

ДИСКРЕТНАЯ МАТЕМАТИКА ДЛЯ ИНЖЕНЕРА

О. П. КУЗНЕЦОВ



О. П. КУЗНЕЦОВ

ДИСКРЕТНАЯ
МАТЕМАТИКА
для
ИНЖЕНЕРА

Издание шестое,
стереотипное



ЛАННОЙ®

САНКТ-ПЕТЕРБУРГ · МОСКВА · КРАСНОДАР

2009

ББК 32.815

К 89

Кузнецов О. П.

К 89 Дискретная математика для инженера. 6-е изд., стер. — СПб: Издательство «Лань», 2009. — 400 с.: ил. — (Учебники для вузов. Специальная литература).

ISBN 978-5-8114-0570-1

В книге изложены основные понятия теории множеств, общей алгебры, логики, теории графов, теории алгоритмов и формальных систем, теории автоматов. По сравнению с изданием 1988 г. заново написаны разделы по теории графов и сложности вычислений.

Для инженеров, специализирующихся в области автоматизированного управления и проектирования, вычислительной техники, информационных технологий, передачи информации, а также для студентов и аспирантов соответствующих специальностей.

ББК 32.815

Обложка
С. ШАПИРО, А. ЛАПШИН

*Охраняется законом РФ об авторском праве.
Воспроизведение всей книги или любой ее части
запрещается без письменного разрешения издателя.
Любые попытки нарушения закона
будут преследоваться в судебном порядке.*

© Издательство «Лань», 2009

© О. П. Кузнецов, 2009

© Издательство «Лань»,
художественное оформление, 2009

ПРЕДИСЛОВИЕ К ТРЕТЬЕМУ ИЗДАНИЮ

Третье издание книги, выходящее через 15 лет после второго издания, заметно от него отличается и содержанием, и составом авторов. Главы, написанные Г. М. Адельсоном-Вельским, в настоящем издании заменены.

Глава 4 «Графы» написана заново и существенно отличается от прежней отбором материала. В частности, в нее включены матричные методы анализа графов, сведения о пространстве циклов и планарных графах. Особое внимание уделено оптимизационным задачам на графах, имеющим широкие приложения.

Глава 9 второго издания заменена новым параграфом 5.5 «Вычислительная сложность и NP-трудные задачи», который содержит современное и более компактное изложение теории NP-трудных задач. Кроме того, в главу 5 добавлены краткие сведения о различных моделях абстрактных машин.

Глава 10 «Линейное программирование» удалена, поскольку ее предмет не принято относить к дискретной математике.

Существенно обновлен список литературы. Как и в первых двух изданиях, этот список не имеет цели отразить историю предмета, приоритеты или дать его полное библиографическое описание. Его задача — указать работы, в которых вопросы, кратко упомянутые в книге, изложены более подробно.

Я глубоко благодарен Г. М. Адельсону-Вельскому за нашу совместную работу над первыми версиями книги.

Благодарю также Т. А. Таран за обширный список замечаний, поправок и уточнений ко второму изданию книги.

О. П. КУЗНЕЦОВ, Москва, октябрь 2003 г.

МНОЖЕСТВА, ФУНКЦИИ, ОТНОШЕНИЯ

1.1. МНОЖЕСТВА И ОПЕРАЦИИ НАД НИМИ

Множества и подмножества. Одними из основных, исходных понятий математики являются понятия *множества* и его *элементов*. Множество состоит из элементов. Принадлежность элемента a множеству M обозначается $a \in M$ (« a принадлежит M »); непринадлежность a множеству M обозначается $a \notin M$ или $a \bar{\in} M$.

Пример 1.1. M_1 — множество* всех натуральных чисел: 1, 2, 3... В дальнейшем будем обозначать его N ; элементы N — натуральные числа. Часто 0 также считают натуральным числом (см., например, [11]); множество, полученное добавлением 0 к N , будем обозначать N_0 .

M_2 — множество всех натуральных чисел, не превосходящих 100.

M_3 — множество всех решений уравнения $\sin x = 1$; элементы M_3 — числа, являющиеся решениями этого уравнения.

M_4 — множество всех чисел вида $\pi/2 \pm k\pi$, где $k \in N_0$.

M_5 — множество всех действительных чисел (в дальнейшем R).

M_6 — футбольная команда «Зенит» (т. е. множество ее футболистов).

M_7 — множество всех футбольных команд высшей лиги в сезоне 2004 г.

Элементами M_7 являются футбольные команды, т. е. множества типа M_6 . Таким образом, множества могут служить элементами других множеств; возможны мно-

* Обозначения M_2 — M_7 сохраняются до конца параграфа.

жества множеств (M_7), множества множеств множеств (множество всех лиг футбольного первенства) и т. д.

Отступление 1.1. Понятие множества, как и любое другое исходное понятие математической теории, не определяется. Ведь всякое определение содержит другие понятия, логически предшествующие определяемому; поэтому по крайней мере первое определение теории обязательно содержит неопределляемые понятия, которые и принимаются за исходные. В качестве исходных обычно выбираются понятия, в понимании которых не возникает существенных разногласий; более точно: различия в понимании которых не нарушают правильности ни одного положения теории. Для теорий, рассматриваемых в данной книге, понятие множества именно таково. Основные принципы построения математических теорий более подробно изложены в гл. 6.

Множество A называется *подмножеством* множества B (обозначение $A \subseteq B$; знак \subseteq называется знаком включения), если всякий элемент A является элементом B . При этом говорят, что B содержит или покрывает A . Множества A и B равны, если их элементы совпадают, иначе говоря, если $A \subseteq B$ и $B \subseteq A$. Второй вариант определения равенства множеств указывает и на наиболее типичный метод доказательства того, что данные множества равны, заключающийся в доказательстве сначала утверждения $A \subseteq B$ («в одну сторону»), а затем $B \subseteq A$ («в другую сторону»). Форму утверждения о равенстве двух множеств имеют многие математические теоремы. Пример — тригонометрическая теорема $M_3 = M_4$, состоящая из двух утверждений: а) всякое решение уравнения $\sin x = 1$ имеет вид $\pi/2 \pm k\pi$ ($M_3 \subseteq M_4$); б) всякое число вида $\pi/2 \pm k\pi$ является решением уравнения $\sin x = 1$ ($M_4 \subseteq M_3$).

Если $A \subseteq B$ и $A \neq B$, то A часто называется собственным, строгим или истинным подмножеством B (обозначение $A \subset B$; знак \subset называется знаком строгого включения).

Заметим, что в случае множества множеств возникает опасность смешения знаков \in и \subset . Например, верно $M_6 \in M_7$, но неверно $M_6 \subset M_7$ (так как M_6 и M_7 — множества разной природы!).

Множества могут быть *конечными* (т. е состоящими из конечного числа элементов) и *бесконечными*. Число

элементов в конечном множестве M называется *мощностью* M и часто обозначается $|M|$. Мощность бесконечно-го множества — более сложное понятие. Оно будет рассмотрено после введения понятия соответствия.

Множество мощности 0, т. е. не содержащее элементов, называется пустым и обозначается \emptyset . Принято считать, что пустое множество является подмножеством любого множества. Пустое множество введено в математике для удобства и единобразия языка. Например, если исследуется множество объектов, обладающих каким-либо свойством, и впоследствии выясняется, что таких объектов не существует, то гораздо удобнее сказать, что исследуемое множество пусто, чем объявлять его несуществующим. Утверждение «множество M непусто» является более компактной формулировкой равносильного ему утверждения «существуют элементы, принадлежащие M ».

Способы задания множеств. Множество может быть задано перечислением (списком своих элементов), порождающей процедурой или описанием характеристических свойств, которыми должны обладать его элементы.

Списком можно задавать лишь конечные множества. Задание типа $N = 1, 2, 3\dots$ — это не список, а условное обозначение, допустимое лишь тогда, когда оно заведомо не вызывает разночтений. Список обычно заключается в фигурные скобки. Например, $A = \{a, b, d, h\}$ означает, что множество A состоит из четырех элементов a, b, d и h .

Порождающая процедура описывает способ получения элементов множества из уже полученных элементов либо из других объектов. Элементами множества считаются все объекты, которые могут быть построены с помощью такой процедуры. Примером служит описание множества M_4 , где исходными объектами для построения являются натуральные числа, а порождающей процедурой — вычисление, описанное формулой $\pi/2 \pm k\pi$. Другой пример — множество $M_{2^n} = 1, 2, 4, 8, 16\dots$, порождающая процедура для которого определяется следующими двумя правилами: 1) $1 \in M_{2^n}$; 2) если $m \in M_{2^n}$, то $2m \in M_{2^n}$. (Правила, описанные таким образом, называются *индуктивными* или *рекурсивными*; о них будет речь в дальнейшем.) Третий пример — множество M_π , заданное следующим образом. Пусть имеется процедура

вычисления цифр разложения числа π в бесконечную десятичную дробь: $\pi = 3,1415926536\dots$ По мере вычисления будем образовывать из последовательно стоящих цифр трехразрядные числа: 314, 159, 265 и т. д. Множество всех таких чисел обозначим M_π .

Весьма распространенной порождающей процедурой является образование множеств из других множеств с помощью операций над множествами, которые будут рассмотрены ниже.

Задание множества описанием свойств его элементов, пожалуй, наиболее обычно. В примере 1.1 так заданы множества M_2, M_3, M_5 ; да и задание M_4 можно интерпретировать как описание свойства его элементов, заключающегося в возможности представить их в виде $\pi/2 \pm k\pi$. Множество M_{2^n} можно задать фразой « M_{2^n} — множество всех целых чисел, являющихся степенями двойки». В случае, когда свойство элементов M может быть описано коротким выражением $P(x)$ (означающим « x обладает свойством P »), M задается при помощи обозначения $M = \{x \mid P(x)\}$, которое читается так: « M — это множество x , обладающих свойством P ». Например, $M_{2^n} = \{x \mid x = 2^k, \text{ где } k \in N_0\}, M_4 = \{x \mid x = \pi/2 \pm k\pi, \text{ где } k \in N_0\}$. К описанию свойств естественно предъявить требование точности и недвусмысленности. Например, множество всех хороших фильмов 1985 г. разные люди зададут разными списками (быть может, пустыми); сами критерии, по которым производится отбор, при этом будут различны. Такое множество нельзя считать строго заданным. Надежным способом точно описать свойство элементов данного множества служит задание распознающей (или, как говорят в математике, разрешающей) процедуры, которая для любого объекта устанавливает, обладает он данным свойством и, следовательно, является элементом данного множества или нет. Например, для M_{2^n} , т. е. для свойства «быть степенью двойки», разрешающей процедурой может служить любой метод разложения целых чисел на простые множители.

Отметим, что в этом примере разрешающая процедура не является порождающей. Однако ее нетрудно сделать таковой: берем последовательно все натуральные числа и каждое из них разлагаем на простые множители;

те числа, которые не содержат множителей, отличных от 2, включаем в M_{2^n} . С другой стороны, порождающая процедура может не быть разрешающей. В этой связи предлагаем читателю поразмыслить над множеством M_π , однако предостережем его от поспешных заключений. К этому множеству мы еще вернемся.

К какому виду принадлежит задание множества M_6 ? Ни к какому: M_6 по существу не задано, а только названо. Задать его можно, например, списком или следующим описанием: « M_6 — множество всех лиц, имеющих удостоверения футбольного клуба «Зенит». Разрешающая процедура для такого описания связана, как легко понять, с проверкой документов.

Отступление 1.2. Рассмотрение способов задания множеств приводит к мысли о том, что само понятие «точно задать множество» нуждается в уточнении. Такое уточнение совсем не просто, а его важность крайне велика и выходит далеко за пределы самой теории множеств. Язык множеств — это универсальный язык математики. Любое математическое утверждение можно сформулировать как утверждение о некотором соотношении между множествами: о равенстве двух множеств (см. ранее $M_3 = M_4$), о непустоте некоторого множества («существует непрерывная нигде не дифференцируемая функция»), о непринадлежности элемента множеству («с помощью циркуля и линейки нельзя построить круг, равновеликий данному квадрату») и т. д. Поэтому анализ способов задания множеств связан с анализом строгости математических утверждений вообще, т. е. с обсуждением самих оснований математики.

В чем трудности вопроса о задании множеств? Одна из основных трудностей заключается в том, что даже из множеств, точность описания которых не вызывает сомнений, с помощью, казалось бы, вполне законных средств можно сконструировать описания множеств, которые приводят к противоречиям — «парадоксам теории множеств», хорошо известным в истории математики. Примером является «множество всех множеств». По смыслу своего описания оно должно содержать все мыслимые множества. Однако оно само содержится в множестве своих подмножеств в качестве элемента. Более точный комментарий этого примера должен опираться на понятие мощности бесконечного множества и будет дан позднее (см. теорему Кантора).

Анализ возникших трудностей привел в первой трети XX в. к бурному развитию области математики, получившей название оснований математики, или метаматематики.

Некоторое представление об этой области будет дано в гл. 5 и 6. Здесь укажем лишь, что одной из ее основных задач является разработка средств задания математических объектов вообще и множеств в частности, которые решали бы все проблемы, связанные с точностью задания, и устранили бы возможные парадоксы.

Таким образом, в первичной простоте понятия «множество», которая декларировалась в начале главы, при более внимательном рассмотрении не оказывается ни простоты, ни первичности. Однако сказанное там остается в силе, и понятие множества будет по-прежнему считаться исходным ввиду сделанной в отступлении 1.1 оговорки, которую здесь сформулируем следующим образом: для тех теорий, использующих понятие множества, которые будут рассматриваться в дальнейшем, знать все сложности, связанные с заданием множеств, не обязательно (за исключением гл. 5 и 6); достаточно зафиксировать некоторые конкретные средства их задания. Со своей стороны обещаем, что эти средства всегда будут конструктивными и не будут вызывать неясностей в их толковании.

Операции над множествами. *Объединением* множеств A и B (обозначение $A \cup B$) называется множество, состоящее из всех тех элементов, которые принадлежат хотя бы одному из множеств A , B . Символически это можно записать так:

$$A \cup B = \{x | x \in A \text{ или } x \in B\}.$$

Аналогично определяется объединение произвольной (в том числе бесконечной) совокупности множеств. Если совокупность содержит небольшое количество множеств, то их объединение описывается явно: $A \cup B \cup C \cup D$ и т. д. В общем случае используется обозначение $\bigcup_{A \in S} A$, которое читается так: «объединение всех множеств A , принадлежащих совокупности S ». Если же все множества совокупности занумерованы индексами, то используются другие варианты обозначений: $\bigcup_{i=1}^k A_i$ (для случая, когда $S = \{A_1, A_2, \dots, A_k\}$), $\bigcup_{i=1}^{\infty} A_i$ (для случая, когда S — бесконечная совокупность и ее множества занумерованы подряд натуральными числами), $\bigcup_{i \in I} A_i$ (для случая, когда набор индексов множеств задан множеством I).

Пример 1.2. а. $A = \{a, b, d\}$, $B = \{b, d, e, h\}$, $A \cup B = \{a, b, d, e, h\}$.

б. $M_3 \cup M_4 = M_3 = M_4$ (так как M_3 и M_4 равны).

в. Обозначим футбольные команды высшей лиги через Φ_i : $M_7 = \{\Phi_1, \Phi_2, \dots, \Phi_{16}\}$. Тогда $\bigcup_{i=1}^{16} \Phi_i$ — множество всех футболистов (но не команд!) высшей лиги. Обобщая последнее замечание, отметим, что всегда $A \subseteq (A \cup B)$, но неверно $A \in (A \cup B)$.

г. Обозначим через N_k множество всех натуральных чисел, делящихся на k и не равных k , а через P — множество всех простых чисел (принято считать, что $1 \notin P$).

Тогда $\bigcup_{i \in P} N_i$ — множество всех составных, т. е. непростых, чисел.

Пересечением множеств A и B (обозначение $A \cap B$) называется множество, состоящее из всех тех и только тех элементов, которые принадлежат и A , и B :

$$A \cap B = \{x \mid x \in A \text{ и } x \in B\}.$$

Аналогично определяется пересечение произвольной (в том числе бесконечной) совокупности множеств. Обозначения для пересечения системы множеств аналогичны приведенным выше обозначениям для объединения.

Пример 1.3. а. $A = \{a, b, d\}$, $B = \{b, d, e, h\}$, $A \cap B = \{b, d\}$.

б. $M_3 \cap M_4 = M_3 = M_4$.

в. $\bigcap_{i=1}^{16} \Phi_i = \emptyset$; более того, для любых i и j $\Phi_i \cap \Phi_j = \emptyset$.

Заметим, что в общем случае из первого свойства не следует второе: например, если $A = \{a, b\}$, $B = \{b, c\}$, $C = \{a, c\}$, то $A \cap B \cap C$ пусто, однако все попарные пересечения непусты. Система множеств, в которой все попарные пересечения множеств пусты, называется *разбиением* множества U всех элементов этих множеств, а множества такой системы называются классами или блоками разбиения. Всякий элемент U входит в один и только в один класс разбиения. Например, M_7 является разбиением множества всех футболистов высшей лиги; классы этого разбиения — команды; всякий футболист из множества $\bigcup_{i=1}^{16} \Phi_i$ может входить только в одну команду. Подробнее о разбиениях см. §§ 1.3 и 2.2.

г. $\bigcap_{i \in P} N_i = \emptyset$ (обозначения те же, что и в п. «г» примера 1.2), так как элемент такого множества должен делиться на все простые числа; ввиду бесконечности множества простых чисел это невозможно.

Разностью множеств A и B (обозначение $A \setminus B$) называется множество всех тех и только тех элементов A , которые не содержатся в B :

$$A \setminus B = \{x \mid x \in A \text{ и } x \notin B\}.$$

В отличие от двух предыдущих операций разность, во-первых, строго двухместна (т. е. определена только для двух множеств), а во-вторых, некоммутативна: $A \setminus B \neq B \setminus A$. Если $A \setminus B = \emptyset$, то $A \subseteq B$.

Симметрической разностью множеств A и B (обозначение $A \Delta B$) называется множество всех тех и только тех элементов A и B , которые содержатся только в одном из этих множеств (т. е. не содержатся в их пересечении):

$$A \Delta B = \{x \mid x \in (A \cup B) \setminus (A \cap B)\}.$$

Пример 1.4. а. $A = \{a, b, d\}$, $B = \{b, d, e, h\}$, $A \setminus B = \{a\}$, $B \setminus A = \{e, h\}$, $A \Delta B = \{a, e, h\}$.

б. $M_3 \setminus M_4 = M_4 \setminus M_3 = \emptyset$.

в. $M_7 \setminus M_6$ — множество всех команд высшей лиги, за исключением «Зенита». Эта запись не вызывает разнотечений, однако, строго говоря, она неточна: из M_7 вычитается не множество M_6 футболистов (это бессмысленно, так как M_6 и M_7 имеют элементы разной природы), а одноэлементное множество $\{M_6\}$ команд. Правильная запись $M_7 \setminus \{M_6\}$. Аналогично этому для множества $A = \{a, b, d\}$ запись $A \setminus a$ неверна, а запись $A \setminus \{a\}$ верна. Напротив, в разности $\bigcup_{i=1}^{16} \Phi_i \setminus M_6$ (множество всех футболистов высшей лиги, не выступающих в «Зените») M_6 является именно множеством футболистов и заключать его в фигурные скобки не следует. В случаях, когда одновременно рассматриваются множества и множества множеств, соблюдение подобных тонкостей не является пустым педантизмом, а предохраняет от возможной путаницы.

Если все рассматриваемые множества являются подмножеством некоторого «универсального» множества U ,

то может быть определена операция дополнения. *Дополнением* (до U) множества A (обозначение \bar{A}) называется множество всех элементов, не принадлежащих A (но принадлежащих U): $\bar{A} = U \setminus A$. Множество U должно быть либо задано, либо очевидно из контекста, в противном случае проще пользоваться выражением $U \setminus A$. Например, из определения \bar{M}_2 очевидно, что M_2 — это множество натуральных чисел, больших 100. Запись же \bar{N} без контекста, т. е. без указания U , неясна — то ли это множество отрицательных целых чисел, то ли множество положительных дробных чисел, то ли пустое множество натуральных чисел.

Операции объединения, пересечения и дополнения часто называют булевыми операциями над множествами. Позднее (в гл. 2 и 3) будет пояснен смысл этого названия и рассмотрены соотношения между этими операциями.

Векторы и прямые произведения. *Вектор* — это упорядоченный набор элементов. Сказанное не следует считать определением вектора, поскольку тогда потребуется давать объяснения по поводу его синонима «упорядоченный набор». Понятие «вектор» (другой синоним — «кортеж») будем считать, как и понятие множества, неопределенным. Элементы, образующие вектор, называются координатами или компонентами вектора. Координаты нумеруются слева направо. Число координат называется длиной или размерностью вектора. Бесконечные векторы рассматриваться не будут. В отличие от элементов множества координаты вектора могут совпадать. Вектор будем заключать в круглые скобки, например $(0, 5, 4, 5)$. Иногда скобки и даже запятые опускаются. Векторы длины 2 часто называются упорядоченными парами (или просто парами), векторы длины 3 — тройками и т. д. Вектор длины n иногда называют n -кой («энкой»).

Два вектора равны, если они имеют одинаковую длину и соответствующие координаты их равны. Иначе говоря, векторы (a_1, \dots, a_m) и (b_1, \dots, b_n) равны, если $m = n$ и $a_1 = b_1, a_2 = b_2, \dots, a_m = b_m$.

Прямым произведением множеств A и B (обозначение $A \times B$) называется множество всех пар (a, b) , таких, что $a \in A, b \in B$. В частности, если $A = B$, то обе координаты

принадлежат A . Такое произведение обозначается A^2 . Аналогично прямым произведением множеств A_1, \dots, A_n (обозначение $A_1 \times \dots \times A_n$) называется множество всех векторов (a_1, \dots, a_n) длины n , таких, что $a_1 \in A_1, \dots, a_n \in A_n$. $A \times \dots \times A$ обозначается A^n .

Пример 1.5. а. Множество $R \times R = R^2$ — это множество точек плоскости, точнее, пар вида (a, b) , где $a, b \in R$ и являются координатами точек плоскости.

Координатное представление точек плоскости, предложенное французским математиком и философом Декартом, исторически первый пример прямого произведения. Поэтому иногда прямое произведение называют декартовым.

б. $A = \{a, b, c, d, e, f, g, h\}$, $B = \{1, 2, \dots, 8\}$. Тогда $A \times B = \{a1, a2, a3, \dots, h7, h8\}$ — множество, содержащее обозначения всех 64 клеток шахматной доски.

в. Рассмотрим множество числовых матриц 3×4 , т. е. матриц вида

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{vmatrix},$$

где a_{ij} принадлежит множеству R действительных чисел. Строки матрицы — это элементы множества R_4 (векторы длины 4). Сама матрица, рассматриваемая как упорядоченный набор (т. е. вектор) строк, — это элемент множества $(R^4)^3 = R^4 \times R^4 \times R^4$. Компоненты матрицы, заданной таким образом, — строки, а не числа. Поэтому $(R^4)^3 \neq R^{12}$. Содержательный смысл этого неравенства в том, что в векторе из R^{12} не содержится никакой информации о строении матрицы; тот же вектор из R^{12} мог бы перечислять элементы матрицы 4×3 или 2×6 , которые как математические объекты вовсе не совпадают с матрицами 3×4 .

Приведенный пример показывает, в частности, что компонентами векторов могут быть также векторы.

г. Пусть A — конечное множество, элементами которого являются символы (буквы, цифры, знаки препинания, знаки операций и т. д.). Такие множества обычно называют *алфавитами*. Элементы множества A^n называются

словами длины n в алфавите A . Множество всех слов в алфавите A — это множество

$$A^* = \bigcup_{i \in N} A^i = A^1 \cup A^2 \cup A^3 \cup \dots$$

При написании слов (которые по нашему определению являются векторами) не принято пользоваться ни запятыми, ни скобками как разделителями; но они могут оказаться символами самого алфавита. Поэтому слово в алфавите A — это просто конечная последовательность символов алфавита A . Например, десятичное целое число — это слово в алфавите цифр $\{0, 1, \dots, 9\}$. Текст, напечатанный на пишущей машинке, является словом в алфавите, определяемом клавиатурой данной машинки (включая знаки препинания и пробел!).

Теорема 1.1. Пусть A_1, A_2, \dots, A_n — конечные множества и $|A_1| = m_1, |A_2| = m_2, \dots, |A_n| = m_n$. Тогда мощность множества $A_1 \times A_2 \times \dots \times A_n$ равна произведению мощностей A_1, A_2, \dots, A_n :

$$|A_1 \times A_2 \times \dots \times A_n| = m_1 m_2 \dots m_n.$$

Эту теорему докажем методом математической индукции. Для $n = 1$ теорема тривиально верна. Предположим, что она верна для $n = k$, и докажем ее справедливость для $n = k + 1$. По предположению, $|A_1 \times A_2 \times \dots \times A_k| = m_1 m_2, \dots, m_k$. Возьмем любой вектор (a_1, \dots, a_k) из $A_1 \times A_2 \times \dots \times A_k$ и припишем справа элемент $a_{k+1} \in A_{k+1}$. Это можно сделать m_{k+1} разными способами; при этом получится m_{k+1} различных векторов из $A_1 \times A_2 \times \dots \times A_{k+1}$. Таким образом, из всех $m_1 \dots m_k$ векторов приписыванием справа элемента из A_{k+1} можно получить $m_1 \dots m_k m_{k+1}$ векторов из $A_1 \times A_2 \times \dots \times A_{k+1}$, причем все они различны, и никаких других векторов в $A_1 \times A_2 \times \dots \times A_{k+1}$ не содержится. Поэтому для $n = k + 1$ теорема верна и, следовательно, верна для любых n . \square^*

Следствие. $|A^n| = |A|^n$.

Эта простая теорема и ее следствие лежат в основе очень многих комбинаторных фактов.

* Знак \square будет обозначать конец доказательства теоремы, т. е. заменять оборот «что и требовалось доказать». Если доказательство опущено или следует из предшествующего текста, то знак \square будет ставиться непосредственно после формулировки теоремы.

Проекции. Проекцией вектора v на i -ю ось (обозначение $\text{пр}_i v$) называется его i -я компонента. Проекцией вектора $v = (a_1, \dots, a_n)$ на оси с номерами i_1, \dots, i_k называется вектор $(a_{i_1}, \dots, a_{i_k})$ длины k (обозначение $\text{пр}_{i_1, \dots, i_k} v$).

Пусть V — множество векторов одинаковой длины. Тогда проекцией множества V на i -ю ось называется множество проекций всех векторов из V на i -ю ось: $\text{пр}_i V = \{\text{пр}_i v \mid v \in V\}$. Аналогично определяется проекция множества V на несколько осей: $\text{пр}_{i_1, \dots, i_k} V = \{\text{пр}_{i_1, \dots, i_k} v \mid v \in V\}$. В частности, если $V = A_1 \times A_2 \times \dots \times A_n$, то $\text{пр}_{i_1, \dots, i_k} V = A_{i_1} \times \dots \times A_{i_k}$. Отметим, что в общем случае $\text{пр}_i V$ — вовсе не обязательно прямое произведение; оно может быть его подмножеством.

Пример 1.6. а. Проекция точки плоскости на 1-ю ось — это ее абсцисса (первая координата); проекция на 2-ю ось — ордината.

б. $V = \{(a, b, d), (c, b, d), (d, b, b)\}$, $\text{пр}_1 V = \{a, c, d\}$, $\text{пр}_2 V = \{b\}$, $\text{пр}_{2,3} V = \{(b, d), (b, b)\}$.

1.2. СООТВЕТСТВИЯ И ФУНКЦИИ

Соответствия. Соответствием между множествами A и B называется подмножество $G \subseteq A \times B$.

Если $(a, b) \in G$, то говорят, что b соответствует a при соответствии G . Множество $\text{пр}_1 G$ называется *областью определения* соответствия, множество $\text{пр}_2 G$ — *областью значений* соответствия. Если $\text{пр}_1 G = A$, то соответствие называется *всюду определенным* или полностью определенным (в противном случае соответствие называется частичным); если $\text{пр}_2 G = B$, то соответствие называется *сюръективным* (сюръекцией).

Множество всех $b \in B$, соответствующих элементу $a \in A$, называется *образом* a в B при соответствии G . Множество всех a , которым соответствует b , называется *прообразом* b в A при соответствии G . Если $G \subseteq \text{пр}_1 G$, то образом множества C называется объединение образов всех элементов C . Аналогично определяется прообраз множества D для любого $D \subseteq \text{пр}_2 G$.

Соответствие G называется *инъективным* (инъекцией), если прообразом любого элемента из $\text{пр}_2 G$ является единственный элемент из $\text{пр}_1 G = A$. Соответствие G

называется *функциональным* (или *однозначным*), если образом любого элемента из $\text{пр}_1 G$ является единственный элемент из $\text{пр}_2 G$. Соответствие G между A и B называется *взаимно однозначным*, или *биекцией* (иногда пишут «*1–1-соответствие*»), если оно всюду определено, сюръективно, функционально и инъектививно.

Пример 1.7. а. Круг G радиуса 1 с центром в точке $(3, 2)$ (рис. 1.1), т. е. множество пар действительных чисел (x, y) , удовлетворяющих соотношению $(x - 3)^2 + (y - 2)^2 \leq 1$, задает соответствие между R и R (осью абсцисс и осью ординат). Образом числа 4 при этом соответствии является единственное число 2, образом числа 3 является отрезок $[1, 3]$ оси ординат; этот же отрезок $[1, 3]$ является образом отрезка $[2, 4]$ оси абсцисс, который, в свою очередь, служит прообразом числа 2. Данное соответствие не является функциональным. Примером функционального соответствия между действительными числами на том же рис. 1.1 служит дуга ABC .

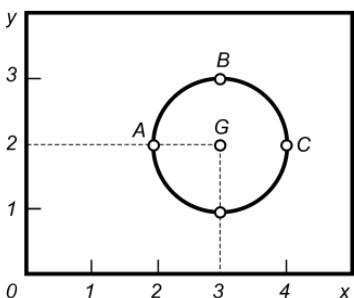


Рис. 1.1

Еще раз напомним, что для задания соответствия надо указать не только множество G , но и множества A и B , т. е. указать, подмножеством какого прямого произведения является G . В данном примере тот же круг G_1 задает и другое соответствие: между отрезком $[2, 4]$ и отрезком $[1, 3]$. При этом по некоторым свойствам соответствия $G_1 \subseteq R^2$ и $G_1 \subseteq [2, 4] \times [1, 3]$ отличаются: например, второе соответствие в отличие от первого всюду определено и сюръективно. Учитывая эти соотношения, следовало бы определять соответствие как тройку множеств (G, A, B) . Тогда не пришлось бы оговариваться, что один круг может задавать два соответствия; это и так было бы ясно из различия троек (G_1, R, R) и $(G_1, [2, 4], [1, 3])$. Однако такие оговорки приходится делать редко: либо множества A и B ясны из контекста, либо различия в их выборе не влияют на исследуемые свойства соответствия. Поэтому «определение через тройку множеств» здесь использоваться не будет.

б. Англо-русский словарь устанавливает соответствие между множеством английских и русских слов. Это соответствие не является функциональным (так как одному английскому слову, как правило, ставится в соответствие несколько русских слов); кроме того, оно практически никогда не является полностью определенным: всегда можно найти английское слово, не содержащееся в данном словаре*.

в. Позиция на шахматной доске представляет собой взаимно однозначное соответствие между множеством оставшихся на доске фигур и множеством занятых ими полей.

г. Различные виды кодирования — кодирование букв азбукой Морзе, представления чисел в различных системах счисления, секретные шифры, входящие и исходящие номера в деловой переписке и др. — являются соответствиями между кодируемыми объектами и присваиваемыми им кодами. Эти соответствия, как правило, обладают всеми свойствами взаимно однозначного соответствия, кроме, быть может, одного — сюръективности. Единственность образа и прообраза в кодировании гарантирует однозначность шифровки и дешифровки. Отсутствие сюръективности означает, что не всякий код имеет смысл, т. е. соответствует какому-либо объекту. Например, кодирование телефонов Москвы семизначными номерами не сюръективно, так как некоторые семизначные номера не соответствуют никаким телефонам.

д. Множество векторов вида $(n, 2^{n-1})$, где $n \in N$, задает взаимно однозначное соответствие между множеством N натуральных чисел и множеством M_{2^n} степеней двойки.

Взаимно однозначные соответствия и мощности множеств. Если между конечными множествами A и B существует взаимно однозначное соответствие, то $|A| = |B|$. Действительно, если это не так, то либо $|A| > |B|$, и тогда, поскольку отображение всюду определено, в A найдутся два элемента, которым соответствует один и тот же элемент $b \in B$ (нарушится инъективность), либо $|A| < |B|$, и тогда, поскольку отображение сюръективно, в B найдутся

* При этом остается в стороне вопрос (вообще говоря, законный), является ли множество английских слов (так же, как и русских) точно заданным множеством.

два элемента, соответствующих одному и тому же $a \in A$ (нарушится функциональность)*.

Этот факт, во-первых, позволяет установить равенство мощностей двух множеств, не вычисляя этих мощностей, а во-вторых, часто дает возможность вычислить мощность множества, установив его взаимно однозначное соответствие с множеством, мощность которого известна или легко вычисляется. В качестве иллюстрации этого приема докажем важную теорему о числе подмножеств конечного множества.

Теорема 1.2. Если для конечного множества A $|A| = n$, то число всех подмножеств A равно 2^n , т. е. $2^{|A|}$.

Занумеруем элементы A номерами от 1 до n : $A = \{a_1, a_2, \dots, a_n\}$ и рассмотрим множество B_n двоичных векторов из нулей и единиц длины n . Каждому подмножеству $A^* \subseteq A$ поставим в соответствие вектор $v = (v_1, v_2, \dots, v_n) \in B_n$ следующим образом:

$$v_i = \begin{cases} 0, & \text{если } a_i \notin A^*; \\ 1, & \text{если } a_i \in A^*. \end{cases}$$

Например, если $A = \{a, b, c, d, e\}$, то подмножеству $\{a, c, d\}$ соответствует вектор $(1, 0, 1, 1, 0)$, а подмножеству $\{b\}$ соответствует вектор $(0, 1, 0, 0, 0)$. Пустому подмножеству любого A соответствует вектор из одних нулей, а самому A — вектор из одних единиц. Очевидно, что установленное соответствие между множеством всех подмножеств A и двоичными векторами длины n является взаимно однозначным и число подмножеств A равно $|B_n|$. А так как B_n является прямым произведением n двухэлементных множеств $\{0, 1\}$, то в силу следствия из теоремы 1.1 $|B_n| = 2^n$. \square

Множества *равномощны*, если между их элементами можно установить взаимно однозначное соответствие. Для конечных множеств это утверждение доказывается, что и было сделано ранее. Для бесконечных множеств оно является определением равномощности. Множества, равномощные N , называются *счетными*. Соответствие, установленное в примере 1.7, д, показывает, что множе-

* Обращаем внимание читателя на то, что в этом простом рассуждении оказываются существенными все четыре свойства взаимно однозначного соответствия.

ство M_{2^n} счетно. Вообще любое бесконечное подмножество N счетно. Действительно, пусть $N' \subset N$. Выберем в N' наименьший элемент и обозначим его n_1 ; в $N' \setminus \{n_1\}$ выберем наименьший элемент и обозначим его n_2 ; наименьший элемент $N' \setminus \{n_1, n_2\}$ обозначим n_3 и т. д. Поскольку для всякого натурального числа имеется лишь конечное множество меньших натуральных чисел, то любой элемент N' рано или поздно получит свой номер. Эта нумерация, т. е. соответствие (n_i, i) , и есть взаимно однозначное соответствие между N' и N .

Множество N^2 счетно. Нумерацию N^2 можно устроить следующим образом. Разобьем N^2 на классы. К первому классу N_1^2 отнесем все пары чисел с минимальной суммой. Такая пара всегда одна: $(1, 1)$. Ко второму классу N_2^2 отнесем все пары чисел с суммой 3: $N_2^2 = \{(1, 2), (2, 1)\}$. В общем случае $N_i^2 = \{(a, b) | a + b = i + 1\}$. Каждый класс N_i^2 содержит ровно i пар. Упорядочим теперь классы по возрастанию индексов i , а пары внутри класса — по возрастанию первого элемента и занумеруем получившуюся последовательность пар номерами 1, 2, 3... Легко видеть, что если $a + b = i + 1$, то пара (a, b) получит номер $1 + 2 + \dots + (i - 1) + a$. Эта нумерация и доказывает счетность N^2 , из которой, в свою очередь, непосредственно следует счетность множества Q^+ положительных рациональных чисел, т. е. дробей вида a/b , где a и b — натуральные числа*. Аналогично доказывается счетность N^3 и вообще N^k для любого натурального k .

Нетрудно понять, что объединение конечного числа счетных множеств M_1, M_2, \dots, M_k счетно. Действительно, перенумеруем сначала все первые элементы множеств, затем все вторые и т. д. Объединение счетного множества конечных множеств также счетно (сначала нумеруем все элементы первого множества, затем все элементы второго множества и т. д.). Из последнего утверждения следует, что множество всех слов в любом конечном

* На примере множества Q^+ видно, что нумерация числового множества может не иметь ничего общего с упорядочением его элементов по величине. В множестве Q^+ нет ни наименьшего элемента, ни двух соседних по величине элементов (для любых двух дробей p_1 и p_2 всегда найдется дробь, лежащая между ними, например $(p_1 + p_2)/2$), однако есть элемент с наименьшим номером и элементы с соседними номерами.

алфавите счетно. Менее очевидно, что счетно и объединение счетного множества счетных множеств. Примером такого объединения является множество $\bigcup_{i \in N} N^i$ всех векторов с натуральными компонентами.

Теорема 1.3 (Кантор). Множество всех действительных чисел интервала $(0, 1)$ не является счетным.

Действительно, предположим, что оно счетно и существует его нумерация. Расположим все числа, изображенные бесконечными десятичными дробями, в порядке этой нумерации:

$$0, a_{11}a_{12}a_{13}\dots$$

$$0, a_{21}a_{22}a_{23}\dots$$

$$0, a_{31}a_{32}a_{33}\dots$$

.....

Рассмотрим любую бесконечную десятичную дробь $0, b_1b_2b_3\dots$, такую, что $b_1 \neq a_{11}$, $b_2 \neq a_{22}$, $b_3 \neq a_{33}$ и т. д. Эта дробь не может войти в указанную последовательность, так как от первого числа она отличается первой цифрой, от второго числа — второй цифрой и т. д. Следовательно, все числа из интервала $(0, 1)$ не могут быть пронумерованы, и множество всех действительных числе интервала $(0, 1)$ *несчетно*. Его мощность называется *континуум*; множества такой мощности называются *континуальными*. Метод, использованный при доказательстве, называется *диагональным методом* Кантора. \square

Множество R всех действительных чисел континуально. Множество всех подмножеств счетного множества также континуально. Это становится ясным, если воспользоваться, как и в теореме 1.2, представлением подмножества в виде последовательности (но теперь уже бесконечной!) нулей и единиц: на i -м месте стоит 1, если i -й элемент множества входит в данное подмножество, и 0 в противном случае. Получаем взаимно однозначное соответствие между подмножествами счетного множества и правильными двоичными дробями, которые, в свою очередь, взаимно однозначно соответствуют множеству чисел интервала $(0, 1)$. Как показывается в теории множеств (с помощью метода, аналогичного диагональному), для множества любой мощности множество его подмножеств имеет более высокую мощность. Поэтому

не существует множества максимальной мощности. Парадокс, приведенный в отступлении 1.2 (парадокс Кантора), как раз и заключается в том, что «множество всех множеств» должно содержать все множества и, следовательно, иметь максимальную мощность, что противоречит результатам теории множеств.

Отображения и функции. *Функцией* называется функциональное соответствие. Если функция f устанавливает соответствие между множествами A и B , то говорят, что функция f имеет тип $A \rightarrow B$ (обозначение $f: A \rightarrow B$). Каждому элементу a из своей области определения функция f ставит в соответствие единственный элемент b из области значений. Это обозначается хорошо известной записью $f(a) = b$. Иногда, если это не вызывает неудобств, используют обозначения fa или af . Элемент a называется *аргументом* функции, b — *значением* функции на a . Полностью определенная функция $f: A \rightarrow B$ называется *отображением* A в B . Образ A при отображении f обозначается $f(A)$. Если соответствие f при этом сюръективно, т. е. каждый элемент B имеет прообраз в A , то говорят, что имеет место отображение A на B (сюръективное отображение). Если $f(A)$ состоит из единственного элемента, то f называется функцией-константой. Отображение типа $A \rightarrow A$ часто называют *преобразованием* множества A .

Функции f и g равны, если их область определения — одно и то же множество A и для любого $a \in A$ $f(a) = g(a)$.

Пример 1.8. а. Функция $f(x) = 2^x$ является отображением N в N и N_0 на M_{2^n} .

б. Всякая нумерация счетного множества является его отображением на N .

в. Функция $f(x) = \sqrt{x}$ не полностью определена, если ее тип $N \rightarrow N$, и полностью определена, если ее тип $N \rightarrow R$ или $R^+ \rightarrow R$ (R^+ — положительное подмножество R).

г. Пусть зафиксирован список $\{a_1, \dots, a_n\}$ всех элементов конечного множества A . Тогда любой вектор $v_i = (a_{i1}, \dots, a_{in})$ из A^n можно рассматривать как описание функции $f_i: A \rightarrow A$ (т. е. преобразования A), определяемой следующим образом: $f_i(a_j) = a_{ij}$, т. е. значение f_i для a_j равно j -й компоненте v_i . Число всех преобразований A равно, следовательно, $|A^n| = n^n$. Аналогично всякую функцию типа $N \rightarrow N$ можно представить бесконечной

последовательностью элементов N , т. е. натуральных чисел; отсюда нетрудно показать, что множество всех преобразований счетного множества континуально.

д. Каждое натуральное число n единственным образом разлагается на произведение простых чисел (простых делителей этого числа). Поэтому, если договориться располагать простые делители n в определенном порядке (например, в порядке убывания), то получим функцию $q(n)$ типа

$$N \rightarrow \bigcup_{i=1}^{\infty} N^i,$$

отображающую N в множество векторов произвольной длины. Например, $q(42) = (2, 3, 7)$, $q(23) = 23$, $q(100) = (2, 2, 5, 5)$. Это отображение не является сюръективным, так как в область значений q не входят векторы, для компонент которых не выполнено условие неубывания, а также векторы с непростыми компонентами.

е. Каждому человеку соответствует множество его знакомых. Если зафиксировать момент времени (например, 10 января 1986 г., 5 ч 00 мин), то это соответствие будет однозначным и явится отображением множества M людей, живущих в этот момент, в множество подмножеств M .

Функция типа $A_1 \times A_2 \times \dots \times A_n \rightarrow B$ называется *n-местной функцией*. В этом случае принято считать, что функция имеет n аргументов: $f(a_1, a_2, \dots, a_n) = b$, где $a_1 \in A_1$, $a_2 \in A_2$, ..., $a_n \in A_n$, $b \in B$. Сложение, умножение, вычитание и деление являются двухместными функциями на R , т. е. функциями типа $R^2 \rightarrow R$. Таблица выигрышер лотереи задает двухместную не полностью определенную функцию, которая устанавливает соответствие между парами из N^2 (серия, номер) и множеством выигрышер.

Пусть дано соответствие $G \subseteq A \times B$. Если соответствие $H \subseteq B \times A$ таково, что $(b, a) \in H$ тогда и только тогда, когда $(a, b) \in G$, то соответствие H называется обратным к G и обозначается G^{-1} . Если соответствие, обратное к функции $f: A \rightarrow B$, является функциональным, то оно называется *функцией, обратной к f*, и обозначается f^{-1} . Так как в обратном соответствии образы и прообразы меняются местами, то для существования функции, обратной к $f: A \rightarrow B$, требуется, чтобы каждый элемент b

из области значений f имел единственный прообраз. Это, в свою очередь, означает, что для функции $f: A \rightarrow B$ обратная функция существует тогда и только тогда, когда f является взаимно однозначным соотношением между своей областью определения и областью значений.

Пример 1.9. а. Функция $\sin x$ имеет тип $R \rightarrow R$. Отрезок $[-\pi/2, \pi/2]$ она взаимно однозначно отображает на отрезок $[-1, 1]$. Поэтому на отрезке $[-1, 1]$ для нее существует обратная функция $\arcsin x$.

б. Ранее приводились примеры кодирующих функций, которые каждому объекту из своей области значений ставят в соответствие некоторый код. Для кодирующих функций обратной будет декодирующая функция, которая каждому коду ставит в соответствие закодированный этим кодом объект. Если кодирующая функция не сюръективна, то декодирующая функция не всюду определена.

Пусть даны функции $f: A \rightarrow B$ и $g: B \rightarrow C$. Функция $h: A \rightarrow C$ называется *композицией* функций f и g (обозначение $f \circ g$), если имеет место равенство $h(x) = g(f(x))$, где $x \in A$. Композиция f и g представляет собой последовательное применение функций f и g ; g применяется к результату f . Часто говорят, что функция h получена *подстановкой* f в g . Знак \circ аналогично умножению часто опускается.

Для многоместных функций $f: A^m \rightarrow B$, $g: B^n \rightarrow C$ возможны различные варианты подстановки f в g , дающие функции различных типов. Например, при $m = 3$, $n = 4$ функция $h_1 = g(x_1, f(y_1, y_2, y_3), x_3, x_4)$ имеет шесть аргументов и тип $B \times A_3 \times B_2 \rightarrow C$, а функция $h_2 = g(f(y_1, y_2, y_3), f(z_1, z_2, z_3), x_3, x_4)$ имеет восемь аргументов и тип $A^6 \times B^2 \rightarrow C$. Особый интерес представляет случай, когда задано множество функций типа $f_1: A_1^m \rightarrow A$, ..., $f_n: A_n^m \rightarrow A$. В этом случае возможны, во-первых, любые подстановки функций друг в друга, а во-вторых, любые переименования аргументов, например переименование x_3 в x_2 , порождающее из функции $f(x_1, x_2, x_3, x_4)$ функцию трех аргументов $f(x_1, x_2, x_2, x_4)$. Функция, полученная из f_1 , ..., f_n некоторой подстановкой их друг в друга и переименованием аргументов, называется *суперпозицией* f_1, \dots, f_n . Выражение, описывающее эту суперпозицию

и содержащее функциональные знаки и символы аргументов, называется *формулой*.

