P(f(x,y))\&(Q(x,g(x))\&\ \&\bar{P}(g(x)))))$.

Шаг 5. Преобразование выражения в предваренную (префиксную) форму. Так как в формуле кванторы существования отсутствуют, то все кванторы общности можно переместить в начало формулы. Формула в таком виде называется ***формулой в предварительной форме***, цепочка кванторов перед ней — ***префиксом***, а следующее за ним бескванторное выражение — матрицей.

Пример к шагу 5.

$\forall x\forall y((\bar{P}(x)\vee\bar{P}(y)\vee P(f(x,y)))\&(\bar{P}(x)\vee Q(x,g(x)))\&(\bar{P}(x)\vee\bar{P}(g(x)))).$

Шаг 6. Приведение матрицы выражения к форме КНФ. Известно, что любая логическая функция может быть представлена в виде КНФ. Для этого чаще всего используется метод эквивалентных преобразований.

Пример к шагу 6. Из алгебры логики известны следующие равенства:

$x_1\vee x_2x_3=(x_1\vee x_2)(x_1\vee x_3)$;

$x_1x_2\vee x_3x_4=(x_1\vee x_3x_4)(x_2\vee x_3x_4)$.

Применим их к полученному после шага 5 выражению:

$$\forall x\forall y(\underbrace{\bar{P}(x)}_{x_1}\vee\underbrace{((\bar{P}(y)\vee P(f(x,y)))}_{x_2}\&\underbrace{Q(x,g(x))\&\bar{P}(g(x)))}_{x_3});$$

$$(\bar{P}(x)\vee\bar{P}(y)\vee P(f(x,y)))\&(\underbrace{\bar{P}(x)}_{x_1}\vee\underbrace{Q(x,g(x))}_{x_2}\&\underbrace{\bar{P}(g(x))}_{x_3})=$$

$$=\forall x\forall y(\bar{P}(x)\vee\bar{P}(y)\vee P(f(x,y))\&(\bar{P}(x)\vee Q(x,g(x)))\&(\bar{P}(x)\vee\bar{P}(g(x)))).$$

Шаг 7. Исключение кванторов общности. Так как в выражении остались кванторы общности, а их порядок несущественен, то можно эти кванторы опустить, т. е. удалить у формулы префикс.

Пример к шагу 7.

$\bar{P}(x) \vee \bar{P}(y) \vee P(f(x,y)) \& (\bar{P}(x) \vee Q(x, g(x))) \& (\bar{P}(x) \vee \bar{P}(g(x)))$.

Шаг 8. Переход от выражения в виде КНФ к множеству предложений. Для этого требуется убрать все операции конъюнкции, а каждый дизъюнкт представить как отдельное предложение, т. е. перейти от выражения вида $x_1 \& x_2$ к выражению $\{x_1, x_1\}$, в котором каждый элемент — предложение.

Пример к шагу 8.

$\{\bar{P}(x) \vee \bar{P}(y) \vee P(f(x,y)), \bar{P}(x) \vee Q(x, g(x)), \bar{P}(x) \vee \bar{P}(g(x))\}$.

Шаг 9. Переименование переменных. Символы переменных должны быть изменены так, чтобы каждый появлялся не более чем в одном предложении. Этот процесс называется ***разделением переменных***.

Пример к шагу 9.

$\{\bar{P}(x_1) \vee \bar{P}(y) \vee P(f(x_1, y)), \bar{P}(x_2) \vee Q(x_2, g(x_2)), \bar{P}(x_3) \vee \bar{P}(g(x_3))\}$.

Примечание. Когда резолюция используется в качестве правила вывода, то множество формул, которое будет использоваться в качестве исходного для доказательства, должно быть представлено в виде множества предложений.

При построении вывода с помощью метода резолюций следует учесть следующее.

1. Если во множестве присутствуют только предложения, содержащие предикаты без переменных, то вывод строится, как и в случае с высказываниями, так как логика высказываний — это частный случай логики предикатов.

2. Для того чтобы применить резолюцию к предложениям, содержащим переменные, необходимо иметь возможность найти такую подстановку, которая, будучи применена к родительским предложениям, приведет к тому, что они будут содержать дополнительные литералы.

Известно, что правило резолюций предполагает нахождение пары дизъюнктов, допускающих построение резольвенты. Для дизъюнктов логики высказываний это очень просто. Для дизъюнктов логики предикатов процесс усложняется, так как дизъюнкты могут содержать функции, переменные и константы, а следовательно, их предварительно необходимо унифицировать.

Пример. Рассмотрим дизъюнкты:

C_1: $P(x) \vee Q(x)$,

C_2: $R(x) \vee \bar{P}(f(x))$.

Так как аргументы предиката P в C_1 и C_2 различны, то невозможно построить на их основе резольвенту, но если подставить $f(a)$ вместо x в C_1 и a вместо x в C_2, то исходные дизъюнкты примут вид

C_1': $P(f(a)) \vee Q(f(a))$,

C_2': $R(a) \vee \bar{P}(f(a))$.

Следовательно, можно получить резольвенту C_3': $Q(f(a)) \vee R(a)$.

В общем случае, подставив $f(x)$ вместо x в C_1, получим C_1'': $P(f(x)) \vee Q(f(x))$.

Предикат $P(f(x))$ в C_1'' и предикат $\bar{P}(f(x))$ в C_2 позволяют получить резольвенту C_3: $Q(f(x)) \vee R(x)$.

Таким образом, если подставлять подходящие термы вместо переменных в исходные дизъюнкты, можно порождать новые дизъюнкты. Отметим, что дизъюнкт C_3 из примера является наиболее общим дизъюнктом в том смысле, что все другие дизъюнкты, порожденные правилом резолюции, будут частным случаем данного дизъюнкта.

Учитывая введенное понятие унификации, можно распространить метод резолюций на исчисление предикатов. При унификации возникает одна трудность: если один их термов есть переменная x, а другой терм содержит x, но не сводится к x, унификация невозможна. Проблема решается путем переименования переменных таким образом, чтобы унифицируемые дизъюнкты не содержали одинаковых переменных.

Определение. Если два или более предиката (с одинаковым знаком) дизъюнкта C имеют наиболее общий унификатор g, то Cg — называется ***склейкой*** C.

Пример. Рассмотрим дизъюнкт $C = P(x) \vee P(f(y)) \vee \bar{Q}(x)$. В этом дизъюнкте первый и второй предикаты имеют наиболее общий унификатор $g = \{f(y)/x\}$. Следовательно, $Cg = P(f(y)) \vee \bar{Q}(f(y))$ есть склейка C.

Определение. Пусть C_1 и C_2 — два дизъюнкта, которые не имеют никаких общих переменных. Пусть L_1 и L_2 — два предиката в C_1 и C_2. Если L_1 и $\overline{L_2}$ имеют наиболее общий унификатор g, то дизъюнкт $(C_1g \backslash L_1g) \cup (C_2g \backslash L_2g)$ называется ***резольвентой*** C_1 и C_2.

Пример. Пусть $C_1 = P(x) \vee Q(x)$ и $C_2 = \bar{P}(a) \vee R(x)$. Так как x входит в C_1 и C_2, то заменим переменную в C_2 и пусть $C_2 = \bar{P}(a) \vee R(y)$. Выбираем $L_1 = P(x)$ и $L_2 = \bar{P}(a)$. L_1 и L_2 имеют наиболее общий унификатор $g = \{a/x\}$. Следовательно, $Q(a) \vee R(y)$ — резольвента C_1 и C_2.

Пример. Пусть задано множество дизъюнктов $\{P(x, f(A)) \vee P(x, f(y)) \vee Q(y), \bar{P}(z, f(A)) \vee \bar{Q}(z)\}$, тогда при $L_1 = \{P(x, f(A))\}$ и $L_2 = \{\bar{P}(z, f(A))\}$ полагаем $g = \{z / x\}$ и получаем резольвенту $P(z, f(y)) \vee \bar{Q}(z) \vee Q(y)$, а при $L_1 = \{P(x, f(A)), P(x, f(y))\}$ и

$L_2 = \{\bar{P}(z, f(A))\}$ полагаем $g = \{z / x, A / y\}$ и получаем резольвенту $Q(A) \vee \bar{Q}(z)$.

Очевидно, что для двух дизъюнктов может существовать более одной резольвенты.

Переформулируем теперь уже известный алгоритм метода резолюций применительно к логике предикатов.

Шаг 1. Если в S есть пустой дизъюнкт, то множество S не выполнимо и алгоритм завершает работу, иначе переходим к шагу 2.

Шаг 2. Найти в исходном множестве S такие дизъюнкты или склейки дизъюнктов C_1 и C_2, которые содержат унифицируемые литералы $L_1 \in C_1$ и $L_2 \in C_2$. Если таких дизъюнктов нет, то исходное множество выполнимо и алгоритм завершает работу, иначе переходим к шагу 3.

Шаг 3. Вычислить резольвенту C_1 и C_2, добавить ее в множество S и перейти к шагу 1.

Пример. Докажем с помощью метода резолюций справедливость следующих рассуждений.

- Некоторые руководители с уважением относятся ко всем программистам.
- Ни один руководитель не уважает бездельников.
- Следовательно, ни один программист не является бездельником.

Введем следующие предикаты:

$C(x)$ — «*x — руководитель*»;

$P(x)$ — «*x — программист*»;

$B(x)$ — «*x — бездельник*»;

$U(x, y)$ — «*x уважает y*».

Тогда посылки, записанные в виде предикатных выражений будут выглядеть так:

- $\exists x(C(x) \& \forall y(P(y) \to U(x, y)))$;
- $\forall x(C(x) \to \forall y(B(y) \to \bar{U}(x, y)))$.

Заключение примет следующий вид:

- $\forall y(P(y) \to \bar{B}(y))$.

Преобразовав посылки и следствие, которое надо взять с отрицанием, в совокупность дизъюнктов, получим множество предложений, невыполнимость которого докажем, используя метод резолюций. Имеем

$$\{C(a),\ \bar{P}(y) \cup U(x, y),\ \bar{C}(x) \vee \bar{B}(y) \vee \bar{U}(x, y), P(b),\ B(b)\}.$$

1) $C(a)$;

2) $\bar{P}(y) \cup U(x,y)$;
3) $\bar{C}(x) \vee \bar{B}(y) \vee \bar{U}(x,y)$;
4) $P(b)$;
5) $B(b)$;
6) $\bar{B}(y) \vee \bar{U}(a,y)$ (1,3) $/ s = \{a/x\}$;
7) $U(a,b)$, (2,4) $/ s = \{b/y\}$;
8) $\bar{B}(b)$, (6,7) $/ s = \{b/y\}$;
9) пустой дизъюнкт (5,8).

В заключение введем еще два определения.

Определение. ***Сколемовской формой*** предикатного выражения называется формула логики предикатов, в которой все вхождения переменных, у которых кванторы существования находятся в области действия кванторов общности, заменены на сколемовские функции.

Определение. ***Клаузальной формой*** называется такая сколемовская форма, матрица которой имеет вид КНФ. Любая сколемовская форма допускает эквивалентную ей клаузальную форму.

ГЛАВА 4

РАСШИРЕНИЯ ТРАДИЦИОННОЙ ЛОГИКИ

4.1. Общие положения модальной логики предикатов

В обычном языке часто говорят о допустимости чего-либо, о гипотетических событиях, при этом большая часть таких фраз может быть истинной или ложной в зависимости от текущей ситуации, точки зрения строящего рассуждения и т. п. В естественном языке подобные ситуации описываются терминами: «возможный», «необходимый», «допустимый» и т. п. — и выражаются вспомогательными глаголами, такими как «должен» и «могу».

Возможность и необходимость называются *алетическими модальностями* или *модальностями возможности*. Так же как кванторы общности и существования вводятся в синтаксис логики первого порядка, можно построить формальный язык, используя понятия «возможно» и «необходимо» как кванторы, действующие на формулы.

Определение. Логическая система, базирующаяся на операторах «возможно, что» и «необходимо, чтобы», называется ***логикой возможного*** или ***алетической логикой***.

Для обозначения модальностей вводится следующая символика:

$\Box$ — обозначает модальность «необходимо». Формула вида $\Box F$ читается так: «необходимо, чтобы F» или «F необходимо»;

$\Diamond$ — обозначает модальность «возможно». Формула вида $\Diamond F$ читается так: «возможно, что F» или «F возможно».

Введенные операторы, как и кванторы $\exists$ и $\forall$, являются двойственными, т. е. имеет место следующее соотношение:

$$\Box F = \overline{\Diamond F}.$$

Замечание. В обычном языке используются и другие модальные формы, которые также переносятся в логику, а именно:

— *дектоническая логика.* Модальности «разрешено»/«обязательно»;

— *эпистемическая логика,* или логика знания. Модальности «знать»/«верить»;

— *временная логика.* Модальности «иногда»/«всегда».

Очевидно, что перечисленный набор охватывает только малую часть возможных модальностей. Ввиду формального сходства между этими различными логическими системами их часто изучают совместно. Для обозначения совокупности таких логик используется термин ***модальная логика***.

Ранее уже были определены операторы для обозначения модальностей, определение было дано аналогично тому, как вводятся кванторы существования и общности в обычной логике первого порядка. Введем следующее общее определение модальных операторов.

Модальный оператор общности Δ является:

— либо именем модального оператора;

— либо выражением, состоящим из имени модального оператора, за которым следует список $(t_1, t_2, ..., t_n)$ термов t_i $(i = \overline{1,n})$.

Модальный оператор существования ∇ определяется через отрицание модального оператора общности

$$\Delta F = \overline{\nabla F}.$$

Замечание. Очевидно, что число модальных формул бесконечно. В философии их вводится и используется огромное количество. Все эти формулы представимы с помощью модальных операторов.

Как и всякая логика, модальная логика наряду с семантическими правилами имеет и определенный синтаксис. Определим его.

Пусть $M_1, M_2,..., M_n$ — модальные операторы. Правило образования модальных формул следующее:

— все правила, построенные из логики предикатов первого порядка, являются также правилами построения в модальной логике предикатов;

— если F — формула и M_i — модальный оператор, то M_iF — формула.

Очевидно, что, как и ранее, одной из основных задач является задача представления утверждений, относящихся к некоторой предметной области, в виде формул модальной логики предикатов. Рассмотрим примеры перевода утверждений в формулы. Будем считать, что синтаксические значения полученных модальных формул должны соответствовать истинным утверждениям.

Пример.

1. Утверждение: возможно, что Сидоров передал Петрову книгу Т. Свана «C++».

Предикат: $P(x, y, z)$ — *x передает y объект z.*

Выражение: $\Diamond P$ (*Сидоров, Петров, Т. Сван «С++»*).

2. Утверждение: невозможно, чтобы Иванов передал кому-нибудь что-либо.

Предикат: $P(x, y, z)$ — *x передает y объект z.*

Выражение: $\overline{\Diamond(\exists x \exists z P(Иванов, z, x))}$.

4.2. Трехзначная семантика для модальной логики предикатов

Семантику модальной логики можно определить аналогично классической. Рассмотрим модальную семантику на примере описания логики, возможного с помощью трехзначной логики.

Бинарная логика с двумя значениями — истина и ложь — является самой элементарной. Истина и ложь образуют два множества высказываний. При этом законы классической логики гласят:

— любое высказывание является элементом хотя бы одного из этих множеств (*закон исключения третьего*);

— никакое высказывание не является элементом сразу двух этих множеств (*закон противоречия*).

Чтобы ввести в эту теорию модальности «возможно» и «необходимо», следует ввести несколько классов высказываний. Введем следующие обозначения:

— N — класс «необходимых» высказываний;

— P — класс «возможных» высказываний;

— I — класс «невозможных» (или «абсурдных») высказываний;

— C — класс «нейтральных» (или «возможно, случайно ложных») высказываний.

Для указанных классов справедлив ряд законов.

1. *Закон противоречий.* Никакое высказывание не принадлежит одновременно N и C или I и P.

2. *Закон следования возможного из необходимого.* Класс N содержится в P.

3. *Закон следования нейтрального из абсурдного.* Класс I содержится в C.

Законы 2 и 3 можно сформулировать так:

— любое необходимое высказывание возможно;

— любое абсурдное высказывание не является необходимым.

Существуют высказывания, которые одновременно возможны и нейтральны. Их называют «проблематичными». Множество таких высказываний обозначается через U. Тогда имеет место закон исключения четвертого:

— любое высказывание принадлежит либо N, либо U, либо I.

Рассмотрим теперь вопрос перевода выбранной теории модальности в алгебраическую форму. Каждому из классов N, U, I соответствует своя интерпретация: «необходимо», «проблематично», «невозможно». Для их обозначения можно использовать три символа: 2, 1, 0, которые обозначают логические значения, сопоставляемые с указанными интерпретациями.

Таким образом, каждому из высказываний можно присвоить одно из этих логических значений. Например, логические операции из классической логики можно было бы использовать и в модальной, определив их следующим образом:

1. Отрицание (табл. 4.1).

Таблица 4.1

x	$\bar{x}$
0	2
2	1
1	0

2. Конъюнкция (табл. 4.2).

Таблица 4.2

	y		
x	**0**	**1**	**2**
0	0	0	0
1	0	1	1
2	0	1	2

3. Дизъюнкция (табл. 4.3).

Таблица 4.3

	y		
x	**0**	**1**	**2**
0	0	1	2
1	1	1	2
2	2	2	2

4. Импликация (табл. 4.4).

Таблица 4.4

	y		
x	**0**	**1**	**2**
0	2	2	2
1	1	2	2
2	0	1	2

5. Эквиваленция (табл. 4.5).

Таблица 4.5

	y		
x	**0**	**1**	**2**
0	2	1	0
1	1	2	1
2	0	1	2

6. Модальные операторы (табл. 4.6).

Таблица 4.6

F	◊*F*	□*F*
0	0	0
1	2	0
2	2	2

Примечание. Приведенные таблицы истинности соответствуют семантике модальной логики, предложенной Лукасевичем.

4.3. Семантика возможных миров и четырехзначная логика

Предположим, что все события происходят в некотором мире — абстрактном, математическом, реальном или другом. Тогда множество истинных событий можно разделить на два множества:

— «случайно истинные». Это истинные события, но они могли бы оказаться ложными, если бы развитие мира шло иначе, например исторические события.

— «необходимо истинные». Это такие события, которые нельзя отрицать, не ставя под сомнения базовые сущности мира, например физические и математические законы.

Аналогично ложные события делятся на «случайно ложные» и «необходимо ложные».

Предположим, что «необходимая истина» (в существующем мире) связана с событием, подтверждающимся не только в существующем, но и во всех возможных мирах.

Интерпретациям «необходимо истинно», «нейтрально истинно», «нейтрально ложно» и «необходимо ложно» можно поставить в соответствие логические значения 3, 2, 1, 0. Таким образом модальная логика, связанная с операторами *необходимо* и *случайно*, оказалась четырехзначной логикой. Деление событий на множества можно представить в виде рисунка (рис. 4.1).

Если предположить, что семантика возможных миров сводится к семантике с двумя возможностями: существующий мир — *X* и возможный мир — *Y*, то каждое из высказываний принимает одно из логических значений — 0, 1, 2, 3 — в соответствии с тем, какому из приведенных ниже условий оно соответствует:

0 — необходимо ложно. Ложно в *X* и в *Y*;

1 — случайно ложно. Ложно в *X* и истинно в *Y*;

2 — случайно истинно. Истинно в *X* и ложно в *Y*;

3 — необходимо истинно. Истинно в *X* и в *Y*.

Рис. 4.1

Такая интерпретация может быть использована для определения основных логических операций для четырехзначной логики, число и задает семантику данной логики.

1. Отрицание (табл. 4.7).

Таблица 4.7

x	$\overline{x}$
0	3
1	2
2	1
3	0

2. Конъюнкция (табл. 4.8).

Таблица 4.8

	y			
x	**0**	**1**	**2**	**3**
0	0	0	0	0
1	0	1	0	1
2	0	0	2	2
3	0	1	2	3

3. Модальные операторы (табл. 4.9).

Таблица 4.9

F	$\Box F$	$\Diamond F$
0	0	0
1	0	3
2	0	3
3	3	3

ГЛАВА 5

ТЕОРИЯ АЛГОРИТМОВ

5.1. Общие сведения об алгоритмах и основные требования к ним

Известно, что в математике существуют различные вычислительные процессы чисто механического характера. В них решение для определенного класса задач выполняется всегда по одним и тем же правилам, которые используются в определенной последовательности (например, нахождение решения квадратного уравнения, выполнение арифметических действий над числами и т. п.), изменяются только исходные данные, к которым применяются указанные вычислительные процедуры для получения результата. Такие процессы принято называть ***алгоритмами***.

Примечание. Считается, что термин «алгоритм» происходит от искаженного имени узбекского математика Аль-Хорезми (*аль-Хорезми — ал Хорезми — ал Горезми — алгоритм*), который еще в IX в. разработал основополагающий трактат по арифметике и алгебре («Книга о восстановлении и противопоставлении»), где и предложил простейшие арифметические алгоритмы.

Определение 1. ***Алгоритм*** — это план действий, состоящий из последовательности понятных человеку операций, приводящих к искомому результату. Алгоритм состоит из элементарных шагов, число которых конечно.

Учитывая, что решаемые с помощью алгоритмов задачи являются однотипными, о них можно говорить как о массовой проблеме, и тогда возможно следующее определение для алгоритма.

Определение 2. Общий, единообразный, точно определяемый способ решения любой задачи из некоторой заданной массовой проблемы называется ***алгоритмом***.

Оба приведенных ниже определения относятся к нестрогим определениям, которые называются *интуитивными*.

Несмотря на то что существует множество всевозможных алгоритмов, используемых при решении различных задач, можно выделить основные требования, применяемые к любому алгоритму.

1. ***Понятность.*** Алгоритм должен быть доступен для понимания определенному классу пользователей.

2. ***Определенность.*** Это требование означает точность формулировок, исключение неоднозначности толкования на любом шаге алгоритма, т. е. при одних и тех же исходных данных задача должна иметь одно и то же решение.

3. ***Дискретность.*** Алгоритм должен быть построен таким образом, что если в начальный момент задается конечный набор исходных величин, то набор величин следующего шага может быть получен по определенному закону из величин предыдущего шага.

4. ***Элементарность шага.*** Закон получения следующего набора величин из предшествующего должен быть простым.

5. ***Массовость.*** Означает универсальность алгоритма для решения любой задачи из некоторого класса и возможность его использования при любых допустимых исходных данных.

6. ***Конечность и результативность.*** Состоит в получении искомого результата после конечного числа шагов.

Для формирования алгоритма используется конечный набор операций $O = \{O_1, O_2, \ldots, O_n\}$.

Совокупность величин, объединенных знаком операции, называется ***оператором***, а сами величины — ***операндами***.

Выделяют следующие типы элементарных операторов:

1. ***Сингулярный*** (когда операндом является одна величина).

2. ***Бинарный*** (выполняет операции с двумя операндами).

Обычно операторы, которые используются для вычислений, обозначаются буквами $A_1, A_2, \ldots, A_n$, здесь индекс имеет смысл метки, выделяющей данный оператор. Тогда алгоритм можно представить как последовательность таких операторов $(A_1, A_2, \ldots, A_n)$, которые выполняются дискретно в порядке их записи. Для обеспечения возможности изменения порядка действий вводят операции отношения — предикаты P_i, где P_i — условный оператор. В зависимости от истинности этого оператора выполняется переход к тому или иному следующему оператору алгоритма. Применяя P_i, можно ветвить алгоритм.

Группа операторов, выполняющихся многократно при одной реализации алгоритма, называется ***циклом***.

Если при решении задачи используется некоторый алгоритм, то процесс обычно разделяют на два этапа.

1. *Составление алгоритма* (описание), т. е. формализация процесса решения для некоторого класса задач.

2. *Реализация алгоритма*, т. е. применение построенного алгоритма к некоторому набору исходных данных для получения результата.

Существует множество способов описания алгоритма, можно выделить следующие *варианты описания*:

1. *Словесное описание.* Словесное описание выполняется на естественном языке, однако при этом должны быть четко выделены шаги. При описании шагов используются формулы, общепринятые математические знаки и символы. Описание считается корректным, если оно может быть воспринято и однозначно интерпретировано другим лицом.

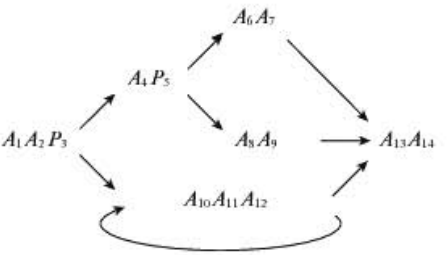

Рис. 5.1

2. *Линейная форма записи.* Записи для алгоритма на рис. 5.1 в линейной форме имеет вид

$$A_1\ A_2\ P_3 \overset{10}{\uparrow} A_4\ P_5 \overset{8}{\uparrow} A_6\ A_7 \overset{13}{\uparrow} A_8\ A_9 \overset{13}{\uparrow} A_{10}\ A_{11}\ P_{12} \overset{10}{\uparrow} A_{13}\ A_{14}.$$

Запись $A_i \overset{m}{\uparrow}$ означает, что после выполнения A_i будет выполняться оператор с меткой m. Запись $P_i \overset{m}{\uparrow}$ означает, что после выполнения P_i в случае его истинности выполняется следующий оператор, в противном случае осуществляется переход на оператор с меткой m.

3. *Представление алгоритма в виде структурной схемы или блок-схемы.* Такая схема представляет собой граф: вершинам соответствуют шаги (действия), а ребрам — переходы, отображающие последовательность выполнения шагов.

В качестве примера на рис. 5.2 приведена блок-схема описанного выше алгоритма.

Существуют государственные стандарты, которые определяют правила выполнения алгоритмов.

В блок-схеме каждый шаг может быть либо элементарным, либо многокомпонентным, представляющим собой самостоятельный алгоритм. Преимуществами такого представления являются:

1) возможность распараллеливания процесса разработки алгоритма;

2) возможность связывания алгоритмов при решении комплексных задач;

3) возможность пошаговой детализации алгоритма.

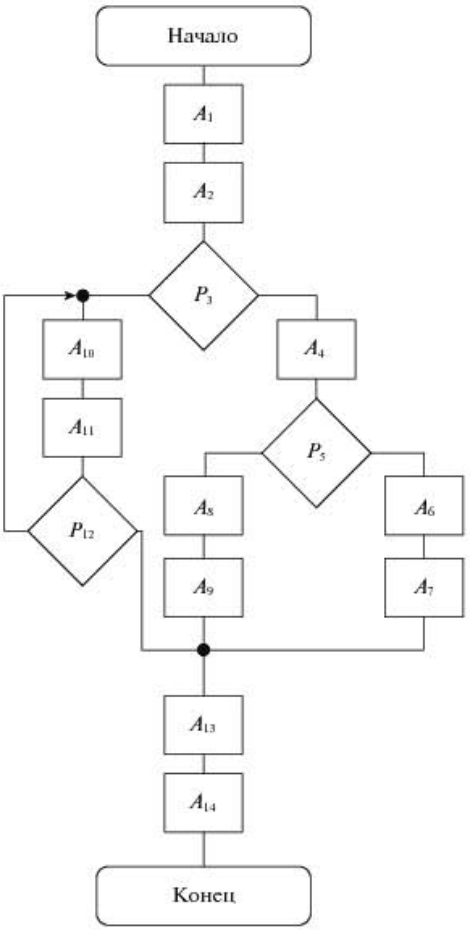

Рис. 5.2

В начале параграфа мы уже приводили два различных определения алгоритма. Различные исследователи теории алгоритмов в процессе ее развития выработали множество определений, это объяснялось тем, что они исходили из различных технических и логических соображений. Однако со временем было доказано, что все эти определения

равносильны, т. е. определяют одно и то же понятие. Поэтому сегодня в подходах к определению алгоритма можно выделить три основных направления.

Первое направление связано с машинной математикой. В нем сущность понятия алгоритма раскрывается путем рассмотрения процессов, осуществляемых в машине. Впервые это было сделано английским математиком Аланом Тьюрингом (*Turing*) в 1937 г. Он предложил самую общую и вместе с тем самую простую концепцию вычислительной машины.

Второе направление связано с уточнением понятия эффективно вычислимой функции. Результатом исследований было выделение особого класса — частично рекурсивных функций, которые имеют строгое математическое определение.

Третье направление связано с понятием нормальных алгоритмов А. А. Маркова, которые трактуют способы обработки некоторых синтаксических конструкций.

Рассмотрим более подробно каждое из этих направлений.

5.2. Рекурсивные функции

Исходная идея построения точного определения алгоритма, опирающегося на понятие рекурсивной функции, состоит в том, что любые данные можно закодировать натуральными числами в некоторой системе счисления, и тогда всякое их преобразование сведется к последовательности вычислительных операций, а результат обработки также будет представлять собой целое число. В данном подходе любой алгоритм, единый для данной числовой функции, вычисляет ее значение, а его элементарными шагами оказываются обычные арифметические и логические операции. Функции, удовлетворяющие таким требованиям, называются *вычислимыми*.

Введем понятие вычислимой функции так, чтобы это не было напрямую связано с понятием алгоритма. Функция вида $f: X \to Y$ будет называться *частичной*, если она определена не для всех $x \in X$. Совокупность тех элементов множества X, для которых определены элементы в Y, называется *областью определения функции*, а совокупность тех элементов Y, которые соответствуют элементам X, называют *областью значений функции*. Если область определения функции из X в Y совпадает с множеством X, то функцию называют *всюду определенной*.

Среди алгоритмических проблем довольно распространенной является такая, которая требует найти алгоритм для решения некоторой задачи, условия которой сформулированы следующим образом. Исходные данные представлены в виде конечной системы целочисленных параметров $x_1, x_2, \ldots, x_n$, а результатом является целое число y. Следовательно, стоит вопрос о существовании алгоритма для вычисления значения числовой функции, зависящей от аргументов $x_1, x_2, \ldots, x_n$.

Определение. Функция $y = f(x_1, x_2, \ldots, x_n)$ называется ***эффективно вычислимой***, если существует алгоритм, позволяющий вычислить ее значение.

Понятие алгоритма в этом определении берется в интуитивном смысле, поэтому и понятие эффективно вычислимой функции оказывается интуитивным. Однако при переходе от алгоритмов к вычислимым функциям возникает одно очень существенное обстоятельство, а именно: совокупность процессов, удовлетворяющих требованиям, предъявляемым к алгоритмам, достаточно обширна и не поддается полному описанию. Совокупность же вычислимых функций, удовлетворяющих перечисленным условиям, ограничена и легко описывается в обычных математических терминах. Переход от понятия алгоритма к понятию вычислимой функции удобен с той точки зрения, что позволяет перенести требования, предъявляемые к алгоритмам, на множество числовых функций. Эта точно описанная совокупность числовых функций, совпадающая с совокупностью всех вычислимых функций при самом широком понимании алгоритма, носит название совокупности ***рекурсивных функций***. Впервые это понятие было введено австрийским математиком Куртом Геделем в 1931 г.

Любая алгоритмическая модель, в том числе рекурсивные функции, должна предусматривать определение элементарных шагов алгоритма и способов построения из них какой-то последовательности преобразований, обеспечивающих необходимую последовательность обработки данных. В рекурсивной модели такими элементарными шагами являются так называемые *простейшие* числовые функции, из комбинации которых строятся более сложные функции. С учетом этого введем следующее определение.

Определение. ***Частично рекурсивная функция*** — это числовая функция, которая строится следующим образом. Определяется конечное число простейших функций и операций. Выполнимость функций должна быть очевидна, а с помощью операций из них можно создавать другие, более сложные функции.

Простейшие функции следующие:

1) $f(x)=x+1$ — *оператор сдвига*, или *функция следования*, позволяет получить любое число натурального ряда;

2) $O(x)=0$ — *оператор аннулирования*, или *константа* 0;

3) $I_n^m(x_1, x_2, \ldots, x_n)=x_m$, где $1\le m\le n$. Выражение I_n^m называется *оператором проектирования* или *селекторной функцией*. Каждая из функций I_n^m равна одному из своих аргументов. Индекс m выделяет m-ю переменную.

Как указано в определении, перечисленные простейшие функции всюду определены и интуитивно вычислимы. Над ними можно выполнить ряд операций (часто используется термин «*оператор*»), которые обладают таким свойством, что их применение к функциям, вычислимым в интуитивном смысле, порождает новые функции, также заведомо вычислимые в интуитивном смысле. Частичные функции, которые можно получить с помощью этих операторов из простейших функций, также будут частично рекурсивными, т. е. класс построенных таким образом частично рекурсивных функций совпадает с классом функций, допускающих алгоритмическое вычисление.

Набор операторов, обеспечивающих преобразование простейших функций, следующий.

1. *Суперпозиция функций.* Пусть задана функция $h(x_1, x_2, \ldots, x_m)$ и функции $q_1(x_1, x_2, \ldots, x_n)$, $q_2(x_1, x_2, \ldots, x_n)$, ... $q_m(x_1, x_2, \ldots, x_n)$, тогда говорят, что функция $\varphi(x_1, x_2, \ldots, x_n)$ получена суперпозицией из функций h и $q_1, q_2, \ldots, q_m$, если имеет место следующее равенство:

$$\varphi(x_1, x_2, \ldots, x_n)=h(q_1(x_1, x_2, \ldots, x_n), q_2(x_1, x_2, \ldots, x_n), \ldots, q_m(x_1, x_2, \ldots, x_n)).$$

Пример. Функции константы 1 и 2 могут быть получены следующей суперпозицией из простейших функций:

$f(0(x))=f(0)=0+1=1.$

$f(f(0(x)))=f(f(0))=f(1)=1+1=2.$

Очевидно, что таким же образом может быть получено любое число натурального ряда.

2. *Оператор примитивной рекурсии.* Говорят, что функция $f(x_1, x_2, \ldots, x_n, y)$ получена из функций $h(x_1, x_2, \ldots, x_n)$ и $\varphi(x_1, x_2, \ldots, x_n, y, z)$ с помощью оператора примитивной рекурсии, если она может быть задана схемой примитивной рекурсии:

$$\begin{cases} f(x_1, x_2, \ldots, x_n, 0) = h(x_1, x_2, \ldots, x_n); \\ f(x_1, x_2, \ldots, x_n, y+1) = \varphi(x_1, x_2, \ldots, x_n, y, f(x_1, x_2, \ldots, x_n, y)). \end{cases}$$

Приведенная схема справедлива для общего случая n-местной функции, т. е. когда $n \geq 1$, в том случае, когда $n = 0$, схема примитивной рекурсии имеет следующий вид:

$$\begin{cases} f(0) = a, \\ f(y+1) = \varphi(y, f(y)), \end{cases}$$

где a — постоянная одноместная функция, равная числу a (т. е. константа).

Определение. Частичная функция $f(x_1, \ldots, x_n)$ называется ***примитивно рекурсивной***, если ее можно получить конечным числом операций суперпозиции и примитивной рекурсии, исходя лишь из простейших функций.

Если функции h и φ интуитивно вычислимы, то будет интуитивно вычислима и f. Например, если $a_1, a_2, \ldots, a_n$ — это набор аргументов, соответствующих $x_1, x_2, \ldots, x_n$, то значения для f могут быть найдены следующим образом:

$f(x_1, x_2, \ldots, x_n, 0) = h(x_1, x_2, \ldots, x_n) = b_0$;

$f(x_1, x_2, \ldots, x_n, 1) = h(x_1, x_2, \ldots, x_n, 0, b_0) = b_1$;

$f(x_1, x_2, \ldots, x_n, 2) = h(x_1, x_2, \ldots, x_n, 1, b_1) = b_2$ и т. д.

Очевидно, что функция f будет всюду определена, если всюду определены функции h и φ.

Пример. Функция двух переменных $S(x, y) = x+y$ может быть получена по схеме примитивной рекурсии.

Известно, что для $x+y$ справедливы следующие выражения:

$x + 0 = x$;

$x + (y + 1) = (x + y) + 1$.

Следовательно, их можно переписать в виде

$S(x, 0) = x$;

$S(x, y + 1) = S(x, y) + 1$,

или иначе

$$\begin{cases} S(x,0) = J_1^1(x); \\ S(x, y+1) = \lambda(S(x+y)). \end{cases}$$

Из приведенных равенств видно, что функция двух переменных $S(x, y)$ получается по схеме примитивной рекурсии из функции одной переменной и функции трех переменных.

3. *Операция минимизации* (μ-*оператор*). Пусть задана некоторая функция $f(x,y)$. Фиксируя значение x, выясняем, при каком y $f(x,y)=0$.

Так как результат решения задачи зависит от x, то наименьшее значение y, при котором имеет место $f(x,y)=0$, является функцией от x, поэтому используется следующие обозначение:

$$\varphi(x)=\mu y\left[f(x,y)=0\right].$$

Приведенная запись трактуется как следующее утверждение: наименьшее y такое, что $f(x, y) = 0$.

Исходя из изложенного можно определить функцию и для нескольких переменных:

$$\varphi(x_1,x_2,\ldots,x_n)=\mu y\left[f(x_1,x_2,\ldots,x_n)=0\right].$$

Переход от функции $f(x_1,x_2,\ldots,x_n,y)$ к функции $\varphi(x_1,x_2,\ldots,x_n)$ принято называть ***применением*** μ-***оператора***.

Алгоритм вычисления функции φ можно представить в следующем виде.

Шаг 1. Вычисляется $f(x_1,x_2,\ldots,x_n,0)$. Если $f(x_1,x_2,\ldots,x_n,0)=0$, то полагается $\varphi(x_1,x_2,\ldots,x_n)=0$.

Если $f(x_1,x_2,\ldots,x_n,0)\neq 0$, то переход на шаг 2.

Шаг 2. Вычисляется $f(x_1,x_2,\ldots,x_n,1)$.

Если $f(x_1,x_2,\ldots,x_n,1)=0$, то полагается $\varphi(x_1,x_2,\ldots,x_n)=1$.

Если $f(x_1,x_2,\ldots,x_n,1)\neq 0$, то переход на шаг 3.

Шаг 3. Берется следующее значение y и повторяется действие, аналогичное предшествующему шагу.

Шаг N. Если перебраны все значения y и для любого из них $f(x_1,x_2,\ldots,x_n,y)\neq 0$, то функция $\varphi(x_1,x_2,\ldots,x_n)$ в этом случае считается неопределенной.

Пример. Рассмотрим функцию $f(x, y) = x - y$, которая может быть получена с помощью оператора минимизации

$f(x, y) = \mu z(y + z = x) = \mu z\,[I_3^2(x, y, z) + I_3^3(x, y, z) = I_3^1(x, y, z)]$.

Вычислим, например, $f(7, 2)$, т. е. значение функции при $y = 2$ и $x = 7$. Положим $y = 2$, а x будем придавать последовательные значения:

$z=0$, $\quad 2+0=2\neq 7$;

$z=1$, $\quad 2+1=3\neq 7$;

$z=2$, $\quad 2+2=4\neq 7$;

$z = 3$, $2 + 3 = 5 \neq 7$;
$z = 4$, $2 + 4 = 6 \neq 7$;
$z = 5$, $2 + 5 = 7 = 7$.

Таким образом, найдено значение функции $f(7,2) = 5$.

Определение. Функция $f(x_1, x_2, \ldots, x_n)$ называется ***частично рекурсивной***, если она может быть получена в конечное число шагов из простейших функций с помощью операций суперпозиции, схем примитивной рекурсии и μ-оператора исходя лишь из простейших функций.

Определение. Функция $f(x_1, x_2, \ldots, x_n)$ называется ***общерекурсивной***, если она частично рекурсивна и всюду определена.

Общерекурсивными можно, например, считать следующие функций:

1) $\lambda(x)$;
2) $O(x)$;
3) $I_n^m(x)$;
4) $f(y,x) = y + x$;
5) $f(y,x) = y \cdot x$.

Как видно из приведенных определений, класс частично рекурсивных функций шире класса примитивно рекурсивных функций, так как все примитивно рекурсивные функции являются всюду определенными, а среди частично рекурсивных функций встречаются функции, не всюду определенные, а также нигде не определенные.

Понятие частично рекурсивной функции является одним из базовых понятий теории алгоритмов. Его значение состоит в том, что, с одной стороны, каждая стандартно заданная частично рекурсивная функция вычислима путем некоторой процедуры механического характера, отвечающей интуитивному представлению об алгоритмах. С другой стороны, какие бы классы точно очерченных алгоритмов ни строились, во всех случаях неизменно оказывалось, что вычислимые посредством них числовые функции являлись частично рекурсивными. Поэтому общепринятой является научная гипотеза, формулируемая как тезис Черча.

Определение. ***Тезис А. Черча:*** Каждая интуитивно вычислимая функция является частично рекурсивной, или другими словами: класс алгоритмически вычислимых частичных числовых функций совпадает с классом всех частично рекурсивных функций.

Приведенный тезис дает алгоритмическое истолкование понятия частично рекурсивной функции. Его нельзя доказать, так как он связы-

вает нестрогое понятие интуитивно вычислимой функции со строгим математическим понятием частично рекурсивной функции. Однако проводимые в течение ряда лет исследования показали целесообразность того, что понятия частично рекурсивной функции и вычислимой частичной функции можно считать эквивалентными.

Тезис Черча оказался достаточным, чтобы придать необходимую точность формулировкам алгоритмических проблем и в ряде случаев сделать возможным доказательство их неразрешимости. Причина заключается в том, что обычно в алгоритмических проблемах математики речь идет не об алгоритмах, а о вычислимости некоторых специальным образом построенных функций. В силу тезиса Черча вопрос о вычислимости функции равносилен вопросу о ее рекурсивности. Понятие рекурсивной функции строгое. Поэтому обычная математическая техника позволяет иногда непосредственно доказать, что решающая задачу функция не может быть рекурсивной.

5.3. Машина Тьюринга

Машина Тьюринга (МТ) — это абстрактное устройство, идея которого была предложена американским математиком Э. Постом и англичанином А. Тьюрингом. Их идея основывалась на следующей посылке. Если для решения некоторой массовой проблемы известен алгоритм, то для его реализации необходимо лишь четко следовать указаниям алгоритма. Автоматизм, присущий этой операции, естественно было бы передать от человека машине. Общий вид МТ приведен на рис. 5.3.

МТ состоит из следующих компонент: управляющего устройства, ленты, устройства обращения к ленте.

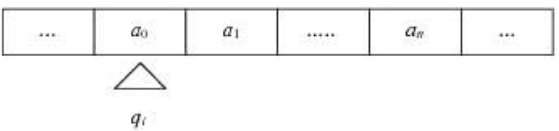

Рис. 5.3

1. ***Управляющее устройство*** может находиться в одном из состояний, образующих множество $Q = \{q_1, q_2, q_3, \ldots, q_i, \ldots, q_n\}$. Множество Q называется внутренним алфавитом МТ. Для любой МТ число состояний конечно и фиксировано. Множество Q выступает в качестве внутренней памяти.

2. ***Лента*** разбита на ячейки. В каждой ячейке должен быть записан один из символов конечного алфавита из множества $A = \{a_1, a_2, a_3, \ldots, a_j, \ldots, a_m\}$. В начале на ленте записывается входное слово, по одной букве в расположенных подряд ячейках. В пустых ячейках записывается знак пробела λ. Лента считается бесконечной, на ней фиксируются исходные и промежуточные данные, а также записывается результат. Лента выступает в качестве внешней памяти. Так как она считается бесконечной, то это свидетельствует о том, что МТ является модельным устройством, поскольку ни одно реальное устройство не может обладать памятью бесконечного размера. Множество A называется внешним алфавитом МТ.

3. ***Устройство обращения к ленте*** (головка) осуществляет считывание и запись данных. В общем случае МТ работает следующим образом.

1. Считывается значение символа a_j, над которым устанавливается головка.

2. В зависимости от пары значений $q_i\, a_j$ производится следующая последовательность действий:

а) запись нового значения a'_j в ту ячейку, над которой стоит головка;

б) смещение головки на одну ячейку вправо или влево;

в) переход в новое состояние q'_i.

Можно сказать, что в МТ реализуется система предельно простых команд обработки информации. Эта система команд обработки дополняется также предельно простой системой команд перемещения головки по ленте: на ячейку влево, на ячейку вправо и остаться на месте, т. е. адрес обозреваемой ячейки в результате выполнения команды может либо измениться на единицу, либо остаться неизменным. Элементарность этих команд означает, что при необходимости обращения к содержимому некоторой ячейки она отыскивается только посредством цепочки отдельных сдвигов на одну ячейку. Это, конечно, удлиняет процесс обработки, но позволяет обойтись без нумерации ячеек и использования команд перехода по адресу, т. е. сокращает количество истинно элементарных шагов, что важно в теоретическом отношении.

Реализация указанной последовательности действий называется ***тактом***, т. е. работа МТ состоит из отдельных тактов.

В зависимости от записанной исходной информации на ленте возможны два варианта работы МТ. Пусть на ленте записана информация B_0 (построенная из символов A), тогда:

1) после конечного числа тактов МТ приходит в заключительное состояние q_z, и на ленте изображается некоторая информация B, также

построенная из символов A. В этом случае говорят, что МТ применима к B_0 и перерабатывает ее в B;

2) если МТ не приходит в состояние q_z, то говорят, что машина не применима к B_0.

Конкретная машина Тьюринга задается перечислением элементов множеств A и Q, а также набором правил преобразования. Очевидно, что различных множеств A, Q и правил преобразования может быть бесконечно много, т. е. и машин Тьюринга также может быть бесконечно много.

Правила преобразования, которые выполняет МТ, определяются системой команд, которые описываются следующим образом:

$$q_i\, a_j \to q_i'\, a_j'\, d_k,$$

где d_k — смещение (R — вправо, L — влево, E — без изменений).

Существует несколько способов представления набора правил преобразования, т. е. работа МТ может быть описана следующим образом:

— системой команд;
— таблицей;
— диаграммой (графом) переходов.

В таблице функционирования МТ возможным состояниям соответствуют строки, строкам — входные символы a_j. На их пересечении записывается следующая команда. Таблица 5.1 имеет следующий вид.

Таблица 5.1

	a_1	a_2	…	a_j	…	a_m	λ	*
q_1								
q_2	$q_2'\, a_1'\, d_k$							
…								
q_i								
…								
q_z								

Примечание. Символ «*» обозначает знак операции.

Диаграмма переходов — это граф, вершинам которого соответствуют состояния МТ, а ребрам — команды (рис. 5.4).

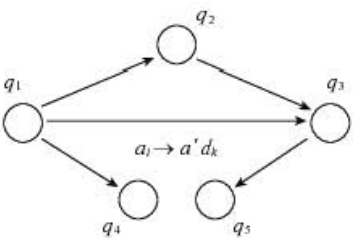

Рис. 5.4

Рассмотрим **примеры реализации МТ**.

Пример 1. Пусть на ленте записаны два натуральных числа в унарном коде (унарный код — это представление числа в виде соответствующей последовательности единиц, например 5 = 11111 = 1^5). Числа разделены символом операции сложения. Требуется реализовать операцию сложения таким образом, чтобы после ее реализации на ленте был записан результат, а управляющее устройство находилось на первом символе результата. Работу МТ опишем в виде системы команд.

Для сложения чисел 2 и 3 начальное состояние МТ имеет вид, показанный на рис. 5.5.

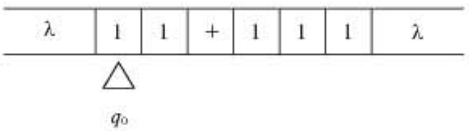

Рис. 5.5

Реализация алгоритма будет базироваться на следующей идее. Из исходного положения требуется дойти до символа «+», заменить его на символ «1», затем дойти до конца записи и заменить последнюю единицу на символ пробела «λ», после чего управляющее устройство следует вернуть в начальное положение. Опишем это в виде системы команд:

$$q_0 1 \to q_1 1\ R;$$
$$q_1 1 \to q_1 1\ R;$$
$$q_1 + \to q_2 1\ R;$$
$$q_2 1 \to q_2 1\ R;$$
$$q_2 \lambda \to q_3 \lambda\ L;$$
$$q_3 1 \to q_4 \lambda\ L;$$
$$q_4 1 \to q_4 1\ L;$$
$$q_4 \lambda \to q_z \lambda\ R.$$

Для приведенного примера алфавиты A и Q имеют вид

$$A = \{1, +, \lambda\}; \quad Q = \{q_1, q_2, q_3, q_4\}.$$

Пример 2. Иногда требуется реализовать задачу таким образом, чтобы на ленте сохранились исходные данные и вместе с ними результаты выполненных действий. Пусть требуется выполнить с помощью МТ умножение двух натуральных чисел, представленных в унитарном коде, с сохранением исходных данных. Тогда начальное состояние МТ для чисел 2 и 4 должно иметь вид, показанный на рис. 5.6, а заключительное — на рис. 5.7.

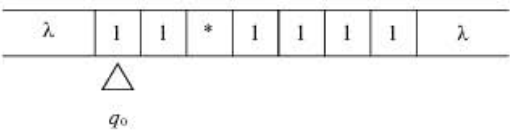

Рис. 5.6

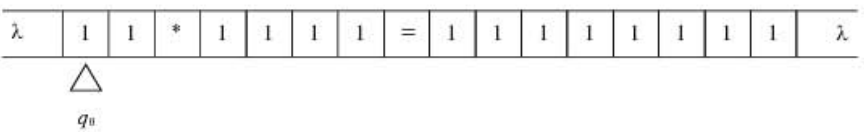

Рис. 5.7

В основе алгоритма будет лежать следующая идея: записав символ «=» после исходных данных, за ним следует повторить второе число столько раз, какое количество единиц содержит первое число. Описание МТ в виде системы команд имеет следующий вид:

$$q_0 1 \to q_1 0\ R; \qquad q_0 * \to q_8 * L;$$
$$q_1 1 \to q_1 1\ R; \qquad q_1 * \to q_2 * R;$$
$$q_2 1 \to q_3 0\ R; \qquad q_2 = \to q_6 = L;$$
$$q_3 = \to q_4 = R; \qquad q_3 \lambda \to q_4 = R;$$
$$q_4 \lambda \to q_5 1\ L; \qquad q_4 1 \to q_4 1\ R;$$
$$q_5 1 \to q_5 1\ L; \qquad q_5 = \to q_5 = L;$$
$$q_5 0 \to q_2 0\ R; \qquad q_6 * \to q_7 * L;$$
$$q_6 0 \to q_6 1\ L; \qquad q_7 1 \to q_7 1\ L;$$
$$q_7 0 \to q_0 0\ R; \qquad q_8 0 \to q_8 1\ L;$$
$$q_8 \lambda \to q_z \lambda\ R.$$

Множества A и Q определены так:

$$A = \{1, *, =, \lambda, 0\} \text{ и } Q = \{q_1, q_2, q_3, \ldots, q_8\}.$$

Полное состояние МТ, однозначно определяющее ее дальнейшее поведение, называется ***конфигурацией*** или ***машинным словом***. Конфигурация определяется следующим образом: $K_i = a_e\, q_i\, a_R$, где q_i — текущее состояние; a_e — последовательность символов, записанных слева; a_R — последовательность символов, записанных справа, включая ячейку, над которой находится головка. Вся совокупность входных конфигураций, при которых машина обеспечивает получение результата, образует *класс решаемых задач*. Очевидно, применять машину Тьюринга для задачи, не входящей в класс решаемых, бессмысленно. С другой стороны, во многих случаях возможно расширение класса решаемых задач за счет создания другой машины Тьюринга. Возникает вопрос: можно ли построить такую универсальную машину (хотя бы на теоретическом уровне), которая решала бы любую задачу? Здесь мы подошли к вопросу об алгоритмической разрешимости, который будет исследован позднее.

Запись $T(\alpha)$ — это обозначение машины T с исходными данными α. Запись $T(\alpha_1\, \alpha_2) = \beta_1\beta_2$ означает, что МТ перерабатывает входное слово $\alpha_1\, \alpha_2$ в выходное слово $\beta_1\beta_2$.

Если для некоторой функции $f(v) = \omega$ существует МТ, которая правильно ее вычисляет, будучи запущенной в начальной конфигурации, то эта функция называется ***правильно вычислимой по Тьюрингу***.

Тьюринг убедительно показал разнообразие возможностей предложенной им конструкции и пришел к выводу, который известен как ***тезис Тьюринга***: *всякий алгоритм может быть реализован соответствующей машиной Тьюринга*.

В разделе 5.2, посвященном частично рекурсивным функциям, рассмотрена операция композиции, которая позволяет строить новую функцию на основе существующих функций. Аналогичную операцию можно реализовать применительно к машинам Тьюринга.

Пусть функция g определена как композиция функций f_1 и f_2, т. е. $g = f_2(f_1(x))$. Результат g получается как результат применения f_2 к результату работы f_1. Для того чтобы эта функция была определена, необходимо и достаточно, чтобы f_1 была определена на x, а f_2 на $f_1(x)$.

Теорема 5.1. Если функции $f_1(x)$ и $f_2(y)$ вычислимы по Тьюрингу, то их композиция $f_2(f_1(x))$ также вычислима по Тьюрингу.

Построенную таким образом машину T называют композицией машин T_1 и T_2 и обозначают $T_2(T_1)$. Приведем пример композиции машин Тьюринга.

По своему устройству МТ крайне примитивна. Она намного проще самых простых компьютеров. Примитивизм состоит в том, что у нее предельно прост набор элементарных операций, производимых головкой, — считывание и запись, а также в том, что доступ к ячейкам памяти в ней происходит не по адресу, как в компьютерах, а путем последовательного перемещения вдоль ленты. По этой причине даже такие простые действия, как сложение с единицей или сравнение двух символов на равенство, МТ производит за несколько шагов, а обычные операции — сложение, умножение или деление — требуют гораздо большего числа элементарных действий. Однако МТ была придумана не как модель или прототип реальных вычислительных машин, а для того чтобы показать принципиальную возможность построения сколь угодно сложного алгоритма из предельно простых операций, причем сами операции и переход от одной к последующей машина выполняет автоматически.

МТ — один из путей уточнения понятия алгоритма. Тьюринг убедительно показал разнообразие возможностей предложенной им конструкции и пришел к выводу, который известен как тезис Тьюринга.

Определение. ***Тезис А. Тьюринга:*** всякий алгоритм может быть задан посредством тьюринговой функциональной схемы и реализован в соответствующей машине Тьюринга.

Этот тезис, также как и тезис Черча, невозможно доказать, так как он связывает нестрогое определение понятия алгоритма со строгим определением машины Тьюринга. Поэтому этот тезис может быть опровергнут, если удастся привести пример алгоритма, который невозможно реализовать с помощью тьюринговой схемы, однако пока все известные до сих пор алгоритмы могут быть заданы посредством тьюринговых схем.

5.4. Нормальные алгоритмы А. А. Маркова

Нормальный алгоритм — понятие, введенное в *конструктивную математику* русским ученым А. А. Марковым. Под конструктивной математикой понимается такое направление в математике, которое, как и классическая математика, имеет своим предметом исследование форм и количественных отношений объективной действительности, но отличается от классической математики некоторыми особенностями. Если в классической математике исследователь имеет дело с объектами, о которых известно, что они обладают такими-то свойствами,

и при этом полностью отвлекается от способов построения такого объекта, то в конструктивной математике исследователь ограничивается *конструктивными объектами*, существование которых считается доказанным лишь тогда, когда указывается способ потенциально осуществимого построения (конструирования) данного объекта. Понятие конструктивного объекта не определяется, а лишь поясняется примерами. В качестве элементарного конструктивного объекта берутся, например, буквы, входящие в то или иное слово. Из этих элементарных конструктивных объектов по некоторым правилам, принятым по соглашению, строится слово, а из слов — еще более сложные конструктивные объекты. Существование такого конструктивного объекта считается доказанным лишь тогда, когда указывается способ потенциально осуществимого построения (конструирования) объекта. Так, например, если применять алгоритм вычитания столбиком к паре (404, 55), то последовательно возникнут такие конструктивные объекты:

404	404	404	404
55	55	55	55
----	----	----	----
	9	49	349

Каждый конструктивный объект определяется непосредственно предшествующим конструктивным объектом. В операциях с конструктивными объектами не применяется абстракция актуальной бесконечности, т. е. бесконечности завершенной, из которой исходит классическая математика. В конструктивной математике руководствуются абстракцией потенциальной осуществимости, согласно которой бесконечное множество не дано в завершенном виде, ибо оно есть бесконечность потенциальная, его можно только неограниченно продолжать строить (конструировать) по определенным правилам.

Другими словами, классическая и конструктивная математики поразному определяют понятие «существование» математического объекта. Если в классической математике объект признается существующим, когда он не несет в себе формально-логического противоречия, то в конструктивной математике существовать — значит быть построенным.

Рассмотрим, каким образом осуществляется переход от ранее введенного интуитивного определения алгоритма к понятию «алгоритм в алфавите A», которое является ключевым понятием в рассматриваемой модели представления алгоритмов.

Определение 1. ***Алгоритмом в алфавите A*** называется всякое общепонятное точное *предписание,* определяющее потенциально осуществимый процесс над словами из A, допускающее любое слово в качестве исходного и последовательно определяющее новые слова в этом алфавите.

Алгоритм *применим к* некоторому слову P, если, отправляясь от этого слова и действуя согласно предписанием, в конце концов получаете новое слово Q, на котором процесс оборвется. Будем тогда говорить, что алгоритм перерабатывает P в Q.

Пример. Сформируем алгоритм, удовлетворяющий введенному определению. Пусть требуется переписать заданное слово P в обратном порядке следования букв, которое задано следующей последовательностью символов: $a_1, a_2, ..., a_n$. Алгоритм будет иметь следующий вид.

Шаг 1. Положить счетчик i равным n. Значение слова Q положить равным пустому слову, т. е. не содержащему ни одного символа.

Шаг 2. Взять символ с индексом i и присоединить его справа к строящемуся слову Q.

Шаг 3. Если счетчик i равен 1, то полученное слово Q есть результат и алгоритм свою работу закончил. Остановка.

Если счетчик i не равен 1, то положить $i = i - 1$ и перейти к шагу 2.

Очевидно, что этот алгоритм представляет собой совершенно точное предписание, применимое к любому слову.

Однако *определение* 1 является слишком общим, так как уточнение понятия «алгоритм в алфавите A» связано с использованием аппарата подстановок, т. е. с построением ассоциативного исчисления. Рассмотрим более уточненный вариант определения.

Определение 2. Будем считать, что ***алгоритм в алфавите A*** задается в виде некоторой системы допустимых подстановок, дополненной общепонятным точным предписанием о том, в каком порядке и как нужно применять допустимые подстановки и когда наступает остановка.

Пример. Рассмотрим вариант определения алгоритма в смысле *определения* 2. Пусть алфавит A содержит три буквы: $A = \{a, b, c\}$, а алгоритм определен с помощью системы подстановок, представленной в табл. 5.2.

Таблица 5.2

№	Комбинация символов	Подстановка
1	*cb*	*cc*
2	*cca*	*ab*
3	*ab*	*bca*

Из следующего указания о способе применения этих подстановок: исходя из произвольного слова P следует просмотреть схему подстановок в том порядке, в каком они выписаны, разыскивая первую формулу, левая часть которой входит в P. Если же такой формулы не найдется, то процесс обрывается сразу же. В противном случае берется первая из найденных формул и делается подстановка ее правой части вместо *первого* вхождения ее левой части в слово P, что дает новое слово P_1, в алфавите A. Затем слово P_1 играет роль P, т. е. вновь берется в качестве исходного, и процесс повторяется. Остановка наступает в том случае, если на n-м шаге получено слово P_n, в которое не входит ни одна из левых частей формул схемы подстановок.

Итак, схема подстановок вместе с указанием, как ими пользоваться, определяет алгоритм в алфавите A. Этот алгоритм перерабатывает слово *babaac* в слово *bbcaaac* (применена 3-я формула), слово *cbacacb* — последовательно в слова: *cbacacb, ccacacb, abcacb, bcacacb, bcacacc* (применены формулы: 1, 2, 3, 1) и, наконец, в *bcacacc*, на котором процесс обрывается; слово *bcacabc* порождает бесконечно повторяющуюся последовательность слов *bcacabc, bcacbcac, bcacccac, bcacabc* (применены формулы: 3, 1, 2) и т. д., т. е. остановка не наступит, и следовательно, к слову *bcacabc* алгоритм не применим.

Очевидно, что в приведенном примере возможны три результата при начале преобразований из исходного слова:

1) тупиковое состояние (например, слово *bbcaaac*);
2) бесконечное кружение по циклу (например, слово *bcacabc*);
3) достаточно долгое движение без попадания в циклы.

С первого взгляда можно решить, что определение алгоритма в смысле *определения 1* у же, чем определение в смысле *определения 2*. Однако оказывается, что на деле такого сужения нет, ибо для любого известного алгоритма в смысле *определения 1* может быть построен эквивалентный ему алгоритм в смысле *определения 2*. Это, конечно, не доказательство того, что *определения 1* и *2* равносильны: такого доказательства не может существовать вообще в силу неточности и расплывчатости обоих определений (везде фигурируют слова «общепонятное, точное предписание»).

Во всяком случае очевидно, что *определение 2* — это шаг вперед по пути уточнения понятия «алгоритм». Уточним теперь понятие эквивалентности алгоритмов.

Определение. Два алгоритма A_1 и A_2 в некотором алфавите называются ***эквивалентными***, если области их применимости совпадают и результаты переработки ими любого слова из общей области при-

менимости также совпадают. Иначе говоря, если алгоритм A_1 применим к некоторому слову P, то и A_2 должен быть применим к этому слову, и наоборот. Оба алгоритма должны перерабатывать слово P в некоторое слово Q. Если же один из алгоритмов не применим к некоторому слову B, то и другой алгоритм должен быть не применим к нему.

Для того чтобы дать точное математическое определение алгоритма, потребовалось сделать еще один шаг по пути дальнейшего уточнения. Этот шаг был сделан А.А. Марковым. Построенный им нормальный *алгоритм*, также как и алгоритм в смысле *определения 2*, выражен в терминах системы подстановок, однако вместо расплывчатого «общепонятного указания» о том, как пользоваться подстановками, Марков дал стандартные, раз и навсегда определенные точные указания о порядке использования подстановок.

Определение. Рассмотрим определение ***нормального алгоритма Маркова***. Задается алфавит A и фиксируется схема подстановок. Алгоритм предписывает исходя из произвольного слова P и алфавита A просмотреть формулы подстановок в том порядке, в каком они заданы в схеме, разыскивая формулу с левой частью, входящей в P. Если такой формулы не найдется, то процесс обрывается. В противном случае берется первая из таких формул и делается подстановка ее правой части вместо первого вхождения ее левой части в P, что дает новое слово P_1 в алфавите A. После первого шага процесса приступают ко второму его шагу, отличающемуся от первого только тем, что роль P играет P_1. Далее делают аналогичный третий шаг и т. д. до тех пор, пока не придется оборвать процесс. Оборваться же он может лишь двумя способами: во-первых, когда получим такое слово P_n, что ни одна из левых частей формул схемы подстановок не будет в него входить; во-вторых, когда при получении слова P_n придется применить последнюю формулу. В обоих этих случаях считаем, что рассматриваемый алгоритм перерабатывает слово P в слово P_n.

Теперь становится видно, что приведенный в *определении 2* пример является примером «почти нормального» алгоритма. Вся разница состоит в том, что там остановка наступает лишь в одном случае, когда ни одна из подстановок не применима, а здесь остановка может наступать в двух случаях.

Различные нормальные алгоритмы отличаются друг от друга лишь алфавитами и системами допустимых подстановок. Чтобы задать какой-либо нормальный алгоритм, достаточно задать алфавит и систему подстановок.

Понятие алгоритма в некотором алфавите уточнено А. А. Марковым следующим образом: *всякий алгоритм в алфавите А эквивалентен некоторому нормальному алгоритму в этом же алфавите*.

Это утверждение является гипотезой; оно не может быть строго доказано, так как, с одной стороны, фигурирует интуитивное понятие «всякий алгоритм», а с другой — строго определенное понятие «нормальный алгоритм». На это утверждение можно смотреть как на закон, который не доказан, но проверен и подтвержден опытом.

В пользу высказанной гипотезы говорит и тот факт, что никому еще не удалось сформулировать пример такого алгоритма в алфавите *А*, для которого нельзя было бы построить эквивалентный ему нормальный алгоритм.

Можно отметить, что теория нормальных алгоритмов строится в рамках абстракции потенциальной осуществимости. *Абстракция потенциальной осуществимости* — один из видов абстракции, при которой мысленно отвлекаются от реальных границ конструктивных возможностей человеческого создания, связанных с ограниченностью жизни человека в пространстве и времени, и руководствуются принципами потенциальной, т. е. становящейся, бесконечности, в отличие от абстракции актуальной, т. е. завершенной, бесконечности.