Пример 1.10. а. Функции $\sin x$ и \sqrt{x} имеют тип $R \rightarrow R$, т. е. отображают одно и то же множество в себя. Поэтому их композиция возможна в произвольном порядке и дает функции $\sin\sqrt{x}$ и $\sqrt{\sin x}$. Заметим, что области определения их различны: первая функция определена на положительной полуоси; вторая функция определена на множестве отрезков $[2k\pi, (2k+1)\pi]$, где $k = 0, \pm 1, \pm 2\dots$. Таким образом, область определения композиции может быть уже областей определения обеих исходных функций и даже казаться пустой.

б. Множество $K = \{k_1, \dots, k_m\}$ команд компьютера отображается в машинные коды этого компьютера т. е. в натуральные числа. Кодирующая функция φ имеет тип $K \rightarrow N$. С помощью суперпозиции этой функции и арифметических функций оказываются возможными арифметические действия над командами (которые сами по себе числами не являются!), т. е. функции $\varphi(k_1) + \varphi(k_2)$, $\varphi(k_1) + 4$ и т. д.

в. В функции $f_1(x_1, x_2, x_3) = x_1 + 2x_2 + 7x_3$ переименование x_2 в x_3 приводит к функции $f_1(x_1, x_2, x_2) = x_1 + 2x_2 + 7x_2$, которая равна функции двух аргументов $f_2(x_1, x_2) = x_1 + 9x_2$. Переименование x_1 и x_3 в x_2 приводит к одноместной функции $f_3(x_2) = 10x_2$.

г. Элементарной функцией в математическом анализе называется всякая функция f , являющаяся суперпозицией фиксированного (т. е. не зависящего от значений аргументов f) числа арифметических функций, а также функций e^x , $\log x$, $\sin x$, $\arcsin x$. Например, функция $\log^2(x_1 + x_2) + 3\sin\sin x_1 + x_3$ элементарна, так как является результатом нескольких последовательных суперпозиций $x_1 + x_2$, x^2 , $\log x$, $3x$, $\sin x$.

д. Всякая непрерывная функция n переменных представима в виде суперпозиции непрерывных функций двух переменных. (Этот результат получен в 1956–1957 гг. в работах А. Н. Колмогорова и В. И. Арнольда и является решением 13-й проблемы Гильберта.)

е. Рассмотрим множество $\{1, 2, 3\}$ и два преобразования этого множества: $\alpha = (1 \rightarrow 3, 2 \rightarrow 3, 3 \rightarrow 1)$ и $\beta = (1 \rightarrow 2, 2 \rightarrow 1, 3 \rightarrow 1)$. Обычно преобразования конечных множеств записывают так:

$$\alpha = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 3 & 1 \end{pmatrix}, \beta = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 1 \end{pmatrix}.$$

Композиция преобразований — это новое преобразование:

$$\alpha\beta = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 1 & 2 \end{pmatrix}, \beta\alpha = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 3 & 3 \end{pmatrix}.$$

Способы задания функций. Наиболее простой способ задания функций — это таблицы, т. е. конечные списки пар $(x, f(x))$. Однако таким образом могут быть заданы только функции, определенные на конечных множествах. Таблицы функций, определенных на бесконечных множествах (например, логарифмических, тригонометрических и т. д.), задают эти функции только в конечном числе точек. Для вычисления значений (приближенных!) функций в промежуточных точках служат правила интерполяции. Другим не менее известным способом задания функций является формула, описывающая функцию как суперпозицию других (исходных) функций (пример 1.10, г). Если способ вычисления исходных функций известен, то формула задает процедуру вычисления данной функции как некоторую последовательность вычислений исходных функций.

Вычисления функций по таблицам, формулам, а также с помощью графиков являются частным видом вычислительных процедур. Для функции, заданной таблицей, процедура заключается в поиске нужной строки таблицы; для формулы — в последовательном вычислении всех функций, входящих в суперпозицию. Существуют вычислительные процедуры, не относящиеся к указанным трем видам. Среди них особо следует выделить рекурсивные процедуры. Рекурсивная процедура задает функцию f , определенную на N_0 , следующим образом: 1) задается значение $f(1)$ (или $f(0)$); 2) значение $f(n + 1)$ определяется через суперпозицию $f(n)$ и других, считающихся известными, функций. Простейшим примером рекурсивной процедуры является вычисление функции $n!$: 1) $0! = 1$; 2) $(n + 1)! = n!(n + 1)$. В правой части последнего равенства имеется формула, однако это задание функции существенно отличается от задания формулой типа примера 1.10, г. Отличие состоит в том, что

для вычисления $8!$ необходимо сначала вычислить $7!$, тогда как в примере 1.10, г вычисление для любой тройки аргументов происходит «непосредственно». Точнее, для вычисления $n!$ требуется n умножений, т. е. число вычислительных шагов растет с ростом аргумента; для вычисления же функции примера 1.10, г требуется всегда одно и то же число шагов (если шагом считать вычисление функции, входящей в суперпозицию), равное восьми.

Наконец, возможны описания функций, которые не содержат способа вычисления функции, хотя сомнений в том, что функция однозначно задана, не возникает. Определим, например, функцию следующим образом:

$$f_\pi(x) = \begin{cases} x, & \text{если } x \in M_\pi; \\ 0, & \text{если } x \notin M_\pi. \end{cases}$$

Из описания M_π (см. § 1.1) не видно, как убедиться в том, что $x \in M_\pi$, поэтому нет гарантии, что $f_\pi(x)$ можно вычислить.

Отступление 1.3. Для функций возникает тот же вопрос, что и для множеств: что значит «задать функцию»? По смыслу нашего определения задать функцию $f: A \rightarrow B$ — это значит описать определяющее ее подмножество $A \times B$, поэтому вопрос сводится к заданию некоторого множества. Однако можно определить понятие функции, не используя языка теории множеств: функция считается заданной, если задана вычислительная процедура, которая по любому заданному значению аргумента выдает соответствующее значение функции.

Функция, определенная таким образом, называется вычислимой. Процедура должна однозначно приводить к результату. Уточнение понятия однозначной и результативной процедуры привело к созданию теории алгоритмов.

Понятие алгоритма может быть принято за исходное при построении всей системы понятий математики. Такой подход к обоснованию математики, называемый конструктивным, допускает только те математические объекты и утверждения, которые могут быть получены с помощью алгоритмов. С этой точки зрения описанная ранее функция f_π вообще не заслуживает названия функции, поскольку для нее не указан алгоритм вычисления. Понятие множества перестает быть первичным; оно определяется с помощью разрешающей или порождающей процедуры (см. § 1.1). Множества, для которых нет таких процедур, просто не рассматриваются.

Достоинства конструктивного подхода достаточно ясны. Однако его последовательное проведение показало, что он требует более радикальной ревизии основных понятий математики, чем это кажется с первого взгляда, и иногда приводит к значительным концептуальным и языковым неудобствам. Поэтому в качестве «математического мировоззрения» конструктивизм разделяется относительно небольшим числом математиков, хотя, пожалуй, никто не отрицает преимуществ конструктивных методов в тех случаях, когда они возможны.

1.3. ОТНОШЕНИЯ

Подмножество R декартова произведения $A_1 \times A_2 \times \dots \times A_n$ называется n -арным (или n -местным) отношением. Говорят, что a_1, a_2, \dots, a_n находятся в отношении R , если $(a_1, a_2, \dots, a_n) \in R$. Пусть, например, V — множество всех точек плоскости, а L — множество всех прямых этой плоскости. Отношение R_{VL} «прямые l_i, l_j пересекаются в точке v_k » — трехместное отношение, определенное на множестве $L \times L \times V$. Оно является множеством всех троек (l_i, l_j, v_k) , таких, что l_i, l_j пересекаются в точке v_k .

Подмножество $R \subseteq M^n$ называется n -местным отношением на множестве M . Одноместное отношение — это просто подмножество M . Такие отношения называют признаками: a обладает признаком R , если $a \in R$ и $R \subseteq M$. Свойства одноместных отношений — это свойства подмножеств M ; поэтому для случая $n = 1$ термин «отношение» употребляется редко. Примером трехместного (или, как говорят, тернарного) отношения является множество троек нападающих в хоккейной команде. Каждый из нападающих находится в этом отношении со всеми теми игроками, с которыми он играет в одной тройке (один нападающий может, вообще говоря, участвовать более чем в одной тройке).

Наиболее часто встречающимися и хорошо изученными являются двухместные, или *бинарные*, отношения. Именно о них будет идти речь в дальнейшем (слово «бинарные» будем опускать). Если a, b находятся в отношении R , это часто записывается как aRb .

Пример 1.11. а. Отношения на N : 1) отношение \leq выполняется для пар $(7, 9)$ и $(7, 7)$, но не выполняется

для пары $(9, 7)$; 2) отношение «иметь общий делитель, отличный от единицы» выполняется для пар $(6, 9), (4, 2), (2, 4), (4, 4)$, но не выполняется для пар $(7, 9)$ и $(9, 7)$; 3) отношение «быть делителем» (т. е. aRb означает « a — делитель b ») выполняется для пар $(2, 4)$ и $(4, 4)$, но не выполняется для пар $(4, 2)$ и $(7, 9)$.

б. Отношение на множестве точек действительной плоскости: 1) отношение «находиться на одинаковом расстоянии от начала координат» выполняется для пары точек $(3, 4)$ и $(-2, \sqrt{21})$, но не выполняется для пары точек $(3, 4)$ и $(1, 6)$: это отношение совпадает с отношением «находиться на одной и той же окружности с центром в начале координат»; 2) отношение «находиться на разном расстоянии от начала координат» выполняется для тех и только тех пар точек, для которых не выполняется предыдущее отношение; 3) отношение «быть симметричным относительно оси x » выполняется для всех пар точек (x_1, y_1) и (x_2, y_2) , удовлетворяющих условию $x_1 = x_2, y_1 = -y_2$.

в. Отношения на множестве людей: «живь в одном городе»; «быть моложе»; «быть сыном»; «быть знакомым».

Пусть дано отношение R на M . Для любого подмножества $M_1 \subseteq M$ естественно определяется отношение R' , называемое сужением R на M_1 , которое получается из R удалением всех пар, содержащих элементы, не принадлежащие M_1 . Иначе говоря, $R' = R \cap M^2_1$. Строго говоря, R и R' — это разные отношения, с разными областями определения. Однако, если не возникает разнотечений, этот педантизм не соблюдается: например, вполне можно говорить об отношении «быть делителем», не уточняя, задано оно на N или каком-либо его подмножестве*.

Для задания бинарных отношений можно использовать любые способы задания множеств (например, список пар, для которых данное отношение выполняется). Отношения на конечных множествах обычно задаются списком или матрицей. Матрица бинарного отношения на множестве $M = \{a_1, \dots, a_m\}$ — это квадратная матрица C порядка m , в которой элемент c_{ij} , стоящий на пересече-

* Отметим, что это допустимо только потому, что совершенно ясно, как перенести это отношение с N на любое подмножество N .

нии i -й строки и j -го столбца, определяется следующим образом:

$$c_{ij} = \begin{cases} 1, & \text{если } a_i R_j; \\ 0 & \text{в противном случае.} \end{cases}$$

Например, для конечного множества $\{1, 2, 3, 4, 5, 6\}$ матрицы отношений 1–3 из примера 1.11, а приведены в табл. 1.1, a – c соответственно.

Для любого множества M отношение E , заданное матрицей, в которой по главной диагонали стоят единицы, а в остальных местах — нули, называется *отношением равенства* на M .

<i>a</i>	1	2	3	4	5	6
1	1	1	1	1	1	1
2	0	1	1	1	1	1
3	0	0	1	1	1	1
4	0	0	0	1	1	1
5	0	0	0	0	1	1
6	0	0	0	0	0	1

<i>b</i>	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	1	0	1	0	1
3	0	0	1	0	0	1
4	0	1	0	1	0	1
5	0	0	0	0	1	0
6	0	1	1	1	0	1

<i>c</i>	1	2	3	4	5	6
1	1	1	1	1	1	1
2	0	1	0	1	0	1
3	0	0	1	0	0	1
4	0	0	0	1	0	0
5	0	0	0	0	1	0
6	0	0	0	0	0	1

Поскольку отношения на M задаются подмножествами M_2 , для них можно определить те же операции, что и над множествами. Например, отношение 2 из примера 1.11, б является дополнением отношения 1, отношение \leq является объединением отношений $<$ и равенства.

Определим еще одну операцию над отношениями. Отношение называется *обратным* к отношению R (обозначение R^{-1}), если $a_i R^{-1} a_j$ тогда и только тогда, когда $a_j R a_i$. Из определения непосредственно следует, что $(R^{-1})^{-1} = R$. Для отношения \leq обратным является отношение \geq .

Свойства отношений. Отношение R называется *рефлексивным*, если для любого $a \in M$ имеет место $a R a$. Главная диагональ его матрицы содержит только единицы. Отношение R называется *антирефлексивным*, если ни для какого $a \in M$ не выполняется $a R a$. Главная диагональ его матрицы содержит только нули. Отношения \leq и «иметь общий делитель» рефлексивны, отношения $<$ и «быть сыном» антирефлексивны. Отношение «быть симметричным относительно оси x » не является ни рефлексивным,

ни антирефлексивным: точка плоскости симметрична сама себе, если она лежит на оси x , и несимметрична сама себе в противном случае.

Отношение R называется *симметричным*, если для пары $(a, b) \in M^2$ из aRb следует bRa (иначе говоря, для любой пары R выполняется либо в обе стороны, либо не выполняется вообще). Матрица симметричного отношения симметрична относительно главной диагонали: $c_{ij} = c_{ji}$ для любых i и j . Отношение R называется *антисимметричным*, если из a_iRa_j и a_jRa_i следует, что $a_i = a_j$. Отношение «быть симметричным относительно оси x » является симметричным*: если первая точка симметрична второй, то и вторая симметрична первой. Пример антисимметричного отношения — отношение \leq : действительно, если $a \leq b$ и $b \leq a$, то $a = b$. Нетрудно убедиться в том, что отношение R симметрично тогда и только тогда, когда $R = R^{-1}$.

Отношение R называется *транзитивным*, если для любых a, b, c из aRb и bRc следует aRc . Отношения «равенство», \leq , «жить в одном городе» транзитивны; отношение «быть сыном» нетранзитивно. Отношение «пересекаться», т. е. «иметь непустое пересечение», заданное на системе множеств, также нетранзитивно. Например, $\{1, 2\}$ пересекается с $\{2, 3\}$, $\{2, 3\}$ пересекается с $\{3, 4\}$, однако $\{1, 2\}$ и $\{3, 4\}$ не пересекаются.

Для любого отношения R отношение \hat{R} , называемое *транзитивным замыканием* R , определяется следующим образом: $a\hat{R}b$, если в M существует цепочка из n элементов $a = a_1, a_2 \dots a_{n-1}, a_n = b$, в которой между соседними элементами выполнено R : $aRa_2, a_2Ra_3, \dots, a_{n-1}Rb$. Если R транзитивно, то $R = \hat{R}$. Действительно, если aRb , то aRb (цепочка состоит из двух элементов a, b), поэтому $R \subseteq \hat{R}$. Если же $a\hat{R}b$, то существует цепочка $aRa_2, a_2Ra_3, \dots, a_{n-1}Rb$. Но так как R транзитивно**, то aRb , поэтому $\hat{R} \subseteq R$. Из включения в обе стороны следует $R = \hat{R}$.

* Эта фраза вовсе не является тавтологией, так как два упоминания в ней термина «симметричный» имеют разный смысл и относятся к двум разным типам объектов: первое упоминание — к свойству пар точек на плоскости, а второе — к свойству отношения между парами точек.

** В этом рассуждении используется тот факт, что транзитивное отношение «распространяется по цепочке» не только из трех (как указано в определении), но и из любого числа элементов.

Транзитивным замыканием отношения «быть сыном» является отношение «быть прямым потомком», являющееся объединением отношений «быть сыном», «быть внуком», «быть правнуком» и т. д.

Транзитивным замыканием отношения «иметь общую стену» для жильцов дома является отношение «жить на одном этаже».

Отношения эквивалентности. Отношение называется *отношением эквивалентности* (или просто *эквивалентностью*), если оно рефлексивно, симметрично и транзитивно.

Пример 1.12. а. Отношение равенства E на любом множестве является отношением эквивалентности. Равенство — это минимальное отношение эквивалентности в том смысле, что при удалении любой пары из E (т. е. любой единицы на диагонали матрицы E) оно перестает быть рефлексивным и, следовательно, уже не является эквивалентностью.

б. Утверждения вида $(a + b) (a - b) = a^2 - b^2$ или $\sin^2 x + \cos^2 x = 1$, состоящие из формул, соединенных знаком равенства, задают бинарное отношение на множестве формул, описывающих суперпозиции элементарных функций (см. пример 1.10, г). Это отношение обычно называется отношением равносильности и определяется следующим образом: формулы равносильны, если они задают одну и ту же функцию. Равносильность, хотя и обозначается знаком $=$, отличается от отношения равенства E , так как оно может выполняться для различных формул (впрочем, можно считать, что знак равенства в таких соотношениях относится не к формулам, а к функциям, которые ими описываются). Отношение E для формул — это совпадение формул по написанию. Оно называется *графическим равенством*.

в. Рассмотрим множество треугольников на плоскости, считая, что треугольник задан, если заданы координаты его вершин. Два треугольника называются конгруэнтными (иногда их называют просто равными), если они при наложении совпадают, т. е. могут быть переведены друг в друга путем некоторого перемещения. Конгруэнтность является отношением эквивалентности на множестве треугольников.

г. Отношение «иметь один и тот же остаток от деления на 7» является эквивалентностью на N . Это отношение выполняется, например, для пар $(11, 46), (14, 70)$ и не выполняется для пар $(12, 13), (14, 71)$.

Пусть на множестве M задано отношение эквивалентности R . Осуществим следующее построение. Выберем элемент $a_1 \in M$ и образуем класс (подмножество M) C_1 , состоящий из a_1 и всех элементов, эквивалентных a_1 ; затем выберем элемент $a_2 \notin C_1$ и образуем класс C_2 , состоящий из a_2 и всех элементов, эквивалентных a_2 , и т. д. Получится система классов $C_1, C_2\dots$ (возможно, бесконечная) такая, что любой элемент из M входит хотя бы в один класс, т. е. $C_i = M$. Эта система классов обладает следующими свойствами: 1) она образует разбиение, т. е. классы попарно не пересекаются; 2) любые два элемента из одного класса эквивалентны; 3) любые два элемента из разных классов неэквивалентны. Все эти свойства немедленно вытекают из рефлексивности, симметричности и транзитивности R . Действительно, если бы классы, например C_1 и C_2 , пересекались, то они имели бы общий элемент b , эквивалентный a_1 и a_2 , но тогда из-за транзитивности R было бы a_1Ra_2 , что противоречит построению C_2 . Аналогично доказываются другие два свойства.

Построенное разбиение, т. е. система классов, называется системой классов эквивалентности по отношению R . Мощность этой системы называется индексом разбиения. С другой стороны, любое разбиение M на классы определяет некоторое отношение эквивалентности, а именно, отношение «входить в один и тот же класс данного разбиения».

Пример 1.13. а. Все классы эквивалентности по отношению равенства E состоят из одного элемента.

б. Формулы, описывающие одну и ту же элементарную функцию, находятся в одном классе эквивалентности по отношению равносильности. В этом примере счетны само множество формул, множество классов эквивалентности (т. е. индекс разбиения) и каждый класс эквивалентности.

в. Разбиение множества треугольников по отношению конгруэнтности имеет континуальный индекс, причем каждый класс также имеет мощность континуум.

г. Разбиение N по отношению «иметь общий остаток от деления на 7» имеет конечный индекс 7 и состоит из 7 счетных классов $0, 7, 14\dots; 1, 8, 15\dots; 2, 9, 16\dots; \dots; 6, 13, 20\dots$.

Отношения порядка. Отношение называется *отношением нестрогого порядка*, если оно рефлексивно, антисимметрично и транзитивно. Отношение называется *отношением строгого порядка*, если оно антирефлексивно, антисимметрично и транзитивно. Оба типа отношений называются отношениями порядка. Элементы a, b сравнимы по отношению порядка R , если выполняется aRb или bRa . Множество M , на котором задано отношение порядка, называется *линейно упорядоченным*, если любые два элемента M сравнимы, и *частично упорядоченным* в противном случае.

Пример 1.14, а. Отношения \leq и \geq для чисел являются отношениями нестрогого порядка, отношения $<$ и $>$ — отношениями строгого порядка. Оба отношения линейно упорядочивают множества N и R .

б. Определим отношения \leq и $<$ на R^n следующим образом: $(a_1, \dots, a_n) \leq (b_1, \dots, b_n)$, если $a_1 \leq b_1, \dots, a_n \leq b_n$; $(a_1, \dots, a_n) < (b_1, \dots, b_n)$, если $(a_1, \dots, a_n) \leq (b_1, \dots, b_n)$ и хотя бы в одной координате i выполнено $a_i < b_i$. Эти отношения определяют частичный порядок на R^n : $(5, 1/2, -3) < < (5, 2/3, -3); (5, 1/2, -3)$ и $(5, 0, 0)$ не сравнимы.

в. На системе подмножеств множества M отношение включения \subseteq задает нестрогий частичный порядок, а отношение строгого включения \subset задает строгий частичный порядок. Например, $\{1, 2\} \subset \{1, 2, 3\}$, а $\{1, 2\}$ и $\{1, 3, 4\}$ не сравнимы.

г. Отношение подчиненности на предприятии задает строгий частичный порядок. В нем несравнимыми являются сотрудники разных отделов.

д. Пусть в списке букв конечного алфавита A порядок букв зафиксирован, т. е. всегда один и тот же, как, например, в русском или латинском алфавите цифр. Тогда этот список определяет линейное упорядочение букв, которое назовем отношением предшествования и обозначим \prec ($a_i \prec a_j$, если a_i предшествует a_j в списке букв). На основе отношения предшествования букв строится отношение предшествования слов, определяемое

следующим образом. Пусть даны слова $\alpha_1 = a_{11} \dots a_{1m}$ и $\alpha_2 = a_{21} \dots a_{2n}$. Тогда $\alpha_1 < \alpha_2$, если и только если либо 1) $\alpha_1 = \beta a_i \gamma$, $\alpha_2 = \beta a_j \delta$ и $a_i < a_j$ (β , γ , δ — некоторые слова, возможно, пустые, a_i и a_j — буквы), либо 2) $\alpha_2 = \alpha_1 \beta$, где β — непустое слово. Это отношение задает упорядочение множества всех конечных слов в алфавите A , которое называется лексикографическим *упорядочением слов*.

Пример 1.15. а. Наиболее известным примером лексикографического упорядочения является упорядочение слов в словарях. Например, лес \prec лето (случай 1 определения: $\beta = \text{лес}$, $c \prec t$, γ пусто, $\delta = 0$), поэтому слово «лес» расположено в словаре раньше слова «лето», лес \prec лесть (случай 2 определения: $\beta = \text{ть}$).

б. Если рассматривать числа в позиционных системах счисления (например, в двоичной или десятичной) как слова в алфавите цифр, то их лексикографическое упорядочение совпадает с обычным, если все сравнимые числа имеют одинаковое число разрядов. В общем же случае эти два вида упорядочения могут не совпадать: например, $10 < 1073$ и $20 < 1073$, но $10 \prec 1073$, а $20 \succ 1073$. Для того чтобы они совпадали, нужно выровнять число разрядов у всех сравниваемых чисел, приписывая слева нули. В данном примере при этом получим $0020 \prec 1073$. Такое выравнивание автоматически происходит при записи целых чисел в компьютере.

в. Лексикографическое упорядочение цифровых представлений дат вида 05.08.86 (пятое августа 1986 года) не совпадает с естественным упорядочением дат от ранних к поздним: например, 05.08.86 лексикографически «старше» третьего числа любого года. Чтобы возрастание дат совпадало с лексикографическим упорядочением, обычное представление надо «перевернуть», т. е. годы поместить слева: 86.08.05. Так обычно делают при представлении дат в памяти компьютера.

ЭЛЕМЕНТЫ ОБЩЕЙ АЛГЕБРЫ

2.1. ОПЕРАЦИИ НА МНОЖЕСТВАХ И ИХ СВОЙСТВА

Алгебры. Функцию типа $\varphi: M^n \rightarrow M$ будем называть *n-арной операцией* на множестве M ; n называется *арностью* операции φ . Множество M вместе с заданной на нем совокупностью операций $\Omega = \{\varphi_1, \dots, \varphi_m, \dots\}$, т. е. система $A = (M; \varphi_1, \dots, \varphi_m, \dots)$, называется *алгеброй*; M называется *основным*, или *несущим, множеством* (или просто *носителем*) алгебры A . Вектор арностей операций алгебры называется ее *тиром*, совокупность операций Ω — *сигнатурой*.

Множество $M' \subset M$ называется *замкнутым* относительно *n-арной* операции φ на M , если $\varphi(M'^n) \subseteq M'$, т. е. если значения φ на аргументах из M' принадлежат M' . Если M' замкнуто относительно всех операций $\varphi_1, \dots, \varphi_m, \dots$ алгебры A , то система $A' = (M'; \varphi_1, \dots, \varphi_m, \dots)$ называется *подалгеброй* A (при этом $\varphi_1, \dots, \varphi_m, \dots$ рассматриваются как операции на M').

Пример 2.1. а. Алгебра $(R; +, \cdot)$ называется *полем действительных чисел*. Обе операции — бинарные, поэтому тип этой алгебры $(2, 2)$. Все конечные подмножества R , кроме $\{0\}$, не замкнуты относительно обеих операций. Подалгеброй этой алгебры является, например, поле рациональных чисел.

б. Пусть $N_p = \{0, 1, 2, \dots, p - 1\}$. Определим на N_p операции \oplus («сложение по модулю p ») и \odot («умножение по модулю p ») следующим образом: $a \oplus b = c$, $a \odot b = d$, где c и d — остатки от деления на p чисел $a + b$ и $a \cdot b$ соответственно. Например, если $p = 7$, то $N_p = \{0, 1, \dots, 6\}$, $3 \oplus 4 = 0$, $3 \odot 4 = 5$, $4 \oplus 6 = 3$. Часто операции \oplus и \odot

обозначают как $a + b \equiv c(\text{mod } p)$, $a \cdot b \equiv d(\text{mod } p)$. Если p — простое число, то алгебра $\{N_p, \oplus, \odot\}$ называется *конечным полем характеристики p* или полем Галуа и обозначается $\text{GF}(p)$.

в. Пусть задано множество U . Множество всех его подмножеств называется *булеаном U* и обозначается через $\mathfrak{B}(U)$. Алгебра $B = (\mathfrak{B}(U); \cup, \cap, \neg)$ называется *булевой алгеброй множеств* над U , ее тип $(2, 2, 1)$. Элементами основного множества этой алгебры являются множества (подмножества U). Для любого $U' \subset U$ $B' = (\mathfrak{B}(U'); \cup, \cap, \neg)$ является подалгеброй B . Например, если $U = \{a, b, c, d\}$, то основное множество алгебры B содержит 16 элементов; алгебра $B' = \{\mathfrak{B}(\{a, c\}); \cup, \cap, \neg\}$ — подалгебра B ; ее основное множество содержит четыре элемента.

г. Множество F одноместных функций на R , т. е. функций $f: R \rightarrow R$, вместе с операцией дифференцирования является алгеброй. Элементы основного множества — функции типа $R \rightarrow R$, единственной операцией этой алгебры служит дифференцирование — унарная операция типа $F \rightarrow F$ (производной функции на R является снова функция на R). Множество элементарных функций (см. пример 1.10, г) замкнуто относительно дифференцирования, поскольку производные элементарных функций элементарны и, следовательно, образует подалгебру данной алгебры.

д. Рассмотрим квадрат с вершинами в точках a_1, a_2, a_3, a_4 , пронумерованных против часовой стрелки, и повороты квадрата вокруг центра в том же направлении, переводящие вершины в вершины. Таких поворотов бесконечное множество: на углы $0, \pi/2, \pi, 3\pi/2, 2\pi, 5\pi/2\dots$, однако они задают всего четыре различных отображения множества вершин в себя, соответствующих первым четырем поворотам.

Таким образом, получаем алгебру с основным множеством $\{a_1, a_2, a_3, a_4\}$ и четырьмя унарными операциями $\alpha, \beta, \gamma, \delta$. Их можно задать табл. 2.1, в которой на пересечении, например, строки a_3 и столбца γ написано значение функции $\gamma(a_3)$.

Операция α , отображающая любой элемент в себя, называется тождественной операцией. Она соответствует нулевому повороту. Подалгебр в этой алгебре нет.

Таблица 2.1

	α	β	γ	δ
a_1	a_1	a_2	a_3	a_4
a_2	a_2	a_3	a_4	a_1
a_3	a_3	a_4	a_1	a_2
a_4	a_4	a_1	a_2	a_3

Таблица 2.2

	α	β	γ	δ
α	α	β	γ	δ
β	β	γ	δ	α
γ	γ	δ	α	β
δ	δ	α	β	γ

е. Множество $O = \{\alpha, \beta, \gamma, \delta\}$ отображений вершин в себя из предыдущего примера вместе с бинарной операцией композиции отображений \circ (см. § 1.2) образует алгебру $\{O; \circ\}$. Элементами множества O являются отображения (повороты).

Композиция отображений — это последовательное выполнение двух поворотов. Она задается табл. 2.2 (в ней на пересечении строки α и столбца γ написан результат композиции $\alpha \circ \gamma$).

Такая таблица, задающая бинарную операцию, называется таблицей Кэли. Множество $\{\alpha, \gamma\}$, т. е. повороты на углы $0, \pi$, образует подалгебру алгебры $\{O; \circ\}$.

Свойства бинарных алгебраических операций. Для того чтобы последующие соотношения выглядели более привычно, условимся результат применения бинарной операции φ к элементам a, b записывать не в функциональном виде $\varphi(a, b)$, а в виде $a\varphi b$, так, как это принято для арифметических операций.

Операция φ называется *ассоциативной*, если для любых элементов a, b, c

$$(a\varphi b)\varphi c = a\varphi(b\varphi c).$$

Выполнение этого условия (свойства ассоциативности) означает, что скобки в выражении $a\varphi b\varphi c$ можно не расставлять. Сложение и умножение чисел ассоциативны, что и позволяет не ставить скобки в выражениях $a + b + c$ и abc . Пример неассоциативной операции — возведение в степень a^b : $(a^b)^c \neq a^{(b^c)}$. Правда, запись a^{b^c} считается допустимой, но служит сокращением выражения $a^{(b^c)}$, а не $(a^b)^c$, которое равно более компактному выражению a^{bc} .

Важным примером ассоциативной операции является композиция отображений.

Операция ϕ называется *коммутативной*, если для любых элементов a, b

$$a\phi b = b\phi a.$$

Сложение коммутативно («от перемены мест слагаемых сумма не меняется»), так же как и умножение; вычитание и деление некоммутативны. Некоммутативным является умножение матриц, например

$$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} 2 & 2 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 3 \\ 0 & 1 \end{pmatrix}, \text{ но } \begin{pmatrix} 2 & 2 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 4 \\ 0 & 1 \end{pmatrix}.$$

Операция ϕ называется *дистрибутивной слева* относительно операции ψ , если для любых a, b, c

$$a\phi(b\psi c) = (a\phi b)\psi(a\phi c),$$

и *дистрибутивной справа* относительно ψ , если

$$(a\psi b)\phi c = (a\phi c)\psi(b\phi c).$$

Дистрибутивность разрешает раскрывать скобки. Например, умножение дистрибутивно относительно сложения слева и справа. Возведение в степень дистрибутивно относительно умножения справа: $(ab)^c = a^c b^c$, но не слева: $a^{bc} \neq a^b a^c$. Сложение недистрибутивно относительно умножения: $a + bc \neq (a + b)(a + c)$. Как будет показано позднее (§ 3.2), операции пересечения и объединения множеств дистрибутивны относительно друг друга слева и справа [см. (3.9) и (3.10)].

Гомоморфизм и изоморфизм. Алгебры с разными типами, очевидно, имеют существенно различное строение. Если же алгебры имеют одинаковый тип, то наличие у них сходства характеризуется с помощью вводимых ниже понятий гомоморфизма и изоморфизма.

Пусть даны две алгебры $A = (K; \phi_1, \dots, \phi_p)$ и $B = (M; \psi_1, \dots, \psi_p)$ одинакового типа. Гомоморфизмом алгебры A в алгебру B называется отображение $\Gamma: K \rightarrow M$, удовлетворяющее условию

$$\Gamma(\phi_i(k_{j1}, \dots, k_{jl(i)})) = \psi_1(\Gamma(k_{j1}), \dots, \Gamma(k_{jl(i)})) \quad (2.1)$$

для всех $i = 1, \dots, p$ [$l(i)$ — arityность операций ϕ_i и ψ_1 , которая у них по условию одинакова] и всех $k_{jr} \in K$. Смысл условия (2.1) в том, что независимо от того, выполнена ли сначала операция ϕ_i в A и затем произведено

отображение Γ , либо сначала произведено отображение Γ , а затем в B выполнена соответствующая операция ψ_1 , результат будет одинаков.

Изоморфизмом алгебры A на алгебру B называется взаимно однозначный гомоморфизм. В этом случае существует обратное отображение $\Gamma^{-1}: M \rightarrow K$, также взаимно однозначное. Пусть $\Gamma(k_j) = m_j$, $m_j \in M$. Тогда $k_j = \Gamma^{-1}(m_j)$. Заменим в условии (2.1) левые части этих равенств на правые и применим Γ^{-1} к обеим частям получившегося равенства. Так как $\Gamma^{-1}\Gamma$ является тождественным отображением: $\Gamma^{-1}\Gamma(a) = a$, то получим:

$$\phi_i(\Gamma^{-1}(m_{i1}), \dots, \Gamma^{-1}(m_{il(i)})) = \Gamma^{-1}\psi_i(m_{i1}, \dots, m_{il(i)}). \quad (2.2)$$

Равенство (2.2) — это то же равенство (2.1) с заменой Γ на Γ^{-1} , элементов K на элементы M и переменой местами ϕ_i и ψ_i ; иначе говоря, Γ^{-1} — это изоморфизм B на A . Итак, если существует изоморфизм A на B , то существует изоморфизм B на A ; при этом алгебры A и B называются *изоморфными*. Мощности основных множеств изоморфных алгебр равны (при гомоморфизме это равенство может не выполняться). Если $A = B$, то изоморфизм называется изоморфизмом на себя, или автоморфизмом; если $B \subset A$, то изоморфизм называется изоморфизмом в себя.

Пример 2.2. а. Пусть Q_N — множество всех целых чисел, Q_{2N} — множество всех четных чисел. Алгебры $(Q_N; +)$ и $(Q_{2N}; +)$ изоморфны; изоморфизмом является отображение $\Gamma_{2n}: n \rightarrow 2n$, причем условие (2.1) здесь имеет вид $2(a + b) = 2a + 2b$. Поскольку $Q_{2N} \subset Q_N$, то Γ_{2n} — изоморфизм $(Q_N; +)$ в себя. Отображение $\Gamma_{-n}: n \rightarrow (-n)$ является для алгебры $(Q_N; +)$ автоморфизмом; условие (2.1) имеет вид $(-a) + (-b) = -(a + b)$. Для алгебры $(Q_N; \cdot)$ Γ_{-n} не является автоморфизмом, так как $(-a)(-b) \neq -(ab)$.

б. Рассмотрим алгебры $(N; +, \cdot)$ и $(N_7; \oplus, \odot)$ (см. пример 2.1) и определим отображение $\Gamma_7: N \rightarrow N_7$ следующим образом: $\Gamma_7(n)$ равно остатку от деления n на 7; иначе говоря, если $n = 7a + b$ ($b < 7$), то $\Gamma_7(n) = b$. Пусть $n_1 = 7a_1 + b_1$; $n_2 = 7a_2 + b_2$. Проверим условие (2.1). Для сложения имеем $\Gamma_7(n_1 + n_2) = \Gamma_7(b_1 + b_2) = b_1 \oplus b_2 = \Gamma_7(n_1) \oplus \Gamma_7(n_2)$. Для умножения имеем $\Gamma_7(n_1 n_2) = \Gamma_7(b_1 b_2) = b_1 \odot b_2 = \Gamma_7(n_1) \odot \Gamma_7(n_2)$. Таким образом, условие (2.1) выполнено и Γ_7 — гомоморфизм. Очевидно, Γ_7 не является

изоморфизмом, так как нет взаимной однозначности. Этот пример показывает, что возможен гомоморфизм бесконечной алгебры (т. е. алгебры с бесконечным основным множеством) в конечную алгебру. При этом N разбивается на семь классов эквивалентности по отношению E_7 : aE_7b , если и только если $\Gamma_7(a) = \Gamma_7(b)$ (см. пример 1.12, г).

в. Изоморфизмом между алгебрами (R_+, \cdot) и $(R, +)$, где R_+ — положительное подмножество R , является отображение $a \rightarrow \log a$. Условие (2.1) имеет вид равенства $\log ab = \log a + \log b$.

г. Рассмотрим алгебры (K, φ) и (M, ψ) , где $K = \{a_1, a_2, a_3, a_4\}$; $M = \{b_1, b_2, b_3, b_4\}$, а бинарные операции φ и ψ заданы таблицами 2.3, а, б.

Отображение Γ : $a_1 \rightarrow b_3, a_2 \rightarrow b_1, a_3 \rightarrow b_2, a_4 \rightarrow b_4$ является изоморфизмом.

Буквальная проверка условия (2.1) состоит в следующем: в клетках (во внутренней части) таблицы φ заменяем a_i на b_j в соответствии с Γ и получаем левую часть (2.1), т. е. таблицу функции $\Gamma_\varphi(a_i, a_j)$; во внешней части таблицы ψ заменяем b_j на a_i и получаем правую часть (2.1); сравнением полученных двух таблиц убеждаемся, что они задают одну и ту же функцию. В действительности достаточно в таблице φ переименовать все a_i в b_j и сравнить полученную таблицу с ψ .

Заметим, что можно было бы рассматривать алгебры (K, φ) и (K, ψ) , где в таблице ψ все b_i заменены на a_i (с тем же индексом). Тогда отображение Γ : $a_1 \rightarrow a_3, a_2 \rightarrow a_1, a_3 \rightarrow a_2, a_4 \rightarrow a_4$ также является изоморфизмом.

д. Булевые алгебры (см. пример 2.1, в), образованные двумя различными множествами U, U' одинаковой мощности, изоморфны: операции у них просто одинаковы, а отображением Γ может служить любое взаимно однозначное соответствие между U и U' .

а	a_1	a_2	a_3	a_4
φ	a_1	a_2	a_3	a_4
a_1	a_3	a_2	a_2	a_1
a_2	a_1	a_4	a_4	a_2
a_3	a_4	a_2	a_2	a_1
a_4	a_1	a_1	a_3	a_3

б	<i>Таблица 2.3</i>			
ψ	b_1	b_2	b_3	b_4
b_1	b_4	b_4	b_3	b_1
b_2	b_1	b_1	b_4	b_3
b_3	b_1	b_1	b_2	b_3
b_4	b_3	b_2	b_3	b_2

Отношение изоморфизма является отношением эквивалентности на множестве алгебр. Рефлексивность его очевидна, симметричность следует из существования обратного изоморфизма, а транзитивность устанавливается следующим образом: если Γ_1 — изоморфизм A на B , Γ_2 — изоморфизм B на C , то изоморфизмом A на C будет композиция Γ_1 и Γ_2 . Классами эквивалентности в разбиении по отношению изоморфизма являются классы изоморфных между собой алгебр.

Понятие изоморфизма является одним из важнейших понятий в математике. Его существование, как видно из последних двух примеров, можно выразить следующим образом: если алгебры A и B изоморфны, то элементы и операции B можно переименовать так, что B совпадает с A . Из условия (2.1) изоморфизма следует, что любое эквивалентное соотношение в алгебре A сохраняется в любой изоморфной ей алгебре A' . Это позволяет, получив такие соотношения в алгебре A , автоматически распространить их на все алгебры, изоморфные A . Распространенное в математике выражение «рассматривать объекты с точностью до изоморфизма» означает, что рассматриваются только те свойства объектов, которые сохраняются при изоморфизме, т. е. являются общими для всех изоморфных объектов. В частности, изоморфизм сохраняет ассоциативность, коммутативность и дистрибутивность.

2.2. ПОЛУГРУППЫ, ГРУППЫ, РЕШЕТКИ

Полугруппы. Полугруппой называется алгебра с одной ассоциативной бинарной операцией. Эта операция обычно называется умножением, поэтому результат ее применения к элементам a и b записывается как $a \cdot b$ или ab . Такая запись называется мультиликативной. В частности, aa принято записывать как a^2 , aaa как a^3 и т. д. В общем случае $ab \neq ba$. Если же умножение коммутативно, то полугруппа называется коммутативной, или абелевой. Если полугруппа содержит такой элемент e , что для любого a $ae = ea = a$, то e называется единицей. Полугруппа с единицей называется моноидом. Единица в полугруппе всегда единственна. Действительно, если

есть две единицы e_1 и e_2 , то $e_1e_2 = e_1$ и $e_1e_2 = e_2$; следовательно, $e_1 = e_2$.

Композиция отображений является ассоциативной операцией. Поэтому всякое множество преобразований (отображений некоторого множества в себя), замкнутое относительно композиции, является полугруппой. Рассмотрим пример. Пусть на множестве $\{1, 2, 3\}$ заданы преобразования

$$\alpha = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 2 \end{pmatrix} \text{ и } \beta = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 3 & 2 \end{pmatrix}.$$

Их произведения имеют вид

$$\alpha = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 3 & 3 \end{pmatrix} \text{ и } \beta = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 2 & 2 \end{pmatrix}.$$

т. е. не совпадают с α и β . Поэтому множество $\{\alpha, \beta\}$ не замкнуто относительно композиции и не образует полугруппы. Однако если к нему добавить преобразования

$$\gamma = \beta^2 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 2 & 3 \end{pmatrix}, \delta = \alpha\beta \text{ и } \xi = \beta\alpha,$$

то можно убедиться, что полученное множество $\Gamma = \{\alpha, \beta, \gamma, \delta, \xi\}$ вместе с операцией композиции образует подгруппу. Таблица Кэли этой полу-

группы имеет вид табл. 2.4.

Если же к Γ добавить тождественное отображение

$$\varepsilon = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix},$$

получим полугруппу с единицей.

Теорема 2.1. Любая полугруппа с единицей изоморфна некоторой полугруппе преобразований.

Действительно, пусть задана подгруппа с множеством $M = \{e, a_1, a_2 \dots\}$. Каждому элементу a_i полугруппы поставим в соответствие преобразование f_i множества M следующим образом: $f_i(x) = xa_i$ для всех $x \in M$. Тогда произведению $a_i a_j$ будет соответствовать преобразование $f_{ij}(x) = xa_i a_j = f_j(f_i(x))$, т. е. композиция преобразований f_i и f_j , следовательно, условие (2.1) гомоморфизма выполнено. Кроме того, разным элементам соответствуют

разные отображения, так как $f_i(e) = a_i$, $f_j(e) = a_j$, и, следовательно, $f_i \neq f_j$. Таким образом, соответствие $a_i \rightarrow f_i(x)$ является изоморфизмом.

Если любой элемент полугруппы $P = (M; \circ)$ можно представить как произведение некоторого числа элементов множества $M_0 \subset M$, то множество M_0 называется порождающим множеством или системой образующих полугруппы P , а его элементы называются *образующими*. В нашем примере образующими являются α и β , так как $\gamma = \beta^2$, $\delta = \alpha\beta$, $\xi = \beta\alpha$. В полугруппе $(N; \circ)$ порождающим множеством служит бесконечное множество простых чисел. Если полугруппа имеет только одну образующую, то все элементы являются степенями этой образующей. Такая полугруппа называется *циклической*. Циклической полугруппой является, например, полугруппа $(N; +)$, так как все натуральные числа — это суммы некоторого количества единиц. Пусть полугруппа P имеет конечное множество образующих $\{a_1, \dots, a_n\}$. Если обозначения операции опустить (как это обычно делается для умножения), то все элементы P можно рассматривать как слова в алфавите $\{a_1, \dots, a_n\}$. Некоторые различные слова могут оказаться равными как элементы. В нашем примере полугруппы преобразований выполняются, например, равенства $\beta^3 = \beta$, $\beta\alpha = \alpha\beta^2$. В коммутативной полугруппе для любых элементов a, b выполняются равенства $ab = ba$. Такие равенства называются *определяющими соотношениями*. Если же в полугруппе нет определяющих соотношений, т. е. любые два различных слова являются различными элементами полугруппы, то полугруппа называется *свободной*.

Всякую полугруппу можно получить из свободной полугруппы введением некоторых определяющих соотношений. Элементы заданной так полугруппы — это слова в алфавите образующих, причем некоторые слова равны (т. е. задают один и тот же элемент) в силу определяющих соотношений. Отношение равенства слов является отношением эквивалентности. Из любого слова, используя определяющие соотношения, легко можно получить различные эквивалентные ему слова. Намного сложнее проблема: для двух данных слов выяснить, можно ли получить одно из другого, используя определяющие соотношения.

Ее исследование оказало значительное влияние на теорию алгоритмов. Более точная постановка этой проблемы будет рассмотрена в § 6.4.

Группы. Группой называется полугруппа с единицей, в которой для каждого элемента a существует элемент a^{-1} , называемый обратным к a и удовлетворяющий условию $aa^{-1} = a^{-1}a = e$. Число элементов группы называется порядком группы. Группа, в которой операция коммутативна, называется коммутативной, или абелевой. Группа, все элементы которой являются степенями одного элемента a , называется циклической. Циклическая группа всегда абелева. Для абелевых групп часто употребляется аддитивная запись: операция обозначается как сложение, а единица обозначается 0.

Пример 2.3. а. Множество рациональных чисел, не содержащее нуля, с операцией умножения является абелевой группой. Обратным к элементу a является элемент $1/a$.

б. Множество целых чисел с операцией сложения является абелевой циклической группой. Роль единицы здесь играет 0, обратным к элементу a является элемент $-a$.

в. Множество невырожденных квадратных матриц порядка n (с отличным от 0 определителем) с операцией умножения является некоммутативной группой.

г. Множество $\{0, 1, 2, 3, 4\}$ с операцией «сложение по mod 5» — конечная абелева циклическая группа. Ее единицей является 0. В этой группе $3^{-1} = 2, 1^{-1} = 4$.