Этот вид абстракции не предполагает индивидуализации каждого элемента бесконечного множества, не предполагает, что может быть осуществлено бесконечное число операций, но основывается на том, что может быть осуществлено любое конечное число операций — шагов, букв, чисел и т. п. Например, при рассмотрении слов в данном алфавите абстракция потенциальной осуществимости будет означать, по А. А. Маркову, отвлечение от практических границ наших возможностей в пространстве, времени и материале при построении слова. Это бывает в том случае, когда отвлекаются от практической невозможности написать на данной доске мелом сколь угодно длинные слова и начинают рассуждать так, как если бы указанная процедура была возможна. Понимать под «построением» с точки зрения абстракции потенциальной осуществимости (дается и другая трактовка) — это значит понимать под «построением» не только практически выполнимое в данных материальных условиях построение, но и построение потенциально осуществимое, т. е. осуществимое в предположении, что после каждого шага процесса построения требуемого слова располагаем возможностями для выполнения следующего шага.

Абстракция потенциальной осуществимости наиболее широко применяется в области информационных технологий и вычислительной

техники, она лежит в основаниях таких теорий, как теория алгоритмов, теория абстрактных автоматов, булевых алгебр и т. п., составляющих теоретический фундамент вычислительных систем. Но теории, применяющие абстракцию потенциальной осуществимости, как и любая другая теория, имеют ограниченную область применения. Показано, что допустимость применения абстракции потенциальной осуществимости ограничена предположением о том, что изменения некоторого свойства объекта не приводят к изменению основных свойств (качеств) данного объекта. Построенные в принятом алфавите слова надо рассматривать как потенциально осуществимые конструктивные объекты.

В заключение можно отметить, что алгоритм А. А. Маркова можно рассматривать наравне с частично рекурсивными функциями и машинами Тьюринга как «стандартную форму» представления для любого заданного алгоритма, т. е. полагается, что вообще любой алгоритм может быть задан в форме нормального алгоритма Маркова. Это утверждение можно рассматривать как гипотезу, и притом значительно менее ясную, чем гипотеза Маркова, о которой шла речь выше, так как она не может даже быть высказана в точных терминах, но интуитивный смысл ее очевиден.

5.5. Сравнительный анализ основных моделей представления алгоритмов

Известно, что все рассмотренные ранее модели алгоритмов возникли в результате попыток решить как теоретические проблемы (например, определение алгоритмической разрешимости задачи), так и практические (например, определение принципов работы устройства, производящего автоматизированную обработку информации). Различные варианты решения проблем привели к построению так называемых абстрактных алгоритмических моделей, которые приведены на рис. 5.8. Очевидно, что следует рассмотреть взаимосвязь отдельных моделей.

Традиционно все алгоритмические задачи принято делить на два больших класса:

1) задачи, связанные с вычислением значения функции;

2) задачи, связанные с распознаванием принадлежности объекта заданному множеству, т. е. проверка обладания объектом заданным свойством.

Рис. 5.8

В первом случае алгоритм Q начинает работать с исходными данными, представленными в виде некоторого слова P, составленным на основе алфавита A, и за конечное число шагов (преобразований) он должен остановиться и выдать результат $P_n = f_q(P)$. Результат зависит, т. е. является функцией, от исходного слова, а также последовательности обработки, т. е. самого алгоритма. При этом вычисление понимается в широком смысле — как алфавитное преобразование.

В задачах, отнесенных ко второму классу, в результате выполнения алгоритма проверяется истинность высказывания, что $x \in M$, и выдается один из двух возможных результатов: истина или ложь. Этот класс можно считать разновидностью первого, поскольку предикат — это функция, принимающая два значения в зависимости от условия. Тем не менее разделение этих классов задач полезно, так как приводит к двум важным понятиям теории алгоритмов: вычислимая функция и разрешимое множество. *Функция* называется *вычислимой*, если имеется алгоритм, вычисляющий ее значение. *Множество* называется *разрешимым*, если имеется алгоритм, который для любого объекта позволяет определить, принадлежит он данному множеству или нет.

Эти определения не являются строгими, так как не позволяют предсказать, окажется ли некоторая функция вычислимой или нет. Поэтому очень важным моментом для построения теории алгоритмов было формулирование тезиса Черча, который утверждал, что всякая частично рекурсивная функция является вычислимой. Другими словами, если функцию удалось простроить с помощью суперпозиции, рекурсии и минимизации из простейших функций, то существует алгоритм, ее вычисляющий. Далее последовал тезис Тьюринга, утверждавший, что для всякой вычислимой функции можно построить машину Тьюринга, которая ее вычисляет. Позднее была доказана теорема о сводимости одной алгоритмической модели к другой, следствием

которой явились утверждения типа: «любую рекурсивную функцию можно вычислить с помощью соответствующей машины Тьюринга» или «для любой задачи, решаемой с помощью машины Тьюринга, существует решающий ее нормальный алгоритм Маркова». Таким образом, все модели оказываются эквивалентными, и это имеет глубокий смысл, так как результат обработки информации определяется алгоритмом и входными данными, но не зависит от вида алгоритмической модели.

5.6. Проблема алгоритмической разрешимости

Всякому алгоритму соответствует задача, для решения которой он был построен. Обратное утверждение в общем случае является неверным по двум причинам: во-первых, одна и та же задача может решаться различными алгоритмами; во-вторых, можно предположить пока, что имеются задачи, для которых алгоритм вообще не может быть построен.

Под термином «алгоритм» в математике обычно понимается процедура, позволяющая путем выполнения последовательности определенных элементарных шагов получать однозначный результат, не зависящий от того, кто эти шаги выполнял, т. е. само построение решения можно считать доказательством существования алгоритма. Однако известен целый ряд математических задач, разрешить которые в общем виде не удавалось. Это в свою очередь привело к проблеме, которую можно сформулировать так: возможно ли, не решая задачи, доказать, что она алгоритмически неразрешима, т. е. невозможно построить алгоритм, который обеспечил бы ее общее решение. Решение этой проблемы очень важно в первую очередь с практической точки зрения, так как сразу позволит избежать затрат времени и ресурсов на решение проблем, которые являются алгоритмически неразрешимыми. Кроме того, она дала толчок развитию и теоретической мысли, потому что понадобилось сначала дать строгое определение алгоритма, без чего обсуждение его существования просто не имело смысла. Построение такого определения, как известно, привело к появлению формальных алгоритмических моделей, что дало возможность математического доказательства неразрешимости ряда задач. Доказательство сводилось к тому, что показывалась невозможность построения рекурсивной функции, решающей задачу, либо невозможность построения машины Тьюринга для нее, либо невозможность построе-

ния нормального алгоритма Маркова. То есть задача считается алгоритмически неразрешимой, если не существует машины Тьюринга, или рекурсивной функции, или нормального алгоритма Маркова, которые ее решают, что естественным образом вытекает из уже показанной ранее эквивалентности алгоритмических моделей.

Первые доказательства алгоритмической неразрешимости касались самой теории алгоритмов. Было доказано, что неразрешима задача установления истинности произвольной формулы исчисления предикатов, т. е. исчисление предикатов неразрешимо (теорема была доказана в 1936 г. Черчем). Кроме того, к алгоритмически неразрешимой относится и так называемая проблема остановки, формулировка которой звучит следующим образом: можно ли по описанию алгоритма и входным данным установить, завершится ли выполнение алгоритма результативной остановкой? Эту проблему легко проиллюстрировать практическим примером. Действительно, одна из самых распространенных ошибок при разработке программ связана с зацикливанием, т. е. ситуацией, когда цикл не завершается и не происходит завершения работы программы и остановки. Неразрешимость проблемы остановки означает, что нельзя создать общий, применимый для любой задачи алгоритм отладки программ. Неразрешимой оказывается и проблема распознавания эквивалентности алгоритмов: нельзя построить алгоритм, который бы для любых двух алгоритмов выяснял, всегда ли они приводят к одному и тому же результату или нет.

Несомненная польза определения алгоритмической неразрешимости состоит в том, что если для задачи доказано, что она алгоритмически неразрешима и это доказательство имеет смысл закона, то можно не тратить усилия на поиск решения, подобно тому, как законы сохранения энергии в физике делают бессмысленными попытки построения вечного двигателя. Однако следует помнить, что алгоритмическая неразрешимость какой-либо задачи в общем виде допускает, что имеют решения какие-либо ее частные случаи. Можно считать справедливым и обратное утверждение: решение частного случая задачи не означает, что ее можно решить в самом общем случае, т. е. не свидетельствует о ее общей алгоритмической разрешимости.

Важность и практическая значимость абстрактных алгоритмических систем состоит в том, что они позволяют оценить возможность нахождения общего решения некоторого класса задач. Существование алгоритмически неразрешимых проблем приводит к тому, что оказывается невозможным построить универсальный алгоритм, пригодный для решения любой задачи, а следовательно, не имеет смысла пред-

принимать шаги по его созданию. С алгоритмически неразрешимыми проблемами часто сталкиваются при решении задач, относящихся к области искусственного интеллекта. Известно, что невозможно полностью автоматизировать такие аспекты человеческой деятельности, как перевод с одного языка на другой, выделение смыслового содержания некоторого текста, обучение других людей, сочинение стихов и т. п.

ГЛАВА 6

НЕЧЕТКИЕ МНОЖЕСТВА И ВЫВОДЫ

На практике очень часто приходится оперировать знаниями, которые работают с некоторым множеством объектов, для которых характерно то, что интуитивно ясно, какие объекты принадлежат к таким множествам, но невозможно определить четкие границы множества. Например, когда говорят о возрасте человека и используют понятие «старик», то не ясно, что имеется в виду — возраст более 50, 60, 70 или 80 лет. Одним из методов изучения множеств без уточнения их границ является теория нечетких множеств. Сформулированная как целостная теория в 1965 г., она активно развивается. Ее положения на сегодняшний день нашли применение во многих областях практической деятельности — начиная от управления сложными объектами и заканчивая разработкой процессоров, работающих по законам нечеткой логики. Рассмотрим некоторые базовые положения теории нечетких множеств с точки зрения представления знаний и построения нечеткого вывода на этих знаниях.

6.1. Обозначение нечетких множеств и функция принадлежности

Из сказанного уже ясно, что под нечетким множеством понимается множество, границы которого невозможно четко определить. Дадим более строгое определение этого понятия.

Определение. Пусть V — полное множество, охватывающее всю предметную область. Нечеткое множество F, которое фактически является подмножеством V, но о котором принято говорить как о множестве, определяется через функцию принадлежности $\mu_F(u)$, где u — элемент множества V. Эта функция отображает элементы и множества V на множество чисел в интервале $[0,1]$, которые указывают степень принадлежности каждого элемента нечеткому множеству F.

Если такое множество V состоит из конечного числа элементов $u_1, u_2, \ldots, u_n$, то нечеткое множество F можно представить в следующем виде:

$$F = \frac{\mu_F(u_1)}{u_1} + \frac{\mu_F(u_2)}{u_2} + \ldots + \frac{\mu_F(u_n)}{u_n} = \sum_{i-1}^{n} \frac{\mu_F(u_i)}{u_i}.$$

Примечание. Операции сложения и деления в приведенной формуле не являются обычными арифметическими операциями, а обозначают, соответственно, совокупность (объединение) элементов и отношение.

В случае непрерывного множества V используется интегральное представление совокупности

$$F = \int_V \frac{\mu_F(u_i)}{u_i}.$$

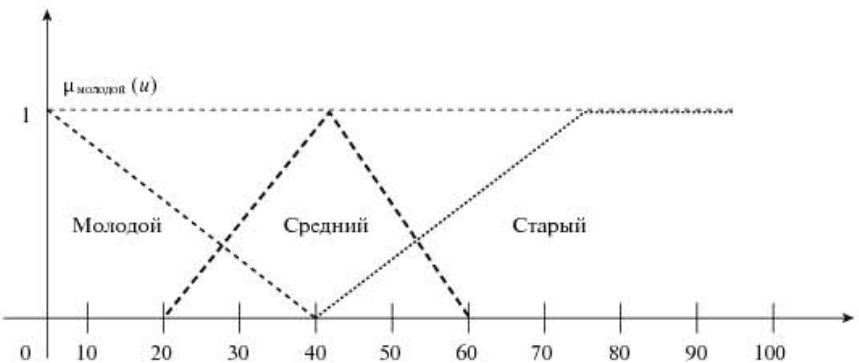

Рис. 6.1

Рассмотрим пример построения нечеткого множества. Пусть полное множество — это множество людей в возрасте 0 до 100 лет. Требуется определить на нем три нечетких множества, которые соответствуют возрастам: «молодой», «средний» и «старый». Функции принадлежности нечетких множеств, обозначающих возраст «молодой», «средний», «старый», можно представить, как показано на рис. 6.1.

Если определить множества возрастов как дискретные, отслеживая только позиции, соответствующие десятилетиям, то множества могут быть представлены в следующем виде:

$$молодой = \mu_{молодой}(u) = \frac{1}{0} + \frac{1}{10} + \frac{0{,}8}{20} + \frac{0{,}3}{30};$$

$$средний = \mu_{средний}(u) = \frac{0{,}5}{30} + \frac{1}{40} + \frac{0{,}5}{50};$$

$$старый = \mu_{старый}(u) = \frac{0{,}4}{50} + \frac{0{,}8}{60} + \frac{1}{70} + \frac{1}{80} + \frac{1}{90}.$$

Примечание. Видно, что в приведенной записи перечислены не все элементы из интервала от 0 до 10, это объясняется тем, что в целях упрощения записи допускается опускать те элементы, функция принадлежности которых равна нулю.

Так как множества хоть и нечеткие, но все же являются множествами, для них могут быть определены базовые операции над множествами, т. е. пересечение, объединение и дополнение.

1. Дополнение множества

$$\overline{F} = \sum_{i-1}^{n} \frac{1-\mu_F(u_i)}{u_i}, \text{ т. е. } \mu_{\overline{F}}(u_i) = 1-\mu_F(u_i).$$

2. Объединение множеств

$$F \cup G = \sum_{i-1}^{n} \frac{\mu_F(u) \vee \mu_G(u)}{u_i},$$

т. е. $\mu_{F\cup G}(u_i) = \mu_F(u_i) \vee \mu_G(u_i)$, где символ «$\vee$» обозначает операцию взятия максимума.

3. Пересечение множеств

$$F \cap G = \sum_{i-1}^{n} \frac{\mu_F(u_i) \wedge \mu_G(u_i)}{u_i},$$

т. е. $\mu_{F\cap G}(u_i) = \mu_F(u_i) \wedge \mu_G(u_i)$, где символ «$\wedge$» обозначает операцию взятия минимума.

Пример. Реализуем рассмотренные операции для множеств «молодой» и «средний»:

1) $\overline{молодой} = \mu_{\overline{молодой}}(u) = \frac{0,2}{20} + \frac{0,7}{30} + \frac{1}{40} + \frac{1}{50} + \ldots + \frac{1}{90}$;

2) $(молодой \cup средний) = \mu_{молодой \cup средний}(u) = \frac{1}{0} + \frac{1}{10} + \frac{0,8}{20} + \frac{0,5}{30} + \frac{1}{40} + \frac{0,5}{50}$;

3) $(молодой \cap средний) = \mu_{молодой \cap средний}(u) = \frac{0}{30}$.

6.2. Нечеткие отношения

Для выполнения нечетких выводов необходимо ввести понятие нечеткого отношения.

Определение. ***Нечетким отношением*** *R* между некоторой проблемной областью, т. е. полным множеством *U*, и другой областью, полным

множеством V, называется нечеткое подмножество прямого произведения $U \times V$, определяемое следующим образом:

$$R = \sum_{i-1}^{n} \sum_{j-1}^{m} \frac{\mu_R(u_i, v_j)}{(u_i, v_j)}, \text{ где } U = \{u_1, u_2 ..., u_n\}, \ V = \{v_1, v_2 ..., v_m\}.$$

Допустим, что существует правило типа «*если F, то G*», использующее нечеткие множества $F \subset U$ и $G \subset V$, тогда один из способов построения нечеткого отношения из соответствующей области множества U в области множества V состоит в следующем:

$$R = F \times G = \sum_{i-1}^{n} \sum_{j-1}^{m} \frac{\mu_F(u_i) \wedge \mu_G(v_i)}{(u_i, v_j)}.$$

Пример. Пусть $U = \{A, B, C, D\}$ — множество людей, а $V = \{1, 2, 3, 4\}$ — множество штанг различного веса. Тогда определим следующим образом нечеткие множества: F — множество сильных людей, G — множество штанг большого веса:

$$F = \frac{1}{A} + \frac{0{,}6}{B} + \frac{0{,}1}{C} + \frac{0}{D};$$

$$G = \frac{0}{1} + \frac{0{,}1}{2} + \frac{0{,}6}{3} + \frac{1}{4}.$$

Пусть имеется правило вида: если u_i — сильный человек, то он занимается штангой большого веса v_j. Тогда можно следующим образом построить нечеткое отношение:

$$R = u_i \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} v_j & & & \\ 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 0{,}1 & 0{,}6 & 1 \\ 0 & 0{,}1 & 0{,}6 & 0{,}6 \\ 0 & 0{,}1 & 0{,}1 & 0{,}1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \qquad (6.1)$$

Для построения полноценного вывода необходимо определить не только понятие отношения, но и правило перехода от одного отношения к другому, которое базируется на понятии свертки отношений.

Определение. ***Сверткой отношений*** называется правило перехода от одного отношения к другому, т. е. пусть R — нечеткое отношение

между областью U и областью V, а S — нечеткое отношение между V и W, тогда нечеткое отношение между U и W определяется как свертка отношений R и S:

$$R \circ S = \sum_{i=1}^{i} \sum_{k=1}^{n} V_{v_j \in v} \frac{\mu_R(u_i, v_j) \wedge \mu_S(v_j, w_k)}{(u_i, w_k)}. \qquad (6.2)$$

Символ «°» обозначает минимаксную свертку, определяемую для выводов с помощью цепочки правил.

Пример. Пусть задано множество чисел $W = \{m_1, m_2, m_3, m_4,\}$ — мышечной массы различного объема и на нем определено нечеткие множество H — большой мышечной массы. Множество $V = \{1, 2, 3, 4\}$, как и в предыдущем примере, это множество штанг различного веса, на котором определено нечеткое множество F — немаленьких весов:

$$F = \frac{0}{1} + \frac{0{,}4}{2} + \frac{0{,}9}{3} + \frac{1}{4};$$

$$H = \frac{0}{m_1} + \frac{0}{m_2} + \frac{0{,}5}{m_3} + \frac{1}{m_4}.$$

Пусть есть правило: если v_j — штанга немаленького веса, то она способствует наращиванию большой мышечной массы w_k. Тогда в соответствии с формулой (6.1) нечеткое отношение S из V в W определяется как

$$S = u_j \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} w_j \\ \begin{array}{cccc} m_1 & m_2 & m_3 & m_4 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0{,}4 & 0{,}4 \\ 0 & 0 & 0{,}5 & 0{,}9 \\ 0 & 0 & 0{,}5 & 1 \end{bmatrix} \end{array}.$$

Если по формуле (6.2) определить минимаксную свертку с нечетким отношением R из предыдущего примера, то из двух правил: «если u_i — сильный человек, то он занимается штангой большого веса v_j» и «если v_j — штанга немаленького веса, то она способствует наращиванию большой мышечной массы w_k» можно построить третье отношение — «если u_i — сильный человек, то он может нарастить большую мышечную массу w_k». Построенное отношение установлено между множествами V в W как результат свертки отношений R и S:

$$R \circ S = \underset{v \in V}{V} \left\{ \mu_R(u,v) \wedge \mu_S(v,w) \right\} =$$

$$= u_i \begin{matrix} \\ 0 \\ 0 \\ 0 \\ 0 \end{matrix} \overset{v_j}{\begin{bmatrix} 0 & 0{,}1 & 0{,}6 & 1 \\ 0 & 0{,}1 & 0{,}6 & 0{,}6 \\ 0 & 0{,}1 & 0{,}1 & 0{,}1 \\ 0 & 0 & 0 & 0 \end{bmatrix}} \circ v_j \overset{w_k}{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0{,}4 & 0{,}4 \\ 0 & 0 & 0{,}5 & 0{,}9 \\ 0 & 0 & 0{,}5 & 1 \end{bmatrix}} = u_i \overset{w_k}{\begin{bmatrix} 0 & 0 & 0{,}5 & 1 \\ 0 & 0 & 0{,}5 & 0{,}6 \\ 0 & 0 & 0{,}1 & 0{,}1 \\ 0 & 0 & 0 & 0 \end{bmatrix}},$$

из первого выбираются строки, которые последовательно умножаются на столбцы. Сочетание строк 1-го и столбцов 2-го дает 1-й элемент в результате.

6.3. Нечеткий вывод

Традиционный дедуктивный вывод (называемый правилом определения) — это вывод Q из P(факта) по правилу $P \to Q$, что записывается как

$$\begin{array}{l} P \to Q \\ \hline P \\ \hline Q \end{array}.$$

Это же обозначение используется в случаях нечетких дедуктивных выводов, если знания — это нечеткие множества (F, G, F', G'), а именно, вывод G' из F' по правилу $F \to G$ записывается так:

$$\begin{array}{l} F \to G \\ \hline P' \\ \hline Q' \end{array}.$$

Эта запись имеет существенную особенность: множества F и F' не обязательно совпадают. Если F и F' близки друг к другу, то их можно сопоставить и получить вывод G' в области их совпадения. Конкретно нечеткие выводы представляются следующим образом.

В первую очередь следует определить нечеткое отношение из правила $F \to G$. Один из способов — это формула (6.1); если есть цепочка из нескольких правил, то отношение — это свертка из формулы (6.2).

Вывод G' определяется из свертки множества и отношения R.

$$G' = \sum_{j=1}^{m} V_{u_i \in V} \frac{\mu_F(u_i) \wedge \mu_R(u_i, v_j)}{v_j}, \qquad (6.3)$$

где $F, F' \subset V, G, G' \subset V = \{v_1, v_2, \ldots, v_m\}$.

Пример. Пусть, как и в предыдущем случае,

$$U = \{A, B, C, D\};$$

$$V = \{1, 2, 3, 4\};$$

$$F = \frac{1}{A} + \frac{0{,}6}{B} + \frac{0{,}1}{C} + \frac{0}{D};$$

$$G = \frac{0}{1} + \frac{0{,}1}{2} + \frac{0{,}6}{3} + \frac{1}{4},$$

кроме того, пусть F' означает «силач»

$$F' = \frac{0{,}9}{A} + \frac{0{,}3}{B} + \frac{0}{C} + \frac{0}{D}$$

при условиях «если u_i — сильный человек, то он занимается штангой большого веса v_j» $(F \to G)$, «u_i — силач» (F').

Из этих правил определим отношение R из U в области V:

$$R = u_i \begin{bmatrix} 0 & 0{,}1 & 0{,}6 & 1 \\ 0 & 0{,}1 & 0{,}6 & 0{,}6 \\ 0 & 0{,}1 & 0{,}1 & 0{,}1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \begin{matrix} \\ v_j \\ \\ \\ \\ \end{matrix}$$

В соответствии с формулой (6.3) определим вывод

$$G' = F' \circ R = [0{,}9 \;\; 0{,}3 \;\; 0 \;\; 0] \circ \begin{bmatrix} 0 & 0{,}1 & 0{,}6 & 1 \\ 0 & 0{,}1 & 0{,}6 & 0{,}6 \\ 0 & 0{,}1 & 0{,}1 & 0{,}1 \\ 0 & 0 & 0 & 0 \end{bmatrix} = [0 \;\; 0{,}1 \;\; 0{,}6 \;\; 0{,}9].$$

По формуле для каждой пары элементов строка из 1-го — столбец из 2-го выбирается минимум, затем из них выбирается максимум, он и заносится в таблицу результата.

Здесь $F' = \frac{0,9}{A} + \frac{0,3}{B} + \frac{0}{C} + \frac{0}{D}$ представлено в виде матрицы $\begin{bmatrix} 0,9 & 0,3 & 0 & 0 \end{bmatrix}$.

В итоговой матрице каждый элемент j представляет значение принадлежности v_j множества G'.

Ответ: «v — это G», где $G' = \frac{0}{1} + \frac{0,1}{2} + \frac{0,6}{3} + \frac{0,9}{4}$.

Примечание. Невозможно строго упорядочить методы обработки нечетких знаний. Адекватность того или иного метода можно оценивать только по конкретным примерам или благодаря эвристическим знаниям.

ГЛАВА 7

ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ И ЯЗЫК ПРОЛОГ

Теоретические положения математической логики и теории алгоритмов широко используются на практике. Особое значение эти дисциплины имеют для развития языков программирования. Хорошо известно, что принципы алгоритмизации, изучаемые в теории алгоритмов, лежат в основе семантики языков процедурного программирования, к которым относятся такие популярные языки, как Си, C++, Паскаль и Бейсик. Менее известные, но широко применяемые в области создания приложений для систем искусственного интеллекта, — языки логического программирования. Один из наиболее популярных языков этой группы — язык Пролог — полностью основывается на теоретических положениях логики предикатов первого порядка. Рассмотрим подробнее особенности программирования на этом языке.

7.1. Основная идея логического программирования и история создания языка Пролог

Основная идея логического программирования состоит в следующем. Программа всегда составляется для решения некоторой задачи. При разработке программы требуется описать предметную область задачи как множество рассматриваемых объектов мира, их свойств и связей между ними. Логическая программа строится как набор утверждений, описывающих исходное множество объектов, а также функций и отношений, определенных на этом множестве. Это статическое описание, само по себе не задающее никакого вычислительного процесса. Удобно считать, что оно определяет базу знаний, содержащую информацию о предметной области решаемой задачи.

Для конкретного применения логической программы необходимо задать ей ***цель***, пытаясь удовлетворить которую, она сама построит

цепочку решения. Простейший пример формулирования цели: каково значение функции (заданной программой) при заданном значении аргумента? По своей семантике понятие цели близко к понятию «запрос» в базах данных. Вычисление ответа на запрос соответствует доказательству существования такого объекта (ответа); в процессе вычисления этот объект строится. Правила, по которым проводятся вычисления, образуют процедурную (или операционную) семантику логических программ. Существование этой семантики и определяет последовательность действий аналогично традиционному представлению программы.

У многих людей, знакомых с разработками в области искусственного интеллекта, при упоминании термина «логическое программирование», обычно возникают ассоциации с языком Пролог. Строго говоря, Пролог не является языком программирования в чистом виде. С одной стороны, это оболочка экспертной системы, с другой — интеллектуальная база знаний; что существенно, она не является реляционной базой. По своей сути Пролог, в отличие от традиционных процедурных языков: Си, Паскаль, Фортран и др., является декларативным языком. Программа на таком языке представляет собой набор логических описаний, определяющих цель, ради которой она написана. Как отмечалось, последовательность решения заложена в математической модели языка, т. е. во внутреннем механизме, который обеспечивает поиск результата на основе исходного логического описания.

Название Пролог произошло от словосочетания ***«программирование с помощью логики»*** (PROgramming in LOGic). Первая версия его была разработана и реализована в 1973 г. группой сотрудников Марсельского университета (Франция) под руководством Алэна Колмерое, занимавшихся задачей перевода с естественного языка. Первое время, в начале 1970-х, Пролог был не очень популярен, до появления первой удачной версии компилятора Дэвида Уоррена для компьютера DEC-10 в Эдинбурге, ставшей своего рода стандартом этого языка. После появления эдинбургской версии на Прологе были успешно реализованы различные проекты, которые затруднительно было выполнить средствами традиционных процедурных языков. Был решен ряд прикладных задач из области молекулярной биологии, проектирования сверхбольших интегральных схем и т. п. Особенно хорошо зарекомендовал себя Пролог в области создания экспертных систем.

Язык имеет строгое математическое обоснование и ориентирован на использование концепций и методов логики предикатов

первого порядка. На сегодняшний день язык Пролог является живым и активно развивающимся, хотя и не имеет такого широкого распространения, как языки Pascal, C++ и созданные на их основе интегрированные среды разработки Delphi, Visual C++, C++ Builder. Объясняется это тем, что перечисленные языки создавались как языки широкого назначения, поэтому в силу своей универсальности они применимы для решения задач практически в любой области. Пролог же создавался в первую очередь как специализированный язык, предназначенный для решения задач, относящихся к области искусственного интеллекта. Именно в этой области в полной мере проявляются все его достоинства и преимущества перед языками широкого назначения.

В настоящее время существует множество реализаций этого языка, наиболее популярные из которых: SWI-Prolog, Arity Prolog, Delphina Prolog, LPA-Prolog и т. д. Из всех существующих версий Пролога в России наиболее популярной оказалась версия, реализованная для IBM-совместимых компьютеров датской компанией Prolog Development Center (PDC) в содружестве с фирмой Borland International. Первая версия этого варианта реализации появилась в 1986 г. и носила название Турбо Пролог. При создании версии были допущены серьезные отступления от принятого стандарта языка. Это была плата за возможность ускорить трансляцию и выполнение программ. Новый компилятор был по достоинству оценен программистами, хотя и вызвал критику за существенные отступления от изначальной модели, его даже часто называли «не совсем чистым Прологом». Фирма Borland распространяла эту версию до 1990 г., а затем компания PDC приобрела все права на использование исходных текстов компилятора и дальнейшее продвижение системы программирования на рынок происходило под названием PDC Prolog. Система постоянно развивалась, и на ее базе в 1996 г. была создана первая версия интегрированной среды разработки под названием Visual Prolog. Интересно отметить, что в ее разработке принимали участие и российские программисты.

В настоящем пособии все примеры реализованы в среде Турбо Пролога или в терминальном режиме Visual Prolog. Этот выбор обоснован, тем, что на начальном этапе изучения языка необходимо ознакомиться с принципами и парадигмой логического программирования, а визуальные возможности в данном случае могут вызвать только дополнительные сложности.

7.2. Структура программы и основная терминология

В учебных пособиях по языкам программирования обычно используют один из двух подходов к представлению материала. При одном подходе изложение начинается с описания структур данных языка, его синтаксиса и т. д., при другом — сначала приводится пример программы, который дает обучаемому возможность составить начальное представление о языке, ознакомиться с некоторыми его особенностями, а затем изложение строится так же, как и в первом случае. Второй способ кажется более предпочтительным, так как сразу вводит читателя в мир нового языка, делает его чуть более знакомым, понятным, улучшает дальнейшее восприятие. Следуя второму подходу, рассмотрим структуру программы на языке Пролог и связанные с ней основные понятия на основе простых примеров.

Пример. Требуется описать группу людей и их пристрастия к различным видам спорта:

```
domains
    person, active = symbol
predicates
    like (person, activ)
clauses
    like (ellen, tennis).
    like (bob, football).
    like (jon, tennis).
    like (mark, basketball).
    like (bill, X) if like (jon, X).
```

Раздел `clauses` содержит множество фактов и правил. Факты `like (ellen, tennis)` и `like (bob, football)` соответствуют следующим утверждениям: «Элен любит теннис» и «Боб любит футбол», остальные организованы аналогично. Для формирования утверждения о Билле использвуется правило `like (bill, X) if like (jon, X)`, которое на естественном языке это звучит так: «Билл любит X, если Джон любит X».

Как уже говорилось ранее, программа на языке Пролог определяет базу знаний, в которой хранятся соответствующие объекты и отношения, заданные на них, т. е. сама по себе такая программа неприменима. Конкретному применению программы соответствует введение цели, для задания которой существует раздел `goal`. Ввод цели можно рассматривать как формулирование запроса к базе знаний, ответ на который

строится самой Пролог-программой. Таким образом, если определять в одной и той же программе разные цели, то можно получить различные варианты ответов. По умолчанию считается, что Пролог ищет все возможные решения для заданной цели.

Например, сформируем запрос `like (ellen, tennis)`, его цель — определить, соответствует ли истине тот факт, что «Элен любит теннис». Пролог выдаст `true`, так как у него есть факт, подтверждающий это утверждение. Аналогичный ответ будет и в случае запроса `like (bill, tennis)`, но его истинность будет подтверждена не фактом, а правилом, которое истинно, так как имеет место факт `like (jon, tennis)`, выполнение которого является условием выполнения правила.

Если ввести `like (bill, football)`, то будет выдано `no solution`, так как нет правила, подтверждающего этот запрос на основе имеющихся фактов, и нет факта, подтверждающего этот запрос.

В правиле `like (bill, X) if like (jon, X)` буква «X» использовалась как ***переменная***, обозначающая неизвестный вид деятельности.

Имена переменных должны начинаться с заглавной (прописной) буквы, вслед за которой могут идти в любом количестве строчные или заглавные буквы, цифры или знаки подчеркивания («_»). Поэтому имена `First_Name`, `True25m`^{3} допустимы, `name1`, `second-name`, `25zrt` — не допустимы. Осмысленный выбор имен переменных, как и в других языках программирования, делает программу на Прологе более наглядной. Максимальная длина имени — 250 знаков.

В Прологе различают ***свободные*** и ***связанные переменные***. Переменная считается ***свободной***, если ей не присвоено никакого значения; как только ей присваивается некоторое значение, переменная становится ***связанной***. Статус переменой существенно влияет на действия, которые можно с ней выполнить. Например, обычная операция «=» работает поразному в зависимости от того, свободная переменная или связанная.

Рассмотрим выражение `N=M`. Если `N` — свободная переменная, а `M` — связанная, то произойдет обычное присваивание. Если обе переменные связанные, то будет выполнена операция сравнения, которая выдаст значение `true` или `false`. Если обе переменные свободные или `M` — свободная, а `N` — связанная, то будет выдано сообщение об ошибке, так как Пролог будет считать такую операцию некорректной. Свободные переменные не могут участвовать в арифметических операциях в качестве аргументов. В Прологе невозможно выполнить операцию инкремента `X=X+1`, так как в этом случае необходимо либо переназначить значение связанной переменной, либо выполнить операцию сложения со свободной переменной, что недопустимо.

Переменные можно использовать не только в правилах, но и в запросах. С учетом этого на основе имеющейся программы можно получить не только подтверждение запроса, но и другие варианты ответа. Если ввести запрос `like (Ind, tennis)`, то для его удовлетворения требуется найти всех, кто любит теннис. Будет найдено три варианта ответа:

```
Ind = ellen
Ind = jon
Ind = bill
```

Домены и предикаты. Факты и правила, входящие в раздел `clauses`, представляют собой *отношения* (*предикаты*), которые связывают один или несколько *объектов*. Все отношения, используемые в программе, должны быть описаны в разделе `predicates`. Для объектов необходимо специфицировать *домены* (*типы данных*), которым принадлежат объекты, участвующие в отношениях. Для этого в Прологе, как и в других языках, зарезервирован набор стандартных типов данных. Описание объектов происходит в разделе `domains`. Например, раздел описаний приведенной выше программы показывает, что в отношении `like` участвуют два объекта, принадлежащие символьному (`symbol`) домену. Очевидно, что несоответствие между объявленными в описании типами доменов и реально указанными в разделе `clauses` фактическими параметрами приведет к ошибке.

Например, `like (12, X)` — неверно, так как `12` не принадлежит символьному домену; `like (ellen, jon, tennis)` приведет к ошибке, так как отношение `like` имеет не три, а два аргумента.

Для описания *отношений* и *объектов* можно выбирать любые имена, удовлетворяющие следующим требованиям:

— *имена объектов* должны начинаться со строчной буквы, за которой может следовать произвольное число символов (букв, цифр, знаков подчеркивания);

— *имена отношений* — произвольные комбинации букв, цифр, знаков подчеркивания, начинающихся со строчных букв.

Например, допустимые отношения:

```
car (model, color, age), nolike (dog, cat).
```

В одном отношении могут участвовать объекты, принадлежащие различным доменам. Например, описание автомобиля, для которого указана марка, цвет, год выпуска и стоимость может выглядеть так:

```
domains
    model, color = symbol
    age = integer
    price = real
```

```
predicates
    car (model, color, age, price)
```

Составные цели. Часто требуется вводить запросы, удовлетворяющие нескольким условиям. Такая цель называется составной целью. Например, для предыдущего описания требуется получить информацию об автомобиле, цена которого меньше 10 000:

```
car (Model, Colort, Age, Price) and Price < 10000.
```

Решая эту задачу, Пролог должен будет удовлетворять двум подцелям, указанным в предикате `car` и отношении «<» (меньше), которое является стандартным (встроенным).

Анонимные переменные. Бывают ситуации, когда для некоторых переменных не требуется конкретного значения, т. е. они могут принимать любые значения и на результат не влияют. Тогда в запросе можно вместо этой переменной указать знак подчеркивания «_», который обозначает анонимную переменную и говорит о том, что значение этого параметра будет игнорироваться. Так, если необходимо узнать только марку автомобиля, можно записать:

```
car (Model, _, _, _, Price) and Price < 10000
```

и получить в ответ информацию о марках всех машин с ценой меньше 10 000.

Анонимную переменную можно использовать везде, где допустимо использование любой другой переменной, она никогда не получает конкретного значения, т. е. знак подчеркивания обозначает переменную, которая не представляет интереса. Анонимные переменные можно использовать в фактах. Например, факт «все читают книги» можно представить так: `read (_, book).`

Решение составных целей и возврат. Как уже говорилось, в Прологе реализован механизм поиска всех возможных решений для заданной цели, поэтому в тех случаях, когда в обычных языках требовалось бы использовать несколько вложенных циклов, в Прологе достаточно правильно определить составную цель.

Пример. Рассмотрим следующую программу, которая содержит данные об учениках класса, для каждого из которых указывается имя и возраст:

```
domains
    child = symbol
    age = integer
predicates
    pupl (child, age)
clauses
    pupl (bob, 9).
```

```
pupl (tom, 10).
pupl (cris, 9).
pupl (mark, 9).
```

Требуется составить таблицу турнира для игры в пинг-понг (по две игры на каждую пару) при условии, что возраст участников — девять лет. Воспользуемся составной целью:

```
pupl (Pers1, 9) and pupl (Pers2, 9) and Pers1 <> Pers2.
```

В качестве результата Пролог выдаст группу ответов, по которым легко оценить ход поиска решения:

```
Pers1 = bob          Pers2 = cris
Pers1 = bob          Pers2 = mark
Pers1 = cris         Pers2 = bob
Pers1 = cris         Pers2 = mark
Pers1 = mark         Pers2 = bob
Pers1 = mark         Pers2 = cris
```

В заключение этого пункта рассмотрим еще один пример — «Пролог в роли свахи».

Пример. Требуется написать программу, которая по заданному набору параметров подобрала бы достойную кандидатуру для свидания. Определим два варианта реализации такой системы. Будем считать, что в обоих случаях подбирается кандидат-мужчина. В качестве параметров для оценки определим следующие: спортсмен, вегетарианец, курильщик.

Для описания каждого кандидата, информация о котором хранится в базе данных, будем использовать предикат `person` с четырьмя параметрами.

```
domains
    sports, vegit, smok = symbol
    name = symbol
predicates
    person (name, sports, vegit, smok)
    ok_person (sports, vegit, smok)
clauses
    person (rob, yes, no, no).
    person (bob, no, no, yes).
    person (pit, yes, yes, no).
    person (smit, yes, no, yes).
    person (bill, yes, no, no).
    ok_person (S, V, Sm) if not (person (_, S, V, Sm)),
         write ("Подходящей кандидатуры нет"), nl,
         exit.
    ok_person (S, V, Sm) if person (X, S, V, Sm),
```

```
        write ("Скорее всего это-", X), fail, nl.
goal
    ok_person (yes, no, yes).
```

Следует еще обратить внимание на то, что Пролог проверяет факты и правила всегда сверху вниз, т. е. по порядку их расположения в тексте программы, поэтому, изменяя порядок следования предикатов, можно изменить и получаемый результат.

Примечание. В программе были использованы предикаты `write`, `exit`, `not`, `nl`, `fail`, которые не описаны в разделе `predicates`. Эти предикаты входят в состав системы и называются ***стандартными.*** Они обычно используются для реализации типовых действий, таких как ввод-вывод, обработка строк и др., и их описывать не нужно. Особое внимание следует обратить на предикат `fail`, который заставляет Пролог искать все возможные решения для поставленной цели.

7.3. Стандартные типы доменов

Пролог может оперировать со следующими ***стандартными типами доменов***:

`char` (литерный) — литера, заключенная между двумя апострофами, например: 'a', '\13', '#', '?';

`integer` (целый) — целые числа в диапазоне от –32768 до 32767, например: 127, –31920;

`real` (вещественный) — числа, возможно со знаком, имеющие целую и дробную часть, разделенные точкой (точка не обязательна, если нет дробной части). Может использоваться экспоненциальная форма записи. Диапазон значений от –1E+307 до 1E+307, например: 42705, –99, 17e23, 64e–94, 65e+25;

`string`(строковый) — произвольная последовательность литер (длиной не более 250 символов), заключенная между двойными кавычками, например: `"Пролог – хороший язык"`, `"123/45"`;

`symbol` (символьный) — для символьных имен допускается два формата:

— последовательность букв, цифр и знаков подчеркивания, начинающаяся со строчной буквы;

— последовательность литер, заключенная между двойными кавычками (используются для имен, имеющих пробелы, или для имен, начинающихся с прописной буквы), например: `number_telephone`, `"инженер Сидоров"`. При использовании типы `symbol` и `string` могут быть взаимозаменяемы, но их внутренняя обработка различна;

`file` (файловый) — имя файла; подробнее этот тип и работа с данными этого типа будут рассмотрены позднее.

Если в декларациях предикатов используются только стандартные домены, то присутствие в программе раздела `domains` не обязательно.

Пример. Определим предикат, описывающий алфавит. По каждому символу задается его обозначение и порядковый номер в алфавите. Предложение для такого предиката могло бы выглядеть так:

```
alf_pos ('a',1).
```

Декларацию предиката можно записать так:

```
predicates
    alf_pos (char, integer)
```

Необходимость в разделе `domains` в этом случае отпадает.

7.4. Организация ввода и вывода

Для ввода информации в Прологе зарезервирован набор стандартных предикатов. Для каждого стандартного домена определен свой предикат. ***Набор стандартных предикатов*** представляет собой:

`readln (X)` — домен для переменной `X` должен иметь строковый или символьный тип. До вызова предиката `readln` переменная `X` должна быть свободной;

`readint (X)` — домен для переменной `X` должен иметь целый тип, и к моменту вызова эта переменная должна быть свободной;

`readreal (X)` — домен для переменной `X` должен иметь тип `real`, к моменту вызова переменная должна быть свободна;

`readchar (X)` — `X` должна принадлежать литерному домену. Предикат `readchar` считывает с текущего устройства одну литеру;

Для вывода информации используются:

`nl` — стандартный предикат, вызывающий перевод строки;

`write (arg1, arg2, arg3, ..., argn)` — предикат может быть вызван с произвольным числом аргументов, печать которых он осуществит. Указанные аргументы могут быть либо константами (из доменов стандартного типа), либо переменными, но переменные обязательно должны быть связанными;

`writef (format, arg1, arg2, arg3, ..., argn)` — в отличие от введенного ранее предикат `write` позволяет осуществлять форматированный вывод, шаблон для которого задает параметр `format`. Общий вид шаблона: `%m.pt`, где `m` — максимальная ширина поля; `p` — точность; `t` — признак формата, который может принимать одно из следующих значений:

`f` — десятичное число в десятичной форме записи;
`e` — десятичное число в экспоненциальной форме записи;
`g` — использование короткого формата (по умолчанию);
`d` — десятичное целое;
`c` — отдельный символ;
`s` — строка символов.

Например, если для осуществления форматного вывода трех значений написать на Прологе программу, состоящую только из одной цели следующего вида:

```
goal
    X = 10.125, Y = 10.125, Z = "10.125",
          writef("%10.5f %10.5e %s",X, Y, Z),
          readchar (_),
```

то в результате будет получена строка:

```
10.12500 1.0125E+01 10.125
```

Пример. Пусть программа на Прологе хранит в виде фактов следующую информацию о членах некоторого клуба: фамилия, адрес, год рождения. Требуется организовать простейший диалог, который позволит запросить данные по фамилии и году рождения, а в качестве результата выдаст сведения об адресе или же сформулирует отрицательный ответ, если человек, о котором введены сведения, не является членом клуба.

```
domains
first_name, adres = string
year = integer
predicates
run_info
analiz_info (first_name, year)
club_man (first_name, adres, year)
povtor (char)
clauses
club_man ("Сидоров М.М.", "пр. Свободы 100", 1949).
club_man ("Петров О.Н.", "пр. Платова 10", 1950).
club_man ("Любимов М.Н.", "ул. Правды 102", 1965).
club_man ("Пахомов И.И.", "пер. Северный 25", 1958).
club_man ("Брыксин Ф.Ф.", "ул. Свешникова 57", 1970).
    run_info
         if write ("Введите информацию об
         интересующем человеке"), nl, write
         ("Фамилия = "),
         readln (Name), write ("Год рождения = "),
         readint (Year), analiz_info (Name, Year),
```

```
        write ("Еще есть вопросы-? (д/н) = >"),
        readchar (Yes_No), povtor (Yes_No),nl.
    analiz_info (N,Y)
        if club_man (N,A,Y),write ("Это член
        клуба"),nl,
    write ("Его адрес->", A), nl.
    analiz_info (_, _) if write ("Это не член
    клуба"),nl.
    povtor ('д') if nl, run_info.
    povtor (_) if exit.
goal
    run_info.
```

7.5. Операции в Прологе

Операции в Прологе можно разделить на *три группы*: арифметические, логические и сравнения. Возможности операций схожи с возможностями аналогичных операций в других алгоритмических языках высокого уровня. Список операций по группам приведен в табл. 7.1.

Арифметические операции и математические функции применимы только к объектам из доменов `real` и `integer`. *Логические операции* применяются к предикатам. *Операции сравнения* могут применяться к объектам любого типа.

Таблица 7.1

Вид операции	Функциональный предикат	Описание
Арифметические операции и математические функции	`X mod Y`	Остаток от деления `X` на `Y`
	`X div Y`	Частное от деления `X` на `Y`
	`abc(X)`	Модуль `X`
	`cos(X)`	Косинус `X`
	`sin(X)`	Синус `X`
	`tan(X)`	Тангенс `X`
	`arctan(X)`	Арктангенс `X`
	`exp(X)`	Экспонента `X`
	`ln(X)`	Логарифм натуральный `X`
	`log(X)`	Логарифм десятичный `X`
	`sqrt(X)`	Корень квадратный `X`
	`+, –, *, /`	Арифметические действия

Продолжение

Вид операции	Функциональный предикат	Описание
Логические операции	not	Отрицание
	and	Логическое умножение
	or	Логическое сложение
Операции сравнения	<, < =, >, > =, <>	Меньше, меньше или равно, больше, больше или равно, не равно

Пример. Написать программу для вычисления значения функции:

$$f(x,y,z)=\begin{cases} x^2, \text{если } yx<z \\ y+z, \text{если } yx=z \\ yx-z, \text{если } yx>z \end{cases}$$

```
predicates
    function(real, real, real, real)
    start
clauses
    function(X, Y, Z, F) if X*Y < Z, F = X*X.
    function(X, Y, Z, F) if X*Y = Z, F = Y+Z.
    function(X, Y, Z, F) if X*Y > Z, F = X*Y-Z.
    start
         if write ("Введите значения"), nl,
         write ("X = "),
         readreal (X), write ("Y = "), readreal (Y),
         write ("Z = "),
         readreal (Z), function (X, Y, Z, F),
         write ("Результат f(x, y, z) = ", F),
         readchar (_).
goal
start.
```

Отметим, что предикат `readchar (_)` использовался для установки режима ожидания ввода, который до нажатия любой клавиши не меняет содержимого экранной области, что дает пользователю возможность просмотреть полученный результат.

7.6. Повторение и рекурсия

В программах часто требуется выполнить одну и ту же операцию несколько раз. Во многих языках программирования имеются операторы

цикла, реализующие повторения фрагментов программы, но в Прологе подобные операторы отсутствуют, поэтому рассмотрим другие возможности реализации повторяющихся действий. Для этого в Прологе используются четыре *основных метода*:

— метод отката после неудачи;
— метод отсечения и отката;
— правило повтора, определяемое пользователем;
— обобщенное рекурсивное правило.

Существует два способа реализации правил, выполняющих одну и ту же операцию многократно: повторение и рекурсия. Правила Пролога, выполняющие повторение, используют откат, а выполняющие рекурсию — самовызов.

Стандартный вид для *повторения*:

```
repeat_rule if <последовательность термов>, fail.
```

Встроенный предикат `fail` организует откат, т. е. в случае невозможности выполнения правила он позволяет вернуться назад и повторно попробовать его реализовать. Это действие осуществляется до тех пор, пока не будут исчерпаны все возможные варианты.

Вид для организации *рекурсии*:

```
repeat_rule if <последовательность термов>, repeat_rule.
```

Правила рекурсии содержат в теле правила сами себя. Правила рекурсии и повтора могут обеспечить одинаковый результат, хотя алгоритмы их реализации отличаются. Каждый из них в конкретной ситуации имеет свои преимущества. Рассмотрим реализацию указанных методов.

Откат после неудачи. Рассмотрим его на следующем примере. Имеется информация о сотрудниках предприятия следующего состава: фамилия, пол и зарплата. Требуется получить информацию обо всех мужчинах.

```
domains
    name = symbol
    sex = char
    plat = real
predicates
    employee (name, sex, plat)
    show_all_man
goal
    write ("Мужчины компании"), nl, show_all_man.
clauses
    employee ("Сидоров", 'М', 150).
    employee ("Петров", 'М', 200).
    employee ("Тихонова", 'Ж', 170).
```

```
    employee ("Иванов", 'M', 160).
    show_all_man if employee (Name, Sex, Plat), Sex = 'M',
         write (Name, Plat), nl, fail.
```

Метод отсечения и отката. В некоторых ситуациях необходим доступ только к ограниченному набору данных. Выше описано, как `fail` осуществляет откат, но для управления выбором необходимо средство управления откатом. Для этого служит предикат `cut`, вместо которого можно использовать символ «`!`». Это предикат, вычисление которого всегда завершается успешно, заставляет забывать все указатели отката, установленные во время попыток вычислить текущую цель.

Пример. Задан список детей, необходимо выдавать имена из списка, пока не встретится имя Дарья, в этом случае выдачу следует прервать.

```
domains
    person = string
predicates
    child (person)
    show_all
    make_person
clauses
    child ("Петр").
    child ("Василий").
    child ("Анастасия").
    child ("Константин").
    child ("Дарья").
    child ("Артем").
    child ("Федор Федорович").
    show_all if child (Name), write (Name), nl,
         make_person (Name), !, fail.
    make_person (Name) if Name = "Дарья".
goal
    show_all.
```

Правило повтора, определяемое пользователем. Также как и предыдущий метод, он использует откат, но в отличие от него, где откат выполняется только в случае искусственно созданного неуспешного результата, в этом методе откат возможен всегда. Вид правила повтора, определяемого пользователем:

```
repeat.
repeat if repeat.
```

Первый предикат `repeat` является фактом, который объявляет этот предикат истинным, он всегда выполняется успешно. Так как имеется еще одно правило для его определения, то указатель отката устанавливается на первый `repeat`. Второй `repeat` — это правило, которое

использует само себя как компоненту (третий `repeat`). Второй `repeat` вызывает третий, этот вызов также выполняется успешно, так как первый `repeat` всегда истинный, т. е. правило `repeat` тоже всегда успешно. Предикат `repeat` вычисляется успешно при каждой новой попытке вызвать его после отката. Таким образом, `repeat` — это рекурсивное правило, которое всегда выполняется успешно.

Такое правило удобно использовать в качестве компоненты других правил для организации бесконечных циклов.

Пример. Напишем программу «эхо», которая считывает строку и дублирует ее на экране. Работа программы завершается после ввода строки `stop`:

```
predicates
  repeat
  analiz (string)
  write_text
  do
clauses
  repeat.
  repeat if repeat.
  write_text if nl, write ("Введите текст, я его
  повторю"), nl,
  write ("Для завершения работы введите слово stop"),
  nl, nl.
  do if repeat, readln (Name), write (Name), nl,
  analiz (Name), !.
  analiz ("stop") if nl, write ("Сеанс окончен.
  До свидания").
  analiz (_) if fail.
goal
  write_text, do.
```

Правило `repeat` стоит первым в разделе утверждений программы, но это не обязательно. В правиле `do` наличие `repeat` вызывает повторение всех стоящих за ней компонент.

Простая рекурсия. Правило, содержащее само себя в качестве компоненты, называется правилом рекурсии, например:

```
write_string if write ("Пролог - это хорошо"), nl,
write_string.
```

Эта программа должна работать бесконечно, выдавая на экран сообщения, но так как при рекурсии растет число элементов данных, используемых рекурсивным процессом, то рано или поздно стек для их хранения переполнится и будет выдано сообщение об ошибке. Для остановки рекурсивного процесса важно иметь правило остановки.

Модернизируем предыдущий пример таким образом, чтобы вывод прекращался после заданного числа сообщений:

```
predicates
    write_string(integer, integer)
    analiz (integer, integer)
clauses
    write_string(N, M) if write ("Пролог отличный
    язык!"),
         nl, analiz (N, M), Ns = N+1, write_string
         (Ns, M).
    analiz (N, M) if N = M, write ("Сеанс окончен"),
    exit.
    analiz (N, M) if N < = M, !.
goal
    write ("Сколько раз следует повторить фразу?"),
         readint (M),
    write_string(1, M).
```

`exit` — это стандартный предикат, обеспечивающий выход в операционную систему или среду Турбо Пролога в зависимости от того, откуда была запущена программа на исполнение.

Метод обобщенного правила рекурсии. Обобщенное правило рекурсии содержит в теле правила само себя. Рекурсия будет конечной, если в правило включено условие выхода, гарантирующее окончание его работы. Тело правила состоит из утверждений и правил, определяющих задачи, которые должны быть выполнены. Общий вид:

<имя правила рекурсии> `if` <термы>,
<условие выхода>,
<термы>,
<имя правила рекурсии>,
<термы>.

Хотя структура и сложнее, чем в изложенном ранее примере, но принципы остаются неизменными.

Пример. Составить программу для нахождения факториала заданного числа:

```
predicates
    factorial (integer, integer)
clauses
    factorial (1, 1) if !.
    factorial (Number, Rez) if Next_number =
    Number-1,
    factorial (Next_number, Ok_Rez),
    Rez = Number *
```

```
    Ok_Rez.
goal
    factorial (5, Rez), write ("5! = ", Rez).
```

7.7. Списки, их представление и обработка

Определение. ***Список*** — это упорядоченный набор объектов. Элементы списка связаны между собой, поэтому с ними можно работать как с группой (т. е. с целым списком) и как с индивидуальными объектами (элементами списка).

Элементы списка записываются в квадратных скобках и разделяются запятыми. Все объекты должны принадлежать к одному домену. Например:

```
[1,2,3,4,7], [dog, cat, canary],
["Иванов","Петров","Сидоров"].
```

При описании доменов для списка указывается тип домена элементов списка и записывается символ «*», который говорит о том, что в списке присутствует ноль или более элементов. Например, описание списка из целых чисел выглядит так:

```
domains
    integerlist = integer*
```

Объекты в списке могут быть какими угодно, в том числе и другими списками, но все элементы списка должны принадлежать одному и тому же домену. Например, список `[[1,2,3], [7,10], [20]]` можно описать так:

```
domains
    object = object1*
    object1 = integer*
```

При обработке списков они разделяются на две части: ***голову*** и ***остаток (хвост)***. *Голова списка* — это его первый элемент, например для списка `[1,2,3]` — это элемент `1`. *Остаток списка* получается удалением из него головы, т. е. в данном случае остаток — это список `[2,3]`.

Для разделения головы и остатка списка используется вертикальная черта «`|`». Следовательно, список с головой `X` и остатком `Y` записывается так: `[X|Y]`. Возможные варианты разделения списка на голову и остаток приведены в табл. 7.2.

Списки могут использоваться в качестве параметров в предикатах и, соответственно, могут обозначаться переменными, которые в ходе поиска решения могут связываться со значениями, причем связь мо-

жет устанавливаться как на уровне всего списка, так и на уровне его отдельных элементов.

В табл. 7.3 приведены примеры установления связи между элементами списка и переменными.

Рассмотрим на примерах ряд наиболее распространенных действий, которые выполняются над списками.

Простой перебор элементов списка. Пусть требуется написать программу, которая распечатает в столбик элементы заданного списка целых чисел:

```
domains
    list = integer*
predicates
    write_a_list (list).
clauses
    write_a_list ( [ ]).
    write_a_list ( [X|Rest]) if write (X), nl,
         write_a_list (Rest).
goal
write_a_list ( [1,5,4,7,9,555]).
```

В первом предложении предписывается завершить выполнение, если элементов в списке больше нет (т. е. список пустой), во втором — вывести голову, перевести строку, а затем обработать остаток.

Таблица 7.2

Список	Голова	Остаток
['a','b','c']	'a'	['b','c']
["word"]	"word"	Пустой список
[]	Не определена	Не определен
[[1,2,3], [2,3,4], []]	[1,2,3]	[[2,3,4], []]

Таблица 7.3

Список 1	Список 2	Связанные переменные
[X,Y,Z]	["Иванов", "читает", "книгу"]	X = "Иванов", Y = "читает", Z = "книгу".
[X|Rest]	[1,2,3]	X=1,Rest=[2,3]
[tom,X,pit]	[Y,eliza,Z]	Y=tom, X=eliza, Z=pit
[7]	[X|Y]	X=7, Y=[]
[1,2]	[3|X]	Сопоставление невозможно, так как головы списков отличаются

Продолжение

[X,Y|R]	[1,2,3,4,5]	X=1, Y=2, R=[3,4,5]
[a,A,Y]	[X,b,X]	X=a, A=b, Y=a
[a|[b|[c|[]]]]	X	X=[a,b,c]

Поиск элемента в списке. Требуется проверить, присутствует ли некоторый элемент в заданном списке:

```
domains
    namelist = name*
    name = symbol
predicates
    element_spisca (name, namelist)
clauses
    element_spisca (Name, [ ])
        if write (Name, "в список не входит"), exit.
    element_spisca (Name, [Name| _ ])
        if write (Name, "есть в списке"), exit.
    element_spisca (Name, [ _ |Ostatok])
        if element_spisca (Name, Ostatok).
```

В этом примере второе предложение проверяет голову списка, остаток интереса не представляет, может быть удовлетворена следующая цель:

```
element_spisca (jon,[jon,ellen,victor]).
```

Если интересующее имя не в начале списка, то третье правило позволяет найти его в списке, для чего проверяется хвост списка.

Разделение заданного списка на части по определенному признаку. Задан список `L` целых чисел и некоторое число `N`. Список нужно разделить на два по следующему правилу: число меньшее либо равное `N` попадает в список `L1`, а большее — в `L2`.

```
domains
    n = integer
    list = integer*
predicates
    sp_razd (n, list, list, list)
clauses
    sp_razd (_, [ ], [ ], [ ]).
    sp_razd (N, [H|Ost], [H|L1], L2)
        if H < = N, sp_razd (N, Ost, L1, L2).
    sp_razd (N, [H|Ost], L1, [H|L2])
        if sp_razd (N, Ost, L1, L2).
goal
    sp_razd (40, [20, 30, 70, 10, 50], L1, L2).
```

Результат разделения списка: `L1 = [20,30,10], L2 = [70,50].`

Объединение списков. Задано два списка `L1` и `L2`, необходимо объединить их в один список `L3` путем присоединения `L2` к `L1`. Последовательность действий при выполнении данной операции можно описать так:

— список `L3` сначала пуст;
— переписываем содержимое списка `L1` в `L3`, пока он не опустеет;
— переписываем `L2` в `L3`.

```
domains
    list = integer*
predicates
    append (list, list, list)
clauses
    append ([ ], L, L).
    append ([N|L1], L2, [N|L3]) if append (L1, L2, L3).
goal
    append ([1, 2, 3], [4, 5], L3).
```

Примечание. Управляя целью, можно получить различные варианты исходных списков, из которых можно получить указанный результат. Для этого достаточно при задании цели ввести результирующий список как связанный объект, а исходные объявить свободными переменными. Например, для цели вида

```
goal
append (L1,L2, [1,2,3])
```

Прологом будет предложено четыре варианта решения:

```
L1 = [ ]          L2 = [1,2,3]
L1 = [1]          L2 = [2,3]
L2 = [1,2]        L2 = [3]
L2 = [1,2,3]      L2 = [ ]
```

Сортировка списка. Заданный список целых чисел требуется отсортировать в порядке возрастания его элементов. Сортировка будет осуществляться по следующему правилу: введем кроме исходного списка промежуточный, в котором будет записан результат сортировки, изначально этот список пуст. Будем извлекать из исходного списка по одному элементу и вставлять их в результирующий таким образом, чтобы список результата оставался упорядоченным, т. е. каждый элемент ставится на свое место. Используемые предикаты будут иметь следующий смысл:

`sort (list, list)` — перебирает элементы исходного списка;

`ins_num (integer, list, list)` — вставляет заданный элемент (первый параметр) в заданный список в нужном месте (второй параметр) и возвращает полученый список в качестве результата (третий параметр);

`ask (integer, list)` — проверяет выполнение условия вставки элемента (первый параметр) в список (второй параметр).

```
domains
    list = integer*
predicates
    sort (list, list)
    ins_num (integer, list, list)
    ask (integer, list)
clauses
    sort ([],L2) if write (L2), exit.
    sort ([N|L1], L2) if ins_num (N, L2, L3),
    sort (L1, L3).
    ins_num (N, [ ], [N]) if !.
    ins_num (N, L1, [N|L1]) if ask (N, L1), !.
    ins_num (N, [X|L1], [X|L2]) if ins_num (N, L1, L2).
    ask (N, [X|L]) if N < = X.
goal
    sort ([10, 5, 16, 7, 4, 1, 3], [ ]).
```

7.8. Работа с файлами

Операции ввода-вывода работают с текущим устройством (чтения или записи). По умолчанию для чтения — это клавиатура, для записи — экран. Текущее устройство может быть переназначено, и в качестве него может выступить реальное физическое устройство (принтер, экран) или файл, который к данному моменту должен быть открыт для соответствующего действия. В Прологе файл может быть открыт для одного из следующих четырех действий: чтения, записи, присоединения к этому файлу новых записей или модификации (чтения или записи).

Файл, открытый для одного из вышеуказанных действий, кроме чтения, по окончании работы должен быть закрыт, иначе все изменения, сделанные в нем, будут потеряны. Можно одновременно открыть несколько файлов, причем направления, по которым перемещаются входные и выходные данные, можно менять, выбирая между открытыми файлами и переназначая текущее устройство.

Имена файлов должны начинаться со строчной буквы и присутствовать в декларации файлового домена, причем в любой программе можно пользоваться только одним файловым доменом. Имена файлов разделяются символом «*;*», например:

```
file = file1; space; fox
```

Следующие четыре символьных имени являются предопределенными и не должны появляться в декларации файлов:

```
printer
screen
keyboard
com1
```

Примечание. Имя `printer` относится к параллельному порту принтера, `com1` — к последовательному.

Стандартные предикаты для открытия и закрытия файлов:

— `openread (SymFN, DosFN)` — открыть файл для чтения; `SymFN` — символьное имя файла в программе, `DosFN` — имя файла в операционной системе (во всех остальных предикатах эти параметры имеют тот же смысл). Если в операционной системе указанное имя недопустимо, выдается сообщение об ошибке;

— `openwrite (SymFN, DosFN)` — открытие для записи. Если файл уже существует, то он уничтожается, иначе в текущем каталоге создается соответствующий элемент;

— `openappend (SymFN, DosFN)` — открытие файла для операции присоединения; если файл не существует, то выдается сообщение об ошибке и выполнение останавливается.

— `openmodify (SymFN, DosFN)` — открытие файла для модификации;

— `closefile (SymFN)` — закрытие файла; результат выполнения всегда истина, даже если файл не был открыт;

— `readdevice (SymFN)` — переназначение текущего устройства чтения при условии, что переменная `SymFN` связана, а файл открыт для чтения. Если переменная свободна, то вызов предиката свяжет ее с именем текущего активного устройства чтения;

— `writedevice (symFN)` — переназначение текущего устройство записи при условии, что файл открыт или для записи, или для присоединения;

— `eof (SymFN)` — проверка совпадения позиции файла во время операции чтения с концом файла. Если да, то предикат вырабатывает значение «истина», в противном случае — «ложь».

Пример. Написать программу, которая будет читать информацию из файла и выводить ее на печать и экран:

```
domains
    file = datafile
predicates
    process
```

```
    read_print
clauses
    process if write ("Введите имя файла"),readln
    (Filename),
        openread (datafile, Filename), readdevice
        (datafile),
        read_print, closefile (datafile).
    read_print if not (eof(datafile)), readchar (Y),
    write (Y),
        writedevice (printer), write (Y),
        writedevice (screen), read_print.
    read_print if readdevice (keyboard), write ("Файл
    пуст"),
        readchar (_).
```

7.9. Работа со строками

Для работы с объектами типа `string` в Прологе предусмотрен набор стандартных предикатов, позволяющих реализовать основной набор действий, выполняемых над строками:

1) `frontchar (Str, Ch, Rest)` — выделяет из строки `Str` первый символ и связывает его с переменной `Ch`, остаток строки связывается с переменной `Rest`.