д. Алгебра $\{O; \circ\}$ из примера 2.1, е, где O — множество поворотов квадрата, а \circ — их композиция, является циклической группой: $\gamma = \beta^2, \delta = \beta^3, \alpha = \beta^4$. Единицей в ней служит тождественное отображение α (поворот на нулевой угол); обратным к данному повороту служит поворот, дополняющий его до 2π : $\beta^{-1} = \delta, \gamma^{-1} = \gamma, \delta^{-1} = \beta$.

е. Рассмотрим множество S всех взаимно однозначных преобразований конечного множества M в себя. Такие преобразования называются подстановками. Алгебра

$$\Sigma_M = \{S_M; \circ\}$$

представляет собой группу, которая называется симметрической. Поскольку число подстановок равно числу пе-

рестановок в списке элементов M , то порядок Σ_M равен $|M|!$ Симметрична группа не является абелевой. Пусть, например,

$$M = \{1, 2, 3, 4\}, \alpha = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 3 & 1 \end{pmatrix}, \beta = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 2 & 3 \end{pmatrix}.$$

Тогда

$$\alpha\beta = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 3 & 2 & 4 \end{pmatrix}, \beta\alpha = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 4 & 3 \end{pmatrix}, \alpha^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 3 & 2 \end{pmatrix}, \beta^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \end{pmatrix}.$$

В любой конечной группе ее операция (умножение) может быть задана таблицей Кэли. Для групп таблица Кэли имеет важную особенность: любой ее столбец содержит все элементы группы. Действительно, если столбец a_i не содержит какого-нибудь элемента, то некоторый другой элемент a_j в нем должен встретиться дважды, скажем, в k -й и l -й строках. Но тогда $a_k a_i = a_j$, $a_l a_i = a_j$ и, следовательно, $a_k a_i = a_l a_i$. Умножая обе части равенства на a_i^{-1} , получаем $a_k = a_l$, что неверно. Таким образом, i -й столбец Кэли, т. е. умножение на a_i , является подстановкой на множестве элементов группы. Проверив, что это соответствие является изоморфизмом (аналогичную проверку мы делали для полугрупп преобразований), получаем теорему Кэли.

Теорема 2.2. Любая конечная группа изоморфна группе подстановок на множестве ее элементов. \square

Из сравнения теорем о связи полугрупп с преобразованиями и групп с подстановками видно, что группа — это полугруппа взаимно однозначных преобразований, причем именно взаимная однозначность гарантирует наличие обратного преобразования. Можно сказать, что в группе при любом числе умножений не теряется информация об исходном элементе: если известно, на что умножали, всегда можно узнать, что умножали. Для полугруппы это верно не всегда. Используя терминологию дискретных систем (например, конечных автоматов, о которых будет идти речь в гл. 7), то же самое можно сказать следующим образом. Пусть имеется дискретная система с конечным числом состояний $S = \{s_1, \dots, s_n\}$, на вход которой может быть подано входное воздействие из множества $\{x_1, \dots, x_m\}$. Всякое входное воздействие однозначно переводит состояние системы в некоторое другое

состояние и, следовательно, является преобразованием множества S . Последовательности воздействий — это композиции преобразований; следовательно, множество всех последовательностей является полугруппой с образующими $\{x_1, \dots, x_m\}$. Если такая полугруппа оказывается группой, то по любой входной последовательности и заключительному состоянию системы можно однозначно определить начальное состояние системы.

Алгебраические системы. Решетки. До сих пор рассматривались алгебры, т. е. множества, на которых заданы операции. Множества, на которых, кроме операций, заданы отношения, называются *алгебраическими системами* [16]. Таким образом, алгебры можно считать частным случаем алгебраических систем (в которых множество отношений пусто). Другим частным случаем алгебраических систем являются *модели* — множества, на которых заданы только отношения. Понятие изоморфизма для алгебраических систем вводится аналогично тому, как это было сделано ранее для алгебр, с той разницей, что к условию (2.1) сохранения операций добавляется условие сохранения отношений при изоморфизме.

Рассмотрим здесь лишь один пример алгебраической системы, который наиболее часто встречается в теоретической алгебре и ее применениях. Этот пример — решетка.

Пусть задано частично упорядоченное множество M . Отношение порядка в дальнейшем будем обозначать \leq . Для элементов a и b из M их верхней гранью называется любой элемент $c \in M$, такой, что $c \geq a$, $c \geq b$, а их нижней гранью — любой элемент $d \in M$, такой, что $d \leq a$, $d \leq b$. В общем случае для некоторых элементов a и b верхняя или нижняя грань может не существовать или быть не единственной, причем различные верхние (или нижние) грани могут быть не сравнимы.

Решеткой называется частично упорядоченное множество, в котором для любых двух элементов a и b существует их пересечение $a \cap b = c$ — такая нижняя грань a и b , что любая другая нижняя грань a и b меньше c ; их объединение $a \cup b = d$ — такая верхняя грань a и b , что любая другая верхняя грань a и b больше d . Таким образом, решетка — это алгебраическая система $\{M; \leq; \cap, \cup\}$.

с одним бинарным отношением и двумя бинарными операциями. Отметим, что операции \cap и \cup здесь понимаются как абстрактные операции алгебраической системы и отличаются от теоретико-множественных операций объединения и пересечения, определенных в § 1.1 (в частных случаях могут с ними совпадать — см. ниже пример 2.4, в).

Пересечение и объединение ассоциативны (предлагаем читателю это доказать!), поэтому можно говорить о пересечении и объединении любого конечного подмножества элементов решетки.

Пример 2.4. а. Любое линейно упорядоченное множество (например, множество целых чисел) можно превратить в решетку, определив для любых $a, b \in M$ $a \cup b = \max(a, b)$, $a \cap b = \min(a, b)$.

б. Определим на N отношение частичного порядка следующим образом: $a \leq b$, если a делит b . Тогда $a \cup b$ — наименьшее общее кратное a и b , $a \cap b$ — наибольший общий делитель a, b . Например, $9 \cup 12 = 36$, $9 \cap 12 = 3$, $5 \cap 7 = 1$, $5 \cup 7 = 35$.

в. Система всех подмножеств $\mathfrak{B}(A) = \{M_i\}$ любого множества частично упорядочена по включению: $M_i \leq M_j$, если и только если $M_i \subseteq M_j$. Эта система является решеткой, элементами которой являются множества, а операциями — обычные теоретико-множественные операции объединения и пересечения (см. пример 2.1, в).

г. Рассмотрим множество B_n двоичных векторов длины n , частично упорядоченное так, как это сделано в примере 1.14, б. Для двоичных векторов это упорядочение выглядит так: $v \leq w$, если в векторе w единицы стоят на всех тех местах, на которых они стоят в v (и, быть может, еще на некоторых). Например, $(010) \leq (011)$, а (010) и (100) не сравнимы. Множество B_n , упорядоченное таким образом, является решеткой; в ней $v \cup w$ — это вектор, в котором единицы стоят на тех (и только тех) местах, где есть единицы либо в v , либо в w , а $v \cap w$ — это вектор, в котором единицы стоят на тех и только тех местах, где единицы есть и в v , и в w . Например, $(010) \cup (100) = (110)$, $(010) \cap (100) = (000)$. При доказательстве теоремы 1.2 было установлено взаимно однозначное соответствие между множеством B_n и системой всех подмножеств любого множества A мощности n . Легко

проверить, что это соответствие является изоморфизмом соответствующих решеток; таким образом, решетка, описанная в примере 2.4, в, и решетка из настоящего примера изоморфны.

д. На множестве $P = \{\pi_1, \dots, \pi_m\}$ всех возможных разбиений конечного множества M решетка строится следующим образом. Частичный порядок: $\pi_i \leq \pi_j$, если любые два элемента M , которые находятся в одном блоке разбиения π_i , находятся в одном блоке разбиения π_j ; иначе говоря, если любой блок π_i является подмножеством некоторого блока π_j . Например, для $M = \{a, b, c, d, e, f\}$ разбиение $\pi_1 = \{ab, c, de, f\}$ меньше разбиения $\pi_2 = \{abf, cde\}$. Минимальным разбиением является разбиение $\pi_{\min} = \{a, b, c, d, e, f\}$, в котором каждый блок состоит из одного элемента; максимальным разбиением является разбиение $\pi_{\max} = \{abcdef\}$, состоящее из одного блока. Пересечение π_i и π_j — это разбиение π_k , в котором два элемента содержатся в одном блоке, если и только если они содержатся в одном блоке и в π_i , и в π_j . Например, для $\pi_3 = \{a, bc, de, f\}$ $\pi_1 \cap \pi_3 = \{a, b, c, d, e, f\}$. Объединение π_i и π_j — это разбиение π_l , в котором два элемента содержатся в одном блоке, если и только если они содержатся в одном блоке в π_i или в π_j . Например, $\pi_1 \cup \pi_3 = \{abc, de, f\}$.

Конечное упорядоченное множество можно изобразить диаграммой, в которой элементам соответствуют точки; из точки a ведет стрелка в точку b , если $a < b$ и нет такого c , что $a < c < b$. Например, решетка B_3 изображается диаграммой на рис. 2.1, а.

На языке диаграмм хорошо иллюстрируются все основные понятия, связанные с решетками: $a \leq b$, если и только если существует путь из стрелок, ведущий из a в b ; верхняя грань a и b — это элемент, в который есть путь из a и из b ; нижняя грань a и b — это элемент, из которого есть путь и в a , и в b .

Когда упорядоченное множество не является решеткой? В двух случаях: 1) когда какие-либо два элемента не имеют верхней или нижней грани (на рис. 2.1, б элементы d и e , c и d не имеют верхней грани, элементы b и c не имеют нижней грани); 2) когда для некоторой пары элементов наименьшая верхняя (или наибольшая нижняя) грань не единственна (на рис. 2.1, в все элемен-

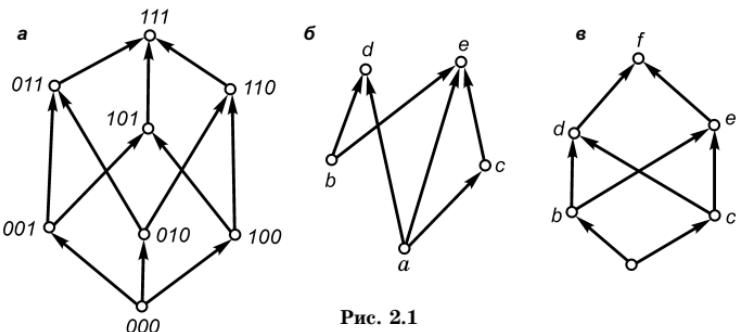


Рис. 2.1

ты имеют верхние и нижние грани, однако b и c имеют две наименьшие и несравнимые верхние грани, d и e имеют две наибольшие нижние грани, поэтому изображенное на этом рисунке множество не является решеткой).

Конкретный пример первого случая можно получить из решетки удалением некоторых ее элементов. На рис. 2.1, a видно, что после удаления $101 B_3$ остается решеткой, а после удаления 111 — нет. Удалением элементов из решетки можно получить и пример второго случая: если в B_4 удалить все элементы, кроме 0000 , 0100 , 0010 , 0111 , 1110 , 1111 , то диаграмма для оставшихся элементов в точности совпадает с рис. 2.1, b .

Решетка, в которой пересечение и объединение существуют для любого подмножества ее элементов, называется полной. Ввиду отмеченной ранее ассоциативности пересечения и объединения конечная решетка всегда полна. Объединение всех элементов полной решетки — это максимальный элемент решетки, называемый *единицей*. Пересечение всех элементов полной решетки — это минимальный элемент решетки, называемый нулем решетки. Решетка из примера 2.4, в всегда полна (в том числе и для бесконечного A). Единицей этой решетки служит само множество (содержащее любое свое подмножество), нулем — пустое множество.

ВВЕДЕНИЕ В ЛОГИКУ

3.1. ЛОГИЧЕСКИЕ ФУНКЦИИ (ФУНКЦИИ АЛГЕБРЫ ЛОГИКИ)

В этой главе особую роль будут играть двухэлементное множество B и двоичные переменные, принимающие значения из B . Его элементы часто обозначают 0 и 1, однако они не являются числами в обычном смысле (хотя по некоторым свойствам и похожи на них). Наиболее распространенная интерпретация двоичных переменных — логическая: «да» — «нет», «истинно» (И) — «ложно» (Л). В контексте, содержащем одновременно двоичные и арифметические величины и функции, эта интерпретация обычно фиксируется явно: например, в языках программирования вводится специальный тип переменной — логическая переменная, значения которой обозначаются `true` и `false`. В данной главе (за исключением § 3.4) логическая интерпретация двоичных переменных не является обязательной; поэтому будем считать, что $B = \{0, 1\}$, рассматривая 0 и 1 как формальные символы, не имеющие арифметического смысла.

Алгебра, образованная множеством B вместе со всеми возможными операциями на нем, называется *алгеброй логики*. *Функцией алгебры логики* (или *логической функцией*) от n переменных называется n -арная операция на B . Первый термин более точен, однако второй более распространен, особенно в приложениях. Он и будет использоваться в дальнейшем. Итак, логическая функция $f(x_1, \dots, x_n)$ — это функция, принимающая значения 0, 1. Множество всех логических функций обозначается P_2 , множество всех логических функций n переменных — $P_2(n)$.

Алгебра, образованная k -элементным множеством вместе со всеми операциями на нем, называется *алгеброй k -значной логики*, а n -арные операции на k -элементном множестве называются *k -значными логическими функциями n переменных*; множество всех k -значных логических функций обозначается P_k . В настоящей книге k -значные логики рассматриваться не будут; речь идет только о логических функциях из P_2 .

Всякая логическая функция n переменных может быть задана таблицей, в левой части которой перечислены все 2^n наборов значений переменных (т. е. двоичных векторов длины n), а в правой части — значения функции на этих наборах. Например, табл. 3.1 задает функцию трех переменных.

Наборы, на которых функция $f = 1$, часто называют единичными наборами функции f , а множество единичных наборов — единичным множеством f . Соответственно наборы, на которых $f = 0$, называют нулевыми наборами f .

В табл. 3.1 наборы расположены в определенном порядке — лексикографическом, который совпадает с порядком возрастания наборов, рассматриваемых как двоичные числа. Этим упорядочением будем пользоваться и в дальнейшем. При любом фиксированном упорядочении наборов логическая функция n переменных полностью определяется вектор-столбцом значений функции, т. е. 2^n . Поэтому число $|P_2(n)|$ различных функций n переменных равно числу различных двоичных векторов длины 2^n , т. е. 2^{2^n} .

Переменная x_i в функции $f(x_1, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n)$ называется *несущественной* (или *фиктивной*), если $f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n) = f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n)$ при любых значениях остальных переменных, т. е. если изменение значения x_i в любом наборе значений x_1, \dots, x_n не меняет значения функции. В этом случае функция $f(x_1, \dots, x_n)$ по существу зависит от $n-1$ переменной, т. е. представляет собой функцию $g(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$ от $n-1$ переменной. Говорят, что функция g получена из

Таблица 3.1

x_1	x_2	x_3	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

функции f удалением фиктивной переменной, а функция f получена из g введением фиктивной переменной, причем эти функции по определению считаются равными. Например, $f(x_1, x_2, x_3) = g(x_1, x_2)$ означает, что при любых значениях x_1 и x_2 $f = g$ независимо от значения x_3 .

Смысл удаления фиктивных переменных очевиден, поскольку они не влияют на значение функции и являются с этой точки зрения лишними. Однако иногда бывает полезно вводить фиктивные переменные. Благодаря такому введению всякую функцию n переменных можно сделать функцией любого большего числа переменных. Поэтому любую конечную совокупность функций можно считать зависящей от одного и того же множества переменных (являющемся объединением множеств переменных всех взятых функций), что часто бывает удобно. В частности, только что доказанное равенство $|P_2(n)| = 2^{2^n}$ справедливо при условии, что $P_2(n)$ содержит все возможные функции n переменных, в том числе и функции с фиктивными переменными.

Примеры логических функций. Логических функций одной переменной — четыре; они приведены в табл. 3.2.

Таблица 3.2

x	ϕ_0	ϕ_1	ϕ_2	ϕ_3
0	0	0	1	1
1	0	1	0	1

Функции ϕ_0 и ϕ_3 — константы 0 и 1 соответственно; их значения не зависят от значения переменной, и, следовательно, переменная x для них несущественна. Функция ϕ_1 «повторяет» x : $\phi_1(x) = x$. Функция $\phi_2(x)$ называется *отрицанием* x (или функцией НЕ) и обозначается \bar{x} , $\neg x$, x' , $\sim x^*$. Ее значение противоположно значению x .

Логических функций двух переменных — 16; они приведены в табл. 3.3.

Функции ψ_0 и ψ_{15} — константы 0 и 1, т. е. функции с двумя несущественными переменными. Отметим, что формально это функции отличаются от ϕ_0 и ϕ_3 в табл. 3.2;

* «Черта сверху» — традиционное обозначение отрицания, которое из-за его привычности сохранено и в этой главе. Однако оно не очень удобно для формальных определений, а в приложениях — для языков программирования, где все символы должны располагаться в строчку. Поэтому все чаще употребляется символ \neg , который будет использоваться в гл. 6.

x_1	x_2	Ψ_0	Ψ_1	Ψ_2	Ψ_3	Ψ_4	Ψ_5	Ψ_6	Ψ_7	Ψ_8	Ψ_9	Ψ_{10}	Ψ_{11}	Ψ_{12}	Ψ_{13}	Ψ_{14}	Ψ_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

все функции в табл. 3.2 — унарные операции на B , а все функции в табл. 3.3 — бинарные операции на B . Однако ранее уже было принято функции, отличающиеся лишь несущественными переменными, считать равными.

Функция $\psi_1(x_1, x_2)$ называется *конъюнкцией* x_1 и x_2 ; ее обозначения: $x_1 \& x_2$, $x_1 \wedge x_2$, $x_1 \cdot x_2$ (во всех случаях знак конъюнкции аналогично умножению часто опускают и пишут x_1x_2). В этой книге конъюнкция будет обозначаться знаком $\&$. Она равна 1, только если x_1 и x_2 равны 1, поэтому ее называют часто функцией И. Еще одно ее название — «логическое умножение», поскольку ее таблица действительно совпадает с таблицей обычного умножения для чисел 0 и 1.

Функция $\psi_7(x_1, x_2)$ называется *дизъюнкцией* x_1 и x_2 ; ее обозначения: $x_1 \wedge x_2$, $x_1 + x_2$. Она равна 1, если x_1 или x_2 равен 1 («или» здесь понимается в неразделительном смысле — хотя бы один из двух). Поэтому ее называют часто функцией ИЛИ.

Функция $\psi_6(x_1, x_2)$ — это *сложение по модулю 2* (см. пример 2.1, б). Ее обозначения: $x_1 \oplus x_2$, $x_1 \Delta x_2$, $x_1 \neq x_2$. Она равна 1, когда значения ее аргументов различны, и равна 0, когда они равны. Поэтому функцию ψ_6 иногда называют *неравнозначностью*.

Функция $\psi_9(x_1, x_2)$ называется *эквивалентностью*, или *равнозначностью*. Ее обозначения: $x_1 \sim x_2$, $x_1 \equiv x_2$. Она равна 1, когда значения ее аргументов равны, и равна 0, когда они различны.

Еще три функции имеют свои названия:

$\psi_{13}(x_1, x_2)$ — *импликация*; обозначения $x_1 \rightarrow x_2$, $x_1 \supset x_2$; читается «если x_1 , то x_2 »;

$\psi_8(x_1, x_2)$ — *стрелка Пирса*; обозначение $x_1 \downarrow x_2$;

$\psi_{14}(x_1, x_2)$ — *штрих Шеффера*; обозначение $x_1 | x_2$.

Остальные функции специальных названий не имеют и, как будет показано позднее, легко выражаются через перечисленные ранее функции.

В функциях ψ_3 и ψ_{12} переменная x_2 фиктивна; из табл. 3.3 видно, что $\psi_3(x_1, x_2) = x_1$, $\psi_{12}(x_1, x_2) = \bar{x}_1$. В функциях ψ_5 и ψ_{10} фиктивна переменная x_1 : $\psi_5(x_1, x_2) = x_2$, $\psi_{10}(x_1, x_2) = \bar{x}_2$.

Таким образом, из 16 функций двух переменных шесть функций имеют фиктивные переменные. С ростом n (числа переменных) доля функций, имеющих фиктивные переменные, убывает и стремится к нулю.

Суперпозиции и формулы. В § 1.2 было введено понятие суперпозиции функций. Напомним, что *суперпозицией функций* f_1, \dots, f_m называется функция f , полученная с помощью подстановок этих функций друг в друга и переименования переменных, а *формулой* называется выражение, описывающее эту суперпозицию. Понятие суперпозиции очень важно в алгебре логики, поэтому рассмотрим его более подробно.

Пусть дано множество (конечное или бесконечное) исходных функций $\Sigma = \{f_1, \dots, f_m\}$. Символы переменных x_1, \dots, x_n, \dots будем считать *формулами глубины 0*. Формула F имеет *глубину* $k + 1$, если F имеет вид $f_i(F_1, \dots, F_{n_i})$, где $f_i \in \Sigma$, n_i — число аргументов f_i , а F_1, \dots, F_{n_i} — формулы, максимальная из глубин которых равна k . F_1, \dots, F_{n_i} называются *подформулами* F ; f_i называется *внешней* или *главной операцией* формулы F . Все подформулы формул F_1, \dots, F_{n_i} также называются подформулами F . Например, $f_2(x_1, x_2, x_3)$ — это формула глубины 1, а $f_3(f_1(x_3, x_1), f_2(x_1, f_3(x_1, x_2)))$ — формула глубины 3, содержащая одну подформулу глубины 2 и две подформулы глубины 1.

В дальнейшем конкретные формулы, как правило, будут иметь более привычный вид, при котором знаки функций стоят между аргументами (такую запись называют *инфиксной*). Например, если f_1 обозначает дизъюнкцию, f_2 — конъюнкцию, а f_3 — сложение по mod 2, то приведенная ранее формула примет вид:

$$(x_3 \vee x_1) \oplus (x_1 \& (x_1 \oplus x_2)). \quad (3.1)$$

Все формулы, построенные описанным ранее образом, т. е. содержащие только символы переменных, скоб-

ки и знаки функций из множества Σ , называются *формулами над Σ* .

Возможны и другие варианты понятия глубины. Например, часто считается, что расстановка отрицаний над переменными не увеличивает глубины; в случае, когда Σ содержит ассоциативную операцию f , можно определить глубину так, что применение f к формулам с той же внешней операцией f не увеличивает глубину формулы. Например, $x_1(x_2 \vee x_3x_4)$ и $x_2x_1(x_2 \vee x_3x_4)$ имеют одну и ту же глубину 3; ДНФ (см. далее) всегда имеет глубину 2.

Всякая формула, выражающая функцию f как суперпозицию других функций, задает способ ее вычисления (при условии, что известно, как вычислить исходные функции). Этот способ определяется следующим очевидным правилом: формулу можно вычислить, только если уже вычислены значения всех ее подформул. Вычислим, например, формулу (3.1) на наборе $x_1 = 1, x_2 = 1, x_3 = 0$. Получим (используя табл. 3.2): $x_3 \vee x_1 = 1; x_1(x_1 \oplus x_2) = x_1 \& 0 = 0; (x_3 \vee x_1) \oplus (x_1(x_1 \oplus x_2)) = 1 \oplus 0 = 1$.

Таким образом, формула каждому набору значений аргументов ставит в соответствие значение функции и, следовательно, может служить наряду с таблицей способом задания и вычисления функции. В частности, по формуле, вычисляя ее на всех 2^n наборах, можно восстановить таблицу функции. О формуле, задающей функцию, говорят также, что она *реализует* или *представляет* эту функцию.

В отличие от табличного задания представление данной функции формулой не единственно. Например, если в качестве исходного множества функций зафиксировать множество $\{\psi_1, \psi_7, \psi_2\}$, т. е. конъюнкцию, дизъюнкцию и отрицание, то функцию штрих Шеффера можно представить формулами

$$\bar{x}_1 \vee \bar{x}_2 \text{ и } \overline{x_1 x_2}, \quad (3.2)$$

а функцию стрелка Пирса — формулами

$$\bar{x}_1 \bar{x}_2 \text{ и } \overline{x_1 \vee x_2}. \quad (3.3)$$

Формулы, представляющие одну и ту же функцию, называются *эквивалентными* или *равносильными* (см. пример 1.12, б). Эквивалентность формул обозначается

знаком равенства; поэтому можно записать $\psi_{14}(x_1, x_2) = \bar{x}_1 \vee \bar{x}_2 = \bar{x}_1 \bar{x}_2$. Как для двух данных формул выяснить, эквивалентны они или нет? Существует стандартный метод, всегда приводящий к ответу: по каждой формуле восстанавливается таблица* функции, а затем полученные две таблицы сравниваются. Иначе говоря, для каждого набора значений переменных проверяется, равны ли на нем значения формул. Этот метод требует $2 \cdot 2^n$ вычислений (если считать, что обе формулы зависят от n переменных) и на практике оказывается слишком громоздким. Существуют и другие методы установления эквивалентности формул и получения новых формул, эквивалентных исходной. К этим методам, называемым эквивалентными преобразованиями формул, мы еще вернемся.

3.2. БУЛЕВА АЛГЕБРА

В этом параграфе будут рассмотрены представления логических функций в виде суперпозиций дизъюнкций, конъюнкций и отрицаний.

Разложение функций по переменным. Совершенная дизъюнктивная нормальная форма. Введем обозначение $x^0 = x$, $x^1 = \bar{x}$. Пусть α — параметр, равный 0 или 1. Тогда $x^\alpha = 1$, если $x = \alpha$, и $x^\alpha = 0$, если $x \neq \alpha$.

Теорема 3.1. Всякая логическая функция $f(x_1, \dots, x_n)$ может быть представлена в следующем виде:

$$\begin{aligned} f(x_1, \dots, x_m, x_{m+1}, \dots, x_n) &= \\ &= \bigvee_{\alpha_1, \dots, \alpha_m} x_1^{\alpha_1} \dots x_m^{\alpha_m} f(\alpha_1, \dots, \alpha_m, x_{m+1}, \dots, x_n), \end{aligned}$$

где $m \leq n$, а дизъюнкция берется по всем 2^m наборам значений переменных x_1, \dots, x_m .

Это равенство называется *разложением по переменным* x_1, \dots, x_m . Например, при $n = 4$, $m = 2$ разложение (3.4) имеет вид:

$$\begin{aligned} f(x_1, x_2, x_3, x_4) &= \bar{x}_1 \bar{x}_2 f(0, 0, x_3, x_4) \vee \bar{x}_1 x_2 f(0, 1, x_3, x_4) \vee \\ &\quad \vee x_1 \bar{x}_2 f(1, 0, x_3, x_4) \vee x_1 x_2 f(1, 1, x_3, x_4). \end{aligned}$$

*До сих пор все, что говорилось о формулах и суперпозициях, было справедливо не только для логических, но и для любых функций вообще (см. § 1.2). Данный же метод проверки эквивалентности годится только для функций с конечными областями определения.

Теорема доказывается подстановкой в обе части равенства (3.4) произвольного набора $(\sigma_1, \dots, \sigma_m, \sigma_{m+1}, \dots, \sigma_n)$ всех n переменных. Так как $x^\alpha = 1$, только когда $x = \alpha$, то среди 2^m конъюнкций $x_1^{\alpha_1}, \dots, x_m^{\alpha_m}$ правой части (3.4) в 1 обратится только одна — та, в которой $\alpha_1 = \sigma_1, \dots, \alpha_m = \sigma_m$. Все остальные конъюнкции равны 0. Поэтому получим:

$$f(\sigma_1, \dots, \sigma_n) = \sigma_1^{\sigma_1} \dots \sigma_m^{\sigma_m} f(\sigma_1, \dots, \sigma_m, \sigma_{m+1}, \dots, \sigma_n) = \\ = f(\sigma_1, \dots, \sigma_n),$$

т. е. тождество. \square

При $m = 1$ из (3.4) получаем разложение функции по одной переменной

$$f(x_1, x_2, \dots, x_n) = \bar{x}_1 f(0, x_2, \dots, x_n) \vee x_1 f(1, x_2, \dots, x_n). \quad (3.5)$$

Ясно, что аналогичное разложение справедливо для любой из n переменных.

Другой важный случай — разложение по всем n переменным ($m = n$). При этом все переменные в правой части (3.4) получают фиксированные значения и функции в конъюнкциях правой части становятся равными 0 или 1, что дает:

$$f(x_1, \dots, x_n) = \bigvee_{f(\sigma_1, \dots, \sigma_n)=1} x_1^{\sigma_1} \dots x_n^{\sigma_n}, \quad (3.6)$$

где дизъюнкция берется по всем наборам $(\sigma_1, \dots, \sigma_n)$, на которых $f = 1$. Такое разложение называется *совершенной дизъюнктивной нормальной формой* (СДНФ) функции f . СДНФ функции f содержит ровно столько конъюнкций, сколько единиц в таблице f ; каждому единичному набору $(\sigma_1, \dots, \sigma_n)$ соответствует конъюнкция всех переменных, в которых x_i взято с отрицанием, если $\sigma_i = 0$, и без отрицания, если $\sigma_i = 1$. Таким образом, существует взаимно однозначное соответствие между таблицей функции $f(x_1, \dots, x_n)$ и ее СДНФ, и, следовательно, СДНФ для всякой логической функции единственна (точнее, единственна с точностью до порядка букв и конъюнкций: это означает, что ввиду коммутативности дизъюнкций и конъюнкций — см. далее (3.8) — формулы, получаемые из (3.6) перестановкой конъюнкций и букв конъюнкции, не различаются и считаются одной и той же СДНФ). Например, функция, заданная табл. 3.1, имеет СДНФ

$$\bar{x}_1 \bar{x}_2 x_3 \vee x_1 \bar{x}_2 \bar{x}_3 \vee x_1 \bar{x}_2 x_3.$$

Единственная функция, не имеющая СДНФ, — это константа 0.

Формулы, содержащие, кроме переменных (и, разумеется, скобок), только знаки функций дизъюнкции, конъюнкции и отрицания, будут называться *булевыми формулами* (напомним, что знак конъюнкции, как правило, опускается).

Соотношение (3.6) приводит к важной теореме.

Теорема 3.2. Всякая логическая функция может быть представлена булевой формулой, т. е. как суперпозиция конъюнкций, дизъюнкций и отрицания.

Действительно, для всякой функции, кроме константы 0, таким представлением может служить ее СДНФ. Константу 0 можно представить булевой формулой $\bar{x}\bar{x}$ [см. далее равенство (3.15)]. \square

Булева алгебра функций и эквивалентные преобразования в ней. Пусть функция f_1 задана формулой F_1 , а функция f_2 — формулой F_2 . Подстановка F_1 и F_2 в дизъюнкцию $x_1 \vee x_2$ дает формулу $F_1 \vee F_2$. Если взять формулу F'_1 , эквивалентную F_1 (т. е. тоже представляющую f_1), и F'_2 , эквивалентную F_2 , то получим формулу $F'_1 \vee F'_2$, эквивалентную $F_1 \vee F_2$ (доказательство эквивалентности можно провести стандартным методом (см. § 3.1); рекомендуем читателю его проделать). Таким образом, дизъюнкцию можно рассматривать как бинарную операцию на множестве логических функций, которая каждой паре функций f_1, f_2 независимо от вида формул, которыми они представлены, однозначно ставит в соответствие функцию $f_1 \vee f_2$. Аналогично этому можно рассматривать конъюнкцию как бинарную операцию, а отрицание — как унарную операцию над функциями.

Алгебра $(P_2; \vee, \&, \neg)$, основным множеством которой является все множество логических функций, а операциями — дизъюнкция, конъюнкция и отрицание, называется *булевой алгеброй логических функций*. Операции булевой алгебры также часто называют *булевыми операциями*.

Фактически мы имеем дело, как правило, не с самими функциями в чистом виде, а с представляющими их формулами, т. е. с алгеброй формул, которых гораздо больше, чем функций, — ведь каждую функцию представляет бесконеч-

ное множество формул. Для того чтобы алгебра формул соответствовала алгебре функций, ей придается следующий вид. Элементами основного множества алгебры формул объявляются не формулы, а классы эквивалентности формул, т. е. классы формул, представляющих одну и ту же функцию. Результатом, например, дизъюнкции классов K_1 и K_2 считается класс всех формул, эквивалентных $F_1 \vee F_2$, где $F_1 \in K_1$, $F_2 \in K_2$. Так, определенная алгебра классов формул называется алгеброй Линденбаума–Тарского. Она изоморфна булевой алгебре функций. Этот изоморфизм настолько очевиден, что часто — особенно в прикладных работах — возникает смешение понятий формулы и функции. Следует ясно представлять себе, что, например, логическая схема на своем выходе реализует функцию от входов; когда же речь идет об эквивалентных преобразованиях, об упрощении и т. д., то имеются в виду преобразования формул, реализующих одну и ту же функцию.

Рассмотрим теперь основные свойства булевых операций.

Ассоциативность:

$$\begin{aligned} \text{а) } &x_1(x_2x_3) = (x_1x_2)x_3; \\ \text{б) } &(x_1 \vee x_2) \vee x_3 = x_1 \vee (x_2 \vee x_3). \end{aligned} \quad (3.7)$$

Коммутативность:

$$\text{а) } x_1x_2 = x_2x_1; \text{ б) } x_1 \vee x_2 = x_2 \vee x_1. \quad (3.8)$$

Дистрибутивность конъюнкции относительно дизъюнкции:

$$x_1(x_2 \vee x_3) = x_1x_2 \vee x_1x_3. \quad (3.9)$$

Дистрибутивность дизъюнкции относительно конъюнкции:

$$x_1 \vee (x_2x_3) = (x_1 \vee x_2)(x_1 \vee x_3). \quad (3.10)$$

Идемпотентность:

$$\text{а) } xx = x; \text{ б) } x \vee x = x. \quad (3.11)$$

Двойное отрицание:

$$\bar{\bar{x}} = x. \quad (3.12)$$

Свойства констант:

$$\left. \begin{aligned} \text{а) } &x \& 1 = x; \text{ б) } x \& 0 = 0; \text{ в) } x \vee 1 = 1; \\ \text{г) } &x \vee 0 = x; \text{ д) } \bar{0} = 1; \text{ е) } \bar{1} = 0. \end{aligned} \right\} \quad (3.13)$$

Правила де Моргана:

$$\text{а) } \overline{x_1x_2} = \bar{x}_1 \vee \bar{x}_2; \text{ б) } \overline{x_1 \vee x_2} = \bar{x}_1 \bar{x}_2. \quad (3.14)$$

Закон противоречия:

$$x\bar{x} = 0. \quad (3.15)$$

Закон «исключенного третьего»:

$$x \vee \bar{x} = 1. \quad (3.16)$$

Соотношения (3.7)–(3.16) можно проверить указанным ранее стандартным методом — вычислением обеих частей равенств на всех наборах значений переменных. Ясно, что результат вычисления не зависит от того, как получены значения переменных, входящих в эти равенства, т. е. от того, являются ли эти переменные независимыми или, в свою очередь, получены в результате каких-то вычислений. Поэтому равенства (3.7)–(3.16) остаются справедливыми при подстановке вместо переменных любых логических функций и, следовательно, любых формул, представляющих эти функции. Важно лишь соблюдать следующее *правило подстановки формулы вместо переменной*: при подстановке формулы F вместо переменной x все вхождения переменной x в исходное соотношение должны быть одновременно заменены формулой F . Например, соотношение $F \vee \bar{F} = 1$, полученное из (3.16), верно для любых F , а соотношение $F \vee \bar{x} = 1$ получено из (3.16) с нарушением правила подстановки и для некоторых F может оказаться неверным. Правило подстановки позволяет получать из соотношений (3.7)–(3.16) новые эквивалентные соотношения.

Зачем нужны эквивалентные соотношения? Во всякой алгебре (в том числе и в булевой алгебре функций) равенство $F_1 = F_2$ означает, что формулы F_1 и F_2 описывают один и тот же элемент алгебры, в данном случае одну и ту же логическую функцию f_1 . Следовательно, если какая-либо формула F содержит F_1 в качестве подформулы, то замена F_1 на F_2 не изменяет элемента булевой алгебры f , над которым производятся операции в формуле F ; поэтому F' , полученная из F такой заменой, эквивалентна F . Это утверждение представляет собой *правило замены подформул*, которое позволяет, используя эквивалентные соотношения, получать формулы, эквивалентные данной.

Подчеркнем разницу между правилами подстановки и замены. При подстановке переменная заменяется на формулу; формула может быть любой, но требуется одновременная ее подстановка вместо всех вхождений пе-

ременной. При замене подформул может быть заменена любая подформула, однако не на любую другую, а только на эквивалентную ей. При этом замена всех вхождений исходной подформулы не обязательна. Пусть имеется эквивалентность $F_1 = F_2$, где F_1 и F_2 содержат переменную x . Если вместо всех вхождений x в $F_1 = F_2$ подставить произвольную формулу F , то получаются новые формулы F'_1 и F'_2 , причем не обязательно $F_1 = F'_1$, $F_2 = F'_2$; однако между собой новые формулы будут эквивалентны: $F'_1 = F'_2$. Если же в F_1 какую-либо подформулу заменить на эквивалентную ей, то получится новая формула F'_1 , эквивалентная F_1 .

Пример 3.1. Возьмем первое из соотношений (3.14) и подставим \bar{x}_1x_3 вместо x_1 . Получим $\bar{x}_1x_3x_2 = \bar{x}_1x_3 \vee \bar{x}_2$, т. е. две формулы, неэквивалентные исходным, но эквивалентные между собой. Если же в правой части нового соотношения \bar{x}_1x_3 заменить формулой $\bar{x}_1 \vee \bar{x}_3$, эквивалентной ей в силу (3.14), а в полученной подформуле \bar{x}_1 заменить на эквивалентную ей в силу (3.12) формулу x_1 , то все формулы в построенной цепи преобразований $\bar{x}_1x_3x_2 \Rightarrow \bar{x}_1x_3 \vee \bar{x}_2 \Rightarrow \bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_2 \Rightarrow x_1 \vee \bar{x}_3 \vee \bar{x}_2$ эквивалентны.

Такие преобразования, использующие эквивалентные соотношения (запас которых можно расширять с помощью правила подстановки) и правило замены, называются *эквивалентными преобразованиями*. Эквивалентные преобразования являются мощным средством доказательства эквивалентности формул, как правило, более эффективным, чем их вычисление на наборах значений переменных.

Рассмотрим некоторые основные эквивалентные преобразования в булевой алгебре и новые соотношения, получаемые с их помощью из (3.7)–(3.16). При этом будем иметь в виду, что в булевой алгебре принято опускать скобки в следующих двух случаях: а) при последовательном выполнении нескольких конъюнкций или дизъюнкций (например, вместо $(x_1x_2)x_3$ пишут $x_1x_2x_3$) — это не вызывает неоднозначности ввиду ассоциативности этих операций (3.7); б) если они являются внешними скобками у конъюнкции: например, вместо $(x_1(x_2 \vee x_3)) \vee (x_4x_5)$ пишут $x_1(x_2 \vee x_3) \vee x_4x_5$. Оба соглашения совершенно

аналогичны общепринятым опусканию скобок для умножения в арифметических формулах*.

1. Упрощение формул (т. е. получение эквивалентных формул, содержащих меньшее число символов).

a) Поглощение:

$$x \vee xy = x; \quad (3.17a)$$

$$x(x \vee y) = x. \quad (3.17b)$$

Докажем подробно первое равенство [для его доказательства используются последовательно соотношения (3.13а), (3.9), (3.13в) и (3.13а)]:

$$x \vee xy = x \& 1 \vee xy = x(1 \vee y) = x \& 1 = x.$$

Второе равенство доказывается с помощью (3.9), (3.11) и первого равенства:

$$x(x \vee y) = xx \vee xy = x \vee xy = x.$$

b) Склейивание:

$$xy \vee x\bar{y} = x. \quad (3.18)$$

Доказательство:

$$xy \vee x\bar{y} = x(y \vee \bar{y}) = x \& 1 = x.$$

v) Обобщенное склеивание:

$$xz \vee y\bar{z} \vee xy = xz \vee y\bar{z}. \quad (3.19)$$

Доказывается с помощью расщепления, т. е. применения (3.18) в обратную сторону и поглощения (3.17):

$$xz \vee y\bar{z} \vee xy = xz \vee y\bar{z} \vee xyz \vee x\bar{y}\bar{z} = xz \vee y\bar{z}.$$

$$\text{г) } x \vee \bar{x}y = x \vee y. \quad (3.20)$$

Доказательство:

$$x \vee \bar{x}y = xy \vee x\bar{y} \vee \bar{x}y = xy \vee x\bar{y} \vee xy \vee \bar{x}y = x \vee y.$$

д) Обобщением равенств (3.17а) и (3.20) является равенство

$$x_1 \vee f(x_1, x_2, \dots, x_n) = x_1 \vee f(0, x_2, \dots, x_n). \quad (3.21)$$

При доказательстве используется разложение по x_1 , (3.17а) и (3.20):

$$x_1 \vee f(x_1, x_2, \dots, x_n) = x_1 \vee \bar{x}_1 f(0, x_2, \dots, x_n) \vee$$

$$\vee x_1 f(1, x_2, \dots, x_n) = x_1 \vee f(0, x_2, \dots, x_n).$$

* Внимательный читатель должен был обратить внимание на то, что эти два соглашения, а также соотношения (3.13) использованы уже в формуле (3.4).

2. Приведение к дизъюнктивной нормальной форме (в том числе к СДНФ).

Элементарными конъюнкциями называются конъюнкции переменных или их отрицаний, в которых каждая переменная встречается не более одного раза. *Дизъюнктивной нормальной формой* (ДНФ) называется формула, имеющая вид дизъюнкции элементарных конъюнкций. Соотношение (3.19) показывает, что ДНФ функции может быть неединственной.

Приведение к ДНФ делается так. Сначала с помощью (3.12) и правил де Моргана (3.14) все отрицания «спускаются» до переменных. Затем раскрываются скобки, с помощью (3.11), (3.15) и (3.16) удаляются лишние конъюнкции и повторения переменных в конъюнкциях, а с помощью (3.13) удаляются константы.

Пример 3.2.

$$\begin{aligned} xy \vee \bar{x}(y \vee xz)(\overline{x(\bar{y} \vee z)} \vee \bar{yz}) &= xy \vee (\bar{xy} \vee \bar{x}xz)(\bar{x} \vee (\bar{\bar{y}} \vee z))\bar{yz} = \\ &= xy \vee x\bar{y}(\bar{x} \vee y\bar{z})(\bar{y} \vee \bar{z}) = xy \vee \bar{x}y(\bar{x}\bar{y} \vee y\bar{y}\bar{z} \vee \bar{x}\bar{z} \vee y\bar{z}) = \\ &= xy \vee \bar{x}y\bar{z} = y(x \vee \bar{x}\bar{z}) = xy \vee y\bar{z}. \end{aligned}$$

Всякую ДНФ можно привести к СДНФ расщеплением конъюнкций, которые содержат не все переменные, например:

$$xy \vee \bar{x}y\bar{z} = xyz \vee xy\bar{z} \vee \bar{x}y\bar{z}.$$

Если из формулы F_1 с помощью некоторых эквивалентных соотношений можно получить формулу F_2 , то F_1 можно получить из F_2 , используя *те же* эквивалентные соотношения; иначе говоря, всякое эквивалентное преобразование обратимо. Это позволяет доказать следующую теорему.

Теорема 3.3. Для любых двух эквивалентных формул F_1 и F_2 существует эквивалентное преобразование F_1 в F_2 с помощью соотношений (3.7)–(3.16).

Действительно, преобразуем F_1 и F_2 в СДНФ. Поскольку F_1 и F_2 эквивалентны, то их СДНФ совпадают. Обратив второе преобразование, получим преобразование $F_1 \Rightarrow \text{СДНФ} \Rightarrow F_2$. \square

Важность этой теоремы в том, что соотношений (3.7)–(3.16) оказывается достаточно для любого эквивалентного преобразования в булевой алгебре.

3. Приведение к конъюнктивной нормальной форме.

Аналогично ДНФ определяется *конъюнктивная нормальная форма* (КНФ) как конъюнкция элементарных дизъюнкций.

От ДНФ к КНФ перейдем следующим образом. Пусть ДНФ F имеет вид $F = k_1 \vee \dots \vee k_m$, где k_1, \dots, k_m — элементарные конъюнкции. Формулу

$$\overline{k_1 \vee \dots \vee k_m}$$

приведем к ДНФ $k'_1 \vee \dots \vee k'_m$. Тогда

$$F = \overline{\overline{k_1 \vee \dots \vee k_m}} = \overline{k'_1 \vee \dots \vee k'_m} = \bar{k}'_1 \bar{k}'_2 \dots \bar{k}'_m.$$

По правилу де Моргана отрицания элементарных конъюнкций преобразуются в элементарные дизъюнкции, что и даст КНФ.

Пример 3.3.

$$\begin{aligned} x\bar{y} \vee \bar{x}y \vee x\bar{z} &= \overline{\overline{x\bar{y}} \vee \overline{\bar{x}y} \vee \overline{x\bar{z}}} = \\ &= \overline{(\bar{x} \vee y)(x \vee \bar{y})(\bar{x} \vee z)} = \overline{\bar{x}\bar{y}} \vee xyz = (x \vee y)(\bar{x} \vee \bar{y} \vee z). \end{aligned}$$

Аналогом СДНФ является СКНФ — совершенная конъюнктивная нормальная форма, каждая элементарная дизъюнкция которой содержит все переменные. Единственная функция, не имеющая СДНФ, — константа 1.

Двойственность. Функция $f_1(x_1, \dots, x_n)$ называется *двойственной* к функции $f_2(x_1, \dots, x_n)$, если $f_1(x_1, \dots, x_n) = \bar{f}_2(\bar{x}_1, \dots, \bar{x}_n)$. Беря отрицание над обеими частями равенства и подставляя $\bar{x}_1, \dots, \bar{x}_n$ вместо x_1, \dots, x_n , получаем $\bar{f}_1(\bar{x}_1, \dots, \bar{x}_n) = \bar{f}_2(\bar{\bar{x}}_1, \dots, \bar{\bar{x}}_n) = f_2(x_1, \dots, x_n)$, т. е. f_2 двойственна к f_1 . Таким образом, отношение двойственности между функциями симметрично. Из определения двойственности ясно, что для любой функции двойственная функция определяется однозначно. В частности, может оказаться, что функция двойственна самой себе. В этом случае она называется *самодвойственной*.