Пример. `frontchar ("Тест", X, Y)` → `X = 'T', Y = "ест";`

2) `fronttoken (Str1, SymP, Rest)` — используется для расщепления строки на список лексем. Последовательность литер воспринимается как одна лексема, если она представляет собой одну из следующих конструкций:

— имя, удовлетворяющее обычному синтаксису;
— число (знак воспринимается как отдельная лексема);
— литера, отличная от пробела.

Если при вызове этого предиката переменная `Str1` связана, то предикат находит первую лексему в строке `Str1` и связывает ее с переменной `SymP`, остальная часть строки связывается с параметром `Rest`. Начальные пробелы в строке игнорируются.

Пример. Рассмотрим простой пример разбиения произвольной введенной строки на лексемы с использованием предиката `fronttoken`.

```
predicates
    start
    work (string)
```

```
clauses
    start if write ("Введите строку для разбора->"),
          readln (St), work (St).
    work ("") if nl, write ("Разбор окончен"),
    readchar (_).
    work (St) if fronttoken (St, S, R), write (S),
    nl, work (R).
goal
    start.
```

Если ввести строку вида "Language-Turbo Prolog", то она будет разбита на следующие лексемы: "Language", "-", "Turbo", "Prolog". А строка "123+abcd-21 b25" — на следующие лексемы: "123", "+", "abcd", "-", "21", "b25";

3) `frontstr (NumChar, Str1, Startst, Str2)` — строка `Str1` разбивается на две части: строка `Startst` будет содержать первые литеры, число которых равно значению параметра `NumChar`, а `Str2` — остальные. Перед вызовом этого предиката два первых параметра должны быть связаны, а два последних свободны.

Пример. `frontstr (5, "бегемот", X, Y)` → `X = "бегем", Y = "от"`;

4) `concat (Str1, Str2, Str3)` — выполняет конкатенацию (слияние) строк `Str1` и `Str2`, результирующая строка — `Str3`. К моменту вызова предиката оба исходных параметра должны быть связаны.

Пример. `concat ("croco", "dile", Animal)` → `Animal = "crocodile"`;

5) `str_len (Str, N)` — определяет длину строки `Str` и связывает значение с переменной `N`.

Пример. `str_len ("Привет", N)` → `N = 6`;

6) `isname (Str)` — проверяет, является ли `Str` именем объекта, удовлетворяющим требованиям Пролога.

Пример. `isname (tom)` → `true; isname (125e)` → `false`.

Примечание. Следует отметить, что все перечисленные выше предикаты обладают традиционным для Пролога эффектом двойного действия, т. е. в зависимости от того, какие из переменных в предикате объявлены как свободные, а какие — как связанные, будет формироваться различный набор результатов работы предиката.

Примеры:

а) `frontchar ("Word", "W", "ord")` — проверяет, действительно ли слово разделено правильно;

б) `frontchar (Str, 'W', "ord")` — определяет исходное слово, т. е. `Str = "Word"`;

в) `frontstr (5,"frontstr", "front", R)` — проверяет, действительно ли первые пять символов в слове `"frontstr"` совпадают с `"front"`, если, например, в цикле обращаться к этому предикату и менять исходную строку, то можно выбрать все строки, начинающиеся с заданной комбинации символов;

4) `str_len ("abc", 3)` — проверяет, действительно ли заданная строка имеет указанное количество символов.

В заключение приведем пример обработки строк.

Пример. Задана строка символов, требуется выделить из нее десять первых гласных символов и занести их в список:

```
domains
    list = char*
predicates
    str_list (integer, list, string)
    glas (char)
    start
clauses
    glas ('a'). glas ('e'). glas ('i').
    glas ('o'). glas ('y'). glas ('u').
    str_list (N, [ ], _) if N = 10, nl.
    str_list (N, [ ], "") if N<10,
    write ("Гласных в строке меньше десяти"), nl.
    str_list (N, [X|List], Str) if frontchar (Str, X,
    Rest),
    glas (X), Ns = N+1, str_list (Ns, List, Rest).
    str_list (N, List, Str)
    if frontchar (Str, _, Rest), str_list (N, List,
    Rest).
    start if write ("Введите строку"), readln (Str),
    str_list (0,List, Str), nl,
    write ("Список гласных- ",List), readchar (_).
goal
    start.
```

7.10. Создание динамических баз данных

Программы работы с базами данных (БД), написанные на Турбо Прологе, можно рассматривать как частный случай систем управления базами данных (СУБД). БД представляет собой набор информации, представленной в доступной для компьютера форме. СУБД — комплекс программных средств, позволяющих работать с информацией,

хранящейся в БД. Обычно СУБД предоставляют набор возможностей манипулирования данными (добавление, редактирование, просмотр, поиск факта по определенному признаку и т. д.).

Существует три известные *модели организации БД*: иерархическая, сетевая и реляционная. В иерархической модели данные представлены набором классов, организованных в виде иерархической системы, образующей иерархию классов. В сетевой — данные содержатся в виде связанных объектов, образующих сеть, в реляционной — данные хранятся в виде таблиц. На сегодняшний день наиболее распространенной является реляционная модель. В Турбо Прологе она реализуется без каких-либо трудностей.

Файл БД представляет собой набор связанных между собой записей. *Запись* — это совокупность полей, каждое из которых содержит некоторую часть данных. Файл БД имеет вид простого файла, однако хранящиеся в нем данные обладают своей внутренней организацией. Данные внутри каждой записи имеют одну и ту же структуру. Часто существует одна специальная запись, в которой описан способ представления данных в записях. Она играет роль «карты», руководствуясь которой программа СУБД работает с данными.

Пример. Требуется выдать сведения о студентах и результатах сдачи ими сессии. Пусть вся информация сведена в некоторую таблицу (табл. 7.4), которая называется `Students`.

Таблица 7.4

Фамилия	Группа	Предмет	Оценка
Сидоров В. В.	II-5	Математика	5
...	...	...	...
Петров М. А.	IV-6	Специальные главы высшей математики	3

Столбцы таблицы принято называть доменами, строки — записями, а ячейки на пересечении строк и столбцов — полями. Таким образом, поле — это часть записи. Обычно все записи файла БД имеют одинаковую длину, поэтому расположение их в файле определяется без труда, однако в Турбо Прологе требование одинаковости длины, в общем, не соблюдается, так как он располагает записи в соответствии с шаблоном, определяемом при описании предиката.

Реляционные БД. Данные в реляционной БД хранятся в таблицах, состоящих из столбцов и строк. Так, табл. 7.4 состоит из четырех столбцов и `N` строк. Имя `Students` — это название всей структуры данных, и оно задает отношение. Имя столбца задает имя атрибута, а набор

атрибутов называется реляционной схемой. Количество атрибутов называется арностью отношения, а число элементов отношения — его мощность.

Базы данных в Турбо Прологе. В Турбо Прологе имеется набор специальных средств для организации БД и работы с ними. Они ориентированы на работу с реляционными БД, так как внутренняя структура представления данных в виде фактов легко сопоставима с табличной формой. Кроме того, механизмы унификации и отката позволяют легко осуществить поиск среди фактов, описывающих БД.

Если трактовать предикат `students (name, gruppa, subject, status)` как предикат из БД Турбо Пролога, то он должен быть соответствующим образом описан. Для этого существует специальный раздел `database`, который должен располагаться перед разделом `predicates`.

Пример.

```
domains
    name, gruppa, subject = string
    status = integer
    p1, p2, p3 = symbol
database
    students (name, gruppa, subject, status)
predicates
    pr1 (p1, p2)
```

Предикаты, описанные в разделе `database`, хранятся в оперативной памяти отдельно от остальных предикатов, поэтому их обработка Прологом ведется быстрее. Для долговременного хранения предикаты, хранящиеся в БД, могут быть перемещены в файл, из которого в случае необходимости они могут быть в полном составе перемещены в оперативную память.

Примечание. Предикаты, образующие БД, представляют собой только факты, записанные в соответствии с синтаксисом Турбо Пролога, правила в БД не могут быть записаны (хотя в других версиях Пролога это возможно).

Действия над предикатами, описанными в `database`, осуществляются так же, как и над другими предикатами, и не требуют специальных форм обращения. Факты из раздела `database` образуют динамическую БД. БД называется динамической, так как во время работы из нее можно удалять любые хранящиеся в ней утверждения или добавлять новые. В этом состоит ее отличие от статических БД, где утверждения являются частью программы. Для динамической БД также характерна возможность обмена с дисковой памятью. Иногда удобно часть определений иметь в статической БД и перемещать их в динамическую в процессе работы программы.

Соответствие терминологии Турбо Пролога в области БД традиционной терминологии в этой области представлено в табл. 7.5.

Таблица 7.5

Турбо Пролог	Реляционная БД
Объект	Атрибут
Отдельное утверждение	Элемент отношения
Предикат БД	Отношение

Стандартные предикаты для работы с динамическими БД. Так как обработка предикатов из БД ничем не отличается от обработки предикатов из раздела `clauses`, то для них можно использовать все рассмотренные раннее действия и правила. Отличием является только их положение в памяти, которое Пролог определяет для хранения БД, поэтому существует набор стандартных предикатов, которые позволяют заносить или удалять информацию из БД:

1) `asserta (Predicat)` — заносит новый факт в БД и помещает его перед другими фактами.

Пример. `asserta (students ("Сидоров","II-5","ЛП",5)).`

2) `assertz (Predicat)` — заносит новый факт в БД и помещает его после всех имеющихся там фактов.

Примечание. Следует помнить, что объект `Predicat`, являющийся параметром предикатов `asserta` и `assertz`, может быть определен в разделе `clauses` либо в явном виде, либо его значение устанавливается в результате работы программы. Кроме того, предикат для добавления в БД может быть построен на основе данных, извлеченных из предикатов раздела `clauses`.

Пример. Из статической БД, содержащей сведения о студентах, поместить в БД факты, содержащие информацию только о студентах, имеющих хотя бы одну двойку:

```
database
    super_students (string, string, string)
predicates
    students (string, string, string, integer)
clauses
    students ("Петров","2-6","ЛП",3).
    …
    append_to_DB if students (Name, Grupp, Subject,
         Status), Status = 2, assertz (super_students
         (Name, Grupp, Subject)), fail.
    append_to_DB if !.
goal
```

```
    append_to_DB.
```

3) `retract (Predicat)` — удаляет утверждение из динамической БД. Этот предикат можно использовать различным образом. Рассмотрим следующие случаи:

а) удалить конкретный факт с заданными параметрами:

```
retract (students ("Сидоров", "2-6")).
```

б) удалить все утверждения, связанные с отдельным предикатом:

```
delete_DB1 if retract (book (_, _, _)), fail.
```

в) полностью очистить содержимое БД:

```
delete_DB2 if retract (_), fail.
```

Очевидно, что для работы с БД необходима комбинация всех трех предикатов. Например, чтобы отредактировать запись, содержащуюся в БД, необходимо получить новые сведения, составить на их основе утверждения, удалить старые и занести новые.

Пример. Написать программу, которая позволит для некоторого студента отредактировать оценки по всем предметам:

```
database
    students (string, string, string, integer)
    dstud (string, string, string, integer)
predicates
    edit_status (string)
clauses
    edit_status (Name) if students (Name, Gr, Sub,
    Status),
        gotowindow (2), clearwindow, write
        (Name, Gr), nl,
        write (Sub), str_int (S1, Status),
        gotowindow (1),
        edit (S1, S2), str_int (S2, S3),
        assertz (dstud (Name, Gr, Sub, S3)),
        retract (students (Name, Gr, Sub, Status)),
        fail.
    edit_status (_) if !.
    append_DB if dstud (N, G, S, St),
        assertz (Students (N, G, S, St)),
    retract (dstud (N, G, S, St)), fail.
    append_DB if !.
goal
    write ("Введите имя студента"), readln (Name),
    makewindow (1, 77, "Оценка", 5, 10, 3, 20),
    makewindow (2, 7, 7, "Name", 0, 0, 5, 25),
        edit_status (Name), append_DB.
```

Кроме предикатов, позволяющих работать с отдельными утверждениями из БД, существует набор предикатов, работающих со всей базой в целом:

1) `save (FileName)` — записывает все факты, содержащиеся в БД, в текстовый файл с именем `FileName`.

Примечание. Перемещаются факты только из динамической БД, а не те, что расположены в разделе `clauses`.

2) `consult (FailName)` — загружает в оперативную память все факты, хранящиеся в файле с именем `FileName`. Предикат `consult` неуместен, если файл с указанным именем отсутствует на диске, если файл содержит ошибку (либо синтаксическую, либо несоответствие описания предикатов из раздела `database` с теми, что хранятся в файле) или если содержимое файла невозможно разместить в памяти из-за отсутствия места.

Пример. Из файла выбрать все записи об автомобиле `Pontiac` и переписать в другой файл, используя БД для промежуточного хранения информации:

```
database
    auto (symbol, integer)
predicates
    p1
clauses
    p1 if consult ("auto. dba"), del_auto, save
    ("auto1.dba").
    del_auto if retract (auto (Mark, _)),
    Mark<>"Pontiac", fail.
    del_auto if !.
goal
    p1.
```

Примечание. Расширение `dba` для обозначения файла БД является общепринятым для Турбо Пролога, но не обязательным, ему можно присвоить любой тип, как обычному текстовому файлу. Каждый факт в файле должен заканчиваться точкой, иначе — ошибка.

7.11. Стандартные предикаты `random` и `findall`

Предикат `random (RealNumber)` используется в качестве датчика случайных чисел и возвращает вещественное число `RealNumber`, удовлетворяющее двойному неравенству `0< = RealNumber <1`.

Пример. Написать программу для выбора случайным образом трех имен из пяти заданных:

```
predicates
    person (integer, symbol)
    rand_int_1_5 (integer)
    rand_person (integer)
goal
    rand_person (3).
clauses
    person (1, fred).
    person (2, bob).
    person (3, jon).
    person (4, stiv).
    person (5, tom).
    rand_int_1_5 (X) if random (Y), X = Y*5+1.
    rand_person (0) if !.
    rand_person (Z) if rand_int_1_5 (N),
        person (N, Name), write (Name), nl, Nz =
        Z-1, rand_person (Nz).
```

Предикат `findall (Var, Predicat, List)` позволяет собрать в список `List` множество значений, которые могут быть порождены для переменной `Var` в результате поиска всех возможных вариантов решения для предиката, связанного с параметром `Predicat`.

Пример. Информация о сотрудниках некоторой организации задана в виде фактов. Определить средний возраст сотрудника:

```
domains
    adress, name = string
    n, age = integer
    list = age*
predicates
    person (name, adress, age)
    summa (list, n, n)
clauses
    person ("Сидоров","ул. Пушкинская 5",50).
    person ("Иванов","пр. Платова 10",25).
    person ("Семенов","ул. Лермонтова 5",50).
    person ("Васильев","пр. Петрова 10",37).
    person ("Сидоров","пер. Печатников 7",1).
    summa ([ ],N, S_age) if S = S_age/N,
        write ("Средний возраст = ",S).
    summa ([M|Ost],N, S_age) if N1 = N+1,
        S1 = S_age+M, summa (Ost, N1, S1).
goal
    findall (Age, person (_, _, Age), L),summa (L, 0, 0).
```

7.12. Составные объекты и их использование

В рассмотренных выше предикатах использованы только данные, относящиеся к шести стандартным типам доменов. Например:

```
owner ("Сидоров", "Ф. Херберт", "Дюна").
```

Каждый из объетов этого предиката представляет сам себя; такие объекты называются ***простыми объектами***. Аналогично структура, состоящая из простых объектов, называется ***простой структурой***.

Утверждение `owner` отражает факт, что Сидоров обладает некоторой книгой. Можно изменить эту структуру, превратив ее в новую форму, более детально описывающую объект принадлежности:

```
owner ("Сидоров", book ("Ф. Херберт", "Дюна")).
```

Объект, представляющий другой объект или совокупность объектов, называется ***составным объектом***. Предикат `owner`, записанный таким образом, называется ***составной структурой***, так как содержит составной объект. Рассмотрим утверждение

```
like ("Иванов", apples, banana, orange),
```

означающее, что «Иванов любит яблоки, бананы, апельсины». Можно объединить все фрукты в отдельной структуре

```
like ("Иванов", fruits (apples, banana, orange)).
```

Терм `fruits` в этом утверждении называется функтором. ***Функтор*** — это терм составного объекта. Функтор составного объекта — это предикат, который вставляется внутри другого предиката. Главный функтор здесь — предикат `like`.

Утверждения и предикаты, использующие составные объекты, описываются в разделе `domains` следующим образом:

```
domains
    pers_like = fruits (t1, t2, t3)
    t1, t2, t3 = symbol
```

В разделе `predicates` составная структура описывается так:

```
predicates
    likes (string, pers_like)
```

Здесь `pers_like` — имя составного объекта, образованного с помощью функтора `fruits`, который одновременно представляет составной объект и функтор.

Если объекты функтора относятся к одному и тому же типу доменов, то объект называется ***однодоменной структурой***, если к разным — ***многодоменной.*** Ссылка на доменную структуру осуществляется по имени функтора.

Пример. Составим описание личной библиотеки. В качестве функтора будет выступать предикат `book` с параметрами: автор, название, издательство, год издания:

```
domains
    pers_library = book (title, autor, publisher,
    year)
    person, title, autor, publisher = string
    year = integer
predicates
    collection (person, pers_library)
clauses
    collection ("Иванов", book ("Программирование на
    C++", "Том Сван", "Binom", 1995)).
    collection ("Иванов", book ("Язык Турбо Пролог",
         "А.Янсон", "Мир", 1994)).
    collection ("Сидоров", book ("J.R.R.Tolkein",
         "Return of the King", "АСТ", 1994)).
```

Рассмотрим несколько вариантов построения запросов.

1. Выдать все книги из коллекции Иванова:

```
goal
    collection ("Иванов", Books)
```

Результат:

```
Books = book ("Программирование на C++", "Том Сван",
"Binom", 1995)
Books = book ("Язык Турбо Пролог", "А.Янсон", "Мир", 1994)
```

2. Для всех книг независимо от коллекционера, изданных в один и тот же заданный год, требуется выдать имя владельца и название:

```
goal
    collection (Person, book (Title, _, _, 1994)).
```

Результат:

```
Person = "Иванов", Title = "Язык Турбо Пролог"
Person = "Сидоров", Title = "Return of the King"
```

Так как составные объекты значительно усложняют структуры описания предикатов, то для их более наглядного представления используют ***структурные диаграммы***. Изображенная на рис. 7.1 диаграмма соответствует описанию приведенного выше примера и является двухуровневым деревом.

Можно внести в программу некоторые изменения и превратить описание в трехуровневую структуру. Для этого объединим `publisher` и `year` в предикат `publication`, а `autor` и `title` — в предикат `volume`. Получим структурную диаграмму, приведенную на рис. 7.2.

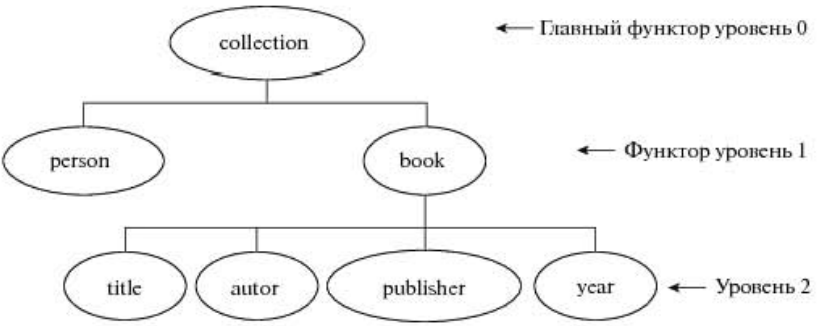

Рис. 7.1

Описание этой структуры будет выглядеть так:

```
domains
    pers_library = book (volume, publication)
    volume = volume (autor, title)
    publication = publication (publisher, year)
    person, autor, title, publisher = string
    year = integer
predicates
    collection (person, pers_library)
clauses
    collection ("Сидоров", book (volume ("J.R.R.
    Tolkein", "Return of the King"),
         publication ("ACT", 1994))).
```

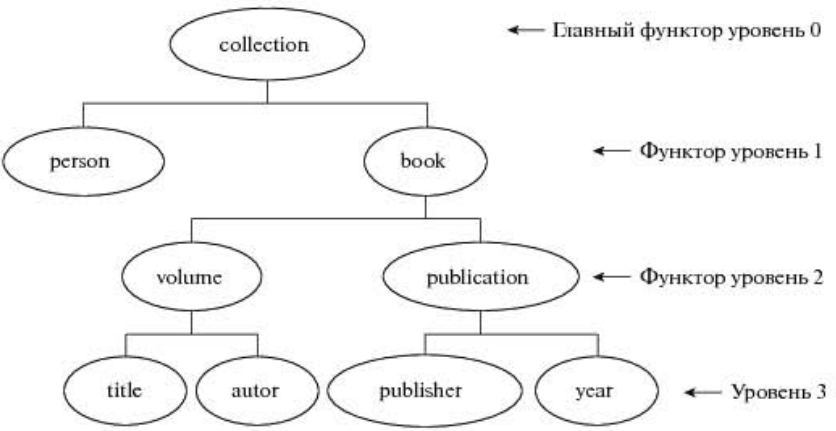

Рис. 7.2

Пример. Требуется сформировать следующий запрос: выдать названия всех книг заданного автора из коллекции Петрова, изданных до 1990 г.:

```
goal
    collection ("Петров", book (volume
    ("П. Андерсон", Title), publication (_, Year))),
        Year < 1990.
```

Использование альтернативных доменов. Представление данных часто требует наличия большого числа структур, а так как все структуры должны быть описаны, то возникают трудности с предикатами и доступом к их объектам. Чтобы избежать этих сложностей, в Прологе реализована возможность формирования ***альтернативных описаний*** для составных объектов. В предыдущем примере была рассмотрена структура, описывающая личные библиотеки. Предположим, что эту структуру надо расширить таким образом, чтобы описание включало не только книги, но и другие предметы, например животных (по каждому — имя и название) и музыкальные компакт-диски (артист и название). Множественное описание строится следующим образом: для предиката, который выступает в роли функтора, в разделе `domains` через `";"` записываются все его возможные определения, а в разделе `clauses` перечисляются все возможные определения для объявляемых предикатов.

Пример:

```
domains
    object = book (autor, title);
    animal (type, name);
    cd (artist, album)
    person, autor, title, type, name, artist, album =
    string
predicates
    owner (person, object)
clauses
    owner ("Сидоров", book ("J.R.R.Tolkein",
        "Return of the King")).
    owner ("Петров", book ("Т.Сван", "С++")).
    owner ("Сидоров", animal ("кот", "Вася")).
    owner ("Иванов", animal ("крокодил", "Гена")).
    owner ("Сидоров", cd ("Madonna", "The best of")).
    owner ("Иванов", cd ("Queen", "Classic")).
```

Запрос 1:

```
owner ("Сидоров", Object)
```

В качестве результата будут выданы все объекты, принадлежащие Сидорову:

```
Object = book ("J.R.R.Tolkein", "Return of the King")
Object = animal ("кот", "Вася")
Object = cd ("Madonna", "The best of")
```

Запрос 2:

```
owner (Person, cd (Artist, _)).
```

В результате будут выданы все компакт-диски, принадлежащие любому владельцу, результат — владелец и исполнитель на диске.

```
Person = Сидоров, Artist = Madonna
Person = Иванов, Artist = Queen
```

Если не использовать множественное описание, то пришлось бы для каждого случая, которые были бы не связаны друг с другом, иметь свой предикат. Например, для рассмотренного случая:

```
domains
    person, autor, title, name = string
    album, artist = string
    cd_library = cd (artist, album)
    book_library = book (autor, title)
    animal_library = animal (type, name)
predicates
    collection_book (person, book_library)
    collection_cd (person, cd_library)
    collection_animal (person, animal_library)
```

7.13. Примеры использования Пролога для решения интеллектуальных задач

7.13.1. Экспертные системы и управление стратегией вывода

Экспертная система — это программа, созданная для выполнения тех видов деятельности, которые доступны только человеку (проектирование, планирование, диагностика, перевод и т. п.).

Программы ЭС обычно работают таким способом, который воспринимается как «интеллектуальный», т. е. они имитируют образ действия человека-эксперта.

Эти программы специфичны, так как обычно используют механизм автоматического рассуждения (вывода) и так называемые слабые мето-

ды, такие как поиск или эвристика, которые существенно отличаются от точных и хорошо аргументированных алгоритмов и не похожи на математические процедуры большинства традиционных разработок.

Разработка правил для ЭС базируется на двух известных концепциях — *прямой и обратной цепочках рассуждений*. Каждая концепция определяет стратегию выбора в конкретных условиях. Если задачу можно построить как последовательность тестов или вопросов, позволяющих уточнить задачу, и таким образом прийти к решению, то о таких задачах говорят, что они используют прямой вывод или прямую цепочку рассуждений.

Прямой вывод применим в тех случаях, когда число потенциальных решений неуправляемо, а количество блоков данных, определяющих начальное состояние проблемы, невелико.

Стратегия прямого вывода должна на основе использования данных и правил привести к правильному заключению. Суть ее состоит в том, что задается последовательность пробных экспериментов, построенных таким образом, что каждый из них позволяет отбросить большую группу потенциальных ответов, благодаря чему значительно сужается пространство поиска. Так продолжается до тех пор, пока не останется один определенный ответ. Если на каждом шаге отбрасывается половина возможных ответов, то прямой вывод по эффективности не уступает двоичному поиску.

Другая модель использования свидетельств и заключений предполагает стратегию обратного вывода. В некоторых задачах имеется всего несколько решений при наличии огромного объема входной информации. В этом случае целесообразно в каждый момент времени рассматривать только один из возможных вариантов решения, а затем собрать и проанализировать все свидетельства, которые могут его подтвердить или опровергнуть.

Проиллюстрируем обе стратегии на примере реализации одной задачи. Создадим ЭС, которая по набору признаков позволит идентифицировать некоторое животное.

Реализация с обратной цепочкой рассуждений. Определим предикаты, которые будут использованы для системы:

```
database
    xpositiv (string, string)
    xnegativ (string, string)
predicates
    do_expert
    do_rezault
    ask (string, string)
```

```
identify (symbol)
it_is (symbol)
positiv (string, string)
negativ (string, string)
remembe (string, string)
clear_db
```

Работа каждой ЭС, в первую очередь, опирается на некоторую базу знаний, содержащую информацию о предметной области экспертизы.

В рассматриваемом случае для этого используются предикаты `identify` и `it_is`. Предикат `identify` содержит в качестве параметра название идентифицируемого животного и определяет правило, на основании которого можно определить животное по заданному набору свойств. Предикат `it_is` позволяет конкретизировать свойства, используемые в `identify`, на более низком уровне:

```
idntify ("жираф") if it_is ("травоядное"),
    positiv ("имеет", "длинную шею"),
    positiv ("имеет", "длинные ноги"),
    positiv ("имеет" "темные пятна"), !.
identify ("орел") if it_is ("птица"),
    positiv ("умеет", "летать"), positiv
    ("используется", "как национальный символ"),!.
identify ("тигр") if it_is ("млекопитающее"),
    positiv ("имеет", "коричневый окрас"),
    positiv ("имеет", "черные полосы").
identify ("сардина") if it_is ("рыба"),
    positiv ("имеет", "маленький размер"),
    positiv ("используется", "в консерве"), !.
it_is ("птица") if not (it_is ("млекопитающее")),
    positiv ("имеет", "крылья"),
    positiv ("имеет", "клюв"), !.
it_is ("млекопитающее") if positiv ("имеет",
"шерсть"), !.
it_is ("рыба") if positiv ("умеет", "плавать"),
    positiv ("живет", "в воде"), !.
```

В данной системе большая часть действий выполняется предикатами `positiv` и `negativ`. Они используются для проверки конкретных атрибутов животных, которые могут быть обнаружены в процессе диалога и записаны в БД. Предикаты `xpositiv` и `xnegativ` служат для хранения образцов положительных и отрицательных ответов, накопленных к данному моменту в динамической БД, фактически они хранят текущую цепочку вывода:

```
positiv (x, y) if xpositiv (x, y),!.
positiv (x, y) if not (negativ (x, y)),!, ask (x, y).
negativ (x, y) if xnegativ (x, y),!.
```

Действия выглядят так: сначала отыскивается положительный ответ для параметров, если его нет — проверяется наличие отрицательного ответа, если такой также отсутствует, то формируется запрос для этих параметров по следующему правилу:

```
ask (x, y) if write ("Вопрос:-", "x", "y," "?"),
    readln (Rez), remembe (x, y, Rez)
```

Предикат `remembe` добавляет в БД предикаты `xpositiv` и `xnegativ` в соответствии с ответом на запрос `ask`. Следует отметить, что отвечать на вопрос следует «да» или «нет», другие варианты ответа анализироваться не будут.

```
remembe (x, y, "да" ) if asserta (xpositiv (x, y)).
remembe (x, y, "нет") if asserta (xnegativ (x, y)),
fail.
```

Осталось определить предикаты, организующие общий вычислительный процесс и интерфейс пользователя:

```
do_expert if makewindow (1, 7, 7, "Экспертная
система",
    1,16, 22,58), do_resault,
    write ("Нажмите любую клавишу"), readchar (_),
    exit.
do_resault if identify (x), !, nl, write ("Это
животное", x), nl, clear_db.
do_resault if nl, write ("Животного нет в базе"),
clear_db.
```

Предикат `clear_db`, очищающий БД по окончании проверки, в данном контексте не имеет особого смысла, так как повторного запуска программы не предусматривается и сразу выходят в DOS. Но если требуется провести повторную проверку, не выходя из программы, необходима небольшая модификация этого предиката. Его определение выглядит так:

```
clear_db if retract (xpositiv, _, _)), fail.
clear_db if retract (xnegativ, _ ,_)), fail.
```

или

```
clear_db if retract (_), fail.
```

Программа с прямой цепочкой рассуждений. Реализуется та же задача, поэтому механизм построения запросов низкого уровня может остаться тот же. Эта версия может быть спроектирована только после тщательного анализа предметной области. Суть метода прямой цепочки рассуждений заключается в составлении вопросов, позволяющих

на каждом шаге отбросить большое количество возможных ответов, так что правильный ответ может быть установлен быстро. Причем задаваемые при каждой проверке вопросы целиком зависят от возможных ответов. Различные ответы подразумевают использование разных проверок. Пусть предметная область описывается представленной на рис. 7.3.

Из схемы видно, что независимо от выбранного маршрута поиска потребуется не более четырех проверок (а иногда достаточно и трех). Основное правило можно построить так:

```
animal if find_animal, have_found ( x),
    write ("Задуманное животное -", x), nl, !.
```

Отсутствует возможность отрицательного ответа:

```
find_animal if test1 (x), test2 (x, y),
test3 (x, y, z), test4 ( x, y, z, -), !.
find animal.
```

Правила `test` (с первого по четвертый) должны охватить все, что может выясниться, соответственно, при первой, второй, третьей и четвертой проверках.

Структура аргументов этих правил определена так, чтобы последующий шаг учитывал результаты всех предыдущих.

Правила проверки предполагают перечисление всех возможностей. Различные комбинации аргументов в задействованном правиле отражают всю историю вывода до указанного места.

Например, `test1` может успешно завершаться одним из двух способов:

`test1 (m)` — животное млекопитающее;

`test1 (n)` — животное не млекопитающее.

Для правила `test2` существуют четыре вероятные комбинации:

`test2 (m, c)` — млекопитающее и плотоядное;

`test2 (m, n)` — млекопитающее и не плотоядное;

`test2 (n, w)` — не млекопитающее и плавает;

`test2 (n, n)` — не млекопитающее и не плавает.

Для `test3` больше всего возможностей — восемь:

`test3 (m, c, s)` — млекопитающее, плотоядное, полосатое;

`test3 (m, c, n)` — млекопитающее, плотоядное, без полос;

`test3 (m, n, l)` — млекопитающее, не плотоядное, живет на суше;

`test3 (m, n, n)` — млекопитающее, не плотоядное, живет не на суше;

`test3 (n, w, t)` — не млекопитающее, плавает, щупальца;

`test3 (n, w, n)` — не млекопитающее, плавает, нет щупальц;

`test3 (n, n, f)` — не млекопитающее, не плавает, летает;

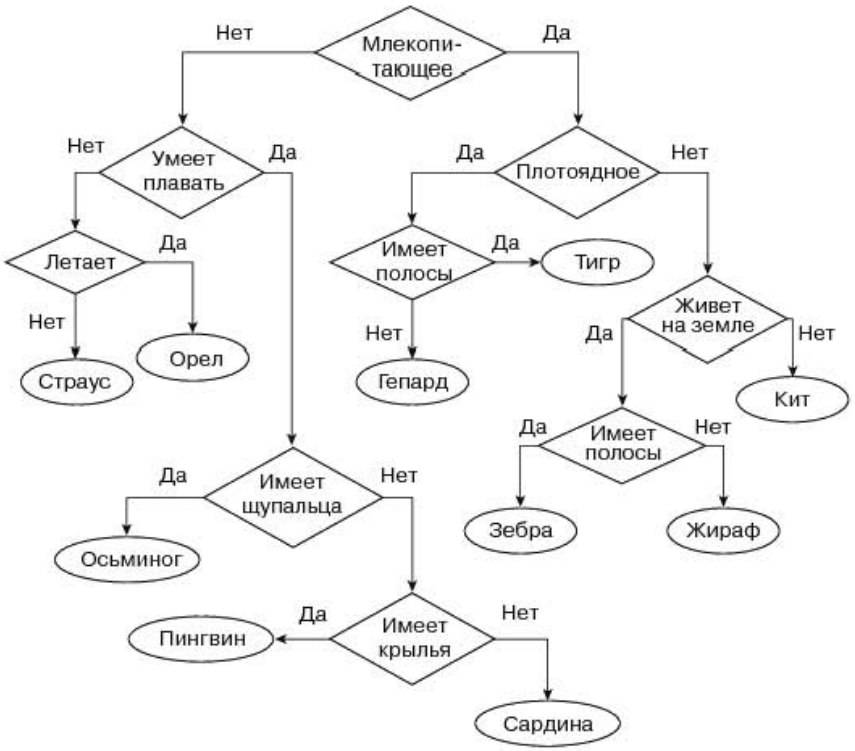

Рис. 7.3

`test3 (n, n, n)` — не млекопитающее, не плавает, не летает.

Для `test4` — четыре варианта:

`test4 (m, n, l, s)` — млекопитающее, не плавает, живет на суше, имеет полосы;

`test4 (m, n, l, n)` — млекопитающее, не плавает, живет на суше, без полос;

`test4 (n, w, n, f)` — не млекопитающее, плавает, без щупалец, имеет крылья;

`test4 (n, w, n, n)` — не млекопитающее, плавает, без щупалец, нет крыльев.

Текст программы, описывающий эти правила, будет выглядеть так:

```
test1 (m) if it_is ("млекопитающее"),!.
test1 (n).
test2 (m, c) if it_is ("плотоядное"),!.
test2 (m, n).
```

```
test2 (n, w) if it_is ("плавает").
test2 (n, n).
test3 (m, c, s) if positiv ("имеет", "полосы"),
    asserta (have_found ("тигр")),!.
test3 (m, c, n) if asserta (have_found (гепард)),! .
test3 (m, n, l) if not (positiv ("живет", "на суше")),! .
test3 (m, n, n) if asserta (have_found ("кит")), ! .
test3 (n, n, f) if positiv ("умеет", "летать"),
    asserta (have_found ("орел")),!.
test3 (n, n, n,) if asserta (have_found ("страус")),!.
test3 (n, w, t) if positiv ("имеет", "щупальца"),
    asserta (have_foun ("осьминог")),!.
test4 (m, n, l, s) if positiv ("имеет", "полосы"),
    asserta (have_found ("зебра")),!.
test4 (m, n, l, n) if asserta (have_found ("жираф")).
test4 (n, w, n, f) if positiv ("имеет", "крылья"),
    asserta (have_found ("пингвин")),!.
test4 (n, w, n, n) if asserta (have_found ("сардина")).
```

7.13.2. Моделирование работы машины Тьюринга

При выполнении заданий студенты должны построить собственную машину Тьюринга, позволяющую реализовать заданную функцию. Машина описывается в виде системы команд. Рассмотрим примеры реализации МТ. Пусть на ленте записаны два натуральных числа в унарном коде (унарный код — это представление числа в виде соответствующей последовательности единиц, например $5 = 11111 = 1^5$). Числа разделены символом операции сложения. Требуется реализовать операцию сложения таким образом, чтобы после ее реализации на ленте был записан результат, а управляющее устройство находилось на первом символе результата. Работу МТ опишем в виде системы команд.

Для сложения чисел 2 и 3 начальное состояние МТ имеет вид, показанный на рис. 7.4.

Реализация алгоритма будет базироваться на следующей идее. Начиная из исходного положения, требуется дойти до символа «+»,

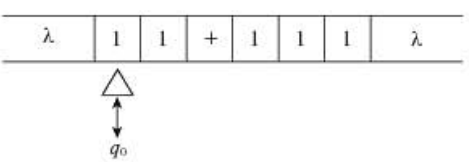

Рис. 7.4

заменить его на символ «1», затем дойти до конца записи и заменить последнюю единицу на символ пробела «λ», после чего работу машины закончить. Опишем изложенное в виде системы команд:

$q_0 1 \to q_0 1\ R$;

$q_0 + \to q_1 1\ R$;

$q_1 1 \to q_1 1\ R$;

$q_1 \lambda \to q_2 \lambda\ L$;

$q_2 1 \to q_z \lambda\ E$.

Для приведенного примера алфавиты A и Q имеют вид

$$A = \{1, +, \lambda\},\ Q = \{q_1, q_2, q_3\}.$$

Внимание. Так как в программе вводимая строка имитирует запись на ленте, а по описанным правилам считается, что вся лента, не занятая значимой (относящейся к текущей функции) информацией, заполнена пробелами, то вводимая строка обязательно должна заканчиваться пробелом. Каждый шаг работы машины отрабатывается после нажатия клавиши <*Enter*>.

Назначение предикатов, реализующих описанную машину:

`tu (integer, string, integer, integer, string, integer)` — описывает команды машины Тьюринга. Первый параметр — текущее состояние, второй — наблюдаемый символ, третий — текущая позиция на ленте, четвертый — новое состояние, пятый — записываемый на ленту символ, шестой — новая позиция, полученная после перемещения считывающего устройства;

`t (integer, string, integer)` — организует общий вычислительный процесс. Первый параметр — текущая позиция на ленте, второй — строка символов, соответствующая записанной на ленте информации, третий — текущее состояние машины;

`str (string, integer, string, string)` — осуществляет процесс записи нового символа на ленту. В строке символов, которой соответствует первый параметр, в позиции, задаваемой вторым параметром, старый символ заменяется на символ, определяемый третьим параметром. Полученная после замены строка связывается с четвертым параметром;

`sym (integer, string, string)` — извлекает из строки, связанной со вторым параметром, символ, стоящий в позиции, определенной первым параметром, и присваивает его значение третьему параметру.

Кроме перечисленных предикатов, определяемых пользователем в приведенном примере, использовано два стандартных предиката,

ориентированных на организацию вывода информации на экран, но не рассмотренных ранее, это field_str и scr_char. Их функции следующие:

field_str (Строка, Столбец, Длина, Строка символов) — записывает или считывает строку символов, начиная из заданной позиции экрана. Начальную позицию для чтения или записи задают параметры «Строка» и «Столбец». «Длина» задает длину считываемой строки. «Строка символов» — строковая переменная. Если переменная свободная, то ей присваивается считанная с экрана строка. Если переменная связанная, то ее значение вписывается в указанную в параметрах экранную область;

scr_char (Строка, Столбец, Символ) — записывает или считывает «Символ» в заданной позиции экрана. Позицию для чтения или записи задают параметры «Строка» и «Столбец». Если переменная свободная, то ей присваивается считанный с экрана символ. Если переменная связанная, то ее значение вписывается в указанную в параметрах позицию экрана.

```
predicates
    tu (integer, string, integer, integer, string,
       integer)
    t (integer, string, integer)
    str (string, integer, string, string)
    sym (integer, string, string)
clauses
    sym (1, S0, S2) if frontstr (1, S0, S2,_).
    sym (N, S0, S2) if N1 = N-1,frontstr (N1, S0,_,
        R),frontstr (1, R, S2,_).
    str (S0, 1, SR, Rez) if frontchar (S0,_, R2),
        concat (SR, R2, Rez).
    str (S0, N, SR, Rez) if N1 = N-1, frontstr
        (N1, S0, S2, R), frontchar (R, _, R2),
concat (S2, SR, R3),concat (R3, R2, Rez).
tu (0, "1", N, 0, "1", N1) if N1=N+1.
tu (0, "+", N, 1, "1", N1) if N1=N+1.
tu (1, "1", N, 1, "1", N1) if N1=N+1.
tu (1, " ", N, 2, " ", N1) if N1=N-1.
tu (2, "1", N, 10, " ", N) if !.
t (_, S, 10) if field_str (0, 1, 50, S),readchar
(_), exit.
t (N, S, Q) if field_str (0, 1, 50, S), scr_char (2,
   N, '^'), sym (N, S, S2), tu (Q, S2, N, Qr, S2r,
   Nr), str (S, N, S2r, Rez), readchar (_), scr_char
```

```
    (2, N, ' '),
t (Nr, Rez, Qr).
goal
    write ("Введите начальную строку  — >"),readln
    (Str), scr_char (0, 0,' '),t (1, Str, 0).
```

ГЛАВА 8

ЗАДАЧИ И ПРИМЕРЫ ИХ РЕШЕНИЯ

8.1. Теория множеств и булева алгебра

Пример 8.1.1. Привести заданную логическую функцию к форме СДНФ с использованием обоих известных способов (табл. 8.1):

$$f(x,y,z)=(x\oplus y)\to \bar{y}\,\&\,\bar{x}\vee x\,\&\,z\,\&\,(y\vee\bar{x})=\overline{(\bar{x}\vee\bar{y})\,\&\,(x\vee y)}\vee$$
$$\vee\bar{x}\,\&\,\bar{y}\vee x\,\&\,y\,\&\,z\vee x\,\&\,z\,\&\,\bar{x}=\overline{\bar{x}\vee\bar{y}}\vee\overline{x\vee y}\vee\bar{x}\,\&\,\bar{y}\vee x\,\&\,y\,\&\,z=$$
$$=x\,\&\,y\vee\bar{x}\,\&\,\bar{y}\vee\bar{x}\,\&\,\bar{y}\vee x\,\&\,y\,\&\,z=x\,\&\,y\vee\bar{x}\,\&\,\bar{y}\vee x\,\&\,y\,\&\,z=$$
$$=x\,\&\,y\,\&\,(z\vee\bar{z})\vee\bar{x}\,\&\,\bar{y}\,\&\,(z\vee\bar{z})\vee x\,\&\,y\,\&\,z=x\,\&\,y\,\&\,z\vee x\,\&\,y\,\&\,\bar{z}\vee$$
$$\vee\bar{x}\,\&\,\bar{y}\,\&\,z\vee\bar{x}\,\&\,\bar{y}\,\&\,\bar{z}\vee x\,\&\,y\,\&\,z=x\,\&\,y\,\&\,z\vee x\,\&\,y\,\&\,\bar{z}\vee$$
$$\vee\bar{x}\,\&\,\bar{y}\,\&\,z\vee\bar{x}\,\&\,\bar{y}\,\&\,\bar{z}$$

Таблица 8.1

x	y	z	$x\oplus y$	$\bar{x}\,\&\,\bar{y}$	$x\,\&\,z\,\&\,(y\vee\bar{x})$	$f(x,y,z)$
0	0	0	0	1	0	1
0	0	1	0	1	0	1
0	1	0	1	0	0	0
1	0	0	1	0	0	0
0	1	1	1	0	0	0
1	0	1	1	0	0	0
1	1	0	0	0	0	1
1	1	1	0	0	1	1

$$f(x,y,z)=\bar{x}\bar{y}\bar{z}\vee\bar{x}\bar{y}z\vee xy\bar{z}\vee xyz.$$

Пример 8.1.2. Привести заданную логическую функцию к форме СКНФ с использованием обоих известных способов (табл. 8.2):

Таблица 8.2

x	y	z	$z \oplus \bar{x}$	$\bar{y} \vee \bar{z} \vee x$	$\bar{x} \to zy$	$f(x, y, z)$
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	1	0	1	1	0	0
1	0	0	0	0	1	0
0	1	1	0	1	1	0
1	0	1	1	1	1	1
1	1	0	0	1	1	0
1	1	1	1	1	1	1

$$f(x,y,z)=(x\vee y\vee\bar{z})(x\vee\bar{y}\vee\bar{z})(\bar{x}\vee\bar{y}\vee z)(\bar{x}\vee y\vee z)(x\vee z\vee y)(x\vee\bar{y}\vee z)$$

$$f(x,y,z)=(z\oplus\bar{x})(\bar{y}\vee\bar{z}\vee x)(\bar{x}\to zy)=$$

$$=(z\vee\bar{x})(\bar{z}\vee\overline{\overline{x}})(x\vee\bar{y}\vee\bar{z})(\overline{\overline{\bar{x}}}\vee zy)=$$

$$=(z\vee\bar{x})(\bar{z}\vee x)(x\vee\bar{y}\vee\bar{z})(x\vee z)(x\vee y)=$$

$$=(z\vee\bar{x}\vee y\bar{y})(\bar{z}\vee x\vee y\bar{y})(x\vee\bar{y}\vee\bar{z})\&(x\vee z\vee y\bar{y})(x\vee y\vee z\bar{z})=$$

$$=(z\vee\bar{x}\vee\bar{y})(z\vee\bar{x}\vee y)(\bar{z}\vee x\vee\bar{y})(\bar{z}\vee x\vee y)(x\vee\bar{y}\vee\bar{z})\&$$

$$\&(x\vee z\vee y)(x\vee z\vee\bar{y})(x\vee y\vee z)(x\vee y\vee\bar{z})=$$

$$=(x\vee y\vee\bar{z})(x\vee\bar{y}\vee\bar{z})(\bar{x}\vee\bar{y}\vee z)\&(\bar{x}\vee y\vee z)(x\vee z\vee y)(x\vee\bar{y}\vee z).$$

Задача 8.1.1. Построить таблицы истинности для заданной логической функции:

а) $f(x,y,z)=\overline{((x\,|\,y)\&(x\downarrow z))}$;

б) $f(x,y,z)=\big((x\vee y)\&(x\vee z)\big)\oplus(x\to y)$;

в) $f(x,y,z)=\overline{(x\,|\,y)\oplus(x\to y)}$;

г) $f(x,y,z)=(x\vee y)\oplus(x\,|\,z)\to(x\downarrow y)$.

Задача 8.1.2. Привести заданную логическую функцию к форме СДНФ, используя табличный способ и способ эквивалентных преобразований:

а) $f(x,y,z)=(x\vee\bar{y}\vee z)\to\overline{(x\to\overline{y\&z})}\vee\bar{x}$;

б) $f(x,y,z)=\big(x\to\bar{x}\&\bar{z}\big)\to\overline{\overline{x}\vee y}\vee\bar{y}\&x$;

в) $f(x,y,z)=\big(x\sim\bar{y}\big)\vee\overline{\overline{x}\to\overline{z}}\vee\bar{x}\&\bar{y}$;

г) $f(x,y,z)=(x\oplus y)\to\bar{y}\,\&\,\bar{x}\vee x\&z\&\left(y\vee\bar{x}\right)$.

Задача 8.1.3. Привести заданную логическую функцию к форме СКНФ, используя табличный способ и способ эквивалентных преобразований:

а) $f(x,y,z)=(x\sim y)\,\&\,(x\oplus z)\,\&\left(x\vee\bar{y}\right)$;

б) $f(x,y,z)=(x\mid y)\,\&\left(x\to\bar{y}\right)\&\left(\bar{x}\oplus\bar{z}\right)$;

в) $f(x,y,z)=\overline{x\downarrow\bar{y}}\,\&\left(\bar{z}\to x\right)\&\left(\bar{z}\vee\bar{x}\right)$;

г) $f(x,y,z)=\left(\bar{z}\to x\right)\&\left(z\mid\bar{y}\right)\&\,(x\oplus y)$.

8.2. Логика высказываний

Задача 8.2.1. Определить, какие из приведенных выражений являются формулами исчисления высказываний:

а) $\left(p_1\to(p_2\vee p_3)\right)\to p_3$;

б) $\left(p_1\,\&(\to p_2)\right)\to\left(p_2\to\overline{p_1}\right)$;

в) $(a\,\&\,b)\vee\to c$;

г) $\left(\overline{p_1}\,\&\,\overline{p_2}\right)\to(p_1\vee p_2)$;

д) $((a\,\&\,b)\vee(\bar{c}))\to d$;

е) $(a\,\&\,b)\vee\&$;

ж) $(a_1\,\&\,a_2)\sim\overline{a_3}$;

з) $(a\sim b)\to(a\vee b)$;

и) $\left(\left(p_1\vee\overline{p_2}\right)\to\left(\overline{p_1}\vee\vee p_2\right)\right)\sim(\vee p_1\vee p_2)$..

Пример 8.2.1. Пусть задана формула $\left(x\to\bar{y}\right)\&\left(\overline{\bar{z}\,\&\,y}\right)$. Выписать все подформулы заданной формулы с распределением их по уровням вложенности, используя табличное представление (*а* — табл. 8.3) и представление в виде дерева (*б* — рис. 8.1).

а)

Таблица 8.3

Подформула	Глубина
$\left(x\to\bar{y}\right)\&\left(\overline{\bar{z}\,\&\,y}\right)$	0
$\left(x\to\bar{y}\right),\ \left(\overline{\bar{z}\,\&\,y}\right)$	1
$x,\ \bar{y},\left(\bar{z}\,\&\,y\right)$	2
$y,\ \bar{z}$	3
z	4

б)

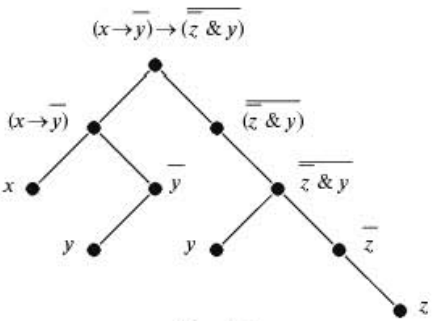

Рис. 8.1

Задача 8.2.2. Выписать все подформулы заданных формул с распределением их по уровням вложенности, используя табличное представление и представление в виде дерева:

а) $\big((x\to y)\&(y\to z)\big)\to\big(\overline{x}\vee z\big)$;

б) $(x\to y)\to((x\to\overline{y})\to\overline{y})$;

в) $\big(p_1\to(p_2\vee p_3)\big)\to p_3$;

г) $\big(p_1\&(p_1\to p_2)\big)\to\big(p_2\to\overline{p_1}\big)$;

д) $\big(\overline{p_1}\&\overline{p_2}\big)\to(p_1\vee p_2)$;

е) $\Big(\big(p_1\vee\overline{p_2}\big)\to\big(\overline{p_1}\vee p_2\big)\Big)\sim(p_1\vee p_2)$;

ж) $(x\to y)\to\big((y\to z)\to(x\to z)\big)$;

з) $(a\to b)\to\big((c\to a)\to(c\to b)\big)$;

и) $(a\to b)\to(a\vee c\to b\vee c)$;

к) $(a\to b)\to\big((c\vee a)\to(c\&b)\big)$;

л) $\big(p_1\vee(p_2\to p_3)\big)\&\,p_3\to(p_1\vee p_2)$;

м) $(a\to c)\to\Big(\big(b\&\overline{a}\big)\to\big(\overline{\overline{a}\vee b}\big)\Big)$;

н) $(a\to b)\to\big((c\vee a)\to(c\&b)\big)$.

Задача 8.2.3. Доказать, используя метод эквивалентных преобразований, справедливость аксиом исчисления высказываний:

а) $x\to(y\to x)$;

б) $\big(x\to(y\to z)\big)\to\big((x\to y)\to(x\to z)\big)$;

в) $(z \to x) \to ((z \to y) \to (z \to x \,\&\, y))$;

г) $(x \to z) \to ((y \to z) \to (x \vee y \to z))$;

д) $(x \to y) \to (\bar{y} \to \bar{x})$.

Пример 8.2.2. Используя алгоритм редукции, доказать общезначимость следующей формулы: $(a \to b) \to ((c \to a) \to (c \to b))$.

Пусть при некоторой интерпретации $(a \to b) \to ((c \to a) \to (c \to b)) = 0$.

Это возможно, только если

$$a \to b = 1; \tag{8.1}$$

$$(c \to a) \to (c \to b) = 0. \tag{8.2}$$

Тогда из (8.1) следует, что возможна одна из следующих комбинаций значений переменных a и b:

$$a = 0, \quad \longleftrightarrow \quad b = 0;$$
$$a = 1, \quad \longleftrightarrow \quad b = 1;$$
$$a = 0, \quad \longleftrightarrow \quad b = 1.$$

Из (8.2) следует $(c \to a) = 1$, это означает, что возможны следующие значения c и a:

$$c = 0, \quad \longleftrightarrow \quad a = 0;$$
$$c = 1, \quad \longleftrightarrow \quad a = 1;$$
$$c = 0, \quad \longleftrightarrow \quad a = 1.$$

Из $c \to b = 0$ имеем $c = 1, \quad b = 0$. Это единственно допустимые для c и b значения, при которых формула принимает значение «ложь». Сопоставляем полученные результаты с ранее рассмотренными возможными значениями переменных.

Оказывается, что при $c = 1$ единственное допустимое значение для a — это $a = 1$, а при $b = 0$ единственное допустимое значение $a = 0$. То есть переменная a должна принимать взаимно исключающие значения, что невозможно. Следовательно, предположение о существовании интерпретации, при которой формула $(a \to b) \to ((c \to a) \to (c \to b))$ принимает значение «ложь», неверно, и это означает ее общезначимость.

Задача 8.2.4. Доказать, используя алгоритм редукции:

а) $(x \to y) \to ((y \to z) \to (x \to z))$;

б) $(a \to b) \to ((c \to a) \to (c \to b))$;

в) $(a \to b) \to (a \vee c \to b \vee c)$;

г) $(a \to b) \,\&\, (b \to c) \to (a \to c)$;

д) $(a \& b \to c) \to (a \to (b \to c))$;
е) $(a \to (b \to c)) \to (a \& b \to c)$;
ж) $(a \to b) \to (a \& c \to b \& c)$;
з) $(a \to b) \to (a \vee c \to b \vee c)$;
и) $((a \to b) \& (a \to c)) \to (a \to (b \& c))$;
к) $((a \to c) \& (b \to c)) \to ((a \vee b) \to c)$;
л) $((a \& b) \to c) \leftrightarrow (a \to (b \to c))$;
м) $a \to (b \to a \& b)$.

Задача 8.2.5. Используя правила эквивалентных преобразования, доказать тождественную истинность выражений из задачи 8.2.4.

Пример 8.2.3. Доказать, используя метод резолюций, что S является логическим следствием множества гипотез H, где $H = \left\{\bar{a} \vee (b \to c), \overline{c \& d} \vee e, f \vee \left(\overline{\bar{d} \vee e}\right)\right\}$, а $S = \bar{a} \vee \bar{b} \vee f$. Сначала преобразуем множество гипотез в множество дизъюнктов:

1) $\bar{a} \vee (b \to c) = \bar{a} \vee \bar{b} \vee c$;
2) $\overline{c \& d} \vee e = \bar{c} \vee \bar{d} \vee e$;
3) $f \vee \left(\overline{\bar{d} \vee e}\right) = f \vee d\bar{e} = (f \vee d)(f \vee \bar{e})$;
4) $\bar{S} = \overline{\bar{a} \vee \bar{b} \vee f} = a \& b \& \bar{f}$.

Для доказательства того, что $H \mid = S$, необходимо и достаточно доказать невыполнимость следующего множества дизъюнктов:

$$H = \{\bar{a} \vee \bar{b} \vee c, \bar{c} \vee \bar{d} \vee e, f \vee d, f \vee \bar{e}, a, b, \bar{f}\}.$$

1) $\bar{a} \vee \bar{b} \vee c$;
2) $\bar{c} \vee \bar{d} \vee e$;
3) $f \vee d$;
4) $f \vee \bar{e}$;
5) a;
6) b;
7) $\bar{f}$;
8) d (3–7);
9) $\bar{b} \vee c$ (1–5);
10) c (6–9);
11) $\bar{e}$ (4–7);
12) $\bar{c} \vee \bar{d}$ (2–11);
13) $\bar{d}$ (10–12);
14) пустой дизъюнкт (8–13).

Задача 8.2.6. Используя метод резолюций для логики высказываний, доказать справедливость вывода для заданного множества гипотез $H = \{h_1, h_2, ..., h_m\}$ и следствия S:

а) $H = \{a \to (\bar{b} \vee c),\ c\&d \to e,\ \bar{f} \to d\&\bar{e}\},\ S = a \to (b \to f)$;

б) $H = \{(a \vee b) \to c\&d,\ (d \vee e) \to f\},\ S = a \to f$;

в) $H = \{a \to b\&c,\ \bar{b} \vee d,\ (e \to \bar{f}) \to \bar{d},\ b \to a\&\bar{e}\},\ S = b \to e$;

г) $H = \{(a \to b)\ \&\ (c \to d),\ (b \to e)\ \&\ (d \to f),\ e\&f,\ a \to c\},\ S = \bar{a}$;

д) $H = \{p \vee \bar{r} \vee \bar{t},\ q,\ r,\ t \vee \bar{p} \vee \bar{r},\ t \vee \bar{q}\},\ S = p\&q\&r$;

е) $H = \{a \sim b,\ b \to c,\ \bar{c} \vee d,\ \bar{a} \to d\},\ S = d$;

ж) $H = \{(a \vee b) \to (c \vee d),\ \overline{\bar{a} \vee g},\ d \to \bar{g}\},\ S = c$;

з) $H = \{(c \to g)\ \&\ (d \to s),\ s\&g \to e,\ \bar{e}\},\ S = \bar{c} \vee \bar{d}$;

и) $H = \{ab \vee \bar{a}\bar{b},\ \bar{b} \vee c,\ c \to d,\ a \vee d\},\ S = d$;

к) $H = \{a\&\bar{g},\ \bar{d} \vee g,\ \overline{\bar{a}\bar{b}} \vee (c \vee d)\},\ S = c$;

л) $H = \{\bar{a} \vee \bar{c} \vee b,\ d \to a,\ \bar{d} \vee b\},\ S = \bar{c}$;

м) $H = \{\bar{a} \vee (b \to c),\ \overline{c\ \&\ d} \vee e,\ f \vee (\overline{\bar{d} \vee e})\},\ S = \bar{a} \vee \bar{b} \vee f$;

н) $H = \{(\bar{a} \vee b)\ \&\ (\bar{c} \vee d),\ (\bar{b} \vee e)\ \&\ (\bar{d} \vee f),\ \bar{e} \vee \bar{f},\ \bar{a} \vee c\},\ S = \bar{a}$.

Пример 8.2.4. Используя метод прямой и обратной дедукции, доказать справедливость вывода для заданного множества гипотез $H = \{h_1, h_2, ..., h_m\}$ и следствия S:

$$H = \{(c \to g)\&(d \to s), s\&g \to e, \bar{e}\},\quad S = \bar{c} \vee \bar{d}.$$

При доказательстве на основе прямой дедукции доказывается справедливость следующей формулы: $h_1\ \&\ h_2\ \&\ \ldots\ \&\ h_n\ \&\ \bar{S} = 0$, а при обратной дедукции следующей: $\bar{h}_1 \vee \bar{h}_2 \vee \ldots \vee \bar{h}_n \vee S = 1$. При построении доказательства по дедукции в качестве механизма воспользуемся методом эквивалентных преобразований.

Прямая дедукция: $cd(c \to g)(d \to s)(sg \to e)\bar{e} = 0$.

Доказательство:

$$cd(c \to g)\&(d \to s)(sg \to e)\bar{e} = cd(\bar{c} \vee g)(\bar{d} \vee s)(\bar{s} \vee \bar{g} \vee e)\bar{e} =$$

$$= (cd\bar{c} \vee cdg)(\bar{d} \vee s)(\bar{e}\bar{s} \vee \bar{g}\bar{e} \vee e\bar{e}) = (cd\bar{d}g \vee cdgs)(\bar{e}\bar{s} \vee \bar{e}\bar{g}) =$$

$$= cdgs\bar{e}\bar{s} \vee cdgs\bar{e}\bar{g} = 0.$$

Обратная дедукция: $\overline{(c \to g)\&(d \to s)} \vee \overline{(s\&g \to e)} \vee \bar{\bar{e}} \vee \bar{c} \vee \bar{d} = 1$.

Доказательство:

$$\overline{(c \to g)\&(d \to s)} \vee \overline{(s\&g \to e)} \vee \bar{\bar{e}} \vee \bar{c} \vee \bar{d} =$$

$$= \overline{\left(\bar{c} \vee g\right) \& \left(\bar{d} \vee s\right)} \vee \overline{\overline{s \& g} \vee e} \vee \bar{\bar{e}} \vee \bar{c} \vee \bar{d} =$$

$$= c \& \bar{g} \vee d \& \bar{s} \vee s \& g \& \bar{e} \vee e \vee \bar{c} \vee \bar{d} = \bar{c} \vee \bar{g} \vee \bar{d} \vee \bar{s} \vee s \& g \vee e =$$

$$= \bar{c} \vee \bar{g} \vee \bar{d} \vee \bar{s} \vee g \vee e = \bar{c} \vee 1 \vee \bar{d} \vee \bar{s} \vee e = 1.$$

Задача 8.2.7. Для примеров из задачи 8.2.6 построить доказательство, используя метод прямой дедукции.

Задача 8.2.8. Для примеров из задачи 8.2.6 построить доказательство, используя метод обратной дедукции.

Пример 8.2.5. Пусть дано множество утверждений, сформулированных на естественном языке, и некоторое заключение, которое из них следует. Требуется превратить их в множество высказываний и доказать справедливость рассуждений, используя метод резолюций. Набор рассуждений следующий:

1) если пойти на первую пару, то надо встать рано, а если играть в преферанс, то лечь придется поздно;
2) если лечь поздно и рано встать, то спать придется мало;
3) мало спать нельзя.

Заключение: надо или не играть в преферанс, или не идти на первую пару.

Введем следующие обозначения для высказываний:

g — встать рано;

d — играть в преферанс;

c — идти на первую пару;

s — лечь поздно спать;

e — мало спать.

Используя введенные обозначения, перейдем от утверждений к следующему набору гипотез: $H_1 = \left(c \to g\right) \& \left(d \to s\right)$, $H_2 = s \& g \to e$, $H_3 = \bar{e}$. Следствие примет вид $A = \bar{c} \vee \bar{d}$.

Теперь построим доказательство, используя метод резолюций. Для этого приведем имеющиеся гипотезы к форме дизъюнктов:

$$H_1 = \left(c \to g\right) \& \left(d \to s\right) = \left(\bar{c} \vee g\right)\left(\bar{d} \vee s\right);$$

$$H_2 = s \& g \to e = \overline{s \& g} \vee e = \bar{s} \vee \bar{g} \vee e;$$

$$H_3 = \bar{e}.$$

Отрицание следствия будет иметь вид $\bar{A} = \bar{\bar{c} \vee \bar{d}} = cd$.

1) $\bar{c} \vee g$;
2) $\bar{d} \vee s$;
3) $\bar{s} \vee \bar{g} \vee e$;

4) $\bar{e}$;
5) c;
6) d;
7) $g(1-5)$;
8) $s(2-6)$;
9) $\bar{s} \vee e$ $(3-7)$;
10) $e(8-9)$;
11) пустой дизъюнкт $(4-10)$.