Пример 3.4. Дизъюнкция двойственна конъюнкции (в силу правил де Моргана); константа 1 двойственна 0; отрицание самодвойствено. Еще один традиционный пример самодвойственной функции — $xy \vee xz \vee yz$.

Пользуясь определением двойственности, нетрудно (прямой выкладкой) доказать следующее утверждение, называемое *принципом двойственности*: если в форму-

ле F , представляющей функцию f , все знаки функций заменить соответственно на знаки двойственных функций, то полученная формула F^* будет представлять функцию f^* , двойственную f . В булевой алгебре принцип двойственности имеет более конкретный вид, вытекающий из ранее приведенных примеров: если в формуле F , представляющей функцию f , все конъюнкции заменить на дизъюнкции, дизъюнкции — на конъюнкции, 1 на 0, 0 на 1, то получим формулу F^* , представляющую функцию f^* , двойственную f .

Если функции равны, то и двойственные им функции также равны. Это позволяет с помощью принципа двойственности получать новые эквивалентные соотношения, переходя от равенства $F_1 = F_2$ с помощью указанных замен к равенству $F_1^* = F_2^*$. Примером пары соотношений, получаемых друг из друга по принципу двойственности, являются два равенства (3.17).

Булева алгебра и теория множеств. В примере 2.1, в были описаны булевые алгебры множеств. Общий термин «булева алгебра» для алгебр множеств и логических функций неслучаен.

Всякая алгебра (2, 2, 1) называется булевой алгеброй, если ее операции удовлетворяют соотношениям (3.7)–(3.16). В алгебре множеств элементами являются подмножества фиксированного («универсального») множества U (напомним, что система всех подмножеств U обозначается через $\mathcal{B}(U)$), операции $\&$ соответствует пересечение, операции \vee — объединение, операции $-$ соответствует дополнение; единицей является само множество U , нулем — пустое множество. Справедливость соотношений (3.7)–(3.16) для алгебры множеств можно показать непосредственной их проверкой. Для этого нужно рассмотреть переменные в них как множества, знаки $\&$, \vee заменить на \cap , \cup и показать, что если какой-либо элемент принадлежит множеству из левой части равенства, то он принадлежит правой части, и наоборот. Эту проверку мы предоставляем читателю.

В предыдущих главах уже отмечалось и использовалось (см. теорему 1.2) взаимно однозначное соответствие Γ между множеством $\mathcal{B}(U)$, где $U = \{a_1, \dots, a_n\}$, и множеством B_n двоичных векторов длины n : каждому

подмножеству $M \subseteq U$ соответствует двоичный вектор $\Gamma(M) = \sigma = (\sigma_1, \dots, \sigma_n)$, где $\sigma_i = 1$, если $a_i \in M$ и $\sigma_i = 0$, если $a_i \notin M$. Булева алгебра $(B_n, \vee, \&, \bar{\cdot})$ на множестве B_n определяется следующим образом: для любых векторов $\sigma = (\sigma_1, \dots, \sigma_n)$ и $\tau = (\tau_1, \dots, \tau_n)$.

$$\begin{aligned}\sigma \vee \tau &= (\sigma_1 \vee \tau_1, \sigma_2 \vee \tau_2, \dots, \sigma_n \vee \tau_n); \\ \sigma \& \tau &= (\sigma_1 \& \tau_1, \sigma_2 \& \tau_2, \dots, \sigma_n \& \tau_n); \\ \bar{\sigma} &= (\bar{\sigma}_1, \bar{\sigma}_2, \dots, \bar{\sigma}_n).\end{aligned}\quad (3.22)$$

Поскольку компоненты (разряды) σ_1 и τ_1 векторов σ и τ принимают значения 0 и 1, то указанные операции над компонентами — это просто логические операции над двоичными переменными; операции над векторами естественно назвать *покомпонентными (поразрядными) логическими операциями над двоичными векторами*. Такие операции (наряду с логическими операциями над переменными) входят, в частности, в систему команд любого современного компьютера. Выполнение их очень просто: вектор $\sigma \vee \tau$ содержит единицы во всех разрядах, в которых есть 1 либо в σ , либо в τ , вектор $\sigma \& \tau$ — в тех разрядах, в которых есть 1 и в σ и в τ ; вектор $\bar{\sigma}$ содержит единицы в тех разрядах, в которых σ содержит нули. Например, если $\sigma = 01011$, $\tau = 11010$, то $\sigma \vee \tau = 11011$, $\sigma \& \tau = 01010$, $\bar{\sigma} = 10100$.

Теорема 3.4. Если $|U| = n$, то булева алгебра $(\mathfrak{B}(U); \cap, \cup, \neg)$ изоморфна булевой алгебре $(B_n; \vee, \&, \bar{\cdot})$.

Взаимно однозначное соответствие Γ между подмножествами U и векторами из B_n было описано ранее. Остается показать, что Γ — изоморфизм, т. е. что для него выполнено условие (2.1), которое в данном случае сводится к трем равенствам: если $\Gamma(M_1) = \sigma$, а $\Gamma(M_2) = \tau$, то

$$\begin{aligned}\Gamma(M_1 \cup M_2) &= \sigma \vee \tau; \\ \Gamma(M_1 \cap M_2) &= \sigma \& \tau; \\ \Gamma(\bar{M}_1) &= \bar{\sigma}.\end{aligned}\quad (3.23)$$

Справедливость их вытекает непосредственно из (3.22): если $a_i \in M_1 \cup M_2$, то i -й разряд вектора $\Gamma(M_1 \cup M_2) = 1$; с другой стороны, это означает, что $a_i \in M_1$ или $a_i \in M_2$, т. е. $\sigma_i = 1$ или $\tau_i = 1$, и, следовательно, i -й разряд вектора $\sigma \vee \tau$ равен 1. Если же $a_i \notin M_1 \cup M_2$, то i -й разряд $\Gamma(M_1 \cup M_2)$ равен 0. Но тогда $a_i \notin M_1$ и $a_i \notin M_2$, следова-

тельно, i -й разряд $\sigma \vee \tau$ также равен 0. Аналогично доказываются остальные два равенства. \square

Эта теорема позволяет заменить теоретико-множественные операции (объединение, пересечение, дополнение) над системой подмножеств поразрядными логическими операциями над двоичными векторами. Такая замена часто используется при программировании, поскольку представление двоичных векторов и поразрядные операции над ними в компьютере реализуются очень просто.

Рассмотрим теперь множество $P_2(m)$ всех логических функций m переменных x_1, \dots, x_m . Оно замкнуто относительно операций $\&$, \vee (результат их применения к функциям из $P_2(m)$ снова дает функцию из $P_2(m)$) и, следовательно, образует конечную булеву алгебру $(P_2(m); \vee, \&, \neg)$, являющуюся подалгеброй булевой алгебры логических функций.

Теорема 3.5. Если $|U| = 2^m$, то булева алгебра множеств $(\mathcal{B}(U); \cap, \cup, \neg)$ изоморфна булевой алгебре функций $(P_2(m); \vee, \&, \neg)$.

Прежде всего отметим, что эти две алгебры равномощны и содержат 2^{2^m} элементов. Кроме того, поскольку все множества одинаковой мощности порождают изоморфные булевые алгебры множеств (см. пример 2.2, д), то теорему достаточно доказать для какого-либо конкретного U , удовлетворяющего условию $|U| = 2^m$. В качестве такого U возьмем множество B_m и, следовательно, будем доказывать изоморфизм между $(\mathcal{B}(B_m); \cup, \cap, \neg)$ и $(P_2(m); \vee, \&, \neg)$.

Обозначим через M_f множество единичных наборов функции f ; тогда набор σ из B_m принадлежит M_f , если и только если $f(\sigma) = 1$. Соответствие $\Gamma(f) = M_f$ между функциями и их единичными множествами является взаимно однозначным соотвествием между $P_2(m)$ и $\mathcal{B}(B_m)$, поскольку различным функциям соответствуют различные множества, и наоборот. (Функцию f , единичным множеством которой служит M , называют *характеристической функцией множества M* .) Покажем, что Γ является изоморфизмом. Для этого достаточно проверить условие (2.1), которое в данном случае сводится к трем равенствам

$$\Gamma(f \vee g) = M_1 \cup M_g;$$

$$\Gamma(f \& g) = M_1 \cap M_g;$$

$$\Gamma(\bar{f}) = \bar{M}_f$$

для любых функций f и g m переменных. Докажем первое из них. Пусть $\sigma \in \Gamma(f \vee g)$. Тогда $f(\sigma) \vee g(\sigma) = 1$, следовательно, $f(\sigma) = 1$ или $g(\sigma) = 1$ и, значит, $\sigma \in M_f$ или $\sigma \in M_g$, откуда следует, что $\sigma \in (M_f \cup M_g)$. Обратный случай: $\sigma \in (M_f \cup M_g)$. Тогда $\sigma \in M_f$ или $\sigma \in M_g$ и, следовательно, $f(\sigma) = 1$ или $g(\sigma) = 1$. Поэтому $f(\sigma) \vee g(\sigma) = 1$ и $\sigma \in \Gamma(f \vee g)$. Аналогично доказываются и остальные два равенства.

Замечание. Во избежание путаницы обращаем внимание читателя на различия объектов в доказанных нами теоремах.

1. В теореме 3.4 фигурировали алгебры со следующими основными множествами: $U = \{a_1, \dots, a_n\}$ — множество произвольной природы и любой конечной мощности n ; $\mathfrak{B}(U)$ — множество подмножеств U мощности 2^n ; B_n — множество двоичных векторов длины n также мощности 2^n .

В теореме 3.5 участвовали: то же множество U , но с дополнительным условием $n = 2^m$ (m — любое натуральное число); B_m — конкретное множество U с этим же условием; $|B_m| = 2^m$; множество $P_2(m)$ логических функций m переменных; $|P_2(m)| = 2^{2^m}$; (B_m) — множество подмножеств B_m ; $|\mathfrak{B}(B_m)| = 2^{2^m}$.

2. Множества B_n и B_m , хотя и имеют одну и ту же природу (состоят из двоичных наборов), использовались в теоремах 3.4 и 3.5 по-разному. В теореме 3.4 была использована структура элементов B_n , благодаря чему над ними оказались возможными поразрядные логические операции. Подмножества B_n не рассматривались. В теореме 3.5 структура элементов B_n не учитывалась, само B_n было выбрано только для естественности и наглядности, зато рассматривалась $\mathfrak{B}(B_n)$ — система подмножеств B_n .

Теоремы 3.4 и 3.5 указывают на тесную связь между множествами и логическими функциями и позволяют переходить от операций над множествами к операциям над функциями и обратно. В частности, они дают возможность непосредственно производить операции над функциями, заданными не формулами, а таблицами или единичными множествами. Из теорем 3.4 и 3.5 следует, что булевы операции над функциями, заданными таблицами, сводятся к поразрядным логическим операциям

Таблица 3.4

x_1	x_2	x_3	f	g	$f \vee g$	fg	\bar{f}
0	0	0	0	0	0	0	1
0	0	1	1	1	1	1	0
0	1	0	0	1	1	0	1
0	1	1	0	0	0	0	1
1	0	0	1	0	1	0	0
1	0	1	1	1	1	1	0
1	1	0	0	0	0	0	1
1	1	1	0	1	1	0	1

над столбцами значений функций. Пример приведен в табл. 3.4, содержащей две функции f и g и результаты булевых операций над ними.

В завершение отметим еще один факт, связывающий логические функции с основными понятиями теории множеств: если $(f \rightarrow g) \equiv 1$, то $M_f \subseteq M_g$. Действительно, если $(f \rightarrow g) \equiv 1$, то из определения импликации (табл. 3.3, функция ψ_{13}) следует, что ни для какого набора σ не может быть одновременно $f(\sigma) = 1$ и $g(\sigma) = 0$. Поэтому если $f(\sigma) = 1$, то $g(\sigma) = 1$, т. е. если $\sigma \in M_f$, то $\sigma \in M_g$ и, следовательно, $M_f \subseteq M_g$. В таком случае говорят, что функция f имплицирует функцию g . При этом если f — элементарная конъюнкция, то f называется импликантом g , а если после удаления буквы (вхождения переменной) f перестает быть импликантом g , то f называется простым импликантом g . Например, для функции $x(y \vee z)$ конъюнкции xy и xz — простые импликанты, а xyz — импликант, но не простой. Отметим, что любая конъюнкция любой ДНФ данной функции является импликантом этой функции.

ДНФ, интервалы и покрытия. Теоретико-множественная интерпретация булевой алгебры функций оказывается очень удобным языком для изучения дизъюнктивных нормальных форм (ДНФ) и построения методов их упрощения. Рассмотрим кратко основные понятия, связанные с ДНФ.

Если функция $f(x_1, \dots, x_m)$ представляется элементарной конъюнкцией всех m переменных, то M_f содержит ровно один элемент множества B_m . Если же f — элементарная конъюнкция k переменных, где $k < m$, то M_f

содержит 2^{m-k} двоичных наборов (так как $m-k$ переменных, не вошедших в эту конъюнкцию, несущественны для f и могут принимать любые 2^{m-k} значений, не изменяя f). Множество M_f такой функции f называется *интервалом*. Например, для $m = 4$ и $f(x_1, x_2, x_3, x_4) = x_2\bar{x}_4$ $M_f = \{0100, 0110, 1100, 1110\}$ и $|M_f| = 2^2 = 4$. В этом случае говорят, что конъюнкция $x_2\bar{x}_4$ (точнее, интервал $M_{x_2\bar{x}_4}$) покрывает множество M_f (и все его подмножества). Представление f в виде ДНФ соответствует представлению ее единичного множества в виде объединения интервалов; в совокупности все конъюнкции ДНФ покрывают все единичное множество f . Верно и обратное: если все элементы некоторого интервала M_k принадлежат M_f , то существует ДНФ функции f , содержащая конъюнкцию k . В частности, СДНФ функции f соответствует просто перечислению элементов M_f . Отношение покрытия между конъюнкциями ДНФ и элементами M_f наглядно задается *таблицей покрытия*, строки которой соответствуют конъюнкциям (т. е. интервалам), столбцы — элементам M_f , а на пересечении строки i со столбцом j стоит какая-либо отметка (например, 1), если i -я конъюнкция покрывает j -й элемент M_f . Например, ДНФ $F = xz \vee y\bar{z} \vee xy$ соответствует таблица покрытия (табл. 3.5).

Таблица 3.5

	010	101	110	111
xz		1		1
$y\bar{z}$	1		1	
xy			1	1

Из табл. 3.5 видно, что интервал M_{xy} покрывается объединением интервалов M_{xz} и $M_{y\bar{z}}$. Поэтому исключение xy из F не изменит единичного множества данной функции и получится теоретико-множественное доказательство соотношения (3.19). Еще более очевидное обоснование имеет закон по-глощения $x \vee xy = x$: интервал M_{xy} всегда покрывается интервалом M_x . Этот закон в терминах ДНФ можно переформулировать следующим образом: любой импликант k ДНФ, не являющийся простым, можно заменить простым импликантом k_0 , покрывающим k ; импликант k_0 получается из k вычеркиванием некоторых букв.

Таким образом, для любой функции f существует ДНФ F , состоящая только из простых импликантов. Ясно, что ДНФ $F \vee k$, где k — простой импликант f , не содержит

жащийся в F , также представляет f . Поэтому дизъюнкция всех простых импликантов f , называемая *сокращенной ДНФ*, также будет представляться f .

Методы упрощения ДНФ (а их в настоящее время известно довольно много) состоят, как правило, из двух этапов. На первом этапе получают список всех простых импликантов, т. е. сокращенную ДНФ. Это можно сделать, например, при помощи эквивалентных преобразований. На втором этапе, используя таблицу покрытия (или аналогичные методы), удаляют «лишние» импликанты, покрываемые другими импликантами. ДНФ, из которой нельзя удалить ни одного импликанта, называется *тупиковой*.

3.3. ПОЛНОТА И ЗАМКНУТОСТЬ

В § 3.1 рассматривались два способа задания логических функций — табличный и формульный. Таблица задает функцию непосредственно как соответствие между двоичными наборами и значениями функции на этих наборах. Этот способ универсален, т. е. пригоден для любой функции, однако громоздок. Формула — гораздо более компактный способ задания функции, однако она задает функцию через другие функции. Поэтому для любой системы функций Σ возникает естественный вопрос: всякая ли логическая функция представима формулой над Σ ? В § 3.2 был получен утвердительный ответ для системы $\Sigma_0 = \{\&, \vee, \neg\}$ (теорема 3.2). В настоящем параграфе будет показано, как решать этот вопрос для произвольной системы Σ .

Функционально полные системы. Система функций Σ называется *функционально полной системой*, если любая логическая функция может быть представлена формулой над Σ , т. е. является суперпозицией функций из Σ . Из теоремы 3.2 следует, что система $\Sigma_0 = \{\&, \vee, \neg\}$ функционально полна. Функционально полной будет и любая система Σ , через функции которой можно выразить дизъюнкцию, конъюнкцию и отрицание. Действительно, для любой логической функции f представляющую ее формулу над Σ можно построить так: взять булеву формулу для f (по теореме 3.2 такая формула обязательно

найдется) и все булевы операции в ней заменить^{*} формулами над Σ , представляющими эти операции. Аналогично доказывается и более общее утверждение: если все функции функционально полной системы Σ' представимы формулами над системой Σ , то Σ также функционально полна. В этом случае будем говорить, что Σ сводится к Σ' . Такое сведение широко используется в дальнейшем.

Пример 3.5. а. Системы $\Sigma_1 = \{\&, \neg\}$ и $\Sigma_2 = \{\vee, \neg\}$ функционально полны. Действительно, из законов де Моргана и двойного отрицания следует, что в каждой из этих двух систем недостающая до Σ_0 функция выражается через остальные две:

$$x_1 \vee x_2 = \overline{\overline{x_1} \& \overline{x_2}}, \quad x_1 \& x_2 = \overline{\overline{x_1} \vee \overline{x_2}}. \quad (3.24)$$

Булева формула $x_1 x_2 \vee \overline{x_2} (x_3 \vee x_4)$ в системе Σ_1 перейдет в формулу

$$\overline{\overline{x_1 x_2} \overline{\overline{x_2} \overline{x_3 x_4}}},$$

а в системе Σ_1 — в формулу $\overline{\overline{x_1} \vee \overline{x_2}} \vee \overline{x_2} \vee \overline{x_3 \vee x_4}$.

С точки зрения функциональной полноты систему Σ_0 можно считать избыточной: она сохраняет свойства полноты и при удалении из нее дизъюнкции или конъюнкции. Отметим, правда, что, как видно из примера, за неизбыточность систем Σ_1 и Σ_2 приходится платить избыточностью формул: ведь каждая замена одной операции на другую по соотношению (3.24) вносит в формулу лишние отрицания.

б. Системы $\Sigma_3 = \{| \}$ (штрих Шеффера) и $\Sigma_4 = \{\downarrow\}$ (стрелка Пирса) функционально полны, так как из (3.2), (3.3) следует, что

$$\begin{aligned} \bar{x} &= x | x = x \downarrow x; \quad x_1 \vee x_2 = \overline{x_1 \downarrow x_2} = \\ &= (x_1 \downarrow x_2) \downarrow (x_1 \downarrow x_2); \quad x_1 x_2 = \overline{x_1 | x_2} = (x_1 | x_2) | (x_1 | x_2), \end{aligned}$$

следовательно, Σ_3 сводится к Σ_1 , а Σ_4 — к Σ_2 .

в. Система $\Sigma_5 = \{\&, \oplus, 1\}$ функционально полна, так как $\bar{x} = x \oplus 1$ и, следовательно, Σ_5 сводится к Σ_1 . На свойствах этой системы остановимся более подробно.

* Такую замену следовало бы описать более точно. Мы этого не делаем из-за громоздкости, надеясь, что она будет ясна из примеров.

Алгебра Жегалкина и линейные функции. Алгебра над множеством логических функций с двумя бинарными операциями $\&$ и \oplus называется *алгеброй Жегалкина*. В алгебре Жегалкина выполняются следующие соотношения:

$$x \oplus y = y \oplus x; \quad (3.25)$$

$$x(y \oplus z) = xy \oplus xz; \quad (3.26)$$

$$x \oplus x = 0; \quad (3.27)$$

$$x \oplus 0 = x, \quad (3.28)$$

а также соотношения булевой алгебры, относящиеся к конъюнкции и константам (3.7а), (3.8а), (3.11а), (3.13, а, б). Отрицание и дизъюнкция выражаются так:

$$\bar{x} = x \oplus 1; \quad (3.29)$$

$$x \vee y = \overline{\bar{x}\bar{y}} = (x \oplus 1)(y \oplus 1) \oplus 1 = xy \oplus x \oplus y. \quad (3.30)$$

Если в произвольной формуле алгебры Жегалкина раскрыть скобки и произвести все возможные упрощения по соотношениям (3.27), (3.28), (3.11а), (3.13, а, б), то получится формула, имеющая вид суммы произведений, т. е. полинома по mod 2. Такая формула называется *полиномом Жегалкина* для данной функции.

От булевой формулы всегда можно перейти к формуле алгебры Жегалкина и, следовательно, полиному Жегалкина, используя равенства (3.29) и (3.30), а также равенство, вытекающее из (3.30): если $f_1 f_2 = 0$, то $f_1 \vee f_2 = f_1 \oplus f_2$. Оно, в частности, позволяет заменять знак дизъюнкции знаком \oplus в случае, когда исходная формула — СДНФ.

Пример 3.6.

$$\begin{aligned} a. (x_1 \vee x_2) (\bar{x}_2 \wedge x_3) &\equiv (x_1 \oplus x_2) \& \bar{x}_2 \oplus x_3 = \\ &= x_1(x_2 \oplus 1)x_3 \oplus x_1x_2x_3 \oplus x_1x_3 \oplus x_1x_2x_3 \oplus x_1(x_2 \oplus 1) = \\ &= x_1x_2x_3 \oplus x_1x_2 \oplus x_1; \end{aligned}$$

$$\begin{aligned} b. x_1\bar{x}_2 \vee \bar{x}_1x_3 &= x_1\bar{x}_2 \oplus \bar{x}_1x_3 = x_1(x_2 \oplus 1) \oplus (x_1 \oplus 1) \& x_3 = \\ &= x_1x_2 \oplus x_1x_3 \oplus x_1 \oplus x_3; \end{aligned}$$

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

Теорема 3.6. Для всякой логической функции существует полином Жегалкина, и притом единственный.

Существование полинома уже доказано. Для доказательства единственности покажем, что между множеством

всех функций от n переменных и множеством всех полиномов Жегалкина от n переменных существует взаимно однозначное соответствие. Число различных членов (т. е. конъюнкций переменных) полиномов от n переменных равно числу всех подмножеств из n элементов, т. е. 2^n (пустому подмножеству соответствует член 1). Число различных полиномов, которые можно образовать из этих конъюнкций, равно числу всех подмножеств множества конъюнкций, т. е. 2^{2^n} (пустому подмножеству конъюнкций соответствует полином 0). Таким образом, число всех полиномов Жегалкина от n переменных равно числу всех функций от n переменных. Так как разным функциям соответствуют разные полиномы (одна и та же формула не может представлять две разные функции), то тем самым между множествами функций и полиномов от n переменных установлено взаимно однозначное соответствие, что и доказывает единственность полинома для каждой функции. \square

Функция, у которой полином Жегалкина имеет вид $\sum \alpha_i x_i \oplus \gamma$, где α_i, γ равны 0 или 1, называется *линейной*. Все функции от одной переменной линейны. Линейными функциями от двух переменных являются сумма по mod 2 и эквивалентность.

Замкнутые классы. Монотонные функции. Множество M логических функций называется *замкнутым классом*, если любая суперпозиция функций из M снова принадлежит M .

Всякая система Σ логических функций порождает некоторый замкнутый класс, а именно класс, состоящий из всех функций, которые можно получить суперпозициями из Σ . Такой класс называется *замыканием* Σ и обозначается $[\Sigma]$. Очевидно, что если M — замкнутый класс, то $[M] = M$, а если M — функционально полная система, то $[M] = P_2$.

Пример 3.7. а. Множество всех дизъюнкций, т. е. функций вида $x_1 \vee x_2 \vee \dots \vee x_n$, является замкнутым классом.

б. Множество всех линейных функций является замкнутым классом, так как подстановка формул вида $\sum \alpha_i x_i \oplus \gamma$ в формулу такого же вида снова дает формулу того же вида.

Важным примером замкнутого класса является класс монотонных функций, к рассмотрению которого мы сейчас переходим.

В гл. 1 (пример 1.14, б) рассматривалось отношение частичного порядка \leq на множестве векторов одинаковой длины. Напомним, что для двух векторов $\sigma = (\sigma_1, \dots, \sigma_n)$ и $\tau = (\tau_1, \dots, \tau_n)$ $\sigma \leq \tau$, если и только если $\sigma_i \leq \tau_i$ для всех $i = 1, \dots, n$. Здесь воспользуемся этим отношением для двоичных векторов.

Функция $f(x_1, \dots, x_n)$ называется *монотонной*, если для любых двоичных наборов σ и τ длины n из того, что $\sigma \leq \tau$, следует, что $f(\sigma) \leq f(\tau)$.

Пример 3.8. а. Константы 0, 1 и функция x монотонны. Отрицание \bar{x} немонотонно.

б. Дизъюнкция и конъюнкция любого числа переменных являются монотонными функциями.

Рассмотрим две функции от трех переменных, данные табл. 3.6. Функция f_1 немонотонная, так как $001 < 101$, а $f_1(001) > f_1(101)$. Функция f_2 монотонна (проверку предоставляем читателю).

Проверка монотонности функции непосредственно по определению требует анализа таблицы функции и может оказаться довольно громоздким делом. Поэтому весьма полезной для опознания монотонности является следующая теорема.

Теорема 3.7. Всякая булева формула, не содержащая отрицаний, представляет монотонную функцию, отличную от 0 и 1; и наоборот, для любой монотонной функции, отличной от 0 и 1, найдется представляющая ее булева формула без отрицаний.

Доказательство. 1. Пусть дана булева формула без отрицаний.

Применим к ней процедуру приведения к ДНФ. Так как соотношения (3.15) и (3.16), приводящие к константам, в формулах без отрицаний неприменимы, то получим невырожденную (отличную от 0 и 1) ДНФ F , также не содержащую отрицаний. Пусть на наборе $\sigma = (\sigma_1, \dots, \sigma_n)$ $F(\sigma) = 1$. Тогда F содержит конъюнкцию (без отрицаний!) $x_{i_1} \dots x_{i_k} = 1$, равную 1 на этом наборе. Следовательно, $\sigma_{i_1} = \dots = \sigma_{i_k} = 1$.

Таблица 3.6

x_1	x_2	x_3	f_1	f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	1
0	1	1	1	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Рассмотрим любой набор τ , больший, чем σ . В нем обязательно $\tau_{i1} = \dots = \tau_{ik} = 1$, поэтому x_{i1}, \dots, x_{ik} обратится в 1 и $F(\tau) = 1$. Таким образом, условие монотонности для F выполнено и функция f , представляемая ДНФ F (а значит, и исходной формулой), монотонна. При этом $f \neq 0$, так как в невырожденной ДНФ для любой конъюнкции найдется набор, обращающий ее в 1; $f \neq 1$, так как на нулевом наборе $(0, 0, \dots, 0) F = 0$.

2. Пусть функция f монотонна и отлична от 1 и 0. Тогда она имеет невырожденную ДНФ и, следовательно, невырожденные импликанты. Предположим, что какой-либо импликант f содержит отрицание над переменной и имеет вид $\bar{x}_i k$ (k — элементарная конъюнкция). На любом наборе σ , в котором $\sigma_i = 0$ и который обращает k в 1, $f(\sigma) = 1$. Рассмотрим набор σ' , полученный из σ заменой σ_i на 1. Так как $\sigma' > \sigma$, то по условию монотонности $f(\sigma') = 1$. Кроме того, σ' обращает в 1 конъюнкцию $x_i k$, которая, следовательно, является импликантом f . Но если $\bar{x}_i k$ и $x_i k$ — импликанты f , то k — также импликант f (это следует из определения импликанта и закона склеивания (3.18)) и, следовательно, $\bar{x}_i k$ не является простым импликантом. Таким образом, простым импликантом монотонной функции f может быть только конъюнкция, не содержащая отрицаний. Дизъюнкция всех простых импликантов f (сокращенная ДНФ) и дает искомую булеву формулу без отрицаний для f . \square

Более подробное изучение импликантов монотонной функции дает следующую картину их строения. Выделим из M_f множество M_f^0 всех минимальных наборов (σ минимален в M_f , если из $\tau < \sigma$ следует, что $\tau \notin M_f$). Каждому минимальному набору σ соответствует простой импликант, являющийся конъюнкцией всех тех переменных, которым в σ соответствуют единицы; и наоборот, каждому простому импликанту f соответствует (аналогичным образом) минимальный набор в M_f . Например, для функции f_2 из табл. 3.6 $M_{f_2}^0 = \{010, 101\}$, поэтому сокращенная ДНФ f_2 имеет вид $x_2 \vee x_1 x_3$.

Следствие. Сокращенная ДНФ монотонной функции является ее минимальной ДНФ.

Действительно, поскольку в сокращенной ДНФ нет отрицаний, для любых двух импликантов k_i и k_j найдется переменная, содержащаяся в k_i , но не в k_j , и переменная, содержащаяся в k_j , но не в k_i . Если положить переменные из k_i равными 1, а остальные переменные — 0, то получим набор σ , который все импликанты, кроме k_i , обращает в 0. Поэтому σ покрывает только k_i , который не может быть удален из сокращенной ДНФ. Следовательно, сокращенная ДНФ — единственная туниковая, а значит, и минимальная ДНФ монотонной функции. \square

Теорема 3.8. Множество всех монотонных функций является замкнутым классом.

Эта теорема непосредственно следует из теоремы 3.7 и того очевидного обстоятельства, что подстановка формул без отрицаний в формулу без отрицаний снова дает формулу без отрицаний.

Следствие. Класс монотонных функций является замыканием системы функций $\{\&, \vee, 0, 1\}$.

Это утверждение вытекает из того, что всякая булева формула без отрицаний является суперпозицией дизъюнкций и конъюнкций. \square

Две теоремы о функциональной полноте. Теперь можно перейти к основной проблеме этого параграфа: какие необходимые и достаточные условия функциональной полноты для произвольной системы функций Σ ? Как отмечалось в начале параграфа, система Σ полна, если дизъюнкция, конъюнкция и отрицание являются суперпозициями функций из Σ . Поэтому будем искать свойства функций, позволяющие выразить через них булевые операции.

Лемма 1 (о немонотонных функциях). Если функция $f(x_1, \dots, x_n)$ немонотонна, то подстановкой констант из нее можно получить отрицание. Точнее: существует такая подстановка $n-1$ константы, что функция оставшейся одной переменной является отрицанием.

Доказательство. Пусть f немонотонна. Тогда существуют наборы σ и τ , такие, что $\sigma < \tau$, $f(\sigma) = 1$, $f(\tau) = 0$. Если σ и τ отличаются k компонентами, то в этих компонентах в σ стоят нули, а в τ единицы. Беря набор σ и заменяя эти компоненты по одному единицами, получаем цепочку $\sigma < \omega^1 < \omega^2 < \dots < \omega^{k-1} < \tau$, в которой любые два набора, стоящие рядом, отличаются только в одной компоненте (такие наборы называются соседними). Ясно, что в такой цепочке найдутся два соседних набора ω^i , ω^{i+1} , таких, что $f(\omega^i) = 1$, $f(\omega^{i+1}) = 0$. Пусть они отличаются в i -й компоненте; тогда $\omega_i^i = 0$, $\omega_i^{i+1} = 1$, остальные их компоненты одинаковы. Подставим эти значения остальных компонент в f . Получим функцию $f(\omega_1^i, \dots, \omega_{i-1}^i, x_i, \omega_{i+1}^i, \dots, \omega_n^i)$ от x_i ; обозначим ее $g(x_i)$. Но $g(0) = g(\omega_i^i) = f(\omega^i) = 1$; $g(1) = g(\omega_i^{i+1}) = f(\omega^{i+1}) = 0$; следовательно, $g(x_i) = \bar{x}_i$. \square

Лемма 2 (о нелинейных функциях). Если функция $f(x_1, \dots, x_n)$ нелинейна, то с помощью подстановки констант и использования отрицаний из нее можно получить дизъюнкцию и конъюнкцию. Точнее, существует представление дизъюнкции и конъюнкции в виде суперпозиции констант, отрицаний и функции f .

Доказательство. Пусть f нелинейна. Тогда ее полином Жегалкина содержит конъюнкции переменных. Выберем самую короткую из них $K = x_{i_1}x_{i_2}\dots x_{i_k}$. Положим $x_{i_3} = \dots = x_{i_k} = 1$, а для всех x_j , не входящих в K , $x_j = 0$. Подстановка этих констант в полином обратит K в $x_{i_1}x_{i_2}$, а остальные конъюнкции в 0, и f примет вид $x_{i_1}x_{i_2} \oplus \alpha x_{i_1} \oplus \beta x_{i_2} \oplus \gamma$, где α, β, γ — коэффициенты, равные 0 или 1 и зависящие от конкретной функции f .

Функции, получающиеся при всех восьми возможных комбинациях значений α, β, γ , приведены в табл. 3.7, в которой для наглядности обозначено $x_{i_1} = x, x_{i_2} = y$, а отрицание в последнем столбце обозначено через N . Поскольку каждая из функций $f_i(x, y)$ ($i = 0, 1, \dots, 7$) — результат подстановки констант в f , то последний столбец табл. 3.7 содержит искомое представление дизъюнкции или конъюнкции в виде суперпозиции отрицаний, констант и исходной функции f . Для перехода от полученного представления к представлению двойственной функции (от конъюнкции к дизъюнкции и наоборот) дополнительно потребуются только отрицания (по закону де Моргана). \square

Таблица 3.7

f_i	α	β	γ	Вид полинома	Эквивалентная булева формула	Искомая суперпозиция
f_0	0	0	0	xy	xy	$xy = f_0(x, y)$
f_1	0	0	1	$xy \oplus 1$	$\bar{x}\bar{y}$	$xy = N(f_1(x, y))$
f_2	0	1	0	$xy \oplus y$	$\bar{x}y$	$xy = f_2(N(x), y)$
f_3	0	1	1	$xy \oplus y \oplus 1$	$\bar{x}\bar{y} = x \vee \bar{y}$	$x \vee y = f_3(x, N(y))$
f_4	1	0	0	$xy \oplus x$	$x\bar{y}$	$xy = f_4(x, N(y))$
f_5	1	0	1	$xy \oplus x \oplus 1$	$\bar{x}\bar{y} = \bar{x} \vee y$	$x \vee y = f_5(N(x), y)$
f_6	1	1	0	$xy \oplus x \oplus y$	$x \vee y$	$x \vee y = f_6(x, y)$
f_7	1	1	1	$xy \oplus x \oplus y \oplus 1$	$\bar{x}\bar{y}$	$x \vee y = N(f_7(x, y))$

Пример 3.9. $f(x_1, x_2, x_3, x_4) = x_1x_3x_4 \oplus x_1x_2x_3x_4 \oplus \oplus x_1 \oplus x_4$. Полагаем $x_4 = 1$, $x_2 = 0$. Тогда $f(x_1, 0, x_3, 1) = x_1x_3 \oplus x_1 \oplus 1$, что соответствует строке f_5 в табл. 3.7. Отсюда получаем

$$x_1 \vee x_3 = f(\bar{x}_1, 0, x_3, 1); x_1x_3 = \overline{\bar{x}_1 \vee \bar{x}_3} = \bar{f}(x_1, 0, \bar{x}_3, 1).$$

Замечание. При традиционных обозначениях переменных в выражениях типа $f(x_1, x_2, x_3, x_4)$, где переменные расположены в естественном порядке индексов, индексы играют двоякую роль: они именуют переменные и нумеруют места в функции. Эти роли следует различать. В примере 3.9 существенны именно места в функции (первое и третье); с помощью той же функции можно получить дизъюнкцию не только x_1 и x_3 , но и любых других переменных. Например, $x_2 \vee x_4 = f(\bar{x}_2, 0, x_4, 1)$, $y \vee z = f(\bar{y}, 0, z, 1)$. Указанное различие хорошо видно при схемной реализации функций: функция от n аргументов реализуется схемой с n входами; номера мест — это номера (или имена) входов схемы, а имена переменных — это имена внешних сигналов (датчиков, выходов других схем и т. д.), подаваемых на входы схемы.

Две доказанные леммы позволяют получить все булевы операции с помощью немонотонных функций, нелинейных функций и констант. Это еще не функциональная полнота в обычном смысле, так как константы с самого начала предполагались данными. Однако такое предположение часто бывает оправданным в различных приложениях и прежде всего в синтезе логических схем (см. гл. 8), где системе логических функций соответствует выбор (серия) типовых логических элементов, а полнота системы означает возможность реализовать с помощью элементов данной схемы любые логические функции. При схемной реализации константы 0 и 1 специальных элементов не требуют. Поэтому имеет смысл ввести ослабленное понятие функциональной полноты: система функций Σ называется *функционально полной в слабом смысле*, если любая логическая функция может быть представлена формулой над системой $\Sigma \cup \{0, 1\}$, т. е. является суперпозицией констант и функций из Σ . Очевидно, что из обычной полноты системы следует ее слабая полнота.

Теорема 3.9 (первая теорема о функциональной полноте). Для того чтобы система функций Σ была функционально полной в слабом смысле, необходимо и достаточно,

чтобы она содержала хотя бы одну немонотонную и хотя бы одну нелинейную функцию.

Н е о б х о д и м о с т ь. Классы монотонных и линейных функций замкнуты и содержат 0 и 1. Поэтому если Σ не содержит немонотонных или нелинейных функций, то их нельзя получить с помощью суперпозиций функций из Σ и констант.

Д о с т а т о ч н о с т ь. Пусть Σ содержит немонотонную и нелинейную функцию. Тогда по лемме 1 подстановкой констант из монотонной функции получаем отрицание, а затем по лемме 2 из нелинейной функции с помощью отрицаний и констант получаем дизъюнкцию и конъюнкцию. \square

Пример 3.10. а. Система $\Sigma_6 = \{\&, \oplus\}$ функционально полна в слабом смысле, так как конъюнкция нелинейна, а сумма по mod 2 немонотонна. Константа 0 получается из соотношения (3.27), однако константу 1 с помощью конъюнкции и суммы по mod 2 получить нельзя, поэтому Σ_6 не является функционально полной системой в обычном (сильном) смысле. Использование константы 1, которое разрешается определением слабой полноты, сводит Σ_6 к полной в сильном смысле системе Σ_6 (см. пример 3.5, в).

б. В функционально полной системе Σ_3 единственная функция — штрих Шеффера — одновременно нелинейна и немонотонна.

в. Проверим на слабую функциональную полноту систему Σ_7 , состоящую из одной функции f_1 , заданной табл. 3.6. Немонотонность f_1 уже установлена. Получим ее полином Жегалкина:

$$\begin{aligned} f_1 = & \bar{x}_1 \bar{x}_2 x_3 \vee \bar{x}_1 x_2 x_3 \vee x_1 \bar{x}_2 \bar{x}_3 \vee x_1 x_2 x_3 = (x_1 \oplus 1)(x_2 \oplus 1)x_3 \oplus \\ & \oplus (x_1 \oplus 1)x_2 x_3 \oplus x_1(x_2 \oplus 1)(x_3 \oplus 1) \oplus x_1 x_2 x_3 = x_1 x_2 \oplus x_1 \oplus x_3. \end{aligned}$$

Следовательно, f_1 нелинейна и Σ_7 — функционально полная в слабом смысле система.

Для формулировки необходимых и достаточных условий сильной полноты рассмотрим еще три замкнутых класса.

Функция $f(x_1, \dots, x_n)$ называется *сохраняющей 0*, если $f(0, 0, \dots, 0) = 0$. Функция $f(x_1, \dots, x_n)$ называется *сохраняющей 1*, если $f(1, 1, \dots, 1) = 1$. Оба класса функций, сохра-

няющих 0 и сохраняющих 1, являются замкнутыми, что проверяется подстановкой констант в суперпозиции.

Напомним, что функция является самодвойственной, если $f(x_1, \dots, x_n) = \bar{f}(\bar{x}_1, \dots, \bar{x}_n)$. Класс самодвойственных функций замкнут. Его замкнутость доказывается прямой выкладкой. (Чтобы избежать громоздких обозначений, далее она проводится не в самом общем виде; обобщение очевидно.) Пусть $f_1(x_1, \dots, x_n), f_2(x_n, x_{n+1}, \dots, x_{n+k})$ — самодвойственные функции. Подставим f_1 в f_2 вместо x_n . Получим $g(x_1, \dots, x_n, x_{n+1}, \dots, x_{n+k}) = f_1(x_1, \dots, x_{n-1}, f_2(x_n, \dots, x_{n+k}))$. Тогда в силу самодвойственности f_1 и f_2

$$\begin{aligned} \bar{g}(\bar{x}_1, \dots, \bar{x}_n, \bar{x}_{n+1}, \dots, \bar{x}_{n+k}) &= \bar{f}_1(\bar{x}_1, \dots, \bar{x}_{n-1}, \bar{f}_2(\bar{x}_n, \dots, \bar{x}_{n+k})) = \\ &= \bar{f}_1(\bar{x}_1, \dots, \bar{x}_{n-1}, \bar{f}_2(x_n, \dots, x_{n+k})) = \\ &= f_1(x_1, \dots, x_{n-1}, f_2(x_n, \dots, x_{n+k})) = g(x_1, \dots, x_{n+k}), \end{aligned}$$

т. е. g является самодвойственной.

Теорема 3.10 (теорема Поста — вторая, основная теорема о функциональной полноте). Для того чтобы система функций Σ была функционально полной (в сильном смысле), необходимо и достаточно, чтобы она содержала: 1) нелинейную функцию; 2) немонотонную функцию; 3) несамодвойственную функцию; 4) функцию, не сохраняющую 0; 5) функцию, не сохраняющую 1.

Необходимость следует из замкнутости пяти классов, упомянутых в условии теоремы.

При доказательстве достаточности отметим следующее. Леммы 1 и 2 используют константы. Поэтому сначала нужно получить константы (из условий 3–5 теоремы) и только потом можно воспользоваться теоремой 3.9.

Прежде всего отметим, что если $f(x_1, \dots, x_n)$ несамодвойственна, то подстановкой в нее x и \bar{x} можно получить константу. Действительно, ввиду несамодвойственности f найдется набор $(\sigma_1, \dots, \sigma_n)$, такой, что $f(\bar{\sigma}_1, \dots, \bar{\sigma}_n) = f(\sigma_1, \dots, \sigma_n)$. Но тогда функция $f_c(x) = f(x^{\sigma_1}, \dots, x^{\sigma_n})$ является константой, так как

$$\begin{aligned} f_c(x) &= f(0^{\sigma_1}, \dots, 0^{\sigma_n}) = f(\bar{\sigma}_1, \dots, \bar{\sigma}_n) = \\ &= f(\sigma_1, \dots, \sigma_n) = f(1^{\sigma_1}, \dots, 1^{\sigma_n}) = f_c(1). \end{aligned}$$

Пусть теперь f_0 не сохраняет 0, f_1 не сохраняет 1, f_2 несамодвойственна (f_0, f_1, f_2 не обязаны быть различными). Если $f_0(1, \dots, 1) = 1$, то функция $\varphi(x) = f_0(x, \dots, x)$

есть константа 1, так как $\phi(0) = 1$ по определению f_0 и $\phi(1) = f_0(1, \dots, 1) = 1$, а функция $\psi(x) = f_1(\phi(x), \dots, \phi(x)) = f_1(1, \dots, 1) = 0$, т. е. $\psi(x)$ есть константа 0. Если же $f_0(1, \dots, 1) = 0$, то $\phi(x) = \bar{x}$, так как $\phi(0) = 1$ по определению f_0 и $\phi(1) = f_0(1, \dots, 1) = 0$. Но тогда из f_2 подстановкой x и \bar{x} получим функцию $f_c(x)$, являющуюся константой, а используя еще раз \bar{x} , получим вторую константу. Итак, в любом случае выполнения условий (3–5) достаточно для получения констант 0, 1. Используя этот факт и теорему 3.9, получаем теорему 3.10. \square

Функция «штрих Шеффера» удовлетворяет всем пяти условиям теоремы Поста и, следовательно, образует функционально полную систему в сильном смысле.

3.4. ЯЗЫК ЛОГИКИ ПРЕДИКАТОВ

Предикаты. Предикатом $P(x_1, \dots, x_n)$ называется функция $P: M_1 \times M_2 \times \dots \times M_n \rightarrow B$, где $B = \{0, 1\}$ — двоичное множество, определенное в начале п. 3.1. Предикат называется однородным, если все его переменные определены на одном и том же множестве M , т. е. $P: M^n \rightarrow B$. Для простоты изложения в дальнейшем мы будем рассматривать однородные предикаты (если не оговорено другое). Таким образом, *n-местный предикат*, определенный на множестве M , — это двузначная функция от n переменных, принимающих значения в произвольном множестве M . M называется *предметной областью* предиката, а x_1, \dots, x_n — *предметными переменными*. Элементы $a_1, \dots, a_n \in M$ называются *предметными константами*.

Для любых M и n существует взаимно однозначное соответствие между n -местными отношениями и n -местными предикатами на M : а) каждому n -местному отношению R соответствует предикат P , такой, что $P(a_1, \dots, a_n) = 1$, если и только если $(a_1, \dots, a_n) \in R$; б) всякий предикат $P(x_1, \dots, x_n)$ определяет отношение R , такое, что $(a_1, \dots, a_n) \in R$; если и только если $P(a_1, \dots, a_n) = 1$. При этом R задает *область истинности* предиката P .

Всякой функции $f: M^n \rightarrow M$ можно поставить в соответствие $(n + 1)$ -местный предикат P , такой, что $P(a_1, \dots, a_n, a_{n+1}) = 1$, если и только если $f(a_1, \dots, a_n) = a_{n+1}$.

Поскольку функция должна быть однозначной, то это соответствие требует, чтобы для любого $a'_{n+1} \neq a_{n+1}$ $P(a_1, \dots, a_n, a'_{n+1}) = 0$. Поэтому обратное соответствие [от $(n+1)$ -местного предиката к n -местной функции] возможно не всегда, а только при выполнении указанного условия.