Задача 8.2.9. Записать формально следующее рассуждение на естественном языке в терминах логики высказываний и доказать его справедливость, используя метод резолюций:

а) посылки: если идет дождь, то нежарко; если светит солнце, то жарко; идет дождь;

заключение: нежарко и не светит солнце;

б) посылки: экзамен сдан вовремя или сессия продлена; если сессия продлена, то не сдана курсовая работа или не зачтены лабораторные работы; курсовая работа сдана, экзамен вовремя не сдан;

заключение: неверно, что если курсовая работа сдана, то лабораторные работы зачтены;

в) посылки: если имеет место денежная эмиссия, то растет курс доллара; если эмиссии нет и инфляция не растет, то курс доллара не растет; инфляция не растет;

заключение: имеет место эмиссия и растет курс доллара или нет эмиссии и курс доллара не растет;

г) посылки: заработная плата возрастет, только если будет инфляция; если будет инфляция, то увеличится стоимость жизни; заработная плата возрастет;

заключение: стоимость жизни увеличится;

д) посылки: если 2 — простое число, то это наименьшее простое число; если 2 — наименьшее простое число, то 1 не есть простое число; число 1 не есть простое число;

заключение: 2 — простое число;

е) посылки: Джон или переутомился, или болен; если он переутомился, то он раздражается; он не раздражается;

заключение: Джон болен;

ж) посылки: если завтра будет холодно, то я надену теплое пальто, если рукав будет починен; завтра будет холодно, а рукав не будет починен;

заключение: я не надену теплое пальто;

з) посылки: если исход скачек будет предрешен сговором или в игорных домах будут орудовать шулеры, то доходы от туризма упадут и город пострадает; если доходы от туризма упадут, полиция будет довольна; полиция никогда не бывает довольна;

заключение: исход скачек не предрешен сговором;

и) посылки: или Вася и Петя одного возраста, или Вася старше Пети; если Вася и Петя одного возраста, то Федя и Петя не одного возраста; если Вася старше Пети, то Петя старше Семена;

заключение: или Федя и Петя не одного возраста, или Петя старше Семена;

к) посылки: если будет идти снег, то машину будет трудно вести; если будет трудно вести машину, то я опоздаю, если не выеду пораньше; идет снег;

заключение: я должен выехать пораньше;

л) посылки: если подозреваемый совершил кражу, то она либо была тщательно подготовлена, либо он имел соучастника; если бы кража была подготовлена тщательно, то, если бы был соучастник, украдено было бы гораздо больше; украдено мало;

заключение: подозреваемый невиновен;

м) посылки: в бюджете возникнет дефицит, если не повысить налоги; если в бюджете имеется дефицит, то государственные расходы на социальные нужды сокращаются;

заключение: если повысить налоги, то государственные расходы на социальные нужды не сокращаются;

н) посылки: намеченная атака удастся, только если захватить противника врасплох или если его позиции плохо защищены; захватить его врасплох можно только тогда, когда он беспечен; он не будет беспечен, если его позиции плохо защищены;

заключение: атака не удастся.

8.3. Логика предикатов

Задача 8.3.1. Пусть заданы предикаты: $P(x)$ — x — четное число и $Q(x)$ — x — число, кратное трем. Предикаты определены на множестве N натуральных чисел. Определить области истинности предикатных выражений:

а) $P(x)\&Q(x)$;

б) $P(x)\vee Q(x)$;

в) $\overline{P}(x)$;

г) $P(x) \to Q(x)$.

Задача 8.3.2. Пусть $P(x)$ обозначает «x — простое число», $E(x)$ — «x — четное число», $O(x)$ — «x — нечетное число» и $D(x, y)$ — «y делится на x». Перевести на русский язык следующие предикатные выражения:

а) $P(7)$;

б) $E(2) \,\&\, P(2)$;

в) $\forall x(D(2,x) \to E(x))$;

г) $\exists x(E(x) \& D(x,6))$;

д) $\exists x(E(x) \& P(x)) \& \exists x((E(x) \& P(x) \& \exists y(x \neq y \& E(y) \& P(y)))$;

е) $\forall x(\overline{E}(x) \to \overline{D}(2,x))$;

ж) $\forall x(E(x) \& \forall y(D(x,y) \to E(y)))$;

з) $\forall x(P(x) \to \exists y(E(y) \& D(x,y)))$;

и) $\forall x(O(x) \to \forall y(P(y) \to \overline{D}(x,y)))$.

Задача 8.3.3. Пусть предикаты определены на множестве N натуральных чисел:

— $A(n)$ — n делится на 3;

— $B(n)$ — n делится на 2;

— $C(n)$ — n делится на 4;

— $D(n)$ — n делится на 6;

— $E(n)$ — n делится на 12.

Определить, какие из предикатных выражений истинные, а какие — ложные:

а) $\forall n\big(C(n) \& D(n) \to E(n)\big)$;

б) $\forall n\big(\overline{E}(n) \to B(n) \& D(n)\big)$;

в) $\exists n\big(B(n) \& C(n) \to \overline{D}(n)\big)$;

г) $\forall n\big(\overline{A}(n) \to \overline{E}(n)\big)$.

Задача 8.3.4. Найти отрицание следующих формул:

а) $\forall x\big(A(x) \to B(x)\big) \& \exists x\big(S(x) \& \overline{R}(x)\big)$;

б) $\forall x \exists y \forall z\big(P(x,y,z) \to Q(x,y,z) \vee \exists x\big(R(x) \sim Q(x)\big)\big)$.

Пример 8.3.1. Доказать тождественную истинность заданного предикатного выражения

$$\forall x\big(P(x) \to \big(P(x) \vee P(y)\big)\big) = \forall x\big(\overline{P}(x) \vee P(x) \vee P(y)\big) =$$

$$= \forall x\big(1 \vee P(y)\big) = \forall x(1) = 1.$$

Пример 8.3.2. Доказать тождественную ложность заданного предикатного выражения

$$\exists x\exists y\Big(\big(F(x)\to F(y)\big)\&\big(F(x)\to\overline{F}(y)\big)\&F(x)\Big)=$$

$$=\exists x\exists y\Big(\big(\overline{F}(x)\vee F(y)\big)\&\big(\overline{F}(x)\vee\overline{F}(y)\big)\&F(x)\Big)=$$

$$=\exists x\exists y\,\big(\overline{F}(x)\vee F(y)\big)\&\big(\overline{F}(x)\&F(x)\vee\overline{F}(y)\&F(x)\big)=$$

$$=\exists x\exists y\,\big(\overline{F}(x)\vee F(y)\big)\&\big(0\vee\overline{F}(y)\&F(x)\big)=$$

$$=\exists x\exists y\Big(\big(\overline{F}(x)\vee F(y)\big)\&\overline{F}(y)\&F(x)\Big)=$$

$$=\exists x\exists y\Big(\overline{F}(x)\&\overline{F}(y)\&F(x)\vee F(y)\&\overline{F}(y)\&F(x)\Big)=\exists x\exists y\big(0\vee 0\big)=0.$$

Определить, какие из приведенных формул являются тождественно истинными, а какие — тождественно ложными:

а) $\forall x\big(q\to p_1(x)\big)\sim\big(q\to\forall x p_1(x)\big)$;

б) $\forall x\big(\Phi_1(x)\to\Phi_2(x)\big)\to\big(\forall x\Phi_1(x)\to\forall x\Phi_2(x)\big)$;

в) $\forall x\big(p_1(x)\to p_2(x)\big)\sim\big(\exists x p_1(x)\to\forall x p_2(x)\big)$;

г) $\exists x R(x)\vee\exists x Q(x)\sim\exists x\big(R(x)\vee Q(x)\big)$;

д) $\forall x\big(p(x)\to\overline{Q}(x)\big)\to\overline{\forall x p(x)\&\forall x Q(x)}$;

е) $\exists x\,\Big(F(x)\to\overline{F(x)}\Big)\&\big(\overline{F}(x)\to F(x)\big)$;

ж) $\exists x\exists y\Big(\big(F(x)\to F(y)\big)\&\Big(F(x)\to\overline{F(y)}\Big)\&F(x)\Big)$.

Задача 8.3.5. Привести к нормальной форме следующие выражения:

а) $\forall x\big(p(x)\to\exists x Q(x)\big)\to\forall x\big(p(x)\to\exists y Q(y)\big)$;

б) $\forall x\Big(\exists y\big(B(x,y)\&T(y)\big)\Big)\to\exists y\big(p(y)\&R(x,y)\big)\&\overline{\exists x p(x)}\to$

$\to\forall x\big(B(x,y)\to\overline{T}(y)\big)$;

в) $(\exists x)\big(R(x)\&\forall y\big(D(y)\to L(x,y)\big)\big)\&\forall x\big(R(x)\big)\to$

$\to\forall y\Big(S(y)\to\overline{L(x,y)}\Big)\&D(x)$.

Задача 8.3.6. Представить утверждения, сформулированные на естественном языке, в виде предикатных выражений:

а) все судьи — юристы ($J(x)$ обозначает «x — судья»; $L(x)$ обозначает «x — юрист);

б) некоторые юристы — жулики ($S(x)$ обозначает «x — жулик»);

в) ни один судья не является жуликом;

г) некоторые судьи — старики, но бодрые ($O(x)$ обозначает «x — старый»; $V(x)$ обозначает «x — бодрый»);

д) судья Сидоров не стар и не бодр (константу «Сидоров» обозначает символ «j»);

е) не все юристы — судьи;

ж) некоторые юристы, являющиеся политиками,— члены Государственной думы ($P(x)$ обозначает «x — политик»; $C(x)$ обозначает «x — член Государственной думы»);

з) ни один член Государственной думы не бодр;

и) все старые члены Государственной думы — юристы;

к) некоторые женщины одновременно являются юристами и членами Государственной думы ($W(x)$ обозначает «x — женщина»);

л) ни одна женщина не является одновременно политиком и домашней хозяйкой ($H(x)$ обозначает «x — домашняя хозяйка»);

м) некоторые женщины-юристы являются домашними хозяйками;

н) все женщины-юристы восхищаются каким-нибудь судьей ($A(x, y)$ обозначает «x — восхищается y);

о) некоторые юристы восхищаются только судьями;

п) некоторые юристы восхищаются женщинами;

р) некоторые жулики не восхищаются ни одним юристом;

с) судья Сидоров не восхищается ни одним жуликом;

т) существуют как юристы, так и жулики, которые восхищаются судьей Сидоровым;

у) только судьи восхищаются судьями;

ф) все судьи восхищаются только судьями.

Пример 8.3.3. Доказать с помощью метода резолюций справедливость следующих рассуждений:

1. Некоторые руководители с уважением относятся ко всем программистам.

2. Ни один руководитель не уважает бездельников.

Следовательно, ни один программист не является бездельником.

Введем следующие предикаты:

$C(x)$ — «x — *руководитель*»;

$P(x)$ — «x — *программист*»;

$B(x)$ — «x — *бездельник*»;

$U(x, y)$ — «x *уважает* y».

Тогда посылки, записанные в виде предикатных выражений, будут выглядеть так:

а) $\exists x(C(x) \& \forall y(P(y) \to U(x, y)))$;

б) $\forall x(C(x) \to \forall y(B(y) \to \bar{U}(x,y)))$.

Заключение примет следующий вид:

$$\forall y(P(y) \to \bar{B}(y)).$$

Преобразовав посылки и следствие, которое надо взять с отрицанием, в совокупность дизъюнктов, получим множество предложений, невыполнимость которого докажем, используя метод резолюций. Имеем

$$\{C(a),\ \bar{P}(y) \vee U(x,y),\ \bar{C}(x) \vee \bar{B}(y) \vee \bar{U}(x,y), P(b),\ B(b)\}:$$

1) $C(a)$;
2) $\bar{P}(y) \vee U(x,y)$;
3) $\bar{C}(x) \vee \bar{B}(y) \vee \bar{U}(x,y)$;
4) $P(b)$;
5) $B(b)$;
6) $\bar{B}(y) \vee \bar{U}(a,y)$; (1–3) $S = \{a/x\}$;
7) $U(x,b)$; (2–4) $S = \{b/y\}$;
8) $\bar{B}(b)$; (6–7) $S = \{a/x,\ b/y\}$;
9) пустой дизъюнкт (5–8).

Задача 8.3.7. Представить утверждения, сформулированные на естественном языке в виде множества предикатных выражений. Доказать их справедливость, используя метод резолюций:

а) все мексиканцы носят сомбреро. Ни одна обезьяна не носит сомбреро. Следовательно, ни одна обезьяна не является мексиканцем;

б) каждый член демократической партии голосует за президента и не любит коммунистов. Некоторые демократы являются предпринимателями. Следовательно, существуют предприниматели, которые не любят коммунистов;

в) все отцы — мужчины. Если у детей один отец, то они единокровны. Брат — это единокровный мужчина. Джон — отец Боба. Джон — отец Сида. Сид — отец Лиз. Следовательно, Сид и Боб — братья;

г) ни один человек не является четвероногим. Все женщины — люди. Следовательно, ни одна женщина не является четвероногой;

д) некоторые республиканцы любят всех демократов. Ни один республиканец не любит ни одного социалиста. Следовательно, ни один демократ не является социалистом;

е) ни один торговец наркотиками не является наркоманом. Некоторые наркоманы привлекаются к ответственности. Следовательно, не все привлекающиеся к ответственности являются торговцами наркотиками;

ж) все рациональные числа являются действительными числами. Некоторые рациональные числа — целые числа. Следовательно, некоторые действительные числа — целые числа;

з) все первокурсники общаются со всеми второкурсниками. Ни один первокурсник не общается ни с одним студентом последнего курса. Следовательно, ни один второкурсник не является студентом последнего курса;

и) некоторым нравится группа «Queen». Некоторые не любят никого, кому нравится группа «Queen». Следовательно, некоторых любят не все;

к) каждый родитель имеет ребенка. Если родитель — женщина, то это — мать. Лиз — женщина. Анна — родитель Лиз. Следовательно, Анна — мать;

л) усли родитель — мужчина, то это отец. Если ребенок — мужчина, то это сын. Иван — мужчина. Сидор — мужчина. Иван — отец Сидора. Следовательно, Иван — отец и Сидор — сын;

м) если два человека являются родственниками третьего, то первый — родственник второго. Каждый — чей-нибудь родственник. Значит, если Иван — родственник Петра, а Петр — родственник Сидора, то Иван — родственник Сидора;

н) таможенники возвращают всех, кто въехал в страну без паспорта. Люди на машинах въехали в страну и были возвращены только другими людьми на машинах. Ни один человек на машине не имел паспорта. Следовательно, некоторые таможенники были на машинах.

Задача 8.3.8. Используя метод резолюций для предикатных выражений для заданного множества гипотез $H = \{h_1, h_2, \ldots, h_n\}$ и утверждения S, доказать справедливость выражения $H \vdash S$:

а) $h_1 = \forall x\,(K(x) \,\&\, \forall y\,(R(y) \to U(x, y)))$,
$h_2 = \forall x\,(K(x) \to \forall y\,(B(y) \to \overline{U}\,(x, y)))$,
$S = \forall y\,(R(y) \to \overline{B}\,(y))$;

б) $h_1 = \forall x\,(T(x) \to \overline{N}\,(x))$,
$h_2 = \exists x\,(N(x) \,\&\, O(x))$,
$S = \exists x\,(N(x) \,\&\, \overline{T}\,(x))$;

в) $h_1 = \forall x\,(C(x) \to W(x) \,\&\, R(x))$,
$h_2 = \exists x\,(C(x) \,\&\, O(x))$,
$S = \exists x\,(P(x) \,\&\, R(x))$;

г) $h_1 = \forall x((E(x) \,\&\, \overline{P}(x)) \to \exists y(R(x, y) \,\&\, D(y)))$,
$h_2 = \exists x((E(x) \,\&\, \underline{M}(x)) \,\&\, \forall y(R(x, y) \to M(y)))$,
$h_3 = \forall x(M(x) \,\&\, \overline{P}(x))$,

$S = \exists x(M(x) \& D(x))$;

д) $h_1 = \forall x(P(x) \to Q(x))$,
$h_2 = \forall x(Q(x) \to R(x))$,
$S = \forall x(P(x) \to R(x))$;

е) $h_1 = \forall x \forall y(F(x, y) \to M(x))$,
$h_2 = \forall x \forall y \forall w(F(x, y) \& F(x, w) \to E(y, w))$,
$h_3 = \forall x \forall y(E(x, y) \& M(x) \to B(x, y))$,
$h_4 = F(Y, H)$,
$h_5 = F(Y, D)$,
$h_6 = F(D, L)$,
$S = B(z, H)$;

ж) $h_1 = \forall z(H(z) \to \overline{Q}(z))$,
$h_2 = \forall y(W(y) \to H(z))$,
$S = \forall x(W(x) \to \overline{Q}(x))$;

з) $h_1 = \forall x \forall y(P(x) \& V(y) \to W(x, y))$,
$h_2 = \forall x \forall y(P(x) \& E(y) \to \overline{W}(x, y))$,
$S = \forall x(V(x) \to \overline{E}(x))$;

и) $h_1 = \forall x \forall y(R(x, y) \& M(x) \to O(x))$,
$h_2 = \forall x \forall y(R(x, y) \& M(y) \to C(y))$,
$h_3 = M(D)$,
$h_4 = M(B)$,
$h_5 = R(D, B)$,
$S = O(D) \& C(B)$;

к) $h_1 = \forall x(M(x) \to W(x))$,
$h_2 = \forall x(N(x) \to \overline{W}(x))$,
$S = \forall x(N(x) \to \overline{M}(x))$;

л) $h_1 = \forall x(D(x) \to \overline{L}(x, K))$,
$h_2 = \exists x(D(x) \& P(x))$,
$S = \exists x(P(x) \& \overline{L}(x, K))$;

м) $h_1 = \exists x(H(x) \& U(x, Q))$,
$h_2 = \exists x \forall y(H(x) \& H(y) \& U(x, Q) \to \overline{L}(x, y))$,
$S = \exists x \exists y(H(x) \& H(y) \& L(x, y))$;

н) $h_1 = \forall z(H(z) \& R(z) \to \overline{Q}(z))$,
$h_2 = \forall y(W(y) \to H(z))$,
$h_3 = R(A)$,
$h_4 = W(x)$,
$S = \forall x(\overline{R}(x) \& \overline{Q}(x))$.

8.4. Теория алгоритмов

Пример 8.4.1. Пусть на ленте записаны два натуральных числа в унарном коде (унарный код — это представление числа в виде соответствующей последовательности единиц, например $4 = 1111 = 1^4$). Числа разделены символом «+». Требуется реализовать операцию сложения таким образом, чтобы после ее реализации на ленте были сохранены исходные данные и записан результат выполнения операции. Управляющее устройство после завершения работы должно находиться на первом символе текущей записи на ленте.

Начальное состояние МТ, например, для чисел 2 и 4 должно иметь вид, представленный на рис. 8.2, а заключительное — на рис. 8.3.

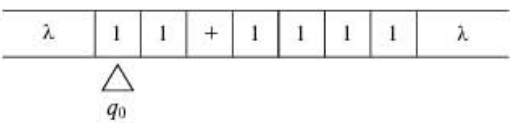

Рис. 8.2

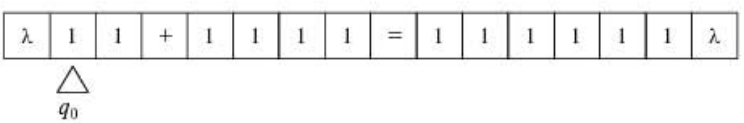

Рис. 8.3

В основе алгоритма будет лежать следующая идея: записав символ «=» после исходных данных, за ним следует скопировать все единицы из первого и второго чисел, после чего вернуть управляющее устройство в первую позицию записи на ленте. Опишем МТ в виде системы команд:

$$q_0 1 \to q_1 0\ R; \qquad q_3 1 \to q_3 1 L;$$
$$q_1 1 \to q_1 1\ R; \qquad q_3 = \to q_3 = L;$$
$$q_1 + \to q_1 + R; \qquad q_3 0 \to q_0 0 R;$$
$$q_1 \lambda \to q_2 = R; \qquad q_0 + \to q_0 + R;$$
$$q_1 = \to q_2 = R; \qquad q_0 = \to q_4 = L;$$
$$q_2 1 \to q_2 1\ R; \qquad q_4 0 \to q_4 1\ L;$$
$$q_2 \lambda \to q_3 1\ L; \qquad q_4 + \to q_4 + L;$$
$$q_3 + \to q_3 + L; \qquad q_4 \lambda \to q_z \lambda\ R.$$

Множества A и Q определены так:

$$A = \{1, +, =, \lambda, 0\} \text{ и } Q = \{q_0, q_1, q_2, q_3, q_4\}.$$

Задача 8.4.1. Построить машину Тьюринга, реализующую заданную функцию. Описание машины должно быть представлено в виде системы команд, графа переходов или таблицы. Считать, что исходные данные записаны на ленте, на ней же они должны быть сохранены после выполнения всех действий. Результаты записываются на ленте после исходных данных и отделяются от них символом «=»:

а) пусть на ленте записана последовательность нулей и единиц. Реализовать машину, которая будет подсчитывать число единиц в исходной последовательности и записывать результат счета;

б) реализовать функцию, которая в выражении в бинарной форме записи будет подсчитывать количество комбинаций вида 010, например 1101101010101 = 11;

в) реализовать функцию, которая в выражении в бинарной форме записи будет подсчитывать количество комбинаций вида 101, например 01101110001011010 = 111;

г) реализовать функцию, которая в выражении в бинарной форме записи будет производить замену комбинации 01 на 1, а 10 на 0. Считать, что такие комбинации в исходном выражении являются взаимно непересекающимися или отделены друг от друга символами, например 01111001110001 = 111011001;

д) $f(x)=\left[\frac{x}{3}\right]$;

е) $f(x)=\left[\frac{x}{2}\right]$;

ж) $f(x,y)=\begin{cases}0, \text{ если } x>y;\\ 1, \text{ если } x\le y;\end{cases}$

з) $f(x,y)=\begin{cases}1, \text{ если } x=y;\\ 0, \text{ если } x>y;\\ 01, \text{ если } x<y;\end{cases}$

и) $f(x,y,z)=\begin{cases}1, \text{ если } x\ge y;\\ z, \text{ если } x<y;\end{cases}$

к) реализовать функцию, которая в выражении в унитарной форме записи будет каждую третью единицу заменять на ноль, например 1111111 = 1101101;

л) $f(x,y)=\left[\frac{x}{y}\right]$. Символ «[]» означает выделение целой части от результата деления. Считать, что $x>y$;

м) $f(x) = 2 * x$;
н) $f(x, y) = x * y$;
о) $f(x, y) = x - y$. Считать, что $x > y$;
п) $f(y) = y^2$.

8.5. Элементы теории нечетких множеств

Задача 8.5.1. Пусть задано множество студентов C = {Иванов, Петров, Сидоров, Федоров}. Определить на нем следующие нечеткие множества: S — трудолюбивых студентов, P — ленивых студентов и U — студентов, успешно сдавших сессию. Найти результаты следующих операций над множествами: $\overline{S}$, $P \cap U$, $S \cup U$.

Задача 8.5.2. Пусть задано два множества X = {a, d, f, r} и Y = {10, 20, 15, 55}, на которых определены нечеткие множества E и T соответственно, определить отношение между E и T, если эти множества заданы следующим образом:

$$E = \frac{0}{a} + \frac{0{,}5}{d} + \frac{0{,}8}{f} + \frac{0{,}9}{r}, \quad T = \frac{0{,}2}{10} + \frac{0{,}5}{20} + \frac{0{,}7}{15} + \frac{1}{55}.$$

Задача 8.5.3. Пусть задано множество студентов C = {Иванов, Петров, Сидоров} и множество курсовых проектов K = {k_1, k_2, k_3, k_4}. Определить на множестве студентов нечеткое множество S — сильных студентов, а на K — множество P сложных курсовых проектов. Определить нечеткое отношение, которое будет соответствовать утверждению: если студент сильный, то он будет защищать сложный курсовой проект.

Задача 8.5.4. Пусть задано множество курсовых проектов K = {k_1, k_2, k_3, k_4}, на котором определено нечеткое множество P непростых курсовых проектов и задано множество D = {d_1, d_2, d_3, d_4, d_5} дипломных проектов, на котором определено нечеткое множество H хороших дипломных проектов. Определить нечеткое отношение, которое будет соответствовать утверждению: если курсовой проект непростой, то из него можно сделать хороший дипломный проект.

Задача 8.5.5. Пусть заданы два нечетких отношения U и B:

$$U = \begin{array}{c} \\ a \\ b \end{array} \begin{array}{c} \begin{array}{ccccc} 4 & 9 & 5 & 2 & 1 \end{array} \\ \begin{bmatrix} 0{,}1 & 0 & 0{,}5 & 0{,}8 & 1 \\ 0{,}9 & 0{,}5 & 0{,}6 & 0{,}4 & 0{,}9 \end{bmatrix} \end{array};$$

$$B = \begin{array}{c} \\ 4 \\ 9 \\ 5 \\ 2 \\ 1 \end{array} \begin{array}{c} \begin{array}{ccc} F & R & M \end{array} \\ \begin{bmatrix} 0{,}1 & 0 & 0{,}5 \\ 0{,}9 & 0{,}5 & 0{,}6 \\ 1 & 0{,}6 & 0{,}4 \\ 0{,}6 & 0 & 0{,}5 \\ 0{,}7 & 1 & 0{,}7 \end{bmatrix} \end{array}.$$

Определить их свертку $C = U \circ B$.

Задача 8.5.6. Используя результаты задач 8.5.3 и 8.5.4, построить свертку нечетких отношений, которая будет соответствовать утверждению: если студент сильный, то у него получится хороший дипломный проект.

8.6. Логическое программирование

8.6.1. Вопросы и задачи для самостоятельного решения

Задача 8.6.1.1. Задана программа на Прологе, описывающая группу программистов с точки зрения владения ими различными языками и средами программирования:

```
domains
    person, language = string
predicates
    program (person, language)
clauses
    program ("Сидоров", "C++").
    program ("Петров", "Pascal")
    program ("Иванов", "Basic")
    program ("Сидоров", "Visual C++").
    program ("Григоров", "Borland C++ Builder").
    program ("Васечкин", "Lisp").
    program ("Тимофеев", "Auto Lisp").
    program ("Григоров", "Vsual Prolog").
    program ("Григоров", "Prolog").
    program ("Иванов", "Visual C++").
    program ("Иванов", "Borland C++ Builder").
    program ("Тимофеев", "Lisp").
```

Сформировать цели для получения ответов на следующие запросы:

а) показать все языки, на которых программирует Сидоров;

б) показать все языки, которыми владеет данная группа программистов (при выводе результата допускаются повторения);

в) проверить, умеет ли Петров программировать на языке Prolog;

г) найти всех программистов, владеющих языком Visual C++.

Считать, что во всех случаях цель будет объявляться как внешняя.

Задача 8.6.1.2. Для запроса `d` из предыдущего примера ответить, чем будут различаться результаты, если:

а) цель будет внутренней;

б) цель будет внешней.

Задача 8.6.1.3. Для заданных сочетаний переменных `X` и `Y` определить результат операции `X=Y`, если:

а) `X = "стол", Y = "стул";`

б) `X – свободная, Y = 7.5;`

в) `X = symbolps, Y – свободная;`

г) `X = "book", Y = "book";`

д) `X = "a", Y = "a";`

е) `X = 7, Y = F/H, где F = 48, H = 8.`

Задача 8.6.1.4. Для заданных фактов из раздела `clauses` привести их описание в разделах `domains` и `predicates`:

а) `collection("Петров","Р.Джордан","Око мира", 1996);`

б) `person("Сидоров","М",1977,"ул.Пушкинская 10").`

Задача 8.6.1.5. Пусть требуется описать все книги, хранящиеся в библиотеке (по каждой указывается автор, название и год издания). Определить набор правил, которые описывают условия взятия книги в библиотеке некоторыми людьми. На естественном языке правила формулируются следующим образом:

— Сидоров возьмет нужную книгу, если она есть в библиотеке;

— Галкин возьмет нужную книгу, если она есть в библиотеке и ее не взял Сидоров;

— Иванов возьмет нужную книгу, если она есть в библиотеке и ее не взял Сидоров;

— Петров возьмет нужную книгу, если ее взял Иванов или Сидоров, но не взял Галкин.

Задача 8.6.1.6. Является ли число 52172 целым с точки зрения Пролога?

Задача 8.6.1.7. Написать программу, которая позволяет вычислить значение функции

$$f(x,y)=\begin{cases}\dfrac{3*x}{2}, \text{ если } x \ge y;\\ 2y+x, \text{ если } x < y.\end{cases}$$

Задача 8.6.1.8. Известны следующие сведения о сотрудниках предприятия: фамилия, пол, зарплата, возраст. Требуется получить информацию обо всех женщинах, чей возраст меньше заданного значения Age, а оклад выше заданной величины Oklad.

Задача 8.6.1.9. Программу «Эхо» из п. 1.7.1 переделать таким образом, чтобы не повторялись строки, содержащие цифры.

Задача 8.6.1.10. Установить, какие из приведенных списков соответствуют своему описанию, а какие нет (табл. 8.4).

Таблица 8.4

Список	Описание
`[125,555,777,9]`	`list=integer*`
`[125,55555,698,7]`	`list=integer*`
`["a","b","c","d"]`	`list=string*`
`["a","b","g","j"]`	`list=string*`
`[ [1,2,3], [abs, kkk]]`	`list1=integer*` `list=list1*`

Задача 8.6.1.11. Определить результат разделения списка на голову и остаток (табл. 8.5).

Таблица 8.5

Запрос
`L = [a|[b,c,d]].`
`L = [a,b,c,d], L2=[2|L]`
`[X|Y] = [a,b,c]`

Задача 8.6.1.12. Найти сумму всех элементов списка вещественных чисел.

Задача 8.6.1.13. Заданный список строк разделить на два списка по следующему признаку: в первый попадут строки, у которых длина больше заданного числа *N*, а во второй — все остальные. Число символов в строке определять путем посимвольного подсчета.

Задача 8.6.1.14. Сравнить два списка целых чисел на равенство по следующему правилу: два списка считаются равными, если они имеют одинаковую длину и у них одинаковые элементы стоят на одних и тех же местах. Сравнение произвести посимвольно.

Задача 8.6.1.15. Написать программу, которая будет преобразовывать заданную строку в список лексем.

Задача 8.6.1.16. Написать программу, которая будет преобразовывать заданный список строк в одну строку.

Задача 8.6.1.17. Для заданных фактов, построенных на основе составных объектов, сформировать их описание для разделов `domains` и `predicates`:

а)
```
person ("Сидоров", "Петр", "Петрович", family
  (wife ("Диана", 1977), children (boy
  ("Вася", 5), girl ("Алена", 3))));
```

б)
```
subject (odm (logic_m ("С. Клини",
  "Математическая логика", 1974), graph
  ("Н.Кристофидес", "Теория графов", 1978)),sai
  (e_system ("Д.Марселлус", "Программирование
  экспертных систем на Прологе", 1996), bz
  ("Т.А.Гаврилов", "Базы знаний", 2000)),
  program (logic ("А.Янсон", "Турбо Пролог",
  1990),
  procedur ("Т.Сван", "Программирование на C++",
  1997))).
```

Задача 8.6.1.18. Для составных объектов из задачи 8.6.1.17 построить соответствующие им структурные диаграммы.

Задача 8.6.1.19. Для задания из задачи 8.6.1.17, *б* изменить описание объекта таким образом, чтобы предикат `subject` имел только один параметр, а не три, как в исходном примере, но при этом параметр мог бы принимать любое из трех возможных (описанных ранее) значений. Использовать при решении задачи альтернативное описание предикатов.

Задача 8.6.1.20. Для составных объектов из задачи 8.6.1.17 построить цели, соответствующие следующей структуре запросов:

а) выдать полную информацию обо всех учебниках по любому предмету;

б) выдать информацию обо всех учебниках по дискретной математике;

в) выдать развернутые сведения (автор, название, год издания) обо всех учебниках по логическому программированию;

г) проверить, есть ли среди книг по теории графов книга, написанная Н. Кристофидесом;

д) выдать названия и авторов всех книг по теории графов, изданных до 1990 г.

При формировании целей предположить, что все они будут внешними, т. е. не надо заботиться об организации вывода результата и искусственной организации поиска всех возможных решений.

Задача 8.6.1.21. Из заданного файла с именем `input.dat`, содержащего строки, переписать в новый файл с именем `output.dat` только те строки, которые начинаются с гласной буквы.

Задача 8.6.1.22. Заданы три файла `f1.dat`, `f2.dat`, `f3.dat`, содержащие строковые данные. Требуется построить список списков, где содержанием списка результата будут три списка, каждый из которых соответствует содержанию одного из исходных файлов.

Задача 8.6.1.23. Программа на Прологе в разделе `clauses` содержит описание библиотеки книг, принадлежащих разным людям. Сформировать БД, в которой будут содержаться сведения только о книгах, принадлежащих Сидорову. Все предикаты должны быть представлены как составные объекты.

Задача 8.6.1.24. Из файла, представляющего собой БД «Хранилище книг», удалить все сведения о книгах, изданных до некоторого заданного года.

Задача 8.6.1.25. В файле, представляющем собой БД «Хранилище книг», исправить во всех хранящихся предикатах владельца с Сидорова на Иванова.

8.6.2. Ответы на вопросы и задачи для самостоятельного решения

Замечание 1. Большинство предлагаемых заданий имеют не единственное решение. Пролог, как и любой язык программирования, допускает в большинстве случаев множество способов реализации одной и той же задачи. Поэтому предлагаемые решения демонстрируют чаще всего один из вариантов решения, который можно рассматривать как рекомендуемый.

Замечание 2. Во многих примерах, там где параметром предиката является объект строкового типа, имеющий длину, превышающую размер экранной области, строки разделяются на две части, размещаемые в разных строках экрана. Например:

```
example ("Это невероятно длинная строка, которая
никак не может быть размещена в пределах одной экранной
строки")
```

Однако следует учитывать, что в Прологе такое разделение невозможно, так как считается ошибкой.