Отступление 3.1. С логической точки зрения двоичные объекты, которые рассматривались в предыдущих параграфах, — это высказывания, которые могут быть истинными или ложными. Формулы — это составные высказывания, истинность которых определяется истинностью входящих в них элементарных высказываний (обозначаемых буквами) и логическими операциями над элементарными высказываниями, причем сами операции (такие, как отрицание, конъюнкция, импликация и т. д.) имеют довольно прозрачный логический смысл. Однако, работая с этими объектами, мы практически не обращались к их логическому содержанию и обходились теоретико-множественной или алгебраической интерпретацией, рассматривая их как функции на двоичных векторах или как элементы алгебр. Этот подход оказался эффективным во многом благодаря тому, что области определения функций алгебры логики конечны и имеют довольно простую структуру. Кроме того, опыт приложений алгебры логики показал, что функционально-алгебраическая интерпретация операций над двоичными объектами оказывается не менее содержательной и плодотворной, чем логическая интерпретация.

С предикатами дело обстоит иначе. Значение логики предикатов, которая как частный случай включает и логику высказываний, заключается не столько в ее собственных конкретных приложениях (хотя такие имеются), сколько в том, что она образует основу логического языка математики. С ее помощью удается формализовать и точно исследовать основные методы построения математических теорий. Логика предикатов является важным средством построения развитых логических языков и формальных систем, которые широко используются в искусственном интеллекте при формализации рассуждений и автоматизации логических выводов. Поэтому логическая интерпретация предикатов является основой, и мы ее будем в дальнейшем придерживаться.

Выражение $P(a_1, \dots, a_n)$ (и другие, более сложные выражения логики предикатов), где $a_1, \dots, a_n \in M$, будем понимать как высказывание « $P(a_1, \dots, a_n) = 1$ » или как « $P(a_1, \dots, a_n)$ истинно», а выражение $P(x_1, \dots, x_n)$, где x_1, \dots, x_n — переменные, как переменное высказывание,

истинность которого определяется подстановкой элементов M вместо x_1, \dots, x_n . При этом $P(x_1, \dots, x_n)$ — это логическая (двоичная) переменная, а x_1, \dots, x_n — нелогические переменные. Поскольку предикаты принимают два значения и интерпретируются как высказывания, из них можно образовывать выражения логики высказываний, т. е. формулы вида $P_1(x_1, x_2) \vee (P_2(x_3, x_4) \& P_1(x_2, x_4))$. Эта формула может рассматриваться и как составная булева формула, описывающая функцию алгебры логики от трех логических переменных $P_1(x_1, x_2)$, $P_1(x_2, x_4)$, $P_2(x_3, x_4)$ [$P_1(x_1, x_2)$ и $P_1(x_2, x_4)$ — разные логические переменные, так как предикат P_1 в этих выражениях зависит от разных переменных], и как составной четырехместный предикат, значение которого определяется четырьмя предметными переменными x_1, x_2, x_3, x_4 .

В дальнейшем, если это не вызовет разнотечений, будем употреблять одинаковые обозначения для отношений и соответствующих им предикатов; при этом, помимо функциональных обозначений вида $P(x)$, $P(x_1, x_2)$, для двухместных предикатов будем пользоваться обозначениями вида x_1Px_2 , которые уже употреблялись в гл. 1 для бинарных отношений.

Пример 3.11. а. Предикат $x_1 > x_2$ — это двухместный предикат, предметной областью которого могут служить любые множества действительных чисел. Высказывание $6 > 5$ истинно, а высказывания $7 > 7$ и $3 > 10$ ложны. Различные подстановки чисел вместо одной предметной переменной дают различные одноместные предикаты: $x_1 > 5$, $x_1 > 0$, $7 > x_2$ и т. д.

б. Великая теорема Ферма утверждает, что для любого целого $n > 2$ не существует натуральных чисел x, y, z , удовлетворяющих равенству $x^n + y^n = z^n$. Если этому равенству поставить в соответствие предикат $P_F(x, y, z, n)$, истинный тогда и только тогда, когда оно выполняется, а через $N(x)$ обозначить предикат « x — натуральное число», то теорема Ферма равносильна утверждению «выражение $N(x) \& N(y) \& N(z) \& N(n) \& (n > 2) \rightarrow \bar{P}_F(x, y, z, n)$ верно для любых чисел x, y, z, n ».

в. В описаниях вычислительных процедур и, в частности, в языках программирования часто встречаются указания типа «повторять цикл до тех пор, пока перемен-

ные x и y не станут равными, либо прекратить вычисление цикла после 100 повторений». Если обозначить через i счетчик повторений, то описанное здесь условие описывается выражением $(x = y) \vee (i > 100)$, а указание в целом принимает вид: «повторять, если $(x = y) \vee (i > 100)$ ».

Кванторы. Пусть $P(x)$ — предикат, определенный на M . Высказывание «для всех x из M $P(x)$ истинно» обозначается $\forall x P(x)$ (множество M не входит в обозначение и должно быть ясно из контекста).

Знак $\forall x$ называется *квантором общности*; другое его обозначение $(\forall x)$. Высказывание «существует такой x из M , что $P(x)$ истинно» обозначается $\exists x P(x)$. Знак $\exists x$ называется *квантором существования*; другое его обозначение $(\exists x)$. Переход от $P(x)$ к $\forall x P(x)$ или к $\exists x P(x)$ называется связыванием переменной x , а также навешиванием квантора на предикат P по переменной x (или на предикат P), иногда — квантификацией переменной x . Переменная, на которую навешен квантор, называется *связанной*; несвязанная переменная называется *свободной*.

Смысл связанных и свободных переменных в предикатных выражениях различен. Свободная переменная — это обычная переменная, которая может принимать различные значения из M ; выражение $P(x)$ — переменное высказывание, зависящее от значения x . Выражение $\forall x P(x)$ не зависит от переменной x и при фиксированных P и M имеет вполне определенное значение. Это, в частности, означает, что переименование связанной переменной, т. е. переход от $\forall x P(x)$ к $\forall y P(y)$, не меняет истинности выражения. Переменные, являющиеся по существу связанными, встречаются не только в логике. Например, в выражениях

$$\sum_{x=1}^{10} f(x) \text{ или } \int_a^b f(x) dx$$

переменная x связана; при фиксированной f первое выражение равно определенному числу, а второе становится функцией от a и b .

Навешивать кванторы можно и на многоместные предикаты, и вообще на любые логические выражения, которые при этом заключаются в скобки. Выражение, на

которое навешивается квантор $\forall x$ или $\exists x$, называется *областью действия квантора*; все вхождения переменной x в это выражение являются связанными. Навешивание квантора на многоместный предикат уменьшает в нем число свободных переменных и превращает его в предикат от меньшего числа переменных.

Пример 3.12. а. Пусть $P(x)$ — предикат « x — четное число». Тогда высказывание $\forall x P(x)$ истинно на любом множестве четных чисел и ложно, если M содержит хотя бы одно нечетное число; высказывание $\exists x P(x)$ истинно на любом множестве, содержащем хотя бы одно четное число, и ложно на любом множестве нечетных чисел.

б. Теорема Ферма (см. пример 3.11, б) формулируется следующим образом:

$$\begin{aligned} \forall x \forall y \forall z \forall n (N(x) \& N(y) \& N(z) \& N(n) \& (n > 2) \rightarrow \\ \rightarrow \bar{P}_F(x, y, z, n)). \end{aligned}$$

в. Рассмотрим двухместный предикат $x \geq y$ на множествах M с отношением нестрогого порядка и различные квантификации его переменных. $\forall x (x \geq y)$ — одноместный предикат от y ; если M — множество неотрицательных чисел, то этот предикат истинен в единственной точке: $y = 0$. $\forall x \forall y (x \geq y)$ — высказывание, истинное на множестве, состоящем из одного элемента, и ложное на любом другом множестве. $\exists x \exists y (x \geq y)$ истинно на любом непустом множестве. Высказывание $\exists x \forall y (x \geq y)$ («существует x , такой, что для любого y $x \geq y$ ») утверждает, что в M имеется единственный максимальный элемент. Оно истинно на любом конечном множестве целых чисел, но ложно на множестве $\{1/2, 2/3, \dots, n/(n+1), \dots\}$ или на множестве двоичных векторов, из которого удален вектор, состоящий из одних единиц. Высказывание $\forall y \exists x (x \geq y)$ утверждает, что для любого элемента y существует элемент x не меньший, чем y ; оно истинно на любом непустом множестве ввиду рефлексивности отношения \geq . Последние два высказывания говорят о том, что перестановка кванторов существования и общности меняет смысл высказывания и условия его истинности.

Истинные формулы и эквивалентные соотношения. При логической (истинностной) интерпретации формул логики предикатов возможны три основные ситуации.

1. Если в области M для формулы F существует такая подстановка констант вместо всех переменных, что F становится истинным высказыванием, то формула F называется выполнимой в области M . Если существует область M , где F выполнима, то F называется просто *выполнимой*. Пример выполнимой формулы: $\exists xP(x, y) \rightarrow \forall xP(x, y)$. Она выполнима, если M состоит из одного элемента.

2. Если формула F выполнима в M при любых подстановках констант, то она называется тождественно истинной в M . Формула, тождественно истинная в любых M , называется *тождественно истинной* или *логически общезначимой*. Например, формула $\exists xP(x, y) \rightarrow \forall xP(x, y)$ тождественно истинна во всех M , состоящих из одного элемента, но может принимать ложные значения в областях, содержащих более одного элемента. Поэтому она не является логически общезначимой. Формула $\forall x(P(x) \vee \bar{P}(x))$ тождественно истинна.

3. Если формула F невыполнима в M , она называется тождественно ложной в M . Если F невыполнима ни в каких M , она называется *тождественно ложной*, или *противоречивой*. Например, формула $P_1(x, y) \& \bar{P}_1(x, z) \& \& P_2(x) \& \bar{P}_2(y) \& \bar{P}_2(z)$ тождественно ложна на любой области M , если $|M| \leq 2$ (предлагаем читателю это проверить). Формула $\exists x(P(x) \& \bar{P}(x))$ тождественно ложна.

Формулы называются *эквивалентными*, если при любых подстановках констант они принимают одинаковые значения. В частности, все тождественно истинные формулы (и все ложные формулы) эквивалентны. Отметим, что если F_1 и F_2 эквивалентны в соответствии с этим определением, то формула $F_1 \sim F_2$ тождественно истинна.

Множество истинных формул логики предикатов входит в любую теорию, и, следовательно, его исследование является важнейшей целью логики предикатов. В частности, как следует из сделанного ранее замечания, в нем содержатся все эквивалентные соотношения логики предикатов. В этом исследовании прежде всего возникают две проблемы: получение истинных формул и проверка формул на истинность. Если вспомнить классификацию способов задания множеств (\S 1.1), то первая проблема — это проблема построения порождающей процедуры, а вторая — проблема разрешающей процедуры

для множества истинных формул. Те же проблемы встают и в логике высказываний. Однако там есть стандартная разрешающая процедура: вычисление формул на наборах значений переменных. С ее помощью порождающую процедуру для множества M_m тождественно истинных высказываний можно организовать следующим образом: строим последовательно все формулы, вычисляем каждую из них на всех наборах и включаем в M_m только те, которые истинны на всех наборах. Аналогичная процедура в логике предикатов сталкивается с большими трудностями, связанными с тем, что предметные и предикатные переменные имеют в общем случае бесконечные области определения. Поэтому прямой перебор всех значений невозможен, и приходится использовать различные косвенные приемы. Покажем их на примере некоторых эквивалентных соотношений

$$\overline{\exists x P(x)} \sim \overline{\forall x P(x)}. \quad (3.31)$$

Пусть для некоторого предиката P и области M левая часть истинна. Тогда не существует $a \in M$, для которого $P(a)$ истинно; следовательно, для всех a $P(a)$ ложно, т. е. $\overline{P(a)}$ истинно, и правая часть истинна. Если же левая часть ложна, то существует $a \in M$, для которого $P(a)$ истинно, и, следовательно, правая часть ложна. Аналогично доказывается

$$\overline{\forall x P(x)} \sim \exists x \overline{P(x)}. \quad (3.32)$$

Докажем теперь дистрибутивность $\forall x$ относительно конъюнкции и $\exists x$ относительно дизъюнкции

$$\forall x(P_1(x) \& P_2(x)) \sim \forall xP_1(x) \& \forall xP_2(x). \quad (3.33)$$

Пусть левая часть соотношения истинна для некоторых P_1 и P_2 . Тогда для любого $a \in M$ истинно $P_1(a) \& P_2(a)$, поэтому $P_1(a)$ и $P_2(a)$ одновременно истинны для любых a , и, следовательно, $\forall xP_1(x) \& \forall xP_2(x)$ истинно. Если же левая часть ложна, то для некоторого $a \in M$ должно либо $P_1(a)$, либо $P_2(a)$, а следовательно, ложно либо $\forall xP_1(x)$, либо $\forall xP_2(x)$, и правая часть ложна. Аналогично доказывается

$$\exists x(P_1(x) \vee P_2(x)) \sim \exists xP_1(x) \vee \exists xP_2(x). \quad (3.34)$$

Если же $\forall x$ и $\exists x$ в этих соотношениях поменять местами, то получатся соотношения, верные лишь в одну сторону:

$$\exists x(P_1(x) \& P_2(x)) \rightarrow \exists xP_1(x) \& \exists xP_2(x); \quad (3.35)$$

$$(\forall xP_1(x) \vee \forall xP_2(x)) \rightarrow \forall x(P_1(x) \vee P_2(x)). \quad (3.36)$$

В таких случаях говорят, что левая часть — более сильное утверждение, чем правая, поскольку она требует для своей истинности выполнения более жестких условий, чем правая. Так, в (3.35) в левой части требуется, чтобы $P_1(a)$ и $P_2(a)$ были истинны для одного и того же a , тогда как в правой части P_1 и P_2 могут быть истинны при различных a_1 и a_2 . В (3.36) левая часть требует, чтобы хотя бы один предикат выполнялся для всех $a \in M$; в правой части достаточно, чтобы один предикат был истинен там, где ложен другой. В этих рассуждениях по существу уже содержатся доказательства; окончательное их уточнение предоставляем читателю. Пример, когда (3.35) и (3.36) в обратную сторону неверны: $P_1(x)$: « x — четное число», $P_2(x)$: « x — нечетное число».

Приведем без доказательства еще несколько соотношений:

$$\forall x \forall y P(x, y) \sim \forall y \forall x P(x, y); \quad (3.37)$$

$$\exists x \exists y P(x, y) \sim \exists y \exists x P(x, y). \quad (3.38)$$

Пример 3.12, в показывает, что перестановка различных кванторов не является эквивалентностью.

Пусть Y — переменное высказывание или формула, не содержащая x . Тогда

$$\forall x(P(x) \& Y) \sim \forall xP(x) \& Y; \quad (3.39)$$

$$\forall x(P(x) \vee Y) \sim \forall xP(x) \vee Y; \quad (3.40)$$

$$\exists x(P(x) \& Y) \sim \exists xP(x) \& Y; \quad (3.41)$$

$$\exists x(P(x) \vee Y) \sim \exists xP(x) \vee Y. \quad (3.42)$$

Эти соотношения означают, что формулу, не содержащую x , можно выносить за область действия квантора, связывающего x .

О методах доказательства в логике предикатов. Метод доказательства общезначимости формул, содержащих переменные, путем непосредственной подстановки в них констант называется методом интерпретаций или методом моделей*.

* Точные понятия интерпретации и модели вводятся в гл. 6, § 6.3.

Подстановка констант позволяет интерпретировать формулу как осмысленное утверждение об элементах конкретного множества M . Поэтому такой метод, выясняющий истинность формулы путем обращения к ее возможному смыслу, называется также семантическим (т. е. смысловым). Метод интерпретаций удобен для доказательства выполнимости формул или их неэквивалентности, поскольку и в том, и в другом случае достаточно найти одну подходящую подстановку (именно так мы поступили ранее, сославшись на пример 3.12, в). Он удобен также для исследования истинности формул на конечных областях. Дело в том, что если область M конечна, $M = \{a_1, \dots, a_n\}$, то кванторы переходят в конечные формулы логики высказываний:

$$\forall x P(x) \sim P(a_1) \ \& \ P(a_2) \ \& \ \dots \ \& \ P(a_n); \quad (3.43)$$

$$\exists x P(x) \sim P(a_1) \ \vee \ P(a_2) \ \vee \ \dots \ \vee \ P(a_n). \quad (3.44)$$

Заменив все кванторы в формуле по этим соотношениям, любую формулу логики предикатов можно перевести в формулу, состоящую из предикатов, соединенных знаками логических операций. Истинность такой формулы на конечной области проверяется конечным числом подстановок и вычислений.

Для бесконечных же областей в общем случае и прежде всего при доказательстве тождественной истинности формул метод интерпретаций, как уже отмечалось, связан с большими трудностями. Поэтому для построения множества истинных формул в логике предикатов выбирается другой путь. Это множество порождается из исходных формул (аксиом) с помощью формальных процедур — правил вывода. Слово «формальный» (которое часто противопоставляется слову «содержательный») подчеркивает здесь то обстоятельство, что при переходе от одних выводимых формул к другим не происходит какого-либо обращения к содержанию, смыслу формул. Используются лишь формальные, внешние свойства последовательностей символов, образующих формулы, причем эти свойства полностью описываются правилами вывода. Важность формального подхода в том, что благодаря ему удается избежать обращения к бесконечной области (что неизбежно при методе интерпретаций) и на каждом шаге вывода оперировать только с конечным множеством символов. Методы рассуждений, использующие только конечные множества конечных объектов, называются финитными.

Множества, порожденные такими формальными методами, называются формальными системами (см. гл. 6).

ГРАФЫ

4.1. ОСНОВНЫЕ ПОНЯТИЯ

С понятием графа обычно связывается его графическое представление, при котором он изображается как множество точек, некоторые из которых соединены линиями. Однако граф отличается от геометрических конфигураций (скажем, фигур, которые также состоят из точек-вершин и линий-сторон) тем, что в графе несущественны расстояния между точками, форма соединяющих линий и углы между ними. Важно лишь, соединена ли данная пара точек линией или нет. Поэтому граф иногда называют топологическим объектом, т. е. объектом, свойства которого не изменяются при растягивании, сжатии, искривлении (но без разрывов и склеиваний). По этой же причине (важно лишь наличие или отсутствие соединения) граф — объект дискретный и может быть задан двумя дискретными множествами: множеством точек, которые будем называть вершинами, и множеством линий, соединяющих некоторые вершины.

Существуют два основных вида графов (и множество их подвидов): ориентированные, в которых линии имеют направление от одной точки к другой, и неориентированные, в которых линии не имеют направления.

Неориентированные графы. Смежность, инцидентность. Неориентированным графом $G(V, E)$ называется объект, заданный парой множеств (V, E) , где V — множество вершин, E — множество линий. Линии будем называть ребрами. Каждое ребро соединяет ровно две вершины. Граф называется *простым*, если каждую пару вершин соединяет не более чем одно ребро. Граф называется

мультиграфом, если хотя бы одну пару вершин соединяют более чем одно ребро. Ребра мультиграфа, соединяющие одну и ту же пару вершин, называются *кратными*.

В простом графе ребро однозначно определяется парой вершин, которые оно соединяет, причем порядок вершин в паре неважен. Поэтому иногда в определении простого графа E определяется как множество неупорядоченных пар вершин. В мультиграфе каждое ребро должно иметь свое собственное имя.

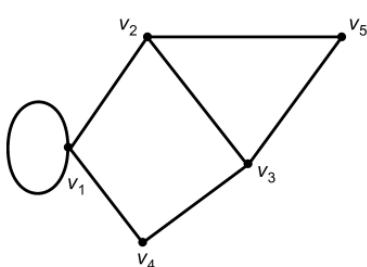
Пример 4.1. Граф G_1 на рис. 4.1 — это простой неориентированный граф, а граф G_2 — неориентированный мультиграф с поименованными ребрами. Ребра e_1 и e_2 , а также ребра e_3 и e_4 — кратные.

Вершины, соединяемые ребром, не обязательно различны. Ребро, соединяющее вершину v_i с самой собой, т. е. пара (v_i, v_i) , называется *петлей*.

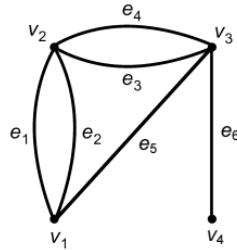
Граф может вовсе не иметь ребер: $E = \emptyset$. Такой граф называется пустым или 0-графом.

Для простого графа существует другой крайний случай — все вершины соединены между собой ребрами. Такой граф называется *полным*, причем различают два варианта полных графов — с петлями и без петель. Полный граф с n вершинами имеет $(n^2 - n)/2$ ребер, если петли не учитывать (число сочетаний из n по 2), и $(n^2 - n)/2 + n = = (n^2 + n)/2$ ребер, если добавить n петель. Понятно, что в мультиграфе ограничения на число ребер нет.

Неориентированный граф задает два отношения между своими элементами: отношение *смежности* и отношение *инцидентности*. Смежность — отношение между вершинами: две вершины называются смежными, если



Простой неориентированный граф G_1



Неориентированный мультиграф G_2

Рис. 4.1

они соединены ребром. Это отношение — обычное бинарное отношение на множестве V , которое для простого графа может быть задано квадратной бинарной (т. е. состоящей из нулей и единиц) матрицей C , в которой клетка $c_{ij} = 1$, если в графе есть ребро (v_i, v_j) , и $c_{ij} = 0$ в противном случае. Оно всегда симметрично, поскольку порядок вершин в паре (v_i, v_j) неважен. Наличие рефлексивности и транзитивности зависит от конкретных свойств графа.

Матрица смежности пустого графа заполнена только нулями, а матрица смежности полного графа с петлями — только единицами. Для мультиграфа матрица смежности уже не является бинарной: в ней $c_{ij} = k$, где k — число кратных ребер, соединяющих вершины v_i и v_j .

Пример 4.2. Матрица смежности графа G_1 , изображенного на рис. 4.1, имеет следующий вид:

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}.$$

Матрица смежности мультиграфа G_2 , изображенного на том же рисунке, содержит не только нули и единицы:

$$\begin{pmatrix} 0 & 2 & 1 & 0 \\ 2 & 0 & 2 & 0 \\ 1 & 2 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

Инцидентность — это отношение между вершинами и ребрами: ребро инцидентно каждой из вершин, которые оно соединяет. Оно может быть задано прямоугольной бинарной матрицей D , в которой число строк равно числу вершин графа, а число столбцов — числу ребер. Клетка $d_{ij} = 1$, если вершина v_i инцидентна ребру e_j , и $d_{ij} = 0$ в противном случае. Для графа G_2 матрица инцидентности приведена в табл. 4.1.

Таблица 4.1						
	e_1	e_2	e_3	e_4	e_5	e_6
v_1	1	1	0	0	1	0
v_2	1	1	1	1	0	0
v_3	0	0	1	1	1	1
v_4	0	0	0	0	0	1

Число ребер, инцидентных вершине v_i , называется *степенью* этой вершины. Степень вершины v_i равна числу единиц в i -й строке матрицы смежности. Сумма степеней вершин равна удвоенному числу ребер, поскольку каждое ребро участвует в степенях двух вершин, т. е. считается в этой сумме два раза. Поскольку эта сумма четна, то и число вершин с нечетными степенями тоже четно. Вершина, степень которой равна 1, называется *концевой*, или *висячей*. Вершина степени 0 называется *изолированной*. Граф называется *однородным* степени k , если степени всех его вершин равны k .

Граф $G'(V', E')$ называется *подграфом* графа $G(V, E)$, если $V' \subset V$, а E' — множество всех ребер G , оба конца которых принадлежат V' . Подграф G' графа G называется *максимальным* по некоторому свойству, если G' обладает этим свойством, а любой подграф графа G , содержащий G' , не обладает им. Подграф G' графа G называется *минимальным* по некоторому свойству, если G' обладает этим свойством, а любой подграф графа G , содержащийся в G' , не обладает им. Граф $G'(V', E')$ называется *частью* графа $G(V, E)$, если $V' \subset V$, а E' — подмножество множества всех ребер G , оба конца которых принадлежат V' .

Всякий подграф графа G является частью G , но не всякая часть — подграф. Подграф G' полностью определяется множеством V' своих вершин. Поэтому его можно обозначать короче: $G'(V')$. Он может быть построен так: в исходном графе G выбираем множество вершин V' и удаляем все ребра, хотя бы один конец которых не принадлежит V' . Часть графа — это подграф, в котором, быть может, удалены некоторые ребра. Например, часть G может содержать все вершины G , но не все его ребра.

Иногда подграф $G'(V', E')$ называют графом, порожденным множеством V' (или просто *вершинно-порожденным графом*).

Пример 4.3. На рис. 4.2 изображены граф G_1 (тот же, что и на рис. 4.1), его подграф, порожденный множеством вершин v_2, v_3, v_5 , и его часть, содержащая все вершины G_1 , но не все его ребра.

Подграф $G'(V')$ содержится в подграфе $G''(V'')$ графа G (обозначение $G' \subseteq G''$), если $V' \subseteq V''$. Отношение включения подграфов графа G является отношением частичного

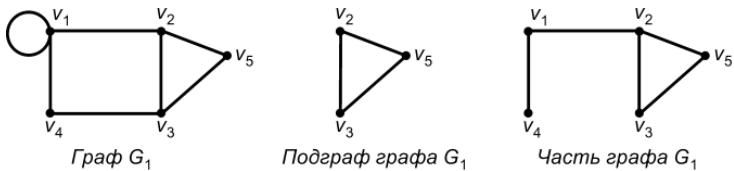


Рис. 4.2

порядка, где единственным максимальным элементом является сам граф G .

Аналогично можно определить реберно-порожденные графы.

Часть $G'(V', E')$ графа $G(V, E)$, где V' — множество всех вершин, инцидентных ребрам из E' , называется *графом, порожденным множеством E'* (обозначение $G'(E')$), или просто *реберно-порожденным графом*.

Реберно-порожденный граф не содержит изолированных вершин.

Часть графа G , которая содержит вершину v_i , все инцидентные ей ребра и все вторые концы этих ребер (т. е. вершины, смежные с v_i), называется *звездой* вершины v_i . Звезда вершины v_i является подграфом, только если никакая пара вершин, смежных с v_i , не соединена ребром.

Если подграф, определяемый заданным множеством вершин, является полным, он называется *кликой*.

Пример 4.4. На рис. 4.3 приведены клика графа G_1 (см. рис. 4.1) и звезда вершины v_2 .

Операции над графиками. Пусть даны два графа $G_1(V_1, E_1)$ и $G_2(V_2, E_2)$.

Объединением $G_1 \cup G_2$ графов G_1 и G_2 называется граф $G(V_1 \cup V_2, E_1 \cup E_2)$, т. е. граф, множества вершин и ребер которого являются объединениями соответствующих множеств графов G_1 и G_2 . *Пересечением* $G_1 \cap G_2$ графов G_1 и G_2 называется граф $G(V_1 \cap V_2, E_1 \cap E_2)$, т. е. граф, множества вершин и ребер которого являются пересечениями соответствующих множеств графов G_1 и G_2 . *Дополнением* \bar{G} графа $G(V, E)$ называется граф $\bar{G}(V, \bar{E})$, т. е. граф, у которого множество вершин — то же, что и в G , а ребро входит в \bar{E} , если и только если оно не входит в E .

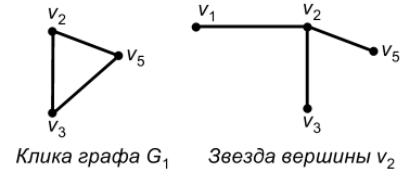
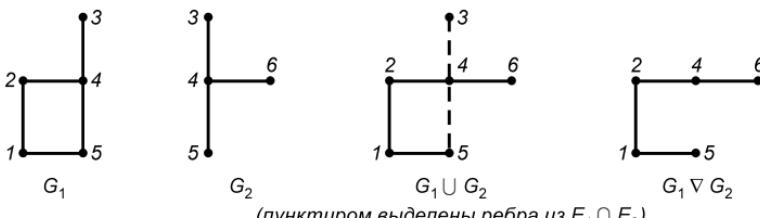


Рис. 4.3

Симметрической разностью $G_1 \Delta G_2$ графов G_1 и G_2 называется реберно-порожденный граф $G_3(E_3)$, где $E_3 = E_1 \Delta E_2 = (E_1 \cup E_2) \setminus (E_1 \cap E_2)$. E_3 — это объединение E_1 и E_2 , из которого удалены ребра, общие для E_1 и E_2 , а V_3 — это объединение V_1 и V_2 , из которого удалены только те совпадающие вершины, которые не инцидентны ребрам из E_3 .

Пример 4.5. На рис. 4.4 приведены два простых графа G_1 , G_2 и графы, полученные из них операциями объединения, пересечения и симметрической разности. Граф $G_1 \cap G_2$ является частью графа $G_1 \cup G_2$ и содержит только ребра (3, 4) и (4, 5), выделенные пунктиром.



(пунктиром выделены ребра из $E_1 \cap E_2$)

Рис. 4.4

Удалением вершины v_i из графа $G(V, E)$ называется операция, которая удаляет из графа G вершину v_i и все инцидентные ей ребра. В результате получается подграф $G(V \setminus v_i)$. *Удалением* ребра e_i из графа $G(V, E)$ называется операция, которая из графа G порождает граф $G'(V, E \setminus e_i)$. Концы ребра e_i не удаляются.

Ориентированные графы. *Ориентированным графом (орграфом)* G называется граф $G(V, E)$, где V — множество вершин, E — множество ориентированных ребер. Ориентированное ребро e , соединяющее вершину v_i с вершиной v_j , задается упорядоченной парой вершин $e = (v_i, v_j)$, имеет начало (вершину v_i , из которой оно выходит) и конец (вершину v_j , в которую оно заходит).

Для орграфа его бинарная матрица смежности C в общем случае несимметрична: клетка $c_{ij} = 1$, если и только если имеется ребро $e = (v_i, v_j)$. Число единиц в этой матрице равно числу ребер графа. (Заметим, что в матрице смежности неориентированного графа петле соответствует одна единица, стоящая на главной диагонали, а остальным ребрам — по две единицы, расположенные в клетках, симметричных относительно главной диагонали.) Если же матрица смежности орграфа G оказывается симметричной, то

это означает, что для каждого ребра (v_i, v_j) в нем имеется ребро (v_j, v_i) , т. е. ребро, соединяющее те же вершины, но в противоположном направлении. Такая матрица совпадает с матрицей смежности неориентированного графа, полученного из G заменой каждой пары противоположно ориентированных ребер (v_i, v_j) и (v_j, v_i) на одно неориентированное ребро (v_i, v_j) . Поэтому симметричный орграф всегда можно заменить простым неориентированным графом, имеющим ту же матрицу смежности.

Понятие инцидентности для орграфов сохраняется: говоря об инцидентности ребра вершине, не различают, инцидентно ли оно начальной или конечной вершине. Однако в матрице инцидентности D удобно начало и конец ребра различать; правда, при этом матрица перестает быть бинарной: клетка $d_{ij} = -1$, если вершина v_j — начало ребра e_j , $d_{ij} = 1$, если вершина v_j — конец ребра e_j , и $d_{ij} = 0$, если соответствующего ребра нет.

Необходимость учитывать ориентацию ребер в орграфе приводит к расщеплению понятия «степень вершины» на две части: *полустепенью захода* вершины v_i называется число ребер, заходящих в v_i ; *полустепенью исхода* v_i — число ребер, выходящих из нее. Полустепень исхода v_i равна числу единиц в i -й строке матрицы смежности, полустепень захода — числу единиц в i -м столбце матрицы смежности.

Понятие подграфа для орграфа остается тем же. Понятие звезды, как и степень, расщепляется на две части. *Полузвезда захода* вершины v_i — это подграф, определяемый вершиной v_i и всеми вершинами, из которых в v_i ведут ребра. *Полузвезда исхода* вершины v_i — это подграф, определяемый вершиной v_i и всеми вершинами, в которые в v_i ведут ребра.

Итак, графы и орграфы могут быть заданы тремя способами:

- непосредственным заданием множеств V и E (например, списком их элементов);
- матрицей смежности или матрицей инцидентности (правда, мультиграф матрицей смежности однозначно задан быть не может, поскольку эта матрица не содержит имен ребер);
- рисунком.

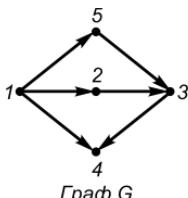
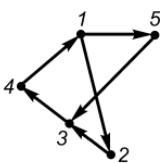


Рис. 4.5



Тот же граф G

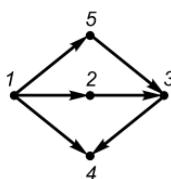


Рис. 4.6

Когда два графа одинаковы? Для первых двух способов задания ответ прост: когда совпадают их описания — списки вершин и ребер или матрицы. По рисунку определить, одинаковы ли графы, сложнее. Один и тот же граф можно изобразить разными рисунками (рис. 4.5), по-разному расположив вершины и придав ребрам разную геометрическую форму и длину.

Например, графы на рис. 4.5 одинаковы и имеют одну и ту же матрицу смежности:

$$C = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}.$$

С другой стороны, два, на первый взгляд, одинаковых рисунка могут задавать разные графы.

Графы на рис. 4.6 геометрически одинаковы. Однако они отличаются нумерацией вершин: вершины v_3 и v_4 во втором графе поменялись местами, из-за чего матрицы смежности и списки ребер у них будут различны. Например, ребро (v_5, v_3) есть в первом графе, но отсутствует во втором: вместо него появилось ребро (v_5, v_4) . Поэтому множества ребер этих графов различны и, следовательно, различны сами графы.

Графы, которые отличаются только нумерацией вершин (и которые с помощью перенумерации можно сделать одинаковыми), называются *изоморфными*. Изоморфизм графов с небольшим числом вершин иногда можно непосредственно увидеть (например, изоморфизм графов на рис. 4.6 очевиден). Однако в общем случае проблема установления изоморфизма графов оказывается сложной в вычислительном отношении задачей.

Графы и бинарные отношения. Между простыми (без кратных ребер) орграфами и бинарными отношениями существует взаимно однозначное соответствие. Всякий

орграф с множеством вершин $V = \{v_1, \dots, v_n\}$ определяет бинарное отношение на множестве V — отношение смежности. Матрица смежности этого графа — это матрица бинарного отношения смежности. Верно и обратное — всякое бинарное отношение R на произвольном множестве $M = \{m_1, \dots, m_n\}$ можно представить орграфом G , вершины которого соответствуют элементам M , а ребро (m_i, m_j) в этом графе существует, если и только если выполняется $m_i R m_j$. Бинарная матрица отношения R одновременно является матрицей смежности орграфа G , а сам граф можно называть графом отношения R .

Граф рефлексивного отношения содержит петли во всех вершинах и, соответственно, единицы во всех элементах главной диагонали. Симметричному отношению соответствует граф с симметрической матрицей смежности. Как было отмечено выше, такой орграф равносителен простому неориентированному графу. Граф транзитивного отношения обладает следующим свойством: если существуют ребра (v_i, v_j) и (v_j, v_k) , то существует ребро (v_i, v_k) . Граф отношения эквивалентности представляет собой совокупность полных подграфов.

Дополнению \bar{R} отношения R (т. е. отношению, которое истинно, когда R ложно) соответствует дополнение графа G до полного орграфа, т. е. орграф \bar{R} , в котором имеются те и только те ребра, которых нет в G . Обратному отношению R^{-1} соответствует орграф G^{-1} , который получен из орграфа G изменением ориентации всех его ребер на противоположные.

4.2. ПУТИ И СВЯЗНОСТЬ В НЕОРИЕНТИРОВАННЫХ ГРАФАХ

Основные определения. Путь в неориентированном графе — это последовательность ребер $(v_{i0}, v_{i1}), (v_{i1}, v_{i2}), \dots (v_{in-1}, v_{in})$, такая, что любые два соседние ребра различны и конец каждого ребра совпадает с началом следующего ребра*. Вершина v_{i0} называется началом пути, вершина v_{in} — концом пути. В мультиграфе при задании

* Неориентированные ребра не имеют начала и конца, однако последовательность ребер определяет порядок их просмотра, и началом ребра пути считается вершина, первая в порядке этого просмотра.

пути нужно указывать имена ребер. Число ребер в пути P называется его *длиной* и обозначается $l(P)$. Путь называется *циклическим*, или просто *циклом*, если $v_{i0} = v_{in}$. Цикл называется *простым*, если любая вершина графа встречается в ней не более чем один раз.

Если конец пути P_1 совпадает с началом пути P_2 , то, приписав справа последовательность ребер P_2 к последовательности ребер P_1 , получим новый путь, ведущий из начала P_1 в конец P_2 . Этот путь будем обозначать P_1P_2 .

Очевидно, что если в неориентированном графе существует путь из v_{i0} в v_{in} , то существует путь из v_{in} в v_{i0} — это тот же путь, «прочтенный» справа налево, т. е. пройденный в обратном направлении.

Одно и то же ребро может встречаться в пути несколько раз. Путь называется *цепью*, если каждое ребро встречается в нем не более одного раза, и *простой цепью* (или *простым путем*), если любая вершина графа встречается в ней не более чем один раз. Простая цепь — это цепь, которая не пересекает сама себя.

Вершины v_i и v_j называются *связанными*, если существует путь с началом в v_i и концом в v_j . В этом случае говорят также, что вершина v_j *достижима* из вершины v_i . Каждая вершина по определению связана сама с собой путем нулевой длины.

Связанность — это бинарное отношение на множестве вершин. Оно рефлексивно (каждая вершина по определению связана сама с собой), симметрично (для каждого пути имеется обратный путь) и транзитивно. Транзитивность означает, что если есть путь из v_i в v_j и путь из v_j в v_k , то есть путь из v_i в v_k . Это очевидно: чтобы получить такой путь, достаточно к последовательности ребер, ведущей из v_i в v_j , приписать справа последовательность ребер, ведущую из v_j в v_k . Таким образом, отношение связности является отношением эквивалентности на множестве вершин графа G и разбивает это множество на непересекающиеся подмножества — классы эквивалентности. Все вершины одного класса связаны между собой, вершины из разных классов между собой не связаны. Подграф, образованный всеми вершинами одного класса, называется *компонентой связности* графа G . Можно дать и другое определение компоненты связности.

Неориентированный граф называется *связным*, если все его вершины связаны между собой. Максимальный (в смысле определения 4.2) связный подграф графа G называется *компонентой связности* графа G . Связный граф состоит из одной компоненты связности.

Пример 4.6. Граф, состоящий из двух компонент связности (рис. 4.7).

Теорема 4.1. Если две вершины связаны между собой, то существует связывающая их простая цепь.

Действительно, если путь, связывающий эти вершины, не является простой цепью, то в нем имеется вершина v , инцидентная более чем двум ребрам этого пути.

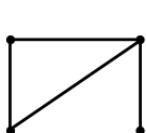


Рис. 4.7

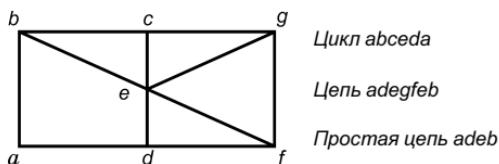


Рис. 4.8

Пусть e_i — первое из этих ребер, e_j — последнее ($j > i + 1$). Тогда из данного пути можно удалить участок от $(i + 1)$ -го ребра до $(j - 1)$ -го. Полученная последовательность останется путем: в ней ребра e_i и e_j станут соседними, и при этом они имеют общую вершину v . Если полученный путь не является простой цепью, то процесс повторяется до получения простой цепи. \square

Пример 4.7. На рис. 4.8 показаны примеры цикла, цепи и простой цепи.

Вершина графа называется *точкой сочленения*, если ее удаление увеличивает число связных компонент графа. Граф называется *разделимым*, если он содержит хотя бы одну точку сочленения, и *неразделимым*, если он не содержит таких точек. Максимальные (см. определение 4.2) неразделимые подграфы графа называются его *блоками*.

Теорема 4.2. Вершина v_i является точкой сочленения связного графа G , если и только если существуют такие вершины v_j и v_k , отличные от v_i , что любой путь между ними проходит через v_i .

Пусть v_i — точка сочленения. Ее удаление дает новый граф G' , содержащий несколько связных компонент.

Выберем вершины v_j и v_k так, чтобы они лежали в разных компонентах. Тогда в G' между ними пути нет. Но в G (в силу его связности) между ними есть пути (по крайней мере один). Значит, именно удаление v_i разорвало эти пути, и следовательно, все они проходят через v_i .

Пусть теперь существуют вершины v_j и v_k , указанные в условии теоремы. Тогда удаление v_i разрывает все пути между ними, граф становится несвязным, и следовательно, v_i — точка сочленения. \square

Разделимые графы называют еще 1-связными. Вообще, i -связным называют граф, для нарушения связности которого надо удалить не менее i вершин. Можно сказать, что число связности i характеризует надежность связности. Если граф изображает, например, сеть коммуникаций, то это число говорит о том, что при повреждении любых ($i - 1$) узлов сеть все еще обеспечивает связь между любыми оставшимися узлами.

Расстояния. Диаметр, радиус, центр. Минимальная из длин простых цепей, связывающих вершины v_i и v_j неориентированного графа, называется *расстоянием* между этими вершинами и обозначается $d(v_i, v_j)$. Поскольку по определению каждая вершина связана сама с собой, то $d(v_i, v_i) = 0$.

Расстояние $d(v_i, v_j)$ удовлетворяет аксиомам метрики:

- 1) $d(v_i, v_j) \geq 0$, причем $d(v_i, v_j) = 0$, если и только если $v_i = v_j$;
- 2) $d(v_i, v_j) = d(v_j, v_i)$;
- 3) $d(v_i, v_j) + d(v_j, v_k) \geq d(v_i, v_k)$ (неравенство треугольника).

Выполнение первых двух свойств очевидно. Доказательство третьего также несложно. Пусть P_{ij} — кратчайшая цепь из v_i в v_j , P_{jk} — кратчайшая цепь из v_j в v_k . Тогда путь $P_{ij}P_{jk}$, длина которого равна $l(P_{ij}P_{jk}) = l(P_{ij}) + l(P_{jk}) = d(v_i, v_j) + d(v_j, v_k)$, ведет из v_i в v_k . Следовательно, $d(v_i, v_k)$ либо равно $d(v_i, v_j) + d(v_j, v_k)$, либо меньше этой суммы, если существует более короткий путь из v_i в v_k , и аксиома 3 выполняется.

Диаметром $d(G)$ графа G называется максимальное из расстояний между его вершинами:

$$d(G) = \max_{v_i, v_j \in G} d(v_i, v_j).$$

Максимальным удалением от вершины v_i называется величина

$$r(v_i) = \max_{v_j \in G} d(v_i, v_j).$$

Вершина v называется центром графа G , если $r(v)$ минимально среди других вершин графа:

$$r(v) = \min_{v_i \in G} r(v_i).$$

Максимальное удаление $r(v)$ от центра v называется радиусом графа G и обозначается $r(G)$.

Пример 4.8. Для графа на рис. 4.9 приведены величины $r(v)$ для всех его вершин, радиус и диаметр.

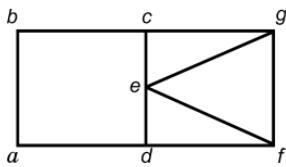


Рис. 4.9

$$\begin{array}{ll} r(a) = 3; & r(b) = 3; \\ r(c) = 2; & r(d) = 3; \\ r(f) = 3; & r(g) = 3; \\ r(G) = 2; & d(G) = 3. \end{array}$$

Число центров и соотношения между радиусом и диаметром в графе могут быть различными. В полном неориентированном графе диаметр и радиус равны единице и все вершины — центры.

Если график G — простая цепь с нечетным числом $2n + 1$ вершин, то $(n + 1)$ -я от начала вершина — единственный центр, $d(G) = 2n$, $r(G) = n$. Если же график G — простая цепь с четным числом $2n$ вершин, то n -я и $(n + 1)$ -я от начала вершины — два центра, $d(G) = 2n - 1$, $r(G) = n - 1$ (рис. 4.10).



Рис. 4.10

В графике G на рис. 4.11, топологически эквивалентном окружности (вернее, тележному колесу), $d(G) = 2$, $r(G) = 1$, т. е. диаметр, как и в геометрическом круге, в два раза больше радиуса.

Эйлеров обход. Цикл в неориентированном графике называется *эйлеровым обходом* или *эйлеровым циклом*, если он содержит все ребра графа в точности по одному разу. Граф называется *эйлеровым*, если в нем существует эйлеров цикл.

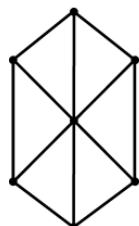


Рис. 4.11

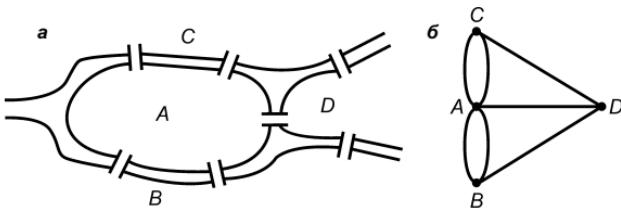


Рис. 4.12

Не всякий граф — эйлеров. Это установил великий математик Л. Эйлер, занимаясь задачей о кенигсбергских мостах. В Кенигсберге во времена Эйлера было семь мостов (рис. 4.12, а). Задача заключается в том, чтобы пройти каждый мост по одному разу и вернуться в исходную точку. Эйлер свел эту задачу к задаче нахождения обхода графа (рис. 4.12, б) и показал, что она не имеет решения.

Необходимые и достаточные условия существования эйлерова цикла, найденные Эйлером, в современных терминах формулируются в следующей теореме.

Теорема 4.3. Неориентированный граф является эйлеровым тогда и только тогда, когда он связан и все степени его вершин четны.

Необходимость. То, что в несвязном графе не может быть цикла, проходящего через все ребра, очевидно. Пусть теперь граф связан, и в нем существует эйлеров цикл с началом в вершине v_1 . Пройдем по этому циклу, помечая пройденные ребра. Через каждую вершину цикл может проходить несколько раз, добавляя каждый раз два помеченных ребра. При каждом проходе через вершину v_1 будет помечено нечетное число ребер (потому что при первом выходе из v_1 было помечено одно ребро), а при проходах через другие вершины — четное число ребер. Поэтому цикл может закончиться в v_1 только в том случае, когда степень v_1 четна (цикл входит по последнему непомеченному ребру, и к нечетному числу помеченных ребер добавляется единица). Во всех остальных вершинах все ребра будут помеченными, т. е. число помеченных ребер в каждой вершине будет равно ее степени, которая тем самым окажется четной.

Достаточность. Покажем, как построить эйлеров цикл в связном графе, у которого все степени вер-

шин четны. Выберем какую-либо вершину v_1 и будем строить из нее путь в произвольном порядке, помечая уже пройденные ребра и включая в путь только те ребра, которые еще не помечены. При входе в вершину, отличную от v_1 , помечается нечетное по счету ребро, при выходе — четное; в силу четности ее степени из нее всегда можно выйти по непомеченому ребру. Поэтому наш путь может закончиться только в v_1 (см. доказательство необходимости), образовав таким образом цикл P_1 . Если при этом все ребра графа оказались помеченными, то искомый эйлеров цикл построен. Если же это не так, то для любой вершины, имеющей инцидентные ей непомеченные ребра, их число четно. Найдем первую такую вершину на пути P_1 (обозначим ее v_2) и будем строить из нее путь по непомеченным ребрам. По указанным выше причинам (четность всех степеней и четность числа непомеченных ребер) этот путь может закончиться только в v_2 , образовав цикл P_2 . Этот цикл вставим в P_1 в точке v_2 . Получим новый цикл P_{12} . Если и после этого остались непомеченные ребра, строим еще один цикл P_3 и вставляем его в P_{12} . Этот процесс построения циклов P_i и вставки их в уже построенный цикл $P_{12\dots i-1}$ продолжаем до тех пор, пока не останется непомеченных ребер. Полученный в конечном счете цикл $P_{12\dots k}$ будет эйлеровым. \square

Пример 4.9. Пример построения эйлерова обхода приведен на рис. 4.13.

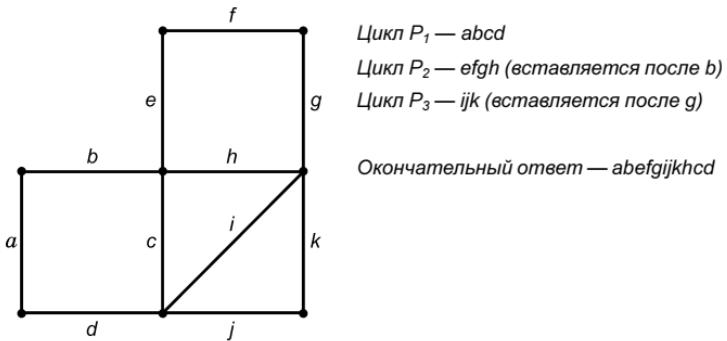


Рис. 4.13

Из метода построения эйлерова цикла вытекает еще один критерий «эйлеровости» связного графа.

Теорема 4.4. Неориентированный граф является эйлеровым тогда и только тогда, когда он связан и является объединением нескольких циклов, не пересекающихся по ребрам.

1. Пусть граф — эйлеров. Тогда эйлеров обход в нем можно построить методом, описанным в предыдущей теореме. Этот обход представляет собой объединение циклов P_1, P_2, \dots, P_k , которые не пересекаются по ребрам, поскольку каждый следующий цикл содержит только непомеченные ребра, т. е. ребра, не вошедшие в предыдущие циклы.

2. Пусть граф связан и является объединением нескольких циклов, не пересекающихся по ребрам. Тогда любое ребро графа принадлежит некоторому циклу. Если через вершину проходит только один цикл, то ее степень равна двум. Если же через вершину проходит несколько циклов, то поскольку они не пересекаются по ребрам, каждый цикл добавляет в степень вершины ровно два ребра. Поэтому все вершины графа имеют четные степени, и, следовательно, граф — эйлеров.

Гамильтонов обход. Цикл в неориентированном графе называется *гамильтоновым обходом* или *гамильтоновым циклом*, если он содержит все вершины графа в точности по одному разу. Граф называется *гамильтоновым*, если в нем существует гамильтонов цикл.

Задача нахождения гамильтонова цикла, поставленная английским математиком Гамильтоном, при всем сходстве ее формулировки с задачей об эйлеровом цикле оказывается гораздо более сложной. Простые критерии существования гамильтонова цикла неизвестны. В то же время интерес к ее решению велик, поскольку она имеет естественную прикладную интерпретацию. Если рассматривать граф как транспортную сеть, вершины которой — города, а ребра — пути между городами, то задача о гамильтоновом цикле оказывается частным случаем известной «задачи о коммивояжере»: объехать все города, побывав в каждом ровно один раз и вернуться в исходный город. Более сложная постановка этой задачи связана со случаем, когда разные пути имеют разную цену в стоимости или длительности; тогда требуется найти обход всех городов с минимальной ценой.

4.3. ПУТИ И СВЯЗНОСТЬ В ОРИЕНТИРОВАННЫХ ГРАФАХ

Виды связности. Путь в ориентированном графе — это последовательность ориентированных ребер (v_{i0}, v_{i1}) , (v_{i1}, v_{i2}) , ... $(v_{i, n-1}, v_{in})$, такая, что конец любого ребра совпадает с началом следующего ребра. Вершина v_{i0} называется началом пути, вершина v_{in} — концом пути.

Ряд понятий для неориентированных графов — длина пути, цикл, цепь, простая цепь — без изменения переносятся на орграфы. Другие понятия, и прежде всего связность и достижимость существенно видоизменяются.

Вершина v_j достижима из вершины v_i , если существует путь с началом в v_i и концом в v_j . По определению полагаем, что любая вершина достижима из себя самой.

Для орграфов верно утверждение, аналогичное теореме 4.1.

Теорема 4.1'. Если вершина v_j достижима из вершины v_i , то существует простой путь из v_i в v_j . \square

Полупуть в ориентированном графе — это последовательность ребер, такая, что любые два соседних ребра различны и имеют общую инцидентную им вершину.

Иначе говоря, полупуть — это путь, который проходится без учета ориентации ребер.

Отношение достижимости между вершинами в орграфах несимметрично: если v_j достижима из v_i , то v_i не обязательно достижима из v_j . Однако полупуть из v_j в v_i в этом случае существует всегда. Возможен случай (рис. 4.14), когда между вершинами нет пути ни в одну, ни в другую сторону, но есть полупуть.



Рис. 4.14

Таким образом, в орграфах существуют различные виды связности, которые описываются следующим определением.

1. Орграф называется *сильно связным*, если любые две вершины достижимы друг из друга (т. е. если между ними существуют пути в обе стороны).

2. Орграф называется *односторонне связным*, если для любой пары вершин существует путь между ними хотя бы в одну сторону.

3. Орграф называется *слабо связным*, если между любой парой вершин существует полупуть.

4. Орграф называется *несвязным*, если между некоторой парой вершин нет полупути (т. е. если он не является слабо связным).

Отметим, что первые три свойства упорядочены по включению: граф, обладающий одним из этих свойств, обладает всеми свойствами, которые в этом определении «ниже» него. Так, сильно связный граф обладает свойствами 2–3 и т. д.

В сильно связном графе любая вершина v_i входит по крайней мере в один цикл, образованный путями из v_i в некоторую другую вершину v_j и обратно из v_j в v_i . Циклы, проходящие через v_i и другие вершины графа, не обязательно все различны. В частности, сильно связный граф, содержащий n вершин, может представлять собой один простой цикл, проходящий через все вершины.

Максимальный сильно связный подграф орграфа G называется *сильно связной компонентой* G .

Каждая вершина v_i орграфа принадлежит ровно одной сильно связной компоненте. Если бы это было не так, то любую пару вершин, принадлежащих разным компонентам, в которые входит v_i , можно было бы связать путями, проходящими через v_i в обе стороны, и, следовательно, эти компоненты можно было бы объединить в одну, что противоречит требованию максимальности компонент.

Таким образом, множество сильно связных компонент образует разбиение орграфа на подграфы. Это разбиение напоминает разбиение на компоненты связности неориентированного графа. Однако если в неориентированном графе между вершинами разных компонент связности нет ребер, то в орграфе такие ребра возможны; но при этом связь между компонентами может существовать только в одну сторону. Более точно: если имеется ребро, идущее из некоторой вершины компоненты K_i в некоторую вершину другой компоненты K_j , то не существует ребер, ведущих из вершин K_j в вершины K_i . В противном случае существовал бы цикл, проходящий через вершины обеих компонент, и эти компоненты можно было бы объединить в одну.

Структура связей между сильно связными компонентами орграфа $G(V, E)$ выясняется в результате построения по графу G нового графа $G'(U, E')$, который строится следующим образом. Каждой сильно связной компоненте K_i графа G поставим в соответствие вершину u_i в G' ; если в G есть ребра, ведущие из вершин K_i в вершины K_j , то в G' проводим соответствующие ребра из u_i в u_j . Полученный граф называется *графом конденсации, конденсированным графом* или просто *конденсацией* графа G . Кроме того, конденсацией называют саму процедуру построения конденсированного графа.

Пример 4.10. Граф, изображенный на рис. 4.15 справа, является результатом конденсации левого графа. Вершина u_1 соответствует сильно связной компоненте, содержащей вершины $\{v_1, v_4, v_5, v_6\}$, вершина u_2 — сильно связной компоненте $\{v_2, v_3, v_8\}$, вершина u_3 — сильно связной компоненте, состоящей из одной вершины v_7 .

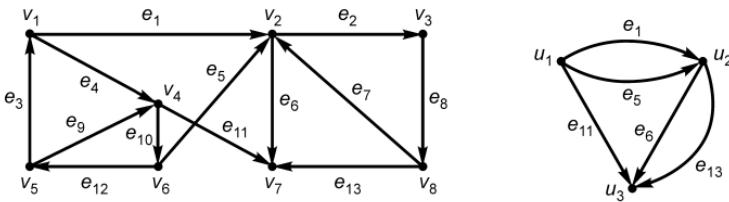


Рис. 4.15

Отметим, что в некоторых книгах граф конденсации определяется несколько по-другому: если в G есть ребра, ведущие из вершин K_i в вершины K_j , то из u_i в u_j проводится только одно ребро.

Ациклические графы. Орграф называется *ациклическим*, если он не содержит циклов.

В общем случае орграф может содержать пути сколь угодно большой длины, поскольку каждый путь может проходить через одну вершину любое число раз. Однако это возможно, только если в графе есть циклы. В ациклическом графе длины путей ограничены, так как вершины в его путях не могут повторяться, т. е. все его пути — простые цепи. Следовательно, в ациклическом орграфе имеются пути максимальной длины, т. е. пути, которые не могут быть продолжены: нельзя добавить ребро ни к их началу, ни к их концу. Отсюда следует,

что в ациклическом графе существует по крайней мере одна вершина, в которую не входит ни одно ребро (такие вершины иногда называют *источниками*), и по крайней мере одна вершина, из которой не выходит ни одно ребро (такие вершины иногда называют *стоками*). Действительно, в ациклическом графе любой путь приводит в вершину, из которой он не может продолжаться. Такая вершина и есть сток. Если же продолжить этот путь от его начала против ориентации, придем в вершину, из которой нельзя выйти против ориентации инцидентных ей ребер, т. е. в вершину, не имеющую входящих ребер. Такая вершина является источником.

Ациклический орграф с одним источником и одним стоком называется *двухполюсным*.

Граф конденсации является ациклическим графом. Действительно, если в графе конденсации G' есть цикл, то в исходном графе G есть цикл, проходящий через несколько сильно связных компонент. Но это невозможно, так как тогда эти компоненты можно было бы объединить в одну.

Топологической сортировкой орграфа называется такая нумерация вершин, что для любого ребра (v_i, v_j) номер его начала меньше номера его конца: $i < j$.

Теорема 4.5. Для орграфа топологическая сортировка существует тогда и только тогда, когда он — ациклический.

Предположим, что для цикла топологическая сортировка возможна. Выберем в цикле произвольную вершину v_i . Цикл содержит ребра (v_i, v_j) и (v_k, v_i) , причем по определению топологической сортировки должно быть $i < j$ и $k < i$. При прохождении вдоль цикла номера вершин должны возрастать и, значит, все они будут больше i . Поэтому, когда мы, двигаясь по циклу, придем в v_k , получим $k > i$, что противоречит предположению.

Для ациклического орграфа G с n вершинами топологическая сортировка осуществляется с помощью следующего алгоритма. Выберем в G какой-либо сток и присвоим ему номер n . Все инцидентные ему ребра — входящие, поэтому их начала будут иметь номера, меньшие n , и, следовательно, для них условие топологической сортировки выполнено. Удалим выбранный сток вместе со

всеми инцидентными ему ребрами. Получим ациклический граф с $n - 1$ вершиной. Выберем в нем какой-либо сток и присвоим ему номер $n - 1$. Будем повторять процедуру удаления стоков и инцидентных им ребер до тех пор, пока не пронумеруем все вершины. Поскольку всякий раз удаляемые ребра будут удовлетворять условию топологической сортировки, получим топологическую сортировку исходного орграфа.

Пример 4.11. На рис. 4.16 приведен пример топологической сортировки.

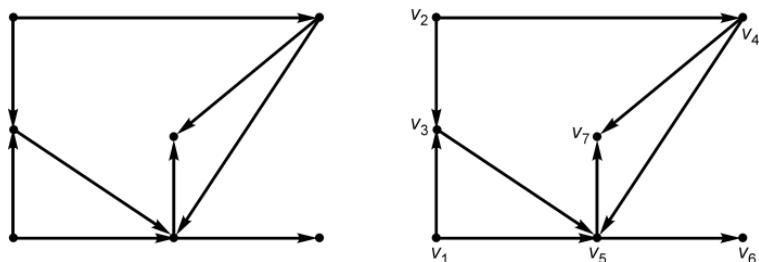


Рис. 4.16.

Матрицы орграфов и их связь с путями. Матрицу смежности $C(G)$ орграфа G можно использовать для подсчета числа различных путей в G . Сама матрица C задает ребра G , т. е. пути длины 1. Оказывается, что матрица C^l (l -я степень C) задает число путей длины l .

Теорема 4.6. Элемент $(i, j) = c_{ij}^{(l)}$ матрицы C^l орграфа G равен числу путей длины l из v_i в v_j .

Напомним правило умножения квадратных матриц (случай неквадратных матриц нам не понадобится). Если даны две квадратные матрицы X и Y одинаковой размерности n , то элемент z_{ij} матрицы $Z = XY$ определяется следующей формулой:

$$z_{ij} = \sum_{k=1}^n x_{ik} y_{kj}. \quad (4.1)$$

Доказательство теоремы проведем индукцией по l . Для $l = 1$ теорема очевидна (см. сказанное ранее). Пусть для некоторого l теорема верна, т. е. элемент $c_{ij}^{(l)}$ матрицы C^l равен числу путей длины l из v_i в v_j . Докажем ее для $l + 1$. Любой путь длины $l + 1$ из v_i в v_j состоит из ребра, ведущего в некоторую вершину v_k , и пути длины l из v_k в v_j . Для произвольного k число путей длины $l + 1$ из v_i в v_j , проходящих на первом шаге через вершину v_k ,

равно $c_{ik}c_{kj}^{(l)}$ (если ребра из v_i в v_k нет, то $c_{ik} = 0$, и $c_{ik}c_{kj}^{(l)} = 0$). Общее число путей длины $l + 1$ из v_i в v_j получим, если просуммируем эту величину по всем k :

$$\sum_{k=1}^n c_{ik}c_{kj}^{(l)}.$$

Эта сумма, как видно из формулы (4.1), равна элементу (i, j) произведения матриц C и C^l , т. е. элементу (i, j) матрицы C^{l+1} , что и доказывает теорему. \square

Следствие 4.1. Элемент (i, j) матрицы $C + C^2 + \dots + C^l$ ографа G равен числу всех путей длины $\leq l$ из v_i в v_j . \square

Использование матриц позволяет получить и перечисление конкретных путей. Для этого всем ребрам графа G присвоим конкретные имена (например, e_1, \dots, e_m) и в матрице C заменим единицы именами соответствующих ребер, т. е. элемент $c_{ij} = 1$ заменим именем ребра, которое соединяет вершину v_i с вершиной v_j . Полученную матрицу обозначим через $H(G)$. Для того чтобы определить произведение матриц этого вида, введем алгебру на множествах путей.

Путь будем рассматривать как слово (последовательность символов) в алфавите $\{e_1, \dots, e_m\}$. Пусть даны два множества путей M_1 и M_2 . Сумма M_1 и M_2 определяется как их обычное теоретико-множественное объединение: $M_1 \cup M_2$, произведение M_1M_2 — как множество, получаемое приписыванием справа к каждому слову из M_1 всех слов из M_2 . Например, если $M_1 = \{e_2e_4e_2, e_3e_1, e_1\}$, $M_2 = \{e_3e_1e_4, e_2\}$, то $M_1M_2 = \{e_2e_4e_2e_3e_1e_4, e_2e_4e_2e_2, e_3e_1e_3e_1e_4, e_3e_1e_2, e_1e_3e_1e_4, e_1e_2\}$. (Такую операцию называют еще *конкатенацией*.) Пустое множество \emptyset играет здесь роль нуля: $M_1\emptyset = \emptyset M_2 = \emptyset$. Поэтому вместо \emptyset будем, как и в матрице C , писать 0. Очевидно, что эта операция некоммутативна. Она имеет простой графовый смысл: если M_1 — множество всех путей, ведущих из v_i в v_j , а M_2 — множество всех путей, ведущих из v_j в v_k , то M_1M_2 — это множество всех путей, ведущих из v_i в v_k и проходящих через v_j .

С помощью этих операций определим произведение $Z = XY$ квадратных матриц X и Y одинаковой размерности n , элементами которых являются множества слов (такие матрицы назовем словарными):

$$z_{ij} = \bigcup_{k=1}^n x_{ik}y_{kj}. \quad (4.2)$$

В этой формуле роль суммы элементов играет теоретико-множественное объединение, а произведения — определенная выше конкатенация. Степень матрицы H определяется тогда по индукции формулой $H^{l+1} = HH^l$.

Теорема 4.7. Элемент $(i, j) = h_{ij}^{(l)}$ матрицы H^l орграфа G представляет собой множество всех путей длины l из v_i в v_j .

Докажем эту теорему индукцией по l . Доказательство почти дословно совпадает с доказательством предыдущей теоремы. Для $l = 1$ теорема очевидна, поскольку любой ненулевой элемент h_{ij} матрицы H — это ребро (т. е. путь длины 1), ведущее из v_i в v_j . Пусть для некоторого l теорема верна, т. е. элемент $h_{ij}^{(l)}$ матрицы H^l равен числу путей длины l из v_i в v_j . Докажем ее для $l + 1$. Любой путь длины $l + 1$ из v_i в v_j состоит из ребра, ведущего в некоторую вершину v_k , и пути длины l из v_k в v_j . Для произвольного k множество путей длины $l + 1$ из v_i в v_j , проходящих на первом шаге через вершину v_k , равно конкатенации $h_{ik}h_{kj}^{(l)}$ (если ребра из v_i в v_k нет, то $h_{ik} = 0$, и $h_{ik}h_{kj}^{(l)} = \emptyset$). Все множество путей длины $l + 1$ из v_i в v_j получим, если объединим множества $h_{ik}h_{kj}^{(l)}$ по всем k :

$$\bigcup_{k=1}^n h_{ik}h_{kj}^{(l)}.$$

Эта сумма, как видно из формулы (4.2), равна элементу (i, j) произведения матриц H и H^l , т. е. элементу (i, j) матрицы H^{l+1} , что и доказывает теорему. \square

Следствие 4.2. Элемент (i, j) матрицы $H + H^2 + \dots + H^l$ орграфа G равен множеству всех путей длины $\leq l$ из v_i в v_j .

Матрицей достижимости орграфа G с n вершинами будем называть квадратную матрицу $T(G)$ размерности $n \times n$, в которой элемент $r_{ij} = 1$, если v_j достижима из v_i , и $r_{ij} = 0$, если v_j недостижима из v_i . Полагаем, что $r_{ii} = 1$ для всех i .

Преобразование произвольной матрицы A , заключающееся в замене на 1 всех ненулевых элементов A , обозначим через $B(A)$.

Теорема 4.8. Для орграфа G с n вершинами $T(G) = B(I + C + C^2 + \dots + C^{n-1})$, где I — единичная матрица.

По теореме 4.1', если v_j достижима из v_i , то существует простая цепь из v_i в v_j . Длина этого пути не превосходит

$n - 1$, поскольку в простой цепи вершины не повторяются. Согласно следствию из теоремы 4.6, в этом случае элемент (i, j) матрицы $I + C + C^2 + \dots + C^{n-1}$ будет ненулевым, откуда и следует наша теорема. \square

В § 4.1 говорилось о том, что каждому простому орграфу G соответствует бинарное отношение \hat{R}_G , матрица которого совпадает с матрицей смежности графа G . Матрица $T(G)$ — это матрица транзитивного замыкания R_G отношения R_G . Действительно, по определению транзитивного замыкания $a\hat{R}_G b$, если в R_G есть цепочка $aR_G a_2, a_2R_G a_3, \dots, a_{n-1}R_G b$, т. е. если b достижима из a в графе G , что и описывается матрицей $T(G)$.

4.4. ДЕРЕВЬЯ

Основные свойства деревьев. Если в ациклическом орграфе «отменить» ориентацию ребер, то в полученном неориентированном графе могут возникнуть циклы. Поэтому неориентированный ациклический граф имеет более специфический вид.

Связный неориентированный граф без циклов называется *неориентированным деревом*. Несвязный неориентированный граф без циклов называется *лесом*; его компоненты связности — деревья.

Очевидно, что в дереве, как и в ациклическом орграфе, все пути — простые.

Пример 4.12. На рис. 4.17 T_1, T_2, T_3 — неориентированные деревья. Следующая теорема описывает основные свойства деревьев.

Теорема 4.9.

1. В неориентированном дереве имеются по крайней мере две концевые вершины (вершины степени 1).

2. Между любыми двумя вершинами дерева имеется ровно один путь.

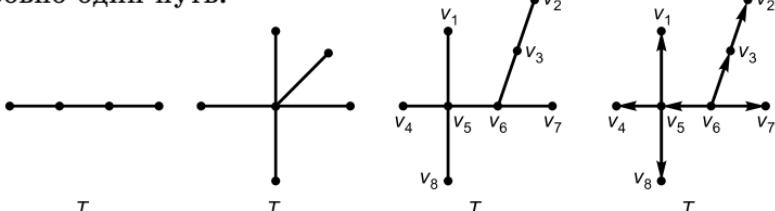


Рис. 4.17

3. Дерево с n вершинами имеет $n - 1$ ребер.

Поскольку в дереве все пути — простые, то в нем есть по крайней мере один максимальный путь, т. е. путь, который нельзя продолжить. Концы этого пути и являются концевыми вершинами.

(Заметим, что в произвольном ациклическом орграфе тоже есть максимальные пути; их концами являются стоки. Однако сток может иметь степень больше единицы. В этом случае продолжить из него путь нельзя не потому, что нет другого инцидентного ему ребра, а потому, что другие ребра — тоже входящие, и выйти по ним из стока нельзя.)

Наличие пути между любой парой вершин следует из связности дерева, а единственность этого пути — из того, что существование двух путей между парой вершин всегда создает цикл.

Третий пункт теоремы проще всего доказать, введя понятие *ориентированного дерева*. Из предшествующего видно, что это понятие нельзя получить, просто заменив в определении дерева «неориентированный граф» на «ориентированный». Ориентированное дерево получается из неориентированного дерева следующей процедурой его ориентации. Выберем в дереве произвольную вершину. Назовем ее *корнем*. Ребра, инцидентные корню, ориентируем в направлении от корня. Для каждой вершины v_i , являющейся концом одного из этих ребер, ориентируем остальные инцидентные ей ребра в направлении от v_i . Продолжаем эту процедуру до тех пор, пока не будут достигнуты концевые вершины. В силу единственности пути между вершинами ни одна вершина не будет достигнута дважды (т. е. не придется ориентировать уже ориентированные ребра), а в силу связности дерева все вершины будут достигнуты. Из этой процедуры видно, что корень не имеет входящих ребер, а все остальные вершины имеют по одному входящему ребру. Поскольку все ребра являются входящими для какой-то вершины и в процессе ориентации число вершин и ребер не изменилось, то отсюда и следует п. 3 теоремы. \square

Оrientированное дерево T_4 на рис. 4.17 получено ориентацией T_3 и имеет корень v_6 . Для ориентированных деревьев свойства 1 и 3 верны всегда, а свойство 2 — нет. Нетрудно видеть, что ориентированное дерево 1) всегда слабо связано; 2) односторонне связано, только если оно

является ориентированной цепью (т. е. когда все вершины лежат на единственном пути из корня); 3) никогда не бывает сильно связным, поскольку сильно связный граф обязательно содержит цикл.

В каких случаях ациклический орграф является ориентированным деревом? Ответ на этот вопрос дает определение ориентированного дерева, не связанное с процедурой ориентации.

Ориентированным деревом называется связный ациклический орграф, в котором только одна вершина, называемая *корнем*, не имеет входящих ребер, а все остальные вершины имеют по одному входящему ребру.

Из одного неориентированного дерева с n вершинами можно получить ровно n различных ориентированных деревьев, так как выбор разных корней всегда дает разные ордеревья. Это следует из того, что из корня достижимы все остальные вершины, сам же корень недостижим ни из какой другой вершины. Однако различные ордеревья, полученные из одного и того же дерева, могут оказаться изоморфными.

Пример 4.13. Если исходное дерево — простая цепь, то выбор в качестве корня концов этой цепи даст две изоморфные ориентированные цепи, а если эта цепь содержит четное число вершин и, следовательно, два центра, то выбор этих центров даст два изоморфных ордерева с двумя путями, ведущими из корня (рис. 4.18).



Рис. 4.18

Центры деревьев. Вернемся теперь к неориентированным деревьям. В § 4.2 было дано определение центра для произвольного неориентированного связного графа. Для неориентированных деревьев центры можно найти с помощью следующего алгоритма.

1. Обозначим исходное дерево G через $G^{(1)}$ и присвоим его концевым вершинам тип 1.

2. Если в полученном на предыдущем шаге графе $G^{(i)}(i = 1, 2, \dots)$ все вершины — концевые, алгоритм заканчивается. В остальных случаях переходим к шагу 3.

3. Удалим в $G^{(i)}$ все концевые вершины вместе с инцидентными им ребрами. Полученный граф обозначим

через $G^{(i+1)}$. Этот граф является деревом, поскольку удаленные ребра не входят ни в какой путь, соединяющий оставшиеся вершины, и, следовательно, их удаление не нарушает связности $G^{(i+1)}$. Концевым вершинам $G^{(i+1)}$ присвоим тип $i+1$. Полагаем $i = i+1$ и переходим к шагу 2.

В результате работы алгоритма каждой вершине исходного дерева присваивается тип. На каждом шаге номер типа увеличивается на единицу, и вершины, оставшиеся к последнему шагу, получают максимальный тип k .

Теорема 4.10. Центрами дерева являются вершины максимального типа k . Дерево имеет либо один, либо два центра, причем в первом случае радиус дерева равен $k - 1$, а во втором — k .

По условию п. 2 алгоритма вершины максимального типа являются концевыми в заключительном дереве $G^{(k)}$, причем неконцевых вершин в $G^{(k)}$ нет. Любое дерево, содержащее более 2 вершин, обязательно содержит неконцевые вершины.

Это интуитивно ясное утверждение можно доказать простой арифметической выкладкой. Сумма степеней вершин непустого графа равна удвоенному числу ребер. Для дерева с n вершинами эта сумма равна $2n - 2$. Если же все вершины дерева — концевые, т. е. имеющие степень 1, то эта сумма в то же время равна n . Но равенство $2n - 2 = n$ верно только при $n = 2$.

Таким образом, возможны только два случая, когда дерево не имеет неконцевых вершин: либо оно пустое и состоит из одной вершины, либо это граф, содержащий две вершины, соединенные ребром. Следовательно, вершин в $G^{(k)}$, т. е. вершин максимального типа, может быть не более двух.

Покажем теперь, что именно эти вершины являются центрами. Отметим сначала, что для любой вершины дерева максимальным удалением может быть только длина пути от нее до некоторой концевой вершины, поскольку путь до неконцевой вершины можно продолжить. Так как на первом шаге алгоритма все концевые вершины удаляются, то все максимальные удаления в полученном графе $G^{(2)}$ уменьшаются на 1. Поэтому $r(G^{(2)}) = r(G) - 1$ и, кроме того, соотношение $r(v) = \min_{v_i \in G} r(v_i)$ сохраняется для

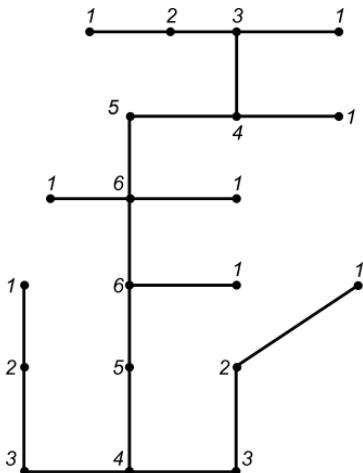


Рис. 4.19

всех вершин, для которых оно выполнялось в G . Поэтому центры в $G^{(2)}$ — те же, что и в G . По индукции заключаем, что $r(G^{(i)}) = r(G) - (i - 1)$ для $i = 1, \dots, k$, причем центры во всех $G^{(i)}$ — одни и те же. Следовательно,

$$r(G^{(k)}) = r(G) - (k - 1), \quad (4.3)$$

и вершины, оставшиеся в $G^{(k)}$, являются центрами не только для $G^{(k)}$, но и для исходного графа G . Если $G^{(k)}$ содержит одну вершину, то $r(G^{(k)}) = 0$, и из (4.3) получаем $r(G) = k - 1$;

а радиусом будет такой путь от центра до концевой вершины, в котором типы соседних вершин отличаются ровно на единицу. Если же $G^{(k)}$ содержит две вершины, то $r(G^{(k)}) = 1$, и из (4.3) получаем $r(G) = k$. Здесь радиус длиннее, чем в предыдущем случае, на одно ребро, соединяющего два центра. \square

Пример 4.14. На рис. 4.19 показаны типы, присвоенные вершинам дерева в соответствии с описанным алгоритмом. Вершины типа 6 — центры.

4.5. ПРОСТРАНСТВО ЦИКЛОВ

Покрывающие деревья (остовы). Цикломатическим числом неориентированного графа G называется величина $\gamma(G) = m - n + k$, где m — число ребер, n — число вершин, k — число связных компонент. Для дерева и леса $\gamma(G) = 0$, для других графов $\gamma(G) > 0$.

Остовом, или *покрывающим деревом*, связного графа $G(V, E)$ называется часть G , которая содержит все его вершины и является деревом. *Хордой* остова графа G называется ребро G , не принадлежащее остову.

Поскольку для связного графа G $k = 1$ и $\gamma(G) = m - n + 1$, то число ребер G равно $m = \gamma(G) + n - 1$. Остов G , как и любое дерево, содержит $n - 1$ ребер. Поэтому для получения остова T в G надо удалить $\gamma(G)$ ребер, которые

и станут хордами T . Следовательно, при любом выборе остова число хорд равно цикломатическому числу $\gamma(G)$.

Очевидно, что любой связный граф имеет хотя бы один остов, а любой несвязный граф остова не имеет.

В последующем алгоритме части исходного графа G , которые возникают в процессе построения покрывающего дерева, будем называть *букетами*.

Алгоритм построения покрывающего дерева для произвольного невзвешенного графа G .

1. Выбрать любое ребро G , не являющееся петлей. Пометить его меткой α и объявить букетом это ребро вместе с его концевыми вершинами.

2. Выбрать любое непомеченное ребро G , не являющееся петлей:

а) если один из концов выбранного ребра принадлежит построенному ранее букету B , а другой конец свободен (не принадлежит ни одному букету), пометить выбранное ребро меткой α , включить его вместе со свободным концом в букет B и перейти к шагу 3;

б) если оба конца выбранного ребра свободны, пометить его меткой α , объявить это ребро вместе с его концевыми вершинами новым букетом и перейти к шагу 3;

в) если концы выбранного ребра принадлежат разным построенным ранее букетам B и C , пометить выбранное ребро меткой α , включить его в букет C и перейти к шагу 3;

г) если оба конца выбранного ребра принадлежат одному букету, пометить его меткой β и перейти к шагу 3;

д) если непомеченных ребер нет, закончить алгоритм.

3. Если все вершины графа G вошли в один букет, закончить алгоритм. Если нет, перейти к шагу 2.

Заметим, что все метки, присвоенные ребрам, в дальнейшем не меняются, а букеты содержат только ребра с пометкой α . Кроме того, на любом шаге алгоритма все построенные букеты образуют лес. Это доказывается индукцией по числу шагов. Для начального шага 1 утверждение очевидно. Предположим, что букеты, построенные на предыдущих шагах, — деревья. Если на данном шаге строится новый букет (пп. 1; 2, б), то он содержит лишь одно ребро, не связанное с построенными букетами, и, следовательно, число деревьев в построенном лесе

увеличивается на 1. Если на данном шаге ребро включается в уже построенный букет (п. 2, а), то полученный граф связан и не создает новых путей в прежнем букете. Поэтому он является деревом, а общее число деревьев в построенном лесе сохраняется. В случае 2, в выбранное ребро соединяет два ранее не связанных дерева в одно, и число деревьев в построенном лесе уменьшается на 1. Наконец, в случае 2, г присоединение ребра к букету образовало бы цикл; поэтому оно в букет не включается.

Таким образом, результатом работы алгоритма является покрывающий лес, т. е. лес, содержащий все вершины исходного графа. Если этот лес состоит из одного букета, то этот букет и является покрывающим деревом. В противном случае граф несвязан. Шаг 3 нужен для того, чтобы не просматривать оставшиеся ребра, когда покрывающее дерево уже построено. Без него можно обойтись, но тогда алгоритм остановится, только если просмотрены все ребра.

Пространство циклов. Пусть $E = \{e_1, \dots, e_m\}$ — множество всех поименованных ребер графа G . Всякий двоичный вектор (кортеж) $w = (w^1, \dots, w^m)$ длины m определяет подмножество E_w ребер графа следующим образом: $e_i \in E_w$, если и только если $w^i = 1$. В свою очередь, E_w определяет реберно-порожденный граф $G(E_w)$, который будем обозначать G_w .

Лемма 4.1. Если $w_3 = w_1 \oplus w_2$, где \oplus — поразрядное сложение по mod 2, то $G_{w_3} = G_{w_1} \Delta G_{w_2}$, где Δ — симметрическая разность графов (см. § 4.1).

Действительно, так как $w^i_3 = 0$, если $w^i_1 = w^i_2$, $w^i_3 = 1$, если $w^i_1 \neq w^i_2$, то E_{w_3} содержит только те ребра из $E_{w_1} \cup E_{w_2}$, которые не являются общими для E_{w_1} и E_{w_2} , т. е. $E_{w_3} = (E_{w_1} \cup E_{w_2}) \setminus (E_{w_1} \cap E_{w_2}) = E_{w_1} \Delta E_{w_2}$. Но граф $G_{w_1} \Delta G_{w_2}$ порожден множеством ребер $E_{w_1} \Delta E_{w_2}$, т. е. совпадает с E_{w_3} . \square

Вектор w длины m называется *циклическим вектором*, если либо w состоит из одних нулей, либо G_w представляет собой один простой цикл или несколько циклов, не пересекающихся по ребрам.

Множество всех циклических векторов графа G обозначим через W_G , а множество всех графов G_w , порожденных циклическими векторами, — через C_G .

Лемма 4.2. Если w_1 и w_2 — циклические векторы, то $w_3 = w_1 \oplus w_2$ — циклический вектор. Иными словами, множество W_G замкнуто относительно операции \oplus .

По теореме 4.4 связный граф является эйлеровым тогда и только тогда, когда он связан и является объединением нескольких циклов, не пересекающихся по ребрам. Граф G_w , соответствующий циклическому вектору w , не обязательно связан, но его компоненты связности — эйлеровы графы. Следовательно, степени всех вершин G_w четны. Поэтому доказательство замкнутости W_G сводится к доказательству того, что, если $w_3 = w_1 \oplus w_2$, то в графе $G_{w3} = G_{w1} \Delta G_{w2}$ четность степеней вершин сохраняется.

Рассмотрим произвольную вершину v графа G_{w3} . Она содержится по крайней мере в одном из графов G_{w1} , G_{w2} . Пусть M_i — множества инцидентных ей ребер в G_i и, соответственно, $|M_i| = k_i$ — ее степень в G_i , $i = 1, 2, 3$. Числа k_1, k_2 — четные. Так как $G_{w3} = G_{w1} \Delta G_{w2}$, то $M_3 = M_1 \Delta M_2$. Поэтому $k_3 = k_1 + k_2 - 2(|M_1 \cap M_2|)$. Следовательно, k_3 — четное число.

Из лемм 4.1 и 4.2 следует, что множество W_G с операцией \oplus и множество C_G с операцией Δ являются алгебрами, а соответствие $w \Rightarrow G_w$ — изоморфизмом. Поэтому в дальнейшем мы не будем различать эти алгебры.

Множество S называется *векторным (линейным) пространством* над полем F , если:

- 1) S — абелева группа (см. § 2.2) с операцией сложения $+$; следовательно, она содержит нулевой элемент;
- 2) для любой пары (s, α) , где $s \in S$, $\alpha \in F$, определена операция умножения: $\alpha s = s'$, где $s' \in S$;
- 3) определенное в п. 2 умножение дистрибутивно относительно сложения:

$$\alpha(s_1 + s_2) = \alpha s_1 + \alpha s_2, (\alpha_1 + \alpha_2)s = \alpha_1 s + \alpha_2 s.$$

Элементы векторного пространства называются *векторами*^{*}.

* Начиная с этого момента, термин «вектор» употребляется в двух несовпадающих смыслах: вектор, определенный в гл. 1, как упорядоченный набор элементов (кортеж), и вектор как элемент векторного пространства. Циклические векторы являются векторами и в первом, и во втором (как будет показано ниже) смысле. Однако терминологическая коллизия налицо. О ее объективной природе, сложившейся исторически, и различиях этих двух смыслов будет сказано ниже в Отступлении 4.1.

Сумма вида $\alpha_1 s_1 + \alpha_2 s_2 + \dots + \alpha_k s_k$ называется *линейной комбинацией* векторов s_1, s_2, \dots, s_k . Из пп. 1 и 2 следует, что любая линейная комбинация векторов пространства S является вектором S .

Множество Q векторов s_1, s_2, \dots, s_k называется линейно-независимым, если ни один из элементов этого множества не является *линейной комбинацией* других векторов из Q .

Базисом векторного пространства S называется линейно-независимое множество R , такое, что любой вектор S может быть представлен линейной комбинацией векторов из R . В теории векторных пространств доказывается, что все базисы данного пространства имеют одинаковую мощность. Эта мощность называется *размерностью векторного пространства*.

Пусть дан граф G . Зафиксируем некоторый его остов T . Простой цикл C_e , порожденный добавлением хорды e в T , называется *базисным циклом*. Циклический вектор w_e , представляющий этот цикл, также называется базисным. Множество всех базисных циклических векторов обозначим через $Z(T)$. Его мощность равна числу хорд, т. е. цикломатическому числу $\gamma(G)$.

Теорема 4.11. Множество W_G всех циклических векторов графа G является векторным пространством над полем Галуа $GF(2)$. Операция сложения двух векторов в W_G определяется как их поразрядное сложение по модулю 2, а операция умножения αw , где $w \in W_G$, $\alpha \in GF(2)$, — как поразрядное умножение числа α на компоненты вектора w . Базисом этого пространства, называемого *пространством циклов* графа G , является множество $Z(T)$, а размерность пространства равна цикломатическому числу $\gamma(G)$.

Напомним, что поле Галуа $GF(2)$ — это частный случай поля Галуа $GF(p)$, определенного в главе 2. Оно представляет собой множество $\{0, 1\}$, на котором определены операции сложения по модулю 2 и умножения:

$$0 \oplus 0 = 0, 0 \oplus 1 = 1 \oplus 0 = 1, 1 \oplus 1 = 0;$$

$$0 \cdot 0 = 0, 0 \cdot 1 = 1 \cdot 0 = 0, 1 \cdot 1 = 1.$$

Докажем сначала, что W_G удовлетворяет определению векторного пространства. Нужные свойства опера-

ций очевидны. Поскольку коэффициенты линейных комбинаций в W_C — это нули и единицы, то любые линейные комбинации в W_C — это суммы некоторых циклических векторов. Из леммы 4.2 следует, что такие суммы — также циклические векторы.

Докажем теперь, что $Z(T)$ является базисом пространства W_G . Линейная независимость $Z(T)$ следует из того, что вектор w_{ei} , порожденный хордой e_i , нельзя представить суммой других элементов $Z(T)$, так как в других циклических векторах $Z(T)$ на i -м месте стоят нули. Покажем, что любой циклический вектор w является суммой некоторых векторов из $Z(T)$, а именно: если G_w содержит хорды e_{i1}, \dots, e_{ir} , то $G_w = C_{ei1} \Delta \dots \Delta C_{eir}$ и, следовательно, $w = w_{ei1} \oplus \dots \oplus w_{eir}$.

Рассмотрим сумму базисных циклов $Q = C_{ei1} \Delta \dots \Delta C_{eir}$ и покажем, что $G_w \Delta Q$ — пустое множество, откуда будет следовать, что $G_w = Q$. Пусть наше предположение неверно и $G_w \Delta Q$ — непустой граф G^* . Так как хорды в G_w и Q совпадают, то G^* не содержит хорд и, следовательно, не содержит циклов. С другой стороны, по теореме 4.4 он является объединением реберно-непересекающихся циклов. Получаем противоречие, откуда следует, что предположение о непустоте $G_w \Delta Q$ неверно и $G_w = Q$, а $w = w_{ei1} \oplus \dots \oplus w_{eir}$. \square

Пример 4.15. Для графа G на рис. 4.20 $n = 7$, $m = 9$, $\gamma(G) = 3$. Три хорды b , d , m выделены пунктиром. Базис $Z(T)$ содержит три циклических вектора w_b , w_d , w_m (табл. 4.2).

Остальные векторы W_G — это их линейные комбинации (см. табл. 4.3).

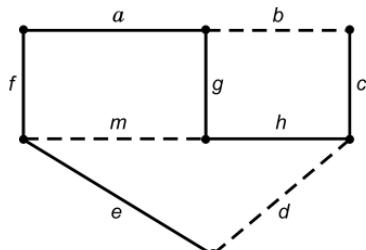


Рис. 4.20

Таблица 4.2

	a	b	c	d	e	f	g	h	m	
$w_b =$	0	1	1	0	0	0	1	1	0	(цикл $bchg$)
$w_d =$	1	0	0	1	1	1	1	1	0	(цикл $afedhg$)
$w_m =$	1	0	0	0	0	1	1	0	1	(цикл $afmg$)

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>m</i>
$w_b \oplus w_d =$	1	1	1	1	1	1	0	0	0
$w_b \oplus w_m =$	1	1	1	0	0	1	0	1	1
$w_d \oplus w_m =$	0	0	0	1	1	0	0	1	1
$w_b \oplus w_d \oplus w_m =$	0	1	1	1	1	0	1	0	1

Отступление 4.1. Терминологическая коллизия, т. е. употребление одного термина в двух несовпадающих смыслах, о чем было сказано выше в примечании, касается не только вектора, но и его размерности. Размерность вектора как упорядоченного набора элементов — это его длина, т. е. число его компонент. В нашем примере она равна 9. Размерность вектора как элемента векторного пространства совпадает с размерностью самого пространства, т. е. с мощностью базиса. В примере она равна 3. Исторически сначала возникло второе понимание вектора. Всякий элемент векторного пространства можно представить линейной комбинацией $\alpha_1 s_1 + \alpha_2 s_2 + \dots + \alpha_k s_k$ векторов базиса. Такое представление удобно стандартизовать, т. е. считать, что, во-первых, порядок элементов базиса зафиксирован, и, во-вторых, эта комбинация всегда содержит все элементы базиса, т. е. полагать, что k — размерность пространства, а некоторые коэффициенты α_i , быть может, равны 0. Тогда достаточно каждый вектор задавать упорядоченным набором коэффициентов — элементов поля F , над которым определено пространство. В нашем примере при алфавитном упорядочении элементов базиса: w_b , w_d , w_m базисному элементу w_d соответствует двоичный набор 010, а линейной комбинации $w_b \oplus w_m$ — двоичный набор 101. Так возникло представление о векторе как упорядоченном наборе, которое в дальнейшем приобрело самостоятельный смысл и стало жить в математике своей жизнью, далеко не всегда имеющей отношение к векторным пространствам. В этом разделе векторы появились сначала просто как двоичные наборы, описывающие реберно-порожденные графы и не имеющие отношения к векторным пространствам. Поэтому их размерность как число компонент не связана с их размерностью в векторном пространстве, которое мы построили позже.

Можно было бы избежать этой коллизии, назвав упорядоченные двоичные наборы не векторами, а, например, кортежами. Но в литературе по графикам понятие «циклический вектор» укоренилось, и мы его сохраним.

4.6. ДВУДОЛЬНЫЕ И ПЛАНАРНЫЕ ГРАФЫ

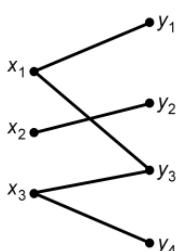
Двудольные и знаковые графы. Двудольным графом $G(X, Y, E)$ называется неориентированный граф, вершины которого можно разбить на два класса X и Y так, что концы каждого ребра принадлежат разным классам. Двудольный граф называется *полным*, если каждая вершина одной доли соединена с каждой вершиной другой доли. Полный двудольный граф принято обозначать символом $K_{m, n}$, где $m = |X|$, $n = |Y|$.

Введенные понятия допускают естественное обобщение. Неориентированный граф называется *k-дольным*, если его вершины можно разбить на k классов так, что концы каждого ребра принадлежат разным классам. Полный k -дольный граф — это k -дольный граф, в котором каждая вершина одной доли соединена с каждой вершиной всех остальных долей.

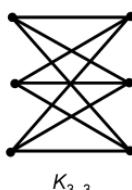
Пример 4.16. На рис. 4.21 представлены двудольный граф $G(X, Y, E)$, полный двудольный граф $K_{3, 3}$ и трехдольный граф $G(X, Y, Z, E)$, где $X = \{x_1, x_2\}$, $Y = \{y_1, y_2, y_3\}$, $Z = \{z_1, z_2\}$.

Теорема 4.12. Граф является двудольным, если и только если длины всех его простых циклов четны.