Ответы к 8.6.1.1:

а) `program ("Сидоров", X);`

б) `program (_, X);`

в) `program ("Петров", "Prolog");`

г) `program (X, "Visual C++").`

Ответы к 8.6.1.2:

а) ответ будет единственным — Сидоров;

б) будут найдены все возможные решения: Сидоров и Иванов.

Ответы к 8.6.1.3 представлены в табл. 8.6.

Таблица 8.6

Вариант	Исходные данные		Результат
а)	X="стол"	Y="стул"	False
б)	X— свободная	Y=7.5	X=7.5
в)	X=symbolps	Y— свободная	Ошибка
г)	X="book"	Y="book"	True
д)	X='a'	Y="a"	False
е)	X=7	Y=F/H, где F=48, H=8	True

Ответы к 8.6.1.4:

а)
```
domains
     name = string
     autor, titul = string
     year = integer
predicates
     collection (name, autor, titul, year)
clauses
     collection ("Петров", "Р.Джордан", "Око мира",
     1996).
```

б)
```
domains
     person_id = string
     sex = char
     year = integer
     adres = string
predicates
     person (person_id, sex, year, adres)
clauses
     person ("Сидоров", "М", 1977, "ул. Пушкинская
     10").
```

Ответы к 8.6.1.5. Для описания наличия книг в библиотеке можно ввести предикат `have_found (autor, name, year)`. Описание воз-

можности взятия книги будет представлено предикатом `have_book (person, name)`, где `person` — фамилия человека, взявшего книгу, а `name` — название книги:

```
domains
    autor, name, person = string
    year = integer
predicates
    have_found (autor, name, year)
    have_book (person, name)
clauses
    have_found ("Р.Джордан","Властелин Хаоса",2000).
    have_found ("Т.Сван",
         "Программирование на Borland C++",1999).
    have_found ("С.Клини", "Математическая логика",
    1978).
    have_found ("А.Егоров", "Маршалы Наполеона",
    2001).
    have_found ("А.Филд",
         "Функциональное программирование", 1996).
    have_book ("Сидоров", X) if have_found (X, _, _).
    have_book ("Галкин", X) if have_found (X, _, _),
         not (have_book ("Сидоров", X)).
    have_book ("Иванов", X) if have_found (X, _, _),
         not (have_book ("Сидоров", X)).
   have_book ("Петров", X) if have_book ("Сидоров",  X),
         not (have_book ("Галкин",X)).
    have_book ("Петров", X) if have_book ("Иванов", X),
         not (have_book ("Галкин", X)).
```

Цель, демонстрирующая возможность взятия заданной книги Сидоровым, могла бы выглядеть так:

```
goal
    have_book ("Сидоров","Математическая логика").
```

Если использовать внешнюю цель, то ответ получится положительным. Аналогичным ответ будет, если сформировать запрос и для Петрова. А вот для Галкина и Иванова положительный ответ получить не удастся, так как правила построены таким образом, что при наличии книги в библиотеке ее всегда берет Сидоров, а взятие им книги не допускает выполнение аналогичного действия для Иванова и Галкина.

Ответ к 8.6.1.6: Нет, так как интервал целых чисел для Пролога определен в границах от –32 768 до 32 767. Значит приведенное в примере число является вещественным.

Ответ к 8.6.1.7. Возможен следующий вариант построения вычисления значения функции:

```
predicates
    function (real, real, real, real)
    start
clauses
    function (X, Y, F) if X > = Y, F = 3*X/2.
    function (X, Y, F) if F = 2*Y+X.
    start if write ("Введите значения
    переменных"),nl,
         write ("X = "), readreal (X),
         write ("Y = "), readreal (Y),
    function (X, Y, Z, F),
    write ("Результат f(x, y) = ",F),
    readchar (_).
```

Ответ к 8.6.1.8. Возможный вариант решения:

```
predicates
    employee (string, char, integer, real)
    show_all (integer, real)
goal
    readint (Age), readreal (Oklad),
    write ("Женщины, соответствующие требованиям"),
         nl, show_all (Age, Oklad).
clauses
    employee ("Сидоров", 'М', 40, 5000).
    employee ("Петров", 'М', 20, 2000).
    employee ("Тихонова", 'Ж', 25, 17000).
    employee ("Иванова", 'Ж', 70, 600).
    employee ("Иванов", 'М', 75, 1600).
    employee ("Глухов", 'М', 19, 2600).
    employee ("Сидорова", 'Ж', 35, 10000).
    show_all (Age, Oklad) if employee (Name, Sex,
    Year, ZPlat),
         Sex = 'М', Year<Age, Oklad>ZPlata
         write (Name), nl, fail.
```

Ответ к 8.6.1.9. Новый вариант программы «Эхо»:

```
predicates
    repeat
    analiz (string)
    write_text
    do
    proverka (string, string)
    number (char)
```

```
clauses
    repeat.
    repeat if repeat.
    number ('0').number ('1').number ('2').number ('3').
    number ('4').number ('5').number ('6').number ('7').
    number ('8').number ('9').
    write_text if nl, write ("Введите текст, я его
    повторю"),nl,
    write ("Строки, содержащие цифры, повторяться не
    будут"),nl,
    write ("Для завершения работы введите слово
    end"),nl, nl.
    do if repeat, readln (Str), proverka (Str, Str),
    analiz (Str),!.
    analiz ("end") if nl, write ("Сеанс окончен. До
    свидания").
    analiz (_) if fail.
    proverka ("", StrIn) if write (StrIn),nl.
    proverka (S, _) if frontchar (S, Ch, _),number
    (Ch),!.
    proverka (S, StrIn) if frontchar (S, _, SRest),
        proverka (SRest, StrIn).
goal
    write_text, do.
```

Ответ к 8.6.1.10: Результаты соответствия приведены в табл. 8.7.

Таблица 8.7

Список	Описание	Результат
`[125,555,777,9]`	`list=integer*`	Соответствует
`[125,55555,698,7]`	`list=integer*`	Не соответствует, так как присутствует элемент 55555, не являющийся целым числом
`["a","b","c","d"]`	`list=string*`	Соответствует
`["a","b","g","j"]`	`list=string*`	Не соответствует, так как присутствует элемент "g", принадлежащий типу `char`
`[[1,2,3], [abs, kkk]]`	`list1=integer*` `list=list1*`	Не соответствует, так как все вложенные списки должны состоять из целых чисел, а список `[abs,kkk]` состоит из объектов типа `symbol`

Ответ к 8.6.1.11. Результаты запросов приведены в табл. 8.8.

Таблица 8.8

Запрос	Описание
`L=[a|[b, c, d]].`	`L=[a, b, c, d]`
`L=[a, b, c, d], L2=[2|L]`	`L=[a, b, c, d]` `L2=[2, a, b, c, d]`
`[X|Y] = [a, b, c]`	`X=a, Y = [b, c]`

Ответ к 8.6.1.12. Сумма элементов списка:

```
domains
    list = real*
predicates
    summa (list, real)
clauses
    summa ([ ], N) if write ("Сумма элементов = ", N).
    summa ([X|R], N) if N1 = N+X, summa (R, N1).
goal
    summa ([1.25, 5.789, 6.987, 4.3],0).
```

Ответ к 8.6.1.13: Разделение списка на два:

```
domains
    n = integer
    list = string*
predicates
    start
    divide (n, list, list, list)
    summa (string, integer, integer)
clauses
    divide (_, [ ], [ ], [ ]).
    divide (N, [H|Ost], [H|L1], L2) if
    summa (H, 0, Rez), Rez>N, divide (N, Ost, L1, L2).
    divide (N, [H|Ost], L1, [H|L2]) if
    divide (N, Ost, L1, L2).
    summa ("", N, N) if !.
    summa (Str, N, M) if frontchar (Str, _, Rest),
    N1 = N+1, summa (Rest,N1, M).
    start if write ("Введите контрольную сумму N="),
          readint (N), L = ["kkffyeh", "fghedh",
          "asderf", "gghghgh445"], divide (N, L, L1,
          L2), write ("L = ", L, "L1 = ", L1,
          "L2 = ", L2), readchar (_).
goal
start.
```

Ответ к 8.6.1.14. Сравнение списков по заданному правилу:

```
domains
list1, list2 = integer*
predicates
eql_list (list1, list2)
clauses
    eql_list (L1, L2) if L1 = [ ], L2 = [ ],
        write ("Списки равны").
    eql_list (L1, L2) if L1 = [ ], not (L2 = [ ]),
        write ("Списки неравны").
    eql_list (L1, L2) if L2 = [ ], not (L1 = [ ]),
        write ("Списки неравны").
    eql_list ([X|L1], [Y|L2]) if X = Y,
        eql_list (L1, L2).
    eql_list (_, _) if write ("Списки не равны").
```

Для различных целей результат будет отличаться; пример цели, дающей положительный ответ:

```
goal
    eql_list ([1,2,3], [1,2,3]).
```

Цели, дающие отрицательные ответы:

а)
```
goal
    eql_list ( [1,2,3], [1,4,3]);
```

б)
```
goal
    eql_list ( [1,2,3], [1,2]).
```

Ответ к 8.6.1.15. Получение списка лексем:

```
domains
    list = string*
predicates
    start
    work (string, list)
clauses
    start if write ("Введите строку для разбора->"),
          readln (St), work (St, List), nl,
          write ("Список лексем = >", List), readchar
          (_).
    work ("", [ ]) if nl, write ("Разбор окончен").
    work (St, [S|List]) if fronttoken (St, S, R),
    work (R, List).
goal
    start.
```

Ответ к 8.6.1.16. Преобразование списка строк в одну строку:

```
domains
list = string*
```

```
predicates
razbor (string, list)
clauses
razbor (Str, [ ]) if
write ("Строка результата = ",Str), readchar (_).
    razbor (Str, [X|Ostatok]) if
    concat (Str, X, New_str), razbor (New_str,
        Ostatok).
goal
    razbor ("", ["hhhdncsss", "hjejhfehj",
    "ee1145ddfvf", "hhhjjjh454gg"]).
```

Ответ к 8.6.1.17. Описания составных объектов будут иметь следующий вид:

а)
```
domains
      name_1, name_2, name_3 = string
      name = string
      year, age = integer
      family = family (wife, children)
      wife = wife (name, year)
      children = children (boy, girl)
      boy = boy (name, age)
      girl = girl (name, age)
predicates
    person (name_1, name_2, name_3, family)
clauses
    person ("Сидоров", "Петр", "Петрович", family
      (wife ("Диана", 1977), children (boy
      ("Саша", 5), girl ("Анна", 3)))).
```

б)
```
domains
      odm = odm (logic_m, graph)
      logic_m = logic_m (autor, name, year)
      graph = graph (autor, name, year)
      sai = sai (e_system, bz)
      e_system = e_system (autor, name, year)
      bz = bz (autor, name, year)
      program = program (logic, procedur)
      logic = logic (autor, name, year)
      procedur = procedur (autor, name, year)
      autor, name = string
      year = integer
predicates
    subject (odm, sai, program)
clauses
    subject (odm (logic_m ("С. Клини",
```

```
"Математическая логика", 1974), graph
("Н.Кристофидес", "Теория графов", 1978)), sai
(e_system ("Д.Марселлус", "Программирование
экспертных систем на Прологе", 1996), bz
("Т.А.Гаврилов", "Базы знаний", 2000)),
program (logic ("А.Янсон", "Турбо Пролог",
1990), procedur ("Т.Сван", "Программирование на
C++", 1997))).
```

Ответ к 8.6.1.18. Структурные диаграммы для составных объектов будут иметь вид, показанный на рис. 8.4 и 8.5:

а)

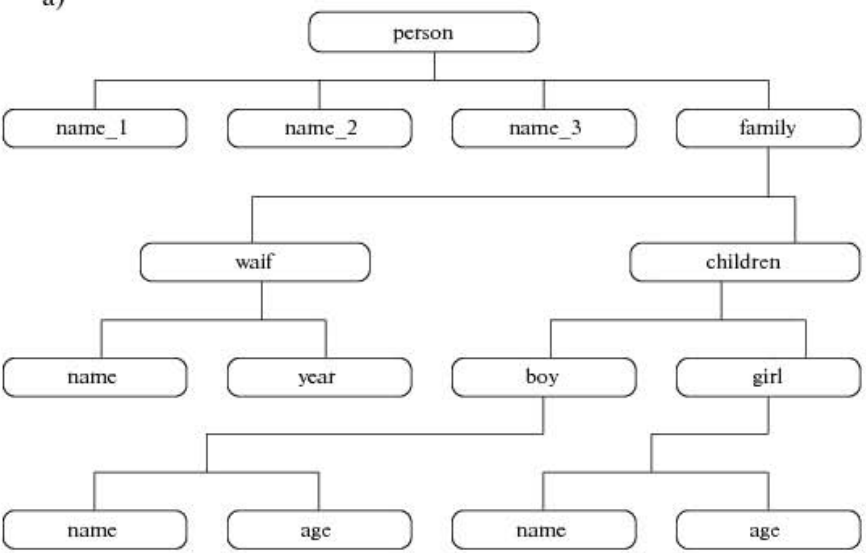

Рис. 8.4

б)

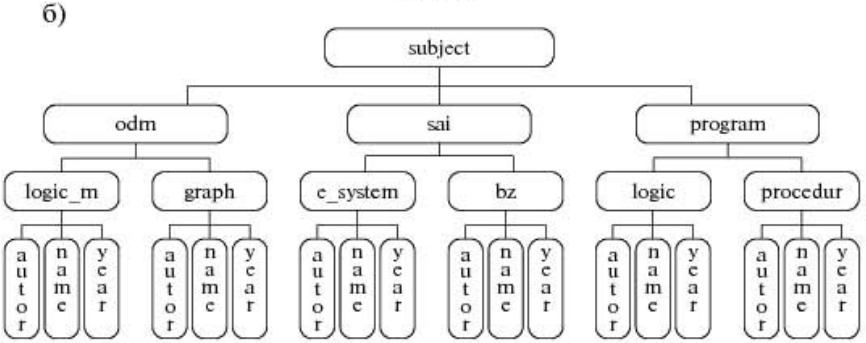

Рис. 8.5

Ответ к 8.6.1.19. Возможен следующий вариант альтернативного описания предикатов:

```
domains
    predmets = odm (type1);
    sai (type2);
    program (type3)
    type1 = logic_m (autor, name, year);
    graph (autor, name, year)
    type2 = e_system (autor, name, year);
    bz (autor, name, year)
    type3 = logic (autor, name, year);
    procedur (autor, name, year)
    autor, name = string
    year = integer
predicates
    subject (predmets)
clauses
    subject (odm (logic_m ("С.Клини",
      "Математическая логика", 1974))).
    subject (odm (graph ("Н.Кристофидес", "Теория
      графов", 1978))).
    subject (sai (e_system ("Д.Марселлус",
      "Программирование экспертных систем на
      Прологе", 1996))).
    subject (sai (bz ("Т.А.Гаврилов", "Базы
      знаний", 2000))).
    subject (program (logic ("А.Янсон", "Турбо
      Пролог", 1990))).
    subject (program (procedur ("Т.Сван",
      "Программирование на С++",1997))).
```

Ответ к 8.6.1.20. Возможны следующие варианты целей:

а) `subject (X);`

б) `subject (odm (X));`

в) `subject (program (logic (Avtor, Name, Year)));`

г) `subject (odm (graph ("Н.Кристофедес", _, _)));`

д)
```
subject (odm (graph (Avtor, Name, Year))),
    Year < = 1990.
```

Ответ к 8.6.1.21. Ввод-вывод из файла:

```
domains
    file = in; out
predicates
    rewrite
    start
```

```
    analiz (string)
    glas (char)
clauses
    glas ('a'). glas ('e'). glas ('u'). glas ('o').
    glas ('i').
    glas ('y').
    start if openread (in, "input. dat"),
    readdevice (in),
    openwrite (out, "output. dat"), writedevice (out),
    rewrite, closefile (in), closefile (out),
    readdevice (keyboard), writedevice (screen).
    rewrite if not (eof(in)), readln (S), analiz (S),
         rewrite.
    rewrite.
    analiz (S) if frontchar (S, Ch, _), glas (Ch),
    write (S), nl.
    analiz (_).
```

Ответ к 8.6.1.22. Создание списка списков:

```
domains
    file = f1; f2; f3
    object = integer*
    list = object*
predicates
    r_list (object, file)
    r_list2 (list)
    start
clauses
    start if openread (f1, "f1.dat"),
             openread (f2, "f2.dat"),
    openread (f3, "f3.dat"), r_list2 (L), closefile (f1),
    closefile (f2), closefile (f3),
    write ("Список списков = >", L).
    r_list2 ( [L1, L2, L3]) if readdevice (f1),
         r_list (L1, f1), readdevice (f2),
         r_list (L2, f2),
         readdevice (f3), r_list (L3, f3),
         readdevice (keyboard).
    r_list ( [H|L], F) if not (eof(F)), readint (H),
    r_list (L, F).
    r_list ( [ ], _).
```

Ответ к 8.6.1.23. Формирование БД:

```
domains
    book = book (string, string, integer)
database
```

```
    book (string, string, integer)
predicates
    owner (string, book)
    ask
    start
clauses
    owner ("Сидоров", book ("С.Клини",
       "Математическая логика", 1974)).
    owner ("Петров", book ("Н.Кристофидес", "Теория
       графов", 1978)).
    owner ("Тимофеев", book ("Д.Марселлус",
       "Программирование экспертных систем", 1996)).
    owner ("Сидоров", book ("Т.А.Гаврилов", "Базы
       знаний", 2000)).
    owner ("Сидоров",book ("Р.Джордан", "Властелин
       Хаоса", 1999)).
    owner ("Сидоров", book ("Г.Гаррисон",
       "Неукротимая планета", 1993)).
    owner ("Григоров",book ("А.Янсон", "Турбо
       Пролог", 1990)).
    owner ("Иванов", book ("Т.Сван",
       "Программирование на С++", 1997)).
    owner ("Сидоров", book ("Дж.Р.Р.Толкиен",
       "Хранители", 1980)).
    ask if owner ("Сидоров", book (Avtor, Title,
       Year)), assertz (book (Avtor, Title, Year)),
       fail.
    ask if !.
    start if ask, save ("sidorov. db").
goal
    start.
```

Файл БД `sidorov. db`, сформированный в результате выполнения цели, будет иметь вид:

```
book ("С.Клини", "Математическая логика", 1974)
book ("Т.А.Гаврилов", "Базы знаний", 2000)
book ("Р.Джордан", "Властелин Хаоса", 1999)
book ("Г.Гаррисон", "Неукротимая планета", 1993)
book ("Дж.Р.Р.Толкиен", "Хранители", 1980)
```

Ответ к 8.6.1.24. Удаление книг, изданных до 1990 г.:

```
database
    book (string, string, integer)
predicates
    ask (integer)
    start
```

```
clauses
    ask (Age) if book (_, _, Year), Year < Age,
        retract (book (_, _, Year)), fail.
    ask (_) if !.
    start if write ("Контрольный год = >"),
        readint (Age), consult ("book. db"), ask (Age),
        save ("book. db").
goal
    start.
```

Вид файла БД:

— до удаления фактов:

```
book ("Н.Кристофидес", "Теория графов", 1978)
book ("Д.Марселлус", "Экспертные системы", 1996)
book ("А.Янсон", "Турбо Пролог", 1990)
book ("Т.Сван", "Программирование на С++", 1997)
book ("С.Клини", "Математическая логика", 1974)
book ("Т.А.Гаврилов", "Базы знаний", 2000)
book ("Р.Джордан", "Властелин Хаоса", 1999)
book ("Г.Гаррисон", "Неукротимая планета", 1993)
book ("Дж.Р.Р.Толкиен", "Хранители", 1980);
```

— после удаления:

```
book ("Д.Марселлус", "Экспертные системы", 1996)
book ("А.Янсон", "Турбо Пролог", 1990)
book ("Т.Сван", "Программирование на С++", 1997)
book ("Т.А.Гаврилов", "Базы знаний", 2000)
book ("Р.Джордан", "Властелин Хаоса", 1999)
book ("Г.Гаррисон", "Неукротимая планета", 1993).
```

Ответ к 8.6.1.25. Изменение имени владельца книги:

```
database
    owner (string, string, string, integer)
predicates
    ask
    start
clauses
    ask if owner ("Сидоров", Avtor, Title, Year),
         retract (owner ("Сидоров", Avtor, Title,
         Year)),
    assertz (owner ("Иванов", Avtor, Title, Year)),
         fail.
    ask if !.
    start if consult ("change.db"), ask, save
    ("change.db").
goal
    start.
```

Вид файла БД:

— до замены:

```
owner ("Сидоров", "С.Клини", "Математическая
    логика", 1974).
owner ("Петров", "Н.Кристофидес", "Теория
    графов", 1978).
owner ("Тимофеев", "Д.Марселлус", "Экспертные
    системы", 1996).
owner ("Сидоров", "Т.А.Гаврилов", "Базы
    знаний", 2000).
owner ("Сидоров", "Р.Джордан", "Властелин
    Хаоса", 1999).
owner ("Сидоров", "Г.Гаррисон", "Неукротимая
    планета", 1993).
owner ("Григоров", "А.Янсон", "Турбо Пролог", 1990).
owner ("Иванов", "Т.Сван", "Программирование на
    C++", 1997).
owner ("Сидоров", "Дж.Р.Р.Толкиен", "Хранители",
    1980);
```

— после замены:

```
owner ("Петров", "Н.Кристофидес", "Теория графов",
    1978)
owner ("Тимофеев", "Д.Марселлус", "Экспертные
    системы", 1996)
owner ("Григоров", "А.Янсон", "Турбо Пролог", 1990)
owner ("Иванов", "Т.Сван", "Программирование на
    C++", 1997)
owner ("Иванов", "С.Клини", "Математическая логика",
    1974)
owner ("Иванов", "Т.А.Гаврилов", "Базы знаний",
    2000)
owner ("Иванов", "Р.Джордан", "Властелин Хаоса",
    1999)
owner ("Иванов", "Г.Гаррисон", "Неукротимая
    планета", 1993)
owner ("Иванов", "Дж.Р.Р.Толкиен", "Хранители",
    1980).
```

ПРИЛОЖЕНИЕ 1

ОСНОВНЫЕ ЛОГИЧЕСКИЕ ФУНКЦИИ

Из множества логических функций выделяется ряд наиболее простых операций, которые имеют ясную логическую интерпретацию:

1) ***отрицание*** (инверсия) $f_1(x) = \bar{x}$ (читается: не x).

Отрицание — это функция, выражающая высказывание, которое истинно, если высказывание x — ложно, и ложно, если x — истинно (табл. П.1.1);

2) ***дизъюнкция*** (логическое сложение) $f_2(x,y) = x \vee y$ (читается: x или y).

Дизъюнкция — это функция, выражающая высказывание, которое истинно тогда и только тогда, когда по крайней мере одно из высказываний — x или y — является истинным (табл. П.1.2);

3) ***конъюнкция*** (логическое умножение) $f_3(x,y) = x \wedge y$ (читается: x и y). Для этой операции применяются также следующие формы записи:

$$f_3(x, y) = xy = x\&y.$$

Конъюнкция — это функция, выражающая высказывание, которое истинно только в том случае, если истинны оба высказывания — x и y (табл. П.1.3);

4) ***импликация*** $f_4(x,y) = x \to y$ (читается: если x, то y).

Функция f_4 принимает значение ложно только тогда, когда x — истинно, а y — ложно (табл. П.1.4);

Таблица П.1.1

x	$\bar{x}$
0	1
1	0

Таблица П.1.2

x_1	x_2	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

Таблица П.1.3

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

Таблица П.1.4

x_1	x_2	$x_1 \to x_2$
0	0	1
0	1	1
1	0	0
1	1	1

5) ***эквивалентность*** (равнозначность) $f_5(x, y) = x \sim y = x \leftrightarrow y$ (читается: x равно y). Функция $f_5 = 1$ тогда и только тогда, когда значения обоих аргументов совпадают (табл. П.1.5);

6) ***сложение по модулю два*** (неравнозначность) $f_6(x,y)=x\oplus y$. Функция f_6 истинна тогда и только тогда, когда значения аргументов различны (табл. П.1.6). Как следует из табл. П.1.5 и П.1.6, функция f_6 является обратной к f_5;

7) ***штрих Шеффера*** $f_7(x,y)=x|y$. Операция, обратная по отношению к конъюнкции (функция ложна, только если оба аргумента истинны) (табл. П.1.7);

8) ***стрелка Пирса*** $f_8(x,y)=x\downarrow y$. Это функция, обратная дизъюнкции (f_8 истинно, только когда x и y ложны) (табл. П.1.8).

Таблица П.1.5

x_1	x_2	$x_1 \sim x_2$
0	0	1
0	1	0
1	0	0
1	1	1

Таблица П.1.6

x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Таблица П.1.7

x_1	x_2	$x_1 \mid x_2$
0	0	1
0	1	1
1	0	1
1	1	0

Таблица П.1.8

x_1	x_2	$x_1 \downarrow x_2$
0	0	1
0	1	0
1	0	0
1	1	0

ПРИЛОЖЕНИЕ 2

СВОЙСТВА ОСНОВНЫХ ЛОГИЧЕСКИХ ФУНКЦИЙ

Основные логические функции обладают следующими свойствами:

1) *коммутативностью:* $a \vee b = b \vee a$; $a \cdot b = b \cdot a$;
2) *ассоциативностью:* $a \vee (b \vee c) = (a \vee b) \vee c$; $a \cdot (b \cdot c) = (a \cdot b) \cdot c$;
3) *идемпотентностью конъюнкции и дизъюнкции:* $a \vee a = a$; $a \cdot a = a$;
4) *дистрибутивностью:*
   а) *конъюнкции относительно дизъюнкции:* $a \cdot (b \vee c) = (a \cdot b) \vee (a \cdot c)$;
   б) *дизъюнкции относительно конъюнкции:* $a \vee (b \cdot c) = (a \vee b) \cdot (a \vee c)$;
5) *двойного отрицания:* $\bar{\bar{a}} = a$;
6) *правилом де Моргана:* $\overline{a \cdot b} = \bar{a} \vee \bar{b}$; $\overline{a \vee b} = \bar{a} \cdot \bar{b}$;
7) *правилом склеивания:* $a \cdot b \vee a \cdot \bar{b} = a$; $(a \vee b) \cdot (a \vee \bar{b}) = a$;
8) *правилом поглощения:* $a \vee a \cdot b = a$; $a \cdot (a \vee b) = a$;
   $a \vee \bar{a} \cdot b = a \vee b$; $a \cdot (\bar{a} \vee b) = a \cdot b$;
9) *действиями с константами*:
   $a \vee 0 = a$; $a \cdot 0 = 0$;
   $a \vee 1 = 1$; $a \cdot 1 = a$;
   $a \vee \bar{a} = 1$; $a \cdot \bar{a} = 0$.

Свойства основных булевых функций доказываются либо путем преобразования выражений, либо на основе сопоставления таблиц истинности правой и левой частей равенства.

ПРИЛОЖЕНИЕ 3

ПРАВИЛА ЭКВИВАЛЕНТНЫХ ПРЕОБРАЗОВАНИЙ

Эквивалентное преобразование осуществляется на основе сопоставления таблиц истинности либо на основе применения свойств основных логических операций.

1. Функция $f_4 : a \to b = \bar{a} \vee b$. Справедливость преобразования доказывается соответствующей таблицей истинности (табл. П.3.1).

Таблица П.3.1

a	b	$\bar{a}$	$a \to b$	$\bar{a} \vee b$
0	0	1	1	1
0	1	1	1	1
1	0	0	0	0
1	1	0	1	1

2. Функция $f_5 : a \sim b = ab \vee \bar{a}\bar{b} = (a \to b)(b \to a)$.
3. Функция $f_6 : a \oplus b = \overline{a \sim b} = \overline{ab \vee \bar{a}\bar{b}} = \overline{ab} \cdot \overline{\bar{a}\bar{b}} = (\bar{a} \vee \bar{b}) \cdot (a \vee b)$.
4. Функция $f_7 : ab = \overline{\bar{a} \cdot \bar{b}}$.
5. Функция $f_8 : a \downarrow b = \overline{a \vee b}$.

ЗАКЛЮЧЕНИЕ

Содержание настоящего учебного пособия соответствует основным разделам программы по дисциплине «Математическая логика и теория алгоритмов», преподаваемой на кафедре «Программное обеспечение вычислительной техники» в Южно-Российском государственном техническом университете (Новочеркасский политехнический институт — НПИ).

Целью изучения дисциплины «Математическая логика и теория алгоритмов» является освоение студентами основных понятий логики высказываний, логики предикатов, модальной логики, теории нечетких множеств и нечеткой логики, основных моделей представления алгоритмов. Пособие содержит также раздел задач и упражнений для практических занятий по дисциплине. Особое место занимает раздел, посвященный логическому программированию, в котором наглядно показано, как основные теоретические положения математической логики применяются в области прикладного программирования.

Пособие позволяет освоить основные положения и математические методы решения задач представления знаний и построения доказательств в формальных системах, построения описания алгоритмов с использованием различных моделей, а также получить практические навыки по использованию методов математической логики и теории алгоритмов для решения практических задач и их программной реализации.

Материал настоящего пособия используется в курсах «Системы искусственного интеллекта», «Теория языков программирования», «Функциональное и логическое программирование».

Предназначено для студентов вузов, обучающихся по специальности 230105 «Программное обеспечение вычислительной техники и автоматизированных систем», а также может быть рекомендовано для студентов специальности 010503 «Математическое обеспечение и администрирование информационных систем» и специальностей направления «Информатика и вычислительная техника» дневной и заочной форм обучения.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Адаменко А., Кучуков А.* Логическое программирование и Visual Prolog. СПб. : БХВ-Петербург, 2003.
2. *Аляев Ю. А. Тюрин С. Ф.* Дискретная математика и математическая логика. М. : Академия, 2006.
3. *Братко И.* Алгоритмы искусственного интеллекта на языке PROLOG. М. : Вильямс, 2004.
4. *Воротников С. М.* Введение в математическую логику / Комсомольский-на-Амуре гос. техн. ун-т. Комсомольск-на-Амуре. 1996.
5. *Гуц А. К.* Математическая логика и теория алгоритмов : учеб. пособие. Омск : Диалог-Сибирь, 2003.
6. *Ершов Ю. Л., Палютин Е. А.* Математическая логика. М. : Наука, 1987.
7. *Иванченко А. Н., Гринченков Д. В., Потоцкий С. И.* Функциональное и логическое программирование / ЮРГТУ (НПИ). Новочеркасск, 2006.
8. *Игошин В. И.* Математическая логика и теория алгоритмов. М. : Академия, 2008.
9. *Клини С. К.* Математическая логика. М. : Мир, 1973.
10. *Кондаков Н. И.* Логический словарь-справочник. М. : Наука, 1975.
11. *Кузнецов О. П., Адельсон-Вельский Г. Н.* Дискретная математика для инженера. М. : Энергоатомиздат, 1988.
12. *Лавров И. А., Максимова Л. Л.* Задачи по теории множеств, математической логике и теории алгоритмов. М. : Наука, 1984.
13. *Лихтарников Л. М., Сукачева Т. Г.* Математическая логика : курс лекций : задачник-практикум и решения. СПб. : Лань, 1998.
14. *Лорьер Ж.-Л.* Системы искусственного интеллекта. М. : Мир, 1991.
15. *Люгер Дж. Ф.* Искусственный интеллект. Стратегии и методы решения сложных проблем. М. : Вильямс, 2003.
16. *Мелихов А. Н., Чефранов А. Г.* Применение ЭВМ для решения задач математической логики / ТРТИ. Таганрог, 1988.
17. *Мендельсон Э.* Введение в математическую логику. М. : Наука, 1984.
18. *Метакидес Г., Нероуд А.* Принципы логики и логического программирования. М. : Факториал, 1998.
19. *Новиков Ф. А.* Дискретная математика для программистов. СПб. : Питер, 2001.
20. *Потоцкий С. И., Гринченков Д. В., Гордиенко А. В.* Дискретная математика/ЮРГТУ (НПИ). Новочеркасск, 2001.
21. *Раца М. Ф.* Выразимость в исчислениях высказываний. Кишинев : Штиинца, 1991.

22. *Тейз А., Грибомон П.* Логический подход к искусственному интеллекту: от классической ложки к логическому программированию. М. : Мир, 1990.
23. *Тэрано Т., Асаи К., Сугэно М.* Прикладные нечеткие системы. М. : Мир, 1993.
24. *Шапиро С. И.* Решение логических и игровых задач. М. : Радио и связь, 1984.