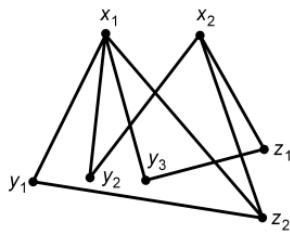
Рассмотрим двудольный граф $G(X, Y, E)$. Поскольку вершины одного класса не смежны, то в любом цикле соседние вершины принадлежат разным классам. Поэтому, если проход по циклу начать с вершины v класса X , то каждое нечетное по счету ребро цикла выходит из X , а каждое четное возвращается в X , и, следовательно, ребро, приходящее в начальную вершину v цикла, может быть только четным, т. е. цикл имеет четную длину.



$G(X, Y, E)$



$K_{3, 3}$



$G(X, Y, Z, E)$

Пусть теперь все простые циклы графа G четны. Без нарушения общности можно считать, что граф связен. Отнесем в класс X произвольную вершину v и все вершины, находящиеся на четном расстоянии от v , а в класс Y — все остальные вершины. Предположим, что вершины w_1 и w_2 класса X соединены ребром (w_1, w_2) . Так как из v в w_1 и w_2 существуют пути четной длины $P(v, w_1), P(v, w_2)$, соответственно, то путь $P(w_1, v)P(v, w_2)(w_2, w_1)$ образует цикл нечетной длины, что противоречит предположению. Предположим теперь, что вершины w'_1 и w'_2 класса Y соединены ребром (w'_1, w'_2) . Поскольку из v в w'_1 и w'_2 существуют пути нечетной длины $P(v, w'_1), P(v, w'_2)$, соответственно, то путь $P(w'_1, v)P(v, w'_2)(w'_2, w'_1)$ образует цикл нечетной длины, что также противоречит предположению. Таким образом, вершины из одного класса не связаны ребрами, и, следовательно, граф G — двудольный. \square

Паросочетанием в неориентированном графе называется множество попарно несмежных ребер. *Задача о максимальном паросочетании* заключается в нахождении паросочетания максимальной мощности. Для двудольного графа одной из наиболее известных интерпретаций задачи о максимальном паросочетании является *задача о назначениях*, которая заключается в следующем. Имеется m работников и n работ. Каждый работник способен выполнять несколько работ; каждую работу могут выполнять несколько работников. Требуется определить назначения по принципу «один работник — одна работа» так, чтобы загрузить максимальное число работников.

Условия этой задачи представляются в виде двудольного графа, в котором вершины класса X соответствуют работникам, вершины класса Y — работам, а наличие ребра (x_i, y_j) означает, что i -й работник может выполнять j -ю работу. Решением этой задачи будет максимальное паросочетание. Метод его нахождения путем сведения к задаче о потоках изложен в § 4.8.

Неориентированный граф называется *знакомым*, если каждому его ребру приписан вес 1 либо -1 . Для краткости эти веса обозначаются знаками «+» и «-». *Знак пути* знакового графа определяется как произведение весов

входящих в него ребер. Знаковый граф называется *сбалансированным*, если все его циклы положительны.

Теорема 4.13 (Харари). Для знакового графа следующие утверждения эквивалентны:

1) граф сбалансирован;

2) для любой пары вершин любые две цепи, их соединяющие, имеют одинаковый знак;

3) вершины графа можно разбить на два класса так, что все ребра, соединяющие вершины одного класса, положительны, а все ребра, соединяющие вершины разных классов, отрицательны.

Пусть вершины v_i и v_j соединены цепями $P_1(v_i, v_j)$ и $P_2(v_i, v_j)$. Тогда цепь $P_1(v_i, v_j) P_2(v_j, v_i)$ образует цикл. Если цепи P_1 и P_2 имеют одинаковый знак, то цикл $P_1(v_i, v_j) P_2(v_j, v_i)$ положителен; если их знаки различны, то этот цикл отрицателен. Поэтому первое и второе утверждения эквивалентны.

Предположим, что вершины графа G разбиты на два класса A и B так, что выполнено третье условие. Рассмотрим произвольный цикл графа G . Если все вершины цикла принадлежат одному классу, то по третьему условию все ребра цикла положительны. Пусть вершины цикла принадлежат разным классам. Если проход по циклу начать с вершины v класса A , то каждое нечетное по счету отрицательное ребро цикла выходит из A в B , а каждое четное отрицательное ребро возвращается в A . Поэтому цикл содержит четное число отрицательных ребер. Следовательно, он положителен, и из третьего утверждения следует первое.

Покажем теперь, что из первого утверждения следует третье, т. е. вершины сбалансированного графа всегда можно разбить на два класса так, чтобы третье условие выполнялось. Это разбиение осуществляется следующим алгоритмом. Выбираем некоторую вершину v и помещаем ее в класс A . Для любого i ($i < n$) проверяем все цепи длины i , ведущие из v в другие вершины. Вершина v помещается в A , если цепь положительна, и в класс B , если цепь отрицательна. В силу второго условия все цепи, соединяющие любую пару вершин, имеют одинаковый знак. Поэтому любая вершина попадет ровно в один класс. Это разбиение удовлетворяет третьему условию.

Действительно, рассмотрим цепь из вершины v . Пока она идет по положительным ребрам, ее вершины находятся в классе A . Первое отрицательное ребро делает эту цепь отрицательной; следовательно, это ребро соединяет вершину класса A с вершиной класса B . Далее, пока цепь идет по положительным ребрам, она остается отрицательной; следовательно, она идет по вершинам класса B . Следующее отрицательное ребро делает цепь положительной и возвращает ее в класс A . Поэтому это ребро также соединяет вершину класса A с вершиной класса B . Продолжая это рассуждение до конца цепи, получим, что все отрицательные ребра цепи соединяют вершины разных классов и, следовательно, третье условие выполнено. \square

Описанный алгоритм разбиения вершин на классы можно использовать для определения сбалансированности графа G . Если в ходе алгоритма некоторая вершина v окажется помещенной в оба класса, то граф не сбалансирован, так как нарушено третье условие. Если же таких вершин не останется, то граф сбалансирован.

Теория знаковых графов и ее приложения подробно изложены в книге Ф. С. Робертса [26].

Планарные графы. Граф *укладывается* на поверхности S , если его можно изобразить на этой поверхности так, что никакие два ребра не будут пересекаться.

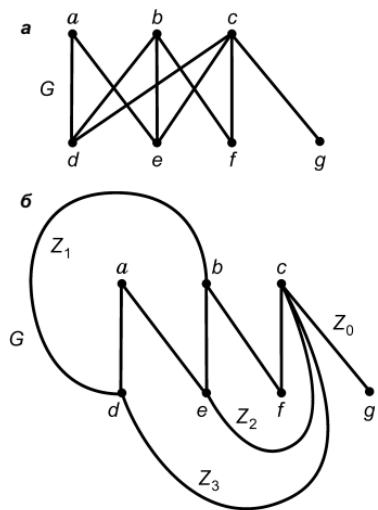


Рис. 4.22

Связный неориентированный граф называется *планарным*, если его можно уложить на плоскость, и *плоским*, если он уже уложен на плоскости. Область плоскости, ограниченная простым циклом и не содержащая никакой другой цикл, называется *внутренней гранью*; единственная внешняя по отношению к графу область называется *внешней гранью*. Плоский граф со всеми его гранями называется *плоской картой*.

Плоская карта отличается от плоского графа тем, что она включает в себя внешнюю грань, т. е. образует полное разбиение плоскости; плоский же график — это лишь часть плоскости.

На рис. 4.22, *a* показан планарный график, изображенный так, что его ребра пересекаются, а на рис. 4.22, *б* тот же график без пересечений ребер, т. е. плоский график. Z_1, Z_2, Z_3 — внутренние грани графа, Z_0 — внешняя грань.

Не всякий график является планарным.

Стандартным примером непланарного графа является график $K_{3,3}$ (см. рис. 4.21), возникающий в известной задаче «о трех домах и трех колодцах», в которой жители домов хотели бы ходить за водой к любому колодцу так, чтобы их тропы не пересекались, т. е. чтобы никогда не встречать своих соседей. Для этого график, соединяющий дома и колодцы, должен быть планарным. Однако он непланарен, так что эта задача не имеет решения. Другой важный пример непланарного графа — полный график K_5 (рис. 4.23). Их непланарность будет показана ниже (следствие 4.5).

Теорема 4.14 (Эйлер). Для плоской карты с n вершинами, m ребрами, r гранями

$$n - m + r = 2. \quad (4.4)$$

Доказательство. Каждой внутренней грани плоского графа G соответствует простой цикл. Множество этих циклов независимо: ни один из них нельзя представить суммой двух других, так как эти суммы дают объединение областей. С другой стороны, любой другой цикл можно получить как сумму циклов, соответствующих граням: каждый такой цикл является внешней границей области, образованной объединением граней. В частности, цикл, соответствующий внешней грани (т. е. внешняя граница плоского графа), равен сумме векторов всех внутренних граней. Таким образом, циклы, соответствующие $r - 1$ внутренним граням графа G , образуют базис пространства циклов, мощность которого по теореме 4.11 равна цикломатическому числу $\gamma(G)$, т. е. $\gamma(G) = r - 1$. Так как для связного графа $\gamma(G) = m - n + 1$, то $r - 1 = m - n + 1$ и $n - m + r = 2$.

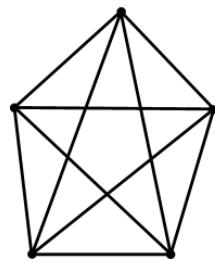


Рис. 4.23

Сам Эйлер сформулировал свою теорему для выпуклых многогранников. Но такие многогранники после некоторого топологического преобразования проектируются на плоскость (с сохранением параметров n , m , r) в виде плоского графа. Например, для куба это преобразование заключается в растяжении во все стороны верхней, либо нижней грани.

Другое доказательство. Докажем теорему индукцией по числу граней. При $r = 1$ единственной гранью графа G является внешняя грань. Такой граф не имеет циклов и, значит, является деревом, так как G связен. Тогда $m = n - 1$, $r = 1$ и $n - m + r = n - (n - 1) + 1 = 2$.

Пусть для любого связного плоского графа с r гранями $n - m + r = 2$. Рассмотрим произвольный связный плоский граф G' с $r' = r + 1$ гранью, n' вершинами и m' ребрами. Удалим произвольное ребро, входящее в границу между двумя гранями, не удаляя его концов. Тогда эти две грани сольются в одну, и мы получим плоский граф G с r гранями, $n = n'$ вершинами и $m = m' - 1$ ребрами. Так как всякая граница между гранями — это цикл, то удаление ребра не нарушает связности. Поэтому G связен, и по предположению индукции для него $n - m + r = 2$. Но тогда для графа G' $n' - m' + r' = n - (m + 1) + r + 1 = n - m + r = 2$. \square

Следствие 4.2. Если G — плоская карта, в которой каждая грань содержит q ребер, то

$$m = \frac{q(n - 2)}{q - 2}. \quad (4.5)$$

Так как любое ребро в G принадлежит двум граням, а каждая грань содержит q ребер, то $qr = 2m$. Подставив значение r из (4.4) в это равенство, получим (4.5). \square

Максимальным планарным графом называется простой планарный граф, который при добавлении любого ребра (без добавления вершин) перестает быть либо простым, либо планарным. Легко видеть, что в максимальном планарном графе все грани ограничены циклами длины 3 (грани с такими границами будем называть треугольниками): внутри грани с более длинными границами всегда можно провести хотя бы одно ребро, а граница длины 2 — это пара кратных ребер, которых в простом графе нет.

Заметим, что если планарный граф содержит ребро, не принадлежащее никакому циклу и, следовательно, никакой границе, то он не является максимальным.

Следствие 4.3. Для максимального планарного графа $m = 3n - 6$.

Получается подстановкой $q = 3$ в (4.5). \square

Отсюда получаем новое следствие.

Следствие 4.4. Если G — простой планарный граф с числом вершин $n \geq 3$, то

$$m \leq 3n - 6 \text{ (неравенство Эйлера).} \quad (4.6)$$

Это следствие, во-первых, дает необходимое условие планарности, а во-вторых, показывает, что в простых планарных графах не может быть слишком много ребер: если для произвольного простого графа без петель $m \leq n(n - 1)/2$, то для планарного графа $m \leq 3n - 6$.

Другой путь доказательства неравенства Эйлера использует следующий факт.

Теорема 4.15. В простом планарном графе с n вершинами, m ребрами и r гранями $3r \leq 2m$.

Сумма R длин всех границ такого графа не меньше $3r$, поскольку граница каждой грани содержит по крайней мере три ребра. В этой сумме ребра посчитаны дважды, так как каждое ребро находится на границе двух граней. Поэтому число граничных ребер равно $R/2$. Так как $R/2 \leq m$, то $3r \leq R \leq 2m$. \square

Из формулы $n - m + r = 2$ получаем: $r = 2 + m - n$. Подставляя значение r в неравенство $3r \leq 2m$, получим: $6 + 3m - 3n \leq 2m$, т. е. $m \leq 3n - 6$.

Следствие 4.5. Графы K_5 и $K_{3,3}$ не являются планарными.

Действительно, для K_5 $m = 10$, $n = 5$; поэтому (4.6) не выполняется, и, следовательно, он непланарен. Для $K_{3,3}$ $m = 9$, $n = 6$, т. е. (4.6) выполняется. Тем не менее он непланарен. Предположим, что он планарен. Тогда по теореме Эйлера число его граней $r = 9 - 6 + 2 = 5$. Так как в силу двудольности $K_{3,3}$ в нем нет циклов длины меньше 4, то, суммируя длины границ всех граней и учитывая, что в этой сумме каждое ребро графа $K_{3,3}$ встретится дважды, получим $2m \geq 4r$, т. е. $4r \leq 18$, и, следовательно, $r < 5$, что противоречит теореме Эйлера. \square

Таким образом, $K_{3,3}$ — пример того, что (4.6) не является достаточным условием планарности.

Необходимые и достаточные условия планарности содержит знаменитая теорема Понtryгина–Куратовского, для формулировки которой введем некоторые понятия.

Слиянием, или *стягиванием*, двух ребер (u, v) , (v, w) , инцидентных вершине степени 2, называется операция, которая эти ребра заменяет одним ребром (u, w) и удаляет вершину v .

Разбиением ребра (u, w) называется операция, обратная слиянию, т. е. добавление новой вершины v степени 2 и замена ребра (u, w) парой ребер (u, v) , (v, w) .

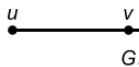
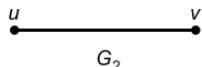


Рис. 4.24



Пример этих операций приведен на рис. 4.24. Переход от графа G_1 к графу G_2 — это стягивание, а переход от графа G_2 к графу G_1 — разбиение.

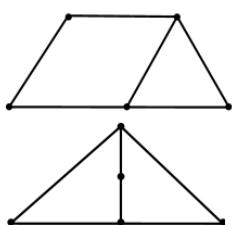


Рис. 4.25

Два графа называются *гомеоморфными*, если они изоморфны или могут быть сделаны изоморфными в результате конечного числа слияний и разбиений.

Пример 4.17. На рис. 4.25 представлены два гомеоморфных графа.

Теорема 4.16 (Понtryгин, Куратовский). Граф планарен, если и только если он не содержит подграфов, гомеоморфных K_5 и $K_{3,3}$. Доказательство этой теоремы весьма громоздко и здесь не приводится. Его можно найти, например, в книге Ф. Харари. □

4.7. РАСКРАСКИ, УСТОЙЧИВОСТЬ, ПОКРЫТИЯ

Раскраски. Раскраска элементов графа в k цветов, или k -раскраска — это разбиение элементов графа на k классов. Рассматривают раскраски вершин и ребер неориентированных графов, а также раскраски граней плоских карт. Раскраска называется правильной, если любая пара смежных элементов окрашена в разные цвета. (Границы плоской карты считаются смежными, если их грани-

цы имеют хотя бы одно общее ребро.) Задача раскраски заключается в нахождении минимального числа цветов, достаточного для правильной раскраски.

Задача о раскраске плоских карт — одна из самых старых проблем теории графов. Еще в конце XIX в. было доказано, что для раскраски любой плоской карты достаточно пяти красок. Высказанная тогда же «гипотеза четырех красок» (всегда достаточно четырех красок) ждала своего решения более полувека и успела стать почти такой же знаменитой, как и теорема Ферма. Однако в 70-х гг. XX в. эта гипотеза была доказана американскими математиками Аппелем и Хакеном. Их доказательство существенно использовало компьютерные вычисления, поскольку было связано с построением и анализом около двух тысяч различных случаев.

В дальнейшем мы ограничимся краткими сведениями о вершинных раскрасках.

Граф называется *k*-раскрашиваемым, если существует его правильная вершинная *k*-раскраска.

Хроматическим числом $\chi(G)$ графа G называется минимальное число k , для которого граф G *k*-раскрашиваемый. Граф G называется *k*-хроматическим, если $\chi(G) = k$, и бихроматическим, если $\chi(G) = 2$.

Пример 4.18. Граф на рис. 4.26 — 3-хроматический.

Теорема 4.17. Граф является бихроматическим, если и только если он двудольный.

Действительно, если граф можно представить как двудольный, то существует разбиение его вершин на два класса X , Y такие, что вершины, находящиеся в одном классе, попарно не смежны. Тогда присваиваем всем вершинам класса X один цвет, а всем вершинам класса Y — другой. Обратно, пусть граф — бихроматический. Отнесем в класс X все вершины, покрашенные в один цвет, а в класс Y — вершины, покрашенные в другой цвет. По определению правильной раскраски, вершины, находящиеся в одном классе, попарно не смежны, и, следовательно, классы X , Y можно рассматривать как доли двудольного графа. \square

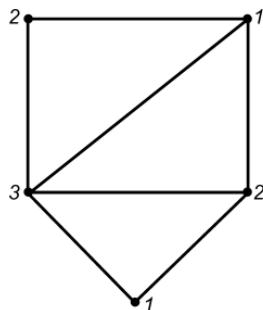


Рис. 4.26

Из этой теоремы легко вытекает следствие, которое может показаться неожиданным.

Следствие 4.6. Всякое дерево является двудольным графом.

Покажем, что вершины любого дерева можно окрасить в два цвета. Алгоритм окраски очень прост. На первом шаге выберем в дереве произвольную вершину и окрасим ее в цвет A . На втором шаге смежные с ней вершины окрасим в цвет B . В общем случае: если вершины, окрашенные на i -м шаге, получили цвет A , то на $(i + 1)$ -м шаге неокрашенные вершины, смежные с вершинами, окрашенными на i -м шаге, красим в цвет B , и наоборот. Такая окраска будет правильной, потому что в дереве нет циклов и, следовательно, вершины, окрашенные на $(i + 1)$ -м шаге, не могут быть смежными с вершинами, окрашенными на первых $i - 1$ шагах. Таким образом, всякое дерево является бихроматическим графом и, следовательно, двудольным. \square

Обозначим через Δ максимальную из степеней вершин в графе.

Теорема 4.18. Для любого простого графа $\chi(G) \leq \Delta + 1$.

Это означает, что любой простой граф можно раскрасить в $\Delta + 1$ цветов.

Метод такой раскраски заключается в следующем. Пусть дано множество $\Delta + 1$ цветов. Первой выбранной вершине присваиваем произвольный цвет из этого множества. В дальнейшем каждой выбираемой вершине v присваиваем цвет, который не присвоен смежным с ней вершинам. Это всегда возможно, так как $d(v) \leq \Delta$, и, следовательно, вершинам, смежным с v , присвоено не более Δ различных цветов. \square

Уточнение этой оценки дает теорема Брукса.

Теорема 4.19 (Брукс). Для полных графов и циклов нечетной длины $\chi(G) = \Delta + 1$. Для всех остальных графов $\chi(G) \leq \Delta$. \square

Внешняя и внутренняя устойчивость. Покрытия. Множество вершин графа G называется *внутренне устойчивым*, или *независимым*, если его вершины попарно не смежны.

Ясно, что если множество M внутренне устойчиво, то и любое его подмножество также внутренне устойчиво.

С другой стороны, множество всех вершин любого непустого графа внутренне устойчивым не является. Поэтому множество всех внутренне устойчивых множеств частично упорядочено по включению, и существуют максимальные внутренне устойчивые множества, т. е. такие, добавление к которым любой вершины нарушает внутреннюю устойчивость. Наибольшая из мощностей максимальных внутренне устойчивых множеств графа G называется *числом внутренней устойчивости* и обозначается $\alpha_0(G)$.

Пример 4.19. В графе на рис. 4.27 внутренне устойчивое множество $\{a, c\}$ максимально по включению, но не наибольшее по мощности. Наибольшим является множество $\{b, d, f\}$; $\alpha_0(G) = 3$.

Известная задача о 8 ферзях — расставить 8 ферзей на доске так, чтобы ни один из них не бил другого — сводится к нахождению наибольшего внутренне устойчивого множества в графе, вершинами которого являются все 64 поля шахматной доски, а полями, смежными с полем v , являются все поля, которые бьет ферзь, стоящий в поле v .

Множество называется *полностью зависимым*, если все его вершины попарно смежны, т. е. если оно является кликой.

Теорема 4.20. Если M внутренне устойчиво в графе $G(V, E)$, то M — клика в $G(V, \bar{E})$.

Действительно, если M внутренне устойчиво в $G(V, E)$, то ни одна пара вершин из M в этом графе не соединена ребрами. Следовательно, все эти пары в $G(V, \bar{E})$ соединены ребрами. \square

Множество X вершин графа G называется *внешне устойчивым*, если любая вершина G смежна с некоторой вершиной из X . Множество X вершин графа G называется *вершинным покрытием*, если оно покрывает все ребра, т. е. любое ребро G инцидентно некоторой вершине из X .

Пример 4.20. а. Множество $\{a, c\}$ на рис. 4.27 внешне устойчиво, но не является вершинным покрытием, так как не покрывает ребра (e, f) и (d, e) .

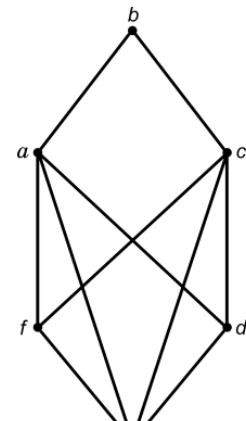


Рис. 4.27

6. Задача о 5 ферзях: расставить 5 ферзей на доске так, чтобы они били все поля.

Ясно, что если множество M внешне устойчиво, то и любое множество вершин, содержащее X , также внешне устойчиво. Поэтому множество всех внешне устойчивых множеств частично упорядочено по включению, и существуют минимальные внешне устойчивые множества, т. е. такие, удаление из которых любой вершины нарушает внешнюю устойчивость. Аналогично, существуют минимальные вершинные покрытия.

Наименьшая из мощностей минимальных вершинных покрытий графа G называется *числом вершинного покрытия* и обозначается $\beta_0(G)$.

Теорема 4.21. Для простого графа $G(V, E) M \subseteq V$ внутренне устойчиво (независимо), если и только если \bar{M} — вершинное покрытие.

Действительно, так как ни одно ребро не содержит в M обоих концов, то концы всех ребер содержатся в \bar{M} . Следовательно, \bar{M} — вершинное покрытие. \square

Теорема 4.22. Для простого графа $G(V, E) \alpha_0 + \beta_0 = n$.

Пусть S^* — максимальное независимое множество, K^* — минимальное вершинное покрытие. Тогда $|S^*| = \alpha_0$, $|K^*| = \beta_0$; по предыдущей теореме $\bar{S}^* = V - S^*$ — вершинное покрытие, $\bar{K}^* = V - K^*$ — независимое множество. Поэтому $|\bar{S}^*| = |V - S^*| = n - \alpha_0 \geq \beta_0$, и, следовательно, $n \geq \alpha_0 + \beta_0$; $|\bar{K}^*| = |V - K^*| = n - \beta_0 \leq \alpha_0$, и, следовательно, $n \leq \alpha_0 + \beta_0$. Объединяя эти два неравенства, получим $\alpha_0 + \beta_0 = n$. \square

Совокупность минимальных внешне устойчивых множеств простого графа можно описать дизъюнктивной нормальной формой без отрицаний, переменными которой

являются вершины графа. Каждая конъюнкция в ней соответствует одному такому множеству.

Алгоритм нахождения этой формулы по матрице смежности графа рассмотрим на примере. Пусть граф задан матрицей смежности в табл. 4.4.

Таблица 4.4

	a	b	c	d	e	f
a	0	1	0	1	1	0
b	1	0	1	0	0	0
c	0	1	0	1	0	1
d	1	0	1	0	1	0
e	1	0	0	1	0	1
f	0	0	1	0	1	0

Строим булево выражение из следующих соображений.

- Вершина a смежна с вершинами b, d, e ; поэтому она может быть покрыта любой из них, а кроме того, самой вершиной a ; это выражается дизъюнкцией $(a \vee b \vee d \vee e)$;
- вершина b покрывается вершиной a или b или c ;
- ...;
- вершина f покрывается вершиной c или e или f .

Получаем булеву формулу:

$$(a \vee b \vee d \vee e)(a \vee b \vee c)(b \vee c \vee d \vee f)$$

$$(a \vee c \vee d \vee e)(a \vee d \vee e \vee f)(c \vee e \vee f).$$

Раскрывая скобки и проведя склеивания по формуле $a \vee ab = a$, получим:

$$ac \vee af \vee be \vee cd \vee ce \vee ade \vee bcf \vee bdf.$$

Каждая из конъюнкций является минимальным внешне устойчивым множеством. Первые пять содержат две буквы, поэтому $\beta_0 = 2$. Используя вместо матрицы смежности матрицу инцидентности, можно аналогичным методом получить булево описание минимальных вершинных покрытий.

4.8. ОПТИМИЗАЦИОННЫЕ ЗАДАЧИ НА ГРАФАХ

В этом параграфе будет рассмотрено несколько оптимизационных задач, имеющих обширные применения. Как правило, эти задачи формулируются для *взвешенных* графов, т. е. для графов, ребрам которых приписаны некоторые (обычно положительные) числа. Интерпретация этих чисел в разных задачах различна.

Минимальные покрывающие деревья. В § 4.5 рассматривалась задача построения покрывающего дерева для невзвешенного неориентированного графа. Так как все покрывающие деревья имеют одинаковое число ребер, то для таких графов задача оптимизации покрывающих деревьев не стоит. Здесь мы рассмотрим задачу построения покрывающего дерева для графа G , ребрам e_i которого приписаны положительные веса $l(e_i)$. Весом $l(T)$ покрывающего дерева T назовем сумму весов входящих в него ребер. Минимальным покрывающим деревом называется покрывающее дерево с минимальным весом.

Теорема 4.23. Алгоритм построения покрывающего дерева, описанный в § 4.5, в котором ребра просматриваются в порядке возрастания их весов, строит минимальное покрывающее дерево.

Доказательство. Пусть алгоритм строит дерево T . Предположим, что минимальным является дерево S , отличное от T . Эти деревья отличаются по крайней мере одним ребром. Пусть ребро $e_1 = (x, y)$ — первое из просмотренных ребер, которое содержится в T , но не содержится в S . Поскольку S — дерево, в нем есть единственная цепь, соединяющая x и y . Обозначим эту цепь через $c(x, y)$. Если ребро e_1 добавить в S , то оно вместе с цепью $c(x, y)$ образует цикл. Этот цикл должен содержать ребро e_2 , которое не содержится в T .

Исключим e_2 из S и включим в него e_1 . Полученный граф (который тоже является покрывающим деревом) обозначим через S' . Так как S минимально, то $l(S') \geq l(S)$, и, следовательно, $l(e_1) \geq l(e_2)$.

Предположим, что e_2 при построении T просматривалось раньше, чем e_1 . Поскольку оно не включено в T , то для него имел место случай 2 г, т. е. оно образует цикл с ребрами T . Однако, поскольку e_1 — первое по возрастанию ребро, которое не содержится в S , все предыдущие ребра, вошедшие в T , должны содержаться и в S . Но, так как для e_2 выполняется случай 2 г, то оно не должно войти в S , что приводит к противоречию. Следовательно, наше предположение неверно, и e_2 при построении T просматривалось позже, чем e_1 , откуда следует, что $l(e_1) < l(e_2)$. Сопоставляя эти два неравенства, получим, что $l(e_1) = l(e_2)$ и потому $l(S') = l(S)$, т. е. S' — также минимальное покрывающее дерево. При этом S' имеет с T на одно общее ребро больше, чем S .

Повторяя это рассуждение с S' в качестве минимального покрывающего дерева, получим новое минимальное покрывающее дерево S'' , которое имеет с T на одно общее ребро больше, чем S' . После конечного числа таких построений получим минимальное покрывающее дерево, совпадающее с T . \square

Кратчайшие пути. Задача о кратчайшем пути, т. е. о нахождении пути минимальной длины в ориентированном графе, имеет несколько вариантов:

- 1) найти кратчайший путь из заданной вершины s в заданную вершину t ;
 - 2) найти кратчайшие пути из вершины s во все остальные вершины графа;
 - 3) найти кратчайшие пути между всеми упорядоченными парами вершин.
- Мы ограничимся первыми двумя вариантами, которые решаются практически одинаково.

Для обычных, т. е. невзвешенных графов длина пути равна числу ребер в нем. Алгоритм нахождения кратчайшего пути для этого случая крайне прост.

1. Начальной вершине s присваиваем ранг 0.

2. Всем вершинам, которые не имеют ранга и в которые входят ребра из вершин i -го ранга, присваиваем ранг $i + 1$.

Если требуется найти путь в заданную вершину t , то алгоритм останавливается, когда t получает ранг. Если же нужно найти пути из s во все вершины, то алгоритм останавливается, когда ранг получат все вершины. Ранг вершины есть длина пути в эту вершину. Доказательство очевидно, поскольку путь в любую вершину $(i + 1)$ -го ранга на единицу длиннее пути в вершину i -го ранга.

Простота этого алгоритма обеспечивается тем, что ранги, присваиваемые вершинам, не пересчитываются. Поэтому каждая вершина рассматривается только один раз.

Во взвешенном графе каждому ребру (x, y) присвоен положительный вес $l(x, y)$, называемый его длиной. Длиной пути называется сумма длин входящих в него ребер. В этом случае задача нахождения кратчайшего пути из вершины s решается алгоритмом, который предложил голландский математик Дейкстра. Если требуется найти кратчайший путь в вершину t , условием окончания является 4, а; если ищутся кратчайшие пути во все вершины, условие окончания — 4, б.

Алгоритм Дейкстры.

1. Положить $d(s) = 0$ и $d(x) = \infty$ для всех вершин $x \neq s$. Пометить s и положить $y = s$.

2. Для всех непомеченных вершин x , смежных с y , пересчитать $d(x)$ по формуле:

$$d(x) = \min\{d(x), d(y) + l(y, x)\}. \quad (4.7)$$

Если $d(x) = \infty$ для всех непомеченных вершин, закончить алгоритм: путей в эти вершины нет.

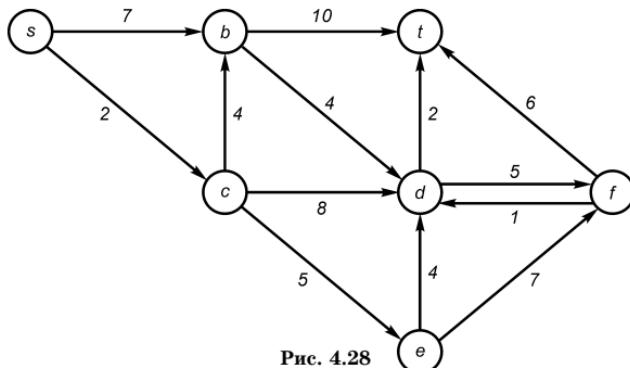
3. Пометить ту из вершин x , для которой $d(x)$ минимально. Кроме того, пометить ребро, ведущее в x , которое дало оценку $d(x)$. Положить $y = x$.

4. а) если $y = t$, закончить алгоритм. Иначе перейти к шагу 2;

б) если непомеченных вершин нет, закончить алгоритм. Иначе перейти к шагу 2.

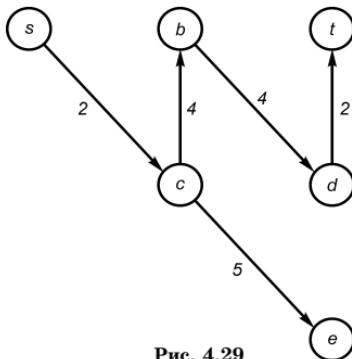
Ниже будет показано, что величина $d(x)$ для помеченной вершины x равна длине пути из s в x , а сам путь идет по помеченным ребрам. Но сначала рассмотрим пример.

Пример 4.21. Дан граф на рис. 4.28. Требуется найти путь из вершины s в вершину t .



Работа алгоритма показана в табл. 4.5. Первая строка соответствует шагу 1 алгоритма. Каждая из последующих строк соответствует одному основному циклу, состоящему из шагов 2 и 3. Значения $d(x)$ помеченных вершин (их окончательные значения) выделены полужирным шрифтом.

Таблица 4.5



Номер цикла	s	y	b	c	d	e	f	t
0	s	∞	∞	∞	∞	∞	∞	∞
1	0	c	7	2	∞	∞	∞	∞
2	0	b	6	2	∞	7	∞	∞
3	0	e	6	2	10	7	∞	16
4	0	d	6	2	10	7	14	16
5	0	t	6	2	10	7	14	12

Обозначим через T_k граф, образованный ребрами и вершинами, помеченными в первых k циклах. Вершину, помеченную в k -м цикле, обозначим через x_k . T_{k+1} получается из T_k добавлением x_{k+1} и помеченного ребра, входящего в x_{k+1} из некоторой вершины T_k . Из этого построения видно, что T_k связан и при любом k в каждую вершину T_k (кроме s) входит ровно одно ребро. Поэтому T_k — ориентированное дерево с корнем в s . На рис. 4.29 представлено дерево T_5 для примера 4.21.

Далее, обозначим через $d_k(x)$ значение $d(x)$ после k -го цикла. Окончательное значение $d(x)$ обозначим через $d_z(x)$. Переменная y алгоритма после k -го цикла принимает значение x_k . Формула (4.7) для $(k+1)$ -го цикла в этих обозначениях примет следующий вид:

$$d_{k+1}(x) = \min\{d_k(x), d_z(x_k) + l(x_k, x)\}. \quad (4.8)$$

Основное свойство величин $d(x)$, порождаемых алгоритмом Дейкстры, описывается следующим утверждением.

Лемма 4.3. Пусть x_i — вершина T_k , помеченная в i -м цикле. Тогда

1) $d_z(x_{i-1}) \leq d_z(x_i)$ для всех $i \leq k$;

2) для любой вершины x , непомеченной после k -го цикла, $d_z(x_k) \leq d(x)$.

Для вершины x_i

$$d_z(x_i) = d_z(x_j) + l(x_j, x_i), \quad (4.9)$$

где j — номер цикла, где $d(x_i)$ менялась последний раз. Если $j = i - 1$, то из (4.8) имеем $d_z(x_i) = d_z(x_{i-1}) + l(x_{i-1}, x_i)$, и первое утверждение выполняется.

Если же $j < i - 1$, то $d_z(x_i) = d_{i-1}(x_i)$ в $(i-1)$ -м цикле не менялась, и величины $d_{i-1}(x_{i-1}), d_{i-1}(x_i)$ рассматривались в этом цикле при выборе новой помеченной вершины. Поскольку была выбрана вершина x_{i-1} , то первое утверждение выполняется и в этом случае.

Аналогично доказывается и второе утверждение. \square

Докажем теперь корректность алгоритма Дейкстры.

Теорема 4.24. Для любого $k < n$ T_k является деревом кратчайших путей, т. е. путь в T_k , ведущий из s в любую его вершину x , является кратчайшим путем из s в x в исходном графе G , а $d_z(x)$ — его длина.

Доказательство проведем индукцией по числу циклов алгоритма.

Дерево T_1 состоит из вершин s , x_1 и ребра (s, x_1) . Вершины x графа G , для которых $d(x) \neq \infty$, смежны с s , причем $d(x) = l(s, x)$. Поскольку $d(x_1)$ минимально, то ребро (s, x_1) — кратчайший путь в G из s в x_1 , так как другие пути из s в x_1 начинаются с ребер, выходящих из s и имеющих больший вес, чем ребро (s, x_1) .

Предположим, что для первых k циклов наше утверждение верно. Тогда T_k содержит вершины s , x_1, \dots, x_k , и по предположению все пути в нем от s к x_1, \dots, x_k — кратчайшие. После $(k+1)$ -го цикла к T_k добавляется вершина $x = x_{k+1}$, для которой $d_{k+1}(x)$ минимально среди всех непомеченных вершин графа G . Докажем, что в T_{k+1} путь $P(s, x_{k+1})$ является кратчайшим путем в G от s к x_{k+1} , а $d_z(x_{k+1})$ — его длина.

Если это не так, то в G существует путь $P'(s, x_{k+1})$, длина которого меньше $d(x_{k+1})$. Выделим в P' последнее ребро (v, x_{k+1}) , т. е. представим его в виде $P'(s, x_{k+1}) = P'(s, v) (v, x_{k+1})$.

Пусть v содержится в T_k , т. е. $v = x_i$, $i < k$. Тогда длина P' равна $D = d_z(x_i) + l(x_i, x_{k+1})$ и $D < d(x_{k+1})$. Но так как x_i смежна с x_{k+1} , то в i -м цикле вершина x_{k+1} должна была получить оценку $d_z(x_i) + l(x_i, x_{k+1}) = D$. Поскольку в ходе алгоритма величины $d(x)$ могут только уменьшаться, то в текущем цикле $d(x_{k+1})$ не может быть больше D . Получаем противоречие. Следовательно, путь $P(s, x_{k+1})$ — кратчайший, и $d(x_{k+1})$ — его длина.

Пусть v не содержится в T_k . Найдем в P' первую вершину w , не содержащуюся в T_k . Так как отрезки кратчайшего пути — тоже кратчайшие пути, то отрезок $P'(s, w)$ — кратчайший. Рассмотрим в нем последнее ребро (u, w) . Так как u содержится в T_k , то, повторяя предыдущее рассуждение, получим, что длина $P'(s, w) = d(w)$. Так как мы предположили, что длина $P'(s, x_{k+1})$ меньше $d(x_{k+1})$, то заведомо $d(w) < d(x_{k+1})$. Но тогда получаем, что в текущем цикле помечена вершина x_{k+1} , для которой величина d не минимальна, что противоречит алгоритму. \square

Сетевое планирование. Задача сетевого планирования заключается в анализе сетевого графика. Сетевой график — это ациклический граф с одним источником и одним стоком. Ребрам графа приписаны положительные числа. Ребро рассматривается как операция, число $t(i, j)$ на ребре

(i, j) — как длительность операции. Вершина v_i рассматривается как событие, которое наступает тогда, когда выполнены все операции, соответствующие ребрам, входящим в v_i . Наступление события v_i означает, что можно начать операции, соответствующие ребрам, выходящим из v_i .

Сетевой график интерпретируется как план сложного проекта, состоящего из различных операций. Начальная вершина (источник) соответствует началу проекта, сток — его окончанию. Некоторые операции выполняются независимо друг от друга; другие могут начаться только после завершения предыдущих операций. Например, при строительстве многоэтажного дома возвведение крыши не может начаться, пока не возведены стены всех этажей, а отделочные работы на первом этаже могут проходить параллельно с возведением крыши.

Если в вершину v_i входит несколько ребер, т. е. для наступления события необходимо завершение нескольких операций, то некоторые из них могут закончиться раньше других. Это означает, что у этих операций есть резерв по времени. Информация о резервах важна для управления проектом, поскольку дает возможность в случае необходимости какие-то операции задержать, перебросить часть ресурсов на операции, задержка которых приведет к задержке всего проекта и т. д. Поэтому задача анализа проекта заключается, во-первых, в определении срока окончания всего проекта и, во-вторых, в выявлении резервов времени для всех операций. Как будет видно из дальнейшего, существуют различные виды резервов.

Для удобства формулировок задач анализа сетевого графика и алгоритмов их решения будем считать, что на графе проведена топологическая сортировка и для любого ребра (i, j) $i < j$. Обозначим через $E(i)$ наиболее ранний возможный срок наступления события i , а через $L(i)$ — наиболее поздний срок наступления события i , не нарушающий срока окончания всего проекта. Эти величины вычисляются следующим образом:

$$E(1) = 0.$$

Для $i = 2, \dots, n$ $E(j) = \max(E(i) + t(i, j))$.

Из алгоритма видно, что $E(j)$ — это максимальная длина пути от 1 до j . Это легко доказывается по индукции.

Ранний срок для заключительной вершины — это минимальный срок окончания всего проекта.

Поздний срок вычисляется «с конца»:

$L(n) = E(n)$ — в случае, если проект должен закончиться как можно раньше; в общем случае можно положить $L(n) \geq E(n)$.

Для $i < n$ $L(i) = \min(L(j) - t(i, j))$. Из алгоритма видно, что $L(n) - L(j)$ — это максимальная длина пути от j до n . Это также доказывается по индукции.

Пример 4.22. Рассмотрим сетевой график на рис. 4.30. Единицей времени будем считать дни. Результаты вычисления ранних и поздних сроков приведены в табл. 4.6.

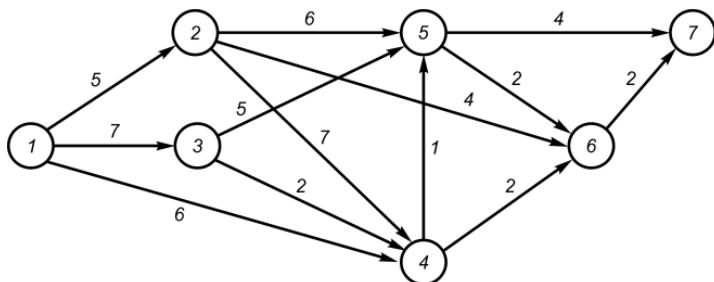


Рис. 4.30

Таблица 4.6

j	1	2	3	4	5	6	7
$E(j)$	0	5	7	12	13	15	17
$L(j)$	0	5	8	12	13	15	17
Резерв событий	0	0	1	0	0	0	0

Поскольку операция (i, j) может начаться не раньше $E(i)$, а закончиться — не позднее $L(j)$, т. е. начаться не позднее $L(j) - t(i, j)$, то максимальная общая задержка начала этой операции, которая не задержит проект в целом, равна

$$L(j) - E(i) - t(i, j).$$

Эта величина называется *полным резервом* (ПР) операции (i, j) .

Для того чтобы операция (i, j) не влияла на временные ограничения последующих операций, она должна закончиться к моменту $E(j)$. Так как начаться она может не раньше $E(i)$, то остающийся резерв времени не превосходит $E(j) - E(i) - t(i, j)$.

Эта величина называется *свободным резервом* (СР) операции (i, j) . Она равна максимальной задержке операции, не влияющей на выполнение последующих операций.

Для того чтобы операция (i, j) не влияла на временные ограничения любых операций проекта, она должна начаться как можно позже, т. е. в момент $L(i)$, а закончиться как можно раньше, т. е. в момент $E(j)$. Остающийся резерв времени равен величине

$$E(j) - L(i) - t(i, j),$$

которая называется *независимым резервом* (НР) операции (i, j) .

Сравнение этих трех величин показывает, что для любой операции (i, j) полный резерв \geq свободный резерв \geq независимый резерв.

Для примера 4.22 все определенные выше величины приведены в табл. 4.7.

Операция, полный резерв которой равен нулю, называется *критической*. Ее задержка приводит к задержке всего проекта.

Так как $L(n) = E(n)$, а $E(n)$ — длина пути максимальной длины от 1 до n , то любая операция на этом пути

Таблица 4.7

Ребра i,j	$E(i)$	$E(j)$	$L(i)$	$L(j)$	$t(i,j)$	ПР	СР	НР	
12	0	5	0	5	5	0	0	0	критическая
13	0	7	0	8	7	1	0	0	
14	0	12	0	12	6	6	6	6	
24	5	12	5	12	7	0	0	0	критическая
25	5	13	5	13	6	2	2	2	
26	5	15	5	15	4	6	6	6	
34	7	12	8	12	2	3	3	2	
35	7	13	8	13	5	1	1	0	
45	12	13	12	13	1	0	0	0	критическая
46	12	15	12	15	2	1	1	1	
56	13	15	13	15	2	0	0	0	критическая
57	13	17	13	17	4	0	0	0	критическая
67	15	17	15	17	2	0	0	0	критическая

является критической и сам путь называется *критическим*. Задержка любой операции на критическом пути приводит к задержке всего проекта.

Из примера 4.22 видно, что хотя резервы событий в графике минимальны (только в вершине 3 резерв ненулевой, а остальные события задерживать нельзя), резервы операций гораздо более разнообразны. Поясним смысл различных резервов.

Операцию 13 можно задержать на один день: $\text{ПР}(13) = 1$, однако при этом лишится резерва следующая операция 35. Поэтому $\text{СР}(13) = 0$.

$\text{СР}(34) = 3$ означает, что если операцию 34 начать на 3 дня позже раннего срока, то это не повлияет на резервы последующих операций. Однако эта задержка возможна только в том случае, если не использован резерв предыдущей операции 13. Если же резерв $\text{ПР}(13) = 1$ желательно сохранить, то возможная задержка 34 уменьшается на 1. Поэтому $\text{НР}(34) = 2$.

Потоки в сетях. Задача о потоке в сети формулируется следующим образом. Дан ориентированный граф, в дальнейшем называемый сетью, с одним источником s (вершиной без входящих ребер) и одним стоком t (вершиной без выходящих ребер). Каждому ребру (i, j) графа приписано положительное число $c(i, j)$, называемое *пропускной способностью* ребра.

Поток в сети — это функция $f((i, j))$, определенная на ребрах графа и удовлетворяющая следующим условиям:

- 1) для любого графа $0 \leq f(i, j) \leq c(i, j);$
- 2) если $i \neq s, i \neq t$, то

$$\sum_j f(i, j) - \sum_j f(j, i) = 0 \quad (\text{закон Кирхгофа});$$

$$3) \sum_j f(s, j) = \sum_j f(j, t) = v.$$

Число v называется *величиной потока*.

Задача состоит в определении функции $f(i, j)$ таким образом, чтобы величина потока была максимальной.

Введем величину $q(i, j) = c(i, j) - f(i, j)$. Обозначим через Q множество ребер, для которых $q(i, j) > 0$, а через R — множество ребер, для которых $f(i, j) > 0$. Эти множества могут пересекаться.

Цепь P (без учета ориентации ребер) из s в t , все ребра которой, ориентированные от s к t (прямые), принадлежат Q , а все ребра, ориентированные от t к s (обратные), принадлежат R , называется *увеличивающей*. На рис. 4.31 ребра (s, a) , (a, b) , (d, t) — прямые, а ребра (c, b) , (d, c) — обратные. Если в сети с заданным потоком существует увеличивающая цепь, то величину потока в сети можно увеличить, изменяя величины потока на ребрах этой цепи. Рассмотрим, как это сделать.

Первое (выходящее из s) и последнее (входящее в t) ребра увеличивающей цепи — всегда прямые. Любую неконцевую вершину j этой цепи можно отнести к одному из трех классов.

1. Входящее в нее ребро (i, j) и выходящее ребро (j, k) — прямые. На этих ребрах поток можно увеличить на одно и то же число, но не более, чем на величину $\min\{q(i, j), q(j, k)\}$ — при этом закон Кирхгофа в вершине j не будет нарушен. На рис. 4.31 к этому классу относится вершина a .

2. Входящее в нее (со стороны s) ребро (j, i) и выходящее (в сторону t) ребро (k, j) — обратные. На этих ребрах поток можно уменьшить на одно и то же число, но не более, чем на величину $\min\{f(j, i), f(k, j)\}$ — при этом закон Кирхгофа также не будет нарушен. На рис. 4.31 к этому классу относится вершина c , для которой входящим ребром считаем (c, b) , а выходящим — (d, c) .

3. Входящее ребро (i, j) — прямое, а выходящее ребро (k, j) — обратное, либо наоборот: входящее — обратное, а выходящее — прямое. В обоих случаях можно на прямом ребре поток увеличить, а на обратном — уменьшить на одну и ту же величину, не превосходящую

$$\min\{q(i, j), f(k, j)\}.$$

На рис. 4.31 к этому классу относятся вершины b (первый случай) и d (второй случай).

На всех прямых ребрах увеличивающей цепи увеличим поток на величину $\min\{\min\{q(i, j)\}, \min\{f(i, j)\}\}$, а на

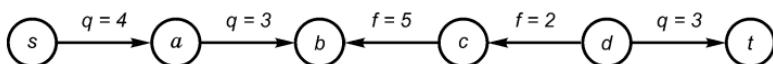


Рис. 4.31

всех обратных ребрах — уменьшим его на ту же величину. Тогда, поскольку первое и последнее ребра цепи — прямые, на эту величину возрастет и общая величина потока. Таким образом, поток в сети можно увеличивать до тех пор, пока в сети есть увеличивающие цепи.

В нашем примере $\min\{\min\{q(i, j)\}, \min\{r(i, j)\}\} = 2$.

Схема алгоритма поиска увеличивающей цепи.

1. Пометить s .

2. Для помеченной вершины i найти смежную с ней непомеченную вершину j , содержащуюся либо в Q , либо в R : а) если $(i, j) \in Q$, пометить j и (i, j) ; б) если $(j, i) \in R$, пометить j и (j, i) .

3. Остановиться, если либо помечена t , либо шаг 2 невозможен. Иначе перейти к шагу 2.

Помеченные вершины и ребра образуют дерево, поскольку ребра, соединяющие две помеченные вершины, не рассматриваются. Поэтому, если t помечена, то тем самым построена единственная помеченная цепь из s в t , которая является увеличивающей. Если же шаг 3 невозможен, а t не помечена, то увеличивающих цепей нет.

Описанная схема становится конкретным алгоритмом, если зафиксировать правила выбора помеченной вершины i и непомеченной вершины j (в случае, когда для i имеется несколько вершин j , удовлетворяющих условиям п. 2). Построенное дерево зависит от выбора этого правила.

Пример 4.23. На рис. 4.32 для помеченной вершины a существуют две вершины, удовлетворяющие условию шага 2, — b и c . При выборе c получим следующее помеченное дерево (рис. 4.33), которое дает увеличивающую цепь $sacbt$.

Таким образом, общий алгоритм нахождения максимального потока формулируется следующим образом.

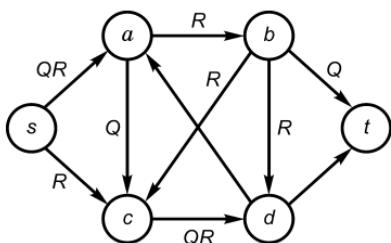


Рис. 4.32

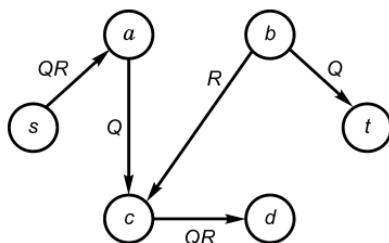


Рис. 4.33.

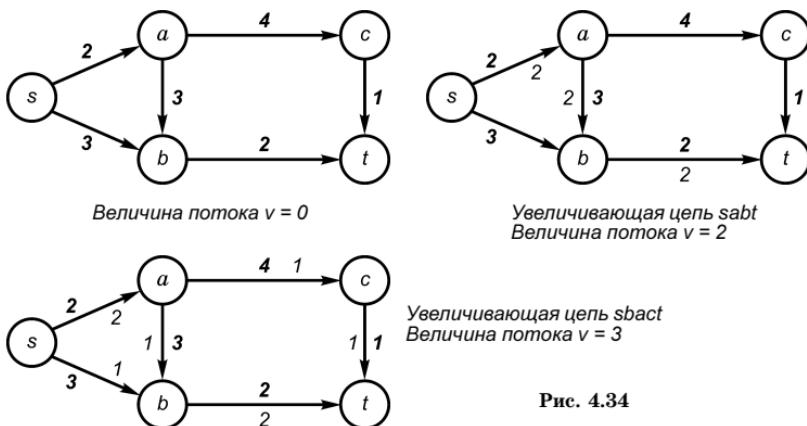


Рис. 4.34

1. Определить начальный поток (например, нулевой).
2. Определить множества Q , R .

3. Найти увеличивающую цепь и провести на ней максимальное увеличение потока. Если такой цепи нет, закончить алгоритм. Иначе перейти к шагу 2.

На рис. 4.34 приведен пример построения максимального потока, которое заканчивается за три цикла. Вершины обозначены буквами, пропускные способности — прямыми жирными цифрами, поток на ребрах — курсивными нежирными цифрами. Отсутствие потока на ребре означает, что он равен нулю.

Теорема 4.25. Поток, построенный описанным алгоритмом, максимален.

Введем понятие разреза. Разрезом связного графа называется множество ребер, удаление которых делает граф несвязным. Весь поток в сети должен проходить через все разрезы. Пропускная способность разреза K равна

$c_K = \sum_{(i, j) \in K} c(i, j)$. Поэтому $\max v = \min_K c_K$. Алгоритм построения потока завершается, когда больше нет увеличивающих цепей. Рассмотрим неориентированное дерево, построенное на последнем шаге алгоритма поиска увеличивающей цепи, и выделим ребра, соединяющие его помеченные ребра с непомеченными. Эти ребра образуют разрез K_0 . Прямые ребра этого разреза не принадлежат Q (иначе бы алгоритм продолжился). Следовательно, для этих ребер $f(i, j) = c(i, j)$, и поток по ним увеличить нельзя. Обратные ребра не принадлежат R , поэтому

для них $f(i, j) = 0$, т. е. поток в них отсутствует, и, значит, обратного потока через них нет. Следовательно, построенный поток через K_0 равен его пропускной способности, и увеличить его нельзя. \square

К задаче о потоках легко сводится задача о максимальном паросочетании в двудольном графе. Пусть дан двудольный граф $G(X, Y, E)$. Ориентируем его ребра от X к Y , введем новые вершины s и t , проведем от s ребра ко всем вершинам X , от всех вершин Y проведем ребра к t и назначим всем ребрам пропускную способность 1. Построим максимальный поток в этом графе. Тогда ребра от X к Y , на которых поток равен 1, образуют максимальное паросочетание в исходном графе $G(X, Y, E)$.

Заключительные замечания. Графы являются одним из наиболее распространенных языков для представления разнообразных математических объектов и формулировки различных теоретических и прикладных задач, выходящих за пределы собственно теории графов. Соответствия между конечными множествами (§ 1.2) могут быть представлены двудольными графами, а решетки (§ 2.2) — ациклическими графами. В этой главе уже говорилось о связи графов и бинарных отношений. Графы используются в математической логике и теории формальных систем (например, для представления вывода), в теории алгоритмов и теоретическом программировании (для представления и анализа алгоритмов и программ), в теории языков и грамматик (для синтаксического анализа текстов). Примеры такого использования будут представлены в последующих главах. Кроме того, они широко применяются в теории управления, исследовании операций, теории кодирования, искусственном интеллекте и т. д.

Трудно назвать прикладную область, где не применяется теория графов. Это — прикладная информатика, экономика, управление, электротехника, проектирование электронных схем и систем коммуникаций. Графы используются для описания и анализа как статических структур (организационных структур, транспортных сетей), так и динамических систем, в которых вершинам соответствуют состояния системы, а ориентированным ребрам — переходы из одного состояния в другое.

ТЕОРИЯ АЛГОРИТМОВ

5.1. ПРЕДВАРИТЕЛЬНОЕ ОБСУЖДЕНИЕ

Несколько вступительных слов. С алгоритмами, т. е. эффективными процедурами, однозначно приводящими к результату, математика имела дело всегда. Школьные методы умножения «столбиком» и деления «углом», метод исключения неизвестных при решении системы линейных уравнений, правило дифференцирования сложной функции, способ построения треугольника по трем заданным сторонам — все это алгоритмы. Однако пока математика имела дело в основном с числами и вычислениями и понятие алгоритма отождествлялось с понятием метода вычисления, потребности в изучении самого этого понятия не возникало. Традиции организации вычислений складывались веками и стали составной частью общей научной культуры в той же степени, что и элементарные навыки логического мышления. Все многообразие вычислений комбинировалось из 10–15 четко определенных операций арифметики, тригонометрии и анализа. Поэтому понятие метода вычисления считалось изначально ясным и не нуждалось в специальных исследованиях.

До середины XIX в. единственной областью математики, работавшей с нечисловыми объектами, была геометрия, и как раз она, не имея возможности опираться на вычислительную интуицию человека, резко отличалась от остальной математики повышенными требованиями к строгости своих рассуждений. До сих пор любой современный шестиклассник, для которого математика — это мир вычислений (и в этом он мало чем отличается от

тического инженера), мучительно привыкает к понятиям доказательства и математического построения и никак не может понять, зачем доказывать равенство отрезков, когда их проще измерить, и зачем строить перпендикуляр с помощью циркуля и линейки, когда есть угольник с «готовым» прямым углом или транспортир.

Такое же мучительное привыкание к новым, более жестким требованиям строгости началось в математике во второй половине XIX в. Оно стимулировалось в основном математикой нечисловых объектов — открытием неевклидовых геометрий, появлением абстрактных алгебраических теорий типа теории групп и т. д. Одним из решающих обстоятельств, приведших к пересмотру оснований математики, т. е. принципов, лежащих в основе математических рассуждений, явилось создание Кантором теории множеств. Довольно быстро стало ясно, что понятия теории множеств в силу своей общности лежат в основе всего здания математики. Однако почти столь же быстро было показано, что некоторые кажущиеся вполне естественными рассуждения в рамках этой теории приводят к неразрешимым противоречиям — парадоксам теории множеств (которые упоминались в гл. 1; более подробно см. [11]). Все это потребовало точного изучения принципов математических рассуждений (до сих пор казавшихся интуитивно ясными) математическими же средствами. Возникла особая отрасль математики — основания математики, или метаматематика.

Опыт парадоксов теории множеств научил математику крайне осторожно обращаться с бесконечностью и по возможности даже о бесконечности рассуждать с помощью финитных методов (см. § 3.4). Существование финитного подхода заключается в том, что он допускает только конечные комплексы действий над конечным числом объектов. Выяснение того, какие объекты и действия над ними следует считать точно определенными, какими свойствами и возможностями обладают комбинации элементарных действий, что можно и чего нельзя сделать с их помощью, — все это стало предметом теории алгоритмов и формальных систем, которая первоначально возникла в рамках метаматематики и стала важнейшей ее частью. Главным внутриматематическим приложением теории

алгоритмов явились доказательства невозможности алгоритмического (т. е. точного и однозначного) решения некоторых математических проблем. Такие доказательства (да и точные формулировки доказываемых утверждений) неосуществимы без точного понятия алгоритма.

Пока техника использовала чисто вычислительные методы, эти высокие проблемы чистой математики ее мало интересовали. В технику термин «алгоритм» пришел вместе с кибернетикой. Если понятие метода вычисления не нуждалось в пояснениях, то понятие процесса управления пришлось вырабатывать практически заново. Понадобилось осознавать, каким требованиям должна удовлетворять последовательность действий (или ее описание), чтобы считаться конструктивно заданной, т. е. иметь право называться алгоритмом. В этом осознании огромную помощь инженерной интуиции оказала практика использования вычислительных машин, сделавшая понятие алгоритма ощутимой реальностью. С точки зрения современной практики, алгоритм — это программа, а критерием алгоритмичности процесса является возможность его запрограммировать. Именно благодаря этой реальности алгоритма, а также потому, что подход инженера к математическим методам всегда был конструктивным, понятие алгоритма в технике за короткий срок стало необычайно популярным (быть может, даже больше, чем в самой математике).

Однако у всякой популярности есть свои издержки. В повседневной практике слово «алгоритм» употребляется слишком широко, теряя зачастую свой точный смысл. Приблизительные описания понятия «алгоритм» (вроде того, которое приведено в первой фразе этого параграфа) часто принимаются за точные определения. В результате за алгоритм зачастую выдается любая инструкция, разбитая на шаги. Появляются такие дикие словосочетания, как «алгоритм изобретения» (а ведь наличие «алгоритма изобретения» означало бы конец изобретательства как творческой деятельности).

Ясное представление о том, что такое алгоритм, важно, конечно, не только для правильного словоупотребления. Оно нужно и при разработке конкретных алгоритмов, особенно когда имеется в виду их последующее

программирование. Однако оно еще важнее при наведении порядка в бурно растущем алгоритмическом хозяйстве. Чтобы ориентироваться в море алгоритмов, захлестнувшем прикладную информатику, необходимо уметь сравнивать различные алгоритмы решения одних и тех же задач, причем не только по качеству решения, но и по характеристикам самих алгоритмов (числу действий, расходу памяти и т. д.). Такое сравнение невозможно без введения точного языка для обсуждения всех этих вопросов; иначе говоря, сами алгоритмы должны стать такими же предметами точного исследования, как и те объекты, для работы с которыми они предназначены.

Строгое исследование основных понятий теории алгоритмов начнется в следующем параграфе. Прежде чем приступить к нему, обсудим на неформальном уровне некоторые основные принципы, по которым строятся алгоритмы, и выясним, что же именно в понятии алгоритма нуждается в уточнении.

Основные требования к алгоритмам. 1. Первое, что следует отметить в любом алгоритме, — это то, что он применяется к исходным данным и выдает результаты. В привычных технических терминах это означает, что алгоритм имеет входы и выходы. Кроме того, в ходе работы алгоритма появляются промежуточные результаты, которые используются в дальнейшем. Таким образом, каждый алгоритм имеет дело с *данными* — входными, промежуточными и выходными. Поскольку мы собираемся уточнять понятие алгоритма, нужно уточнить и понятие данных, т. е. указать, каким требованиям должны удовлетворять объекты, чтобы алгоритмы могли с ними работать.

Ясно, что эти объекты должны быть четко определены и отличимы как друг от друга, так и от «необъектов». Во многих важных случаях хорошо понятно, что это значит: к таким алгоритмическим объектам относятся числа, векторы, матрицы смежностей графов, формулы. Изображения (например, рисунок графа) представляются менее естественными в качестве алгоритмических объектов. Если говорить о графике, то дело даже не в том, что в рисунке больше несущественных деталей и два человека один и тот же график изобразят по-разному (в конце концов, раз-

ные матрицы смежности тоже могут задавать один и тот же граф с точностью до изоморфизма), а в том, что матрица смежности легко разбивается на элементы, причем из элементов всего двух видов (нулей и единиц) состоят матрицы любых графов, тогда как разбить на элементы рисунок гораздо труднее. Наконец, с такими объектами, как «хорошая книга» или «осмысленное утверждение», с которыми легко управляется любой человек (но каждый по-своему!), алгоритм работать откажется, пока они не будут описаны как данные с помощью других, более подходящих объектов.

Вместо того чтобы пытаться дать общее словесное определение четкой определенности объекта, в теории алгоритмов фиксируют конкретные конечные наборы исходных объектов (называемых элементарными) и конечный набор средств построения других объектов из элементарных. Набор элементарных объектов образует конечный алфавит исходных символов (цифр, букв и т. д.), из которых строятся другие объекты; типичным средством построения являются индуктивные определения, указывающие, как строить новые объекты из уже построенных. Простейшее индуктивное определение — это определение некоторого множества слов, классическим примером которого служит определение идентификатора в языках программирования; идентификатор — это либо буква, либо идентификатор, к которому приписана справа буква или цифра. Слова конечной длины в конечных алфавитах (в частности, числа) — наиболее обычный тип алгоритмических данных, а число символов в слове (длина слова) — естественная единица измерения объема обрабатываемой информации. Более сложный случай алгоритмических объектов — формулы. Они также определяются индуктивно и также являются словами в конечном алфавите, однако не каждое слово в этом алфавите является формулой. В этом случае обычно основным алгоритмам предшествуют вспомогательные, которые проверяют, удовлетворяют ли исходные данные нужным требованиям. Такая проверка называется синтаксическим анализом (см. гл. 7).

2. Данные для своего размещения требуют памяти.
Память обычно считается однородной и дискретной, т. е.

состоит из одинаковых ячеек, причем каждая ячейка может содержать один символ алфавита данных. Таким образом, единицы измерения объема данных и памяти согласованы. При этом память может быть бесконечной. Вопрос о том, нужна ли одна память или несколько и, в частности, нужна ли отдельная память для каждого из трех видов данных (входных, выходных и промежуточных), решается по-разному.

3. Алгоритм состоит из *отдельных элементарных шагов*, или *действий*, причем множество различных шагов, из которых составлен алгоритм, конечно. Типичный пример множества элементарных действий — система команд компьютера. Обычно элементарный шаг имеет дело с фиксированным числом символов (это удобно, например, для измерения времени работы алгоритма числом проделанных шагов), однако это требование не всегда выполняется. Например, в компьютерах есть команды типа «память — память», работающие с полями памяти переменной длины.

4. Последовательность шагов алгоритма *детерминирована*, т. е. после каждого шага либо указывается, какой шаг делать дальше, либо дается команда остановки, после чего работа алгоритма считается *законченной*.

5. Естественно от алгоритма потребовать *результативности*, т. е. остановки после конечного числа шагов (зависящего от данных) с указанием того, что считать результатом. В частности, всякий, кто предъявляет алгоритм решения некоторой задачи, например вычисления функции $f(x)$, обязан показать, что алгоритм останавливается после конечного числа шагов (как говорят, *сходится*) для любого x из области задания f . Однако проверить результативность (сходимость) гораздо труднее, чем требования, изложенные в пп. 1–4. В отличие от них сходимость обычно не удается установить простым просмотром описания алгоритма; общего же метода проверки сходимости, пригодного для любого алгоритма A и любых данных x , как будет показано далее, вообще не существует. Обсуждение трудностей, связанных с распознаванием сходимости, см. в § 5.4.

6. Следует различать: а) описание алгоритма (инструкцию или программу); б) механизм реализации алго-

ритма (например, компьютер), включающий средства пуска, остановки, реализации элементарных шагов, выдачи результатов и обеспечения детерминированности, т. е. управления ходом вычисления; в) процесс реализации алгоритма, т. е. последовательность шагов, которая будет порождена при применении алгоритма к конкретным данным. Будем предполагать, что описание алгоритма и механизм его реализации конечны (память, как уже говорилось, может быть бесконечной, но она не включается в механизм). Требования к конечности процесса реализации совпадают с требованиями результативности (см. п. 5).

Пример 5.1. Рассмотрим следующую задачу: дана последовательность P из n положительных чисел (n — конечное, но произвольное число); требуется упорядочить их, т. е. построить последовательность R , в которой эти же числа расположены в порядке возрастания. Почти сразу же приходит в голову следующий простой способ ее решения: просматриваем P и находим в ней наименьшее число; вычеркиваем его из P и выписываем его в качестве первого числа R ; снова обращаемся к P и находим в ней наименьшее число; приписываем его справа к полученной части R и так далее, до тех пор, пока в P не будут вычеркнуты все числа.

Возникает естественный вопрос: что значит «и так далее». Для большей ясности перепишем описание способа решения в более четкой форме, разбив его на шаги и указав переходы между шагами.

Шаг 1. Ищем в P наименьшее число.

Шаг 2. Найденное число приписываем справа к R (в начальный момент R пуста) и вычеркиваем его из P .

Шаг 3. Если в P нет чисел, то переходим к шагу 4. В противном случае переходим к шагу 1.

Шаг 4. Конец. Результатом считать последовательность R , построенную к данному моменту.

Многие читатели сочтут такое описание достаточно ясным (и даже излишне формальным), чтобы, пользуясь им, однозначно получить нужный результат. Однако это впечатление ясности опирается на некоторые неявные предположения, к правильности которых мы привыкли, но которые нетрудно нарушить. Например, что

значит «дана последовательность чисел»? Является ли таковой последовательность $\sqrt[3]{3}$, $\sqrt[5]{2}$, $(1, 2)^\pi$? Очевидно, да, однако в нашем описании ничего не сказано, как найти наименьшее число среди таких чисел. В нем вообще не говорится о том, как искать наименьшие числа, по-видимому, предполагается, что речь идет о числах, представленных в виде десятичных дробей, и что известно, как их сравнивать.

Итак, необходимо уточнить формы представления данных. При этом нельзя просто заявить, что допустимо любое представление чисел. Ведь для каждого представления существует свой алфавит (который, помимо цифр, может включать запятые, скобки, знаки операций и функций и т. д.) и свой способ сравнения чисел (например, способ перевода в десятичную дробь), тогда как конечность алфавита требует фиксировать его заранее, а конечность описания алгоритма позволяет включить в него лишь заранее фиксированное число способов сравнения. Фиксация представления чисел в виде десятичных дробей также не решает всех проблем. Сравнение 10–20-разрядных чисел уже не может считаться элементарным действием: сразу нельзя сказать, какое из чисел больше: 908115577001,15 или 32899901467,0048. В машинных алгоритмах само представление числа еще требует дальнейшего уточнения: нужно, во-первых, ограничить число разрядов (цифр) в числе, так как от этого зависит, сколько ячеек памяти будет занимать число, а во-вторых, договориться о способе размещения десятичной запятой в числе, т. е. выбрать представление в виде числа с фиксированной или плавающей запятой, поскольку способы обработки этих двух представлений различны. Наконец, любой, кто имел дело с программированием, отметит, что на шаге 1 требуется узнать две вещи: само наименьшее число (чтобы записать его в R) и его место в P , т. е. его адрес в той части памяти, где хранится P (чтобы вычеркнуть его из P), а следовательно, нужно иметь средства указания этого адреса.

Таким образом, даже в этом простом примере несложный анализ показывает, что описанию, которое выглядит вполне ясным, еще далеко до алгоритма. Мы столкнулись здесь с необходимостью уточнить почти все основ-

ные характеристики алгоритма, которые отмечались ранее: алфавит данных и форму их представления, память и размещение в ней элементов P и R , элементарные шаги (поскольку шаг 1 явно не элементарен). Кроме того, становится ясным, что выбор механизма реализации (скажем, человека или компьютера) будет влиять и на сам характер уточнения: у человека требования к памяти, представлению данных и к элементарности шагов гораздо более слабые и «укрупненные», отдельные незначительные детали он может уточнить сам.

Пожалуй, только два требования к алгоритмам в приведенном описании выполнены в достаточной мере (они-то и создают впечатление ясности). Довольно очевидна сходимость алгоритма: после выполнения шагов 1 и 2 либо работа заканчивается, либо из P вычеркивается одно число; поэтому ровно после n выполнений шагов 1 и 2 из P будут вычеркнуты все числа и алгоритм остановится, а R будет результатом. Кроме того, не вызывает сомнения детерминированность: после каждого шага ясно, что делать дальше, если учесть, что здесь и в дальнейшем используется общепринятое соглашение — если шаг не содержит указаний о дальнейшем переходе, то выполняем шаг, следующий за ним в описании. Поскольку использованные в примере средства обеспечения детерминированности носят довольно общий характер, остановимся на них несколько подробнее.

Блок-схемы алгоритмов. Связи между шагами можно изобразить в виде графа. Для примера 5.1 график изображен на рис. 5.1.

Такой график, в котором вершинам соответствуют шаги, а ребрам — переходы между шагами, называется *блок-схемой* алгоритма. Его вершины могут быть двух видов: вершины, из которых выходит одно ребро (их называют операторами), и вершины, из которых выходит два ребра (их называют логическими условиями, или предикатами). Кроме того, имеется единственный оператор конца,

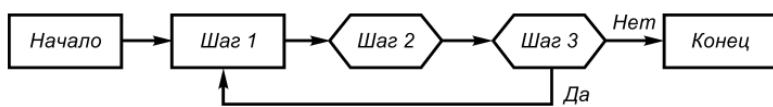


Рис. 5.1

из которого не выходит ни одного ребра, и единственный оператор начала. Важной особенностью блок-схемы является то, что связи, которые она описывает, не зависят от того, являются ли шаги элементарными или представляют собой самостоятельные алгоритмы, — как говорят в программировании, блоки (по существу шаг 1 таковым и является). Возможность «разбlocчивать» алгоритм хорошо известна в программировании и широко там используется: большой алгоритм разбивается на блоки, которые можно раздать для программирования разным лицам. Для данного блока неважно, как устроены другие блоки; для программирования блока достаточно знать, где лежит вся исходная информация, какова форма ее представления, что должен делать блок и куда записать результат.

С помощью блок-схем можно, наоборот, несколько алгоритмов, рассматриваемых как блоки, связать в один большой алгоритм. В частности, если алгоритм A_1 , вычисляющий функцию $f_1(x)$, соединен с алгоритмом A_2 , вычисляющим функцию $f_2(x)$ (рис. 5.2), и при этом исходными данными для A_2 служит результат A_1 , то полученная блок-схема задает алгоритм, вычисляющий $f_2(f_1(x))$, т. е. композицию f_1 и f_2 (см. § 1.2). Такое соединение алгоритмов называется *композицией* алгоритмов.

На блок-схеме хорошо видна разница между описанием алгоритма и процессом его реализации. Описание — это граф; процесс реализации — это путь в графе. Различные пути в одном и том же графе возникают при различных данных, которые создают разные логические условия в точках разветвления. Отсутствие сходимости означает, что в процессе вычисления не появляется условий, ведущих к концу, и процесс идет по бесконечному пути (зацикливается).

При всей наглядности языка блок-схем не следует, однако, переоценивать его возможности. Он достаточно груб и отражает связи лишь по управлению (что делать в следующий момент, т. е. какому блоку передать управление), а не по информации (где этому блоку брать исходные данные). Например, рис. 5.2 при сделанной оговорке относительно данных изображает вычисление $f_2(f_1(x))$, однако он же мог изображать последовательное

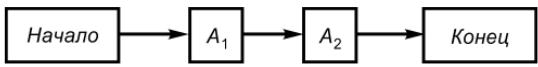


Рис. 5.2

вычисление двух независимых функций $f_1(5)$ и $f_2(100)$, порядок которых несуществен. Блок-схемы не содержат сведений ни о данных, ни о памяти, ни об используемом наборе элементарных шагов. В частности, надо иметь в виду, что если в блок-схеме нет циклов, это еще не значит, что нет циклов в алгоритме (они могут быть в каком-нибудь неэлементарном блоке). По существу блок-схемы — это не язык, а средство, правда, очень удобное, для одной цели — описания детерминизма алгоритма. Оно будет неоднократно использоваться в дальнейшем с одним видоизменением: условия, т. е. точки разветвления, могут быть не только двоичными, но и многозначными; важно лишь, чтобы верным был ровно один из возможных ответов. Примеры таких условий: а) $x < 0$, $x = 0$, $x > 0$; б) $x < 5$, $5 \leq x < 20$, $x = 20$, $x < 20$.

О подходах к уточнению понятия «алгоритм». Ранее были сформулированы основные требования к алгоритмам. Однако понятия, использованные в этих формулировках (такие, как ясность, четкость, элементарность), сами нуждаются в уточнении. Очевидно, что их словесные определения будут содержать новые понятия, которые снова потребуют уточнения, и т. д. Поэтому в теории алгоритмов принимается другой подход: выбирается конечный набор исходных объектов, которые объявляются элементарными, и конечный набор способов построения из них новых объектов. Этот подход был уже использован при обсуждении вопроса о данных: уточнением понятия «данные» в дальнейшем будем считать множества слов в конечных алфавитах. Для уточнения детерминизма будут использоваться либо блок-схемы и эквивалентные им словесные описания (типа того, который приведен в примере 5.1), либо описание механизма реализации алгоритма. Кроме того, нужно зафиксировать набор элементарных шагов и договориться об организации памяти. После того как это будет сделано, получится конкретная алгоритмическая модель.

Алгоритмические модели, рассматриваемые в этой главе, претендуют на право считаться формализацией понятия

«алгоритм». Это значит, что они должны быть универсальными, т. е. допускать описание любых алгоритмов. Поэтому может возникнуть естественное возражение против предлагаемого подхода: не приведет ли выбор конкретных средств к потере общности формализации? Если иметь в виду основные цели, стоявшие при создании теории алгоритмов, — универсальность и связанную с ней возможность говорить в рамках какой-либо модели о свойствах алгоритмов вообще, то это возражение снимается следующим образом. Во-первых, доказывается сводимость одних моделей к другим, т. е. показывается, что всякий алгоритм, описанный средством одной модели, может быть описан средствами другой. Во-вторых, благодаря взаимной сводимости моделей в теории алгоритмов удалось выработать инвариантную по отношению к моделям систему понятий, позволяющую говорить о свойствах алгоритмов независимо от того, какая формализация алгоритма выбрана. Эта система понятий основана на понятии вычислимой функции, т. е. функции, для вычисления которой существует алгоритм. Общее понятие вычислимости и его свойства будут более подробно рассмотрены в § 5.4.

Тем не менее, хотя общность формализации в конкретной модели не теряется, различный выбор исходных средств приводит к моделям разного вида. Можно выделить три основных типа универсальных алгоритмических моделей, различающихся исходными эвристическими соображениями относительно того, что такое алгоритм. Первый тип связывает понятие алгоритма с наиболее традиционными понятиями математики — вычислениями и числовыми функциями. Наиболее развитая и изученная модель этого типа — *рекурсивные функции* — является исторически первой формализацией понятия алгоритма. Второй тип основан на представлении об алгоритме как о некотором детерминированном устройстве, способном выполнять в каждый отдельный момент лишь весьма примитивные операции. Такое представление не оставляет сомнений в однозначности алгоритма и элементарности его шагов. Кроме того, эвристика этих моделей близка к компьютерам и, следовательно, к инженерной интуиции. Основной теоретической моделью этого типа (созданной

в 30-х гг. — раньше появления реальных вычислительных машин!) является машина Тьюринга. Наконец, третий тип алгоритмических моделей — это преобразования слов в произвольных алфавитах, в которых элементарными операциями являются подстановки, т. е. замены куска слова (подслова) другим словом. Преимущества этого типа — в его максимальной абстрактности и возможности применить понятие алгоритма к объектам произвольной (не обязательно числовой) природы. Впрочем, как будет ясно из дальнейшего, модели второго и третьего типа довольно близки (их взаимная сводимость доказывается просто) и отличаются в основном эвристическими акцентами. Примеры моделей этого типа — канонические системы Поста (см. § 6.4) и нормальные алгоритмы Маркова [48].

5.2. МАШИНЫ ТЬЮРИНГА

Основные определения. Машина Тьюринга состоит: 1) из управляющего устройства, которое может находиться в одном из состояний, образующих конечное множество $Q = \{q_1, \dots, q_n\}$; 2) ленты, разбитой на ячейки, в каждой из которых может быть записан один из символов конечного алфавита $A = \{a_1, \dots, a_m\}$; 3) устройства обращения к ленте, т. е.читывающей и пишущей головки, которая в каждый момент времени обозревает ячейку ленты, в зависимости от символа в этой ячейке и состояния управляющего устройства записывает в ячейку символ (быть может, совпадающий с прежним или пустой, т. е. стирает символ), сдвигается на ячейку влево или вправо или остается на месте; при этом управляющее устройство переходит в новое состояние (или остается в старом). Среди состояний управляющего устройства выделены начальное состояние q_1 и заключительное состояние, которое будем обозначать q_z (z здесь понимается не как числовая переменная, а как мнемонический знак конца). В начальном состоянии машина находится перед началом работы; попав в заключительное состояние, машина останавливается.

Таким образом, память машины Тьюринга — это конечное множество состояний (внутренняя память) и лента (внешняя память). Лента бесконечна в обе стороны,

однако в начальный момент времени только конечное число ячеек ленты заполнено непустыми символами, остальные ячейки пусты, т. е. содержат пустой символ λ (пробел). Из характера работы машины следует, что и в любой последующий момент времени лишь конечный отрезок ленты будет заполнен символами. Поэтому важна не фактическая (как говорят в математике, актуальная) бесконечность ленты, а ее неограниченность, т. е. возможность писать на ней сколь угодно длинные, но конечные слова. Данные машины Тьюринга — это слова в алфавите ленты; на ленте записываются и исходные данные, и окончательные результаты. Элементарные шаги машины — это считывание и запись символов, сдвиг головки на ячейку влево и вправо, а также переход управляющего устройства в следующее состояние.

Детерминированность машины, т. е. последовательность ее шагов, определяется следующим образом: для любого внутреннего состояния q_i и символа a_i однозначно заданы: а) следующее состояние q'_i ; б) символ a'_j , который нужно записать вместо a_j в ту же ячейку (стирание символа будем понимать как запись пустого символа λ); в) направление сдвига головки d_k , обозначаемое одним из трех символов: L (влево), R (вправо), E (на месте). Это задание может описываться либо системой правил (команд), имеющих вид

$$q_i a_j \rightarrow q'_i a'_j d_k, \quad (5.1)$$

либо таблицей, строкам которой соответствуют состояния, столбцам — входные символы, а на пересечении строки q_i и столбца a_j записана тройка символов $q'_i a'_j d_k$, и, наконец, блок-схемой, которую будем называть диаграммой переходов. В этой диаграмме состояниям соответствуют вершины, а правилу вида (5.1) — ребро, ведущее из q_i в q'_i , на котором написано $a_j \rightarrow a'_j d_k$. Условие однозначности требует, чтобы для любого j и любого $i \neq z$ в системе команд имелась одна команда, аналогичная (5.1), с левой частью $q_i a_j$; состояние q_z в левых частях команд не встречается. На диаграмме переходов это выражается условием, что из каждой вершины, кроме q_z , выходят ровно m ребер, причем на разных ребрах левые части различны; в вершине q_z нет выходящих ребер.

В дальнейшем договоримся опускать символы q'_i и a'_j , если $q'_i = q_i$, $a'_j = a_j$.

Полное состояние машины Тьюринга, по которому однозначно можно определить ее дальнейшее поведение, определяется ее внутренним состоянием, состоянием ленты (т. е. словом, записанным на ленте) и положением головки на ленте. Полное состояние будем называть *конфигурацией*, или машинным словом, и обозначать тройкой $\alpha_1 q_i \alpha_2$, где q_i — текущее внутреннее состояние, α_1 — слово слева от головки, а α_2 — слово, образованное символом, обозреваемым головкой, и символами справа от него, причем слева от α_1 и справа от α_2 нет непустых символов. Например, конфигурация с внутренним состоянием q_i , в которой на ленте записано $abcde$, а головка обозревает d , запишется как $abcq_i de$. Стартовой начальной конфигурацией назовем конфигурацию вида $q_1 \alpha$, т. е. конфигурацию, содержащую начальное состояние, в которой головка обозревает крайний левый символ слова, написанного на ленте. Аналогично стандартной заключительной конфигурацией назовем конфигурацию вида $q_z \beta$. Ко всякой незаключительной конфигурации K машины T применима ровно одна команда вида (5.1), которая K переводит в конфигурацию K' . Это отношение между конфигурациями обозначим $K \xrightarrow{T} K'$; если из контекста ясно, о какой машине T идет речь, индекс T будем опускать. Если для K_1 и K_n существует последовательность конфигураций K_1, K_2, \dots, K_n , такая, что $K_1 \xrightarrow{T} K_2 \xrightarrow{T} \dots \xrightarrow{T} K_n$, обозначим это $K_1 \xrightarrow{T} K_n$. Например, если в системе команд машины T имеются команды $q_2 a_5 \rightarrow q_3 a_4 R$ и $q_3 a_1 \rightarrow q_4 a_2 L$, то $q_2 a_5 a_1 a_2 \rightarrow a_4 q_3 a_1 a_2 \rightarrow \dots \rightarrow q_4 a_4 a_2 a_2$ и, следовательно, $q_2 a_5 a_1 a_2 \xrightarrow{T} q_4 a_4 a_2 a_2$. Последовательность конфигураций $K_1 \xrightarrow{T} K_2 \xrightarrow{T} K_3 \xrightarrow{T} \dots$ однозначно определяется исходной конфигурацией K_1 и полностью описывает работу машины T , начиная с K_1 . Она конечна, если в ней встретится заключительная конфигурация, и бесконечна в противном случае.

Пример 5.2. а. Машина с алфавитом $A = \{1, \lambda\}$, состояниями $\{q_1, q_2\}$ и системой команд $q_1 1 \rightarrow q_1 1 R$, $q_1 \lambda \rightarrow q_1 1 R$ из любой начальной конфигурации будет работать бесконечно, заполняя единицами всю ленту вправо от начальной точки.

6. Для любой машины T , если $K_1 \xrightarrow{T} K_i \xrightarrow{T} K_j$ и $K_i = K_j$, последовательность $K_1 \xrightarrow{T} K_i \xrightarrow{T} K_j \xrightarrow{T} \dots$ является бесконечной: ее отрезок $K_1 \xrightarrow{T} K_j$ будет повторяться (зацикливаться).

Если $\alpha_1 q_1 \alpha_2 \xrightarrow{T} \beta_1 q_z \beta_2$, то будем говорить, что машина T перерабатывает слово $\alpha_1 \alpha_2$ в слово $\beta_1 \beta_2$, и обозначать это $T(\alpha_1 \alpha_2) = \beta_1 \beta_2$. Запись $T(\alpha)$ иногда будем употреблять и в другом смысле — просто как обозначение машины T с исходными значениями α .

Для того чтобы говорить о том, что могут делать машины Тьюринга, необходимо уточнить, как будет интерпретироваться их поведение и как будут представляться данные. Исходными данными машины Тьюринга будем считать записанные на ленте слова в алфавите исходных данных $A_{\text{исх}}$ ($A_{\text{исх}} \subseteq A$) и векторы из таких слов (словарные векторы над $A_{\text{исх}}$). Это значит, что для каждой машины будут рассматриваться только те начальные конфигурации, в которых на ленте записаны словарные векторы над $A_{\text{исх}}$. Запись на ленте словарного вектора $(\alpha_1, \dots, \alpha_k)$ назовем правильной, если она имеет вид $\alpha_1 * \alpha_2 * \dots * \alpha_{k-1} * \alpha_k$, где $*$ — специальный символ-разделитель (маркер), не входящий в $A_{\text{исх}}$. Для любого вектора $V_{\text{исх}}$ над $A_{\text{исх}}$ машина T либо работает бесконечно, либо перерабатывает его в словарный вектор в алфавите, который назовем алфавитом результатов и обозначим $A_{\text{рез}}$; $A_{\text{исх}}$ и $A_{\text{рез}}$ могут пересекаться и даже совпадать. В ходе работы на ленте могут появляться символы, не входящие в $A_{\text{исх}}$ и $A_{\text{рез}}$ и образующие промежуточный алфавит $A_{\text{пр}}$ (содержащий, в частности, разделитель). Таким образом, алфавит ленты $A = A_{\text{исх}} \cup A_{\text{рез}} \cup A_{\text{пр}}$. В простейшем случае $A_{\text{исх}} = A_{\text{рез}}$ и $A = A_{\text{исх}} \cup \{\lambda\}$.

Пусть f — функция, отображающая множество векторов над $A_{\text{исх}}$ в множество векторов над $A_{\text{рез}}$. Машина T правильно вычисляет функцию f , если: 1) для любых V и W , таких, что $f(V) = W$,

$$q_1 V^* \xrightarrow{T} q_z W^*,$$

где V^* и W^* — правильные записи V и W соответственно; 2) для любого V , такого, что $f(V)$ не определена, машина T , запущенная в стандартной начальной конфигурации $q_1 V^*$, работает бесконечно. Если для f существует машина T ,

которая ее правильно вычисляет, функция f называется *правильно вычислимой по Тьюрингу*.

С другой стороны, всякой правильно вычисляющей машине Тьюринга, т. е. машине, которая, начав со стандартной начальной конфигурации $q_1\alpha$, может остановиться только в стандартной заключительной конфигурации $q_z\beta$, можно поставить в соответствие вычисляемую ей функцию. Две машины Тьюринга с одинаковым алфавитом $A_{\text{исх}}$ будем называть эквивалентными, если они вычисляют одну и ту же функцию. В частности, машины из примеров 5.2 эквивалентны, так как они вычисляют одну и ту же нигде не определенную (пустую) функцию.

Пример 5.3. Если машина T содержит команды $q_i a_j \rightarrow q'_i a'_j E$ и $q'_i a'_j \rightarrow q''_i a''_j d_k$, то, заменив эти две команды командой $q_i a_j \rightarrow q''_i a''_j d_k$, получим машину T' , эквивалентную T . Путем таких преобразований можно в машине T убрать все команды, содержащие E , для случая, когда q'_i — незаключительное состояние; при этом может сократиться число состояний (некоторые q'_i не войдут в правые части новых команд и станут недостижимыми из q_1). Если q'_i — заключительное состояние, то, введя новое заключительное состояние q_{n+1} и заменив команду $q_i a_j \rightarrow q'_i a'_j E$ на $m + 1$ команду $q_i a_j \rightarrow q'_i a'_j R$, $q'_i a'_j \rightarrow q_{n+1} a_1 L$, ..., $q'_i a_m \rightarrow q_{n+1} a_m L$ (m — число букв в A), получим машину, также эквивалентную T . Таким образом, для любой машины T существует эквивалентная ей машина, не содержащая в командах E ; поэтому можно рассматривать машины, головки которых на каждом шаге движутся.

Определения, связанные с вычислением функций, заданных на словарных векторах, даны с явным «запасом общности» и имеют в виду переработку нечисловых объектов. В дальнейшем это понадобится; однако в ближайшее время будут рассматриваться числовые функции, точнее, функции, отображающие N в N . Договоримся представлять натуральные числа в единичном (*унарном*) коде, т. е. для всех числовых функций $A_{\text{исх}} = \{1\}$ либо $A_{\text{исх}} = \{1, *\}$ и число x представляется словом $1\dots 1 = 1^x$, состоящим из x единиц. Таким образом, числовая функция $f(x_1, \dots, x_n)$ правильно вычислнима по Тьюрингу, если существует машина T , такая, что

$$q_1 1^{x_1} * 1^{x_2} * \dots * 1^{x_n} \xrightarrow[T]{} q_z 1^y$$

когда $f(x_1, \dots, x_n) = y$, и T работает бесконечно, начиная с $q_1 1^{x_1} * 1^{x_2} * \dots * 1^{x_n}$, когда $f(x_1, \dots, x_n)$ не определена.

Начав с довольно общих определений абстрактных машин, мы затем ввели при определении вычислений на этих машинах ряд ограничений, связанных с правильной записью, правильным вычислением, представлением чисел в весьма неэкономном единичном коде и т. д. Не теряется ли при этом общность? Действительно, возможны другие определения вычисления: например, можно в заключительной конфигурации допускать символы из промежуточного алфавита $A_{\text{пр}}$, а результатом считать слово в $A_{\text{рез}}$, которое получится, если символы из $A_{\text{пр}}$ выкинуть и «сдвинуть» оставшиеся куски. В частности, если $A_{\text{рез}} = \{1\}$, то результатом будет число, равное числу единиц на ленте в заключительной конфигурации. Оказывается, что потери общности не происходит; в частности, если функция вычислима в последнем смысле, то она правильно вычислима. Не будем заниматься детальным доказательством этого утверждения в общем виде, однако проиллюстрируем его справедливость на примерах (см. далее пример 5.7). Поэтому обычно прилагательное «правильный» будем опускать и говорить просто о функциях, вычислимых по Тьюрингу.

Пример 5.4. а. Сложение. Во введенном ранее представлении чисел сложить числа a и b — это значит слово $1^a * 1^b$ переработать в слово 1^{a+b} , т. е. удалить разделитель $*$ и сдвинуть одно из слагаемых, скажем первое, к другому. Это преобразование осуществляется машина T_+ с четырьмя состояниями и следующей системой команд (первая команда введена для случая, когда $a = 0$ и исходное слово имеет вид $*1^b$):

$$\begin{aligned} q_1 * &\rightarrow q_z \lambda R; \\ q_1 1 &\rightarrow q_2 \lambda R; \\ q_2 1 &\rightarrow q_2 1 R; \\ q_2 * &\rightarrow q_3 1 L; \\ q_3 1 &\rightarrow q_3 1 L; \\ q_3 \lambda &\rightarrow q_z \lambda R. \end{aligned}$$

В этой системе команд перечислены не все сочетания состояний машины и символов ленты: опущены те из

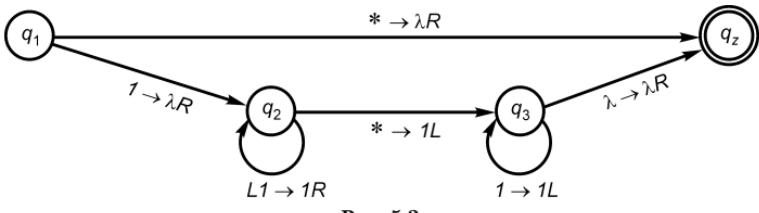


Рис. 5.3

них, которые при стандартной начальной конфигурации никогда не встречаются. Опускать ненужные команды будем в дальнейшем; в таблицах это будет отмечено прочерками. Диаграмма переходов T_+ приведена на рис. 5.3; заключительное состояние отмечено двойным кружком.

б. Копирование (перезапись) слова, т. е. переработка слова α в $\alpha * \alpha$. Для чисел эту задачу решает машина $T_{\text{коп}}$, система команд которой приведена в табл. 5.1.

Таблица 5.1

	1	λ	*	0
q_1	$q_2 0R$	$q_2 \lambda R$	$q_1 * L$	$q_1 1L$
q_2	$q_2 1R$	$q_3 * R$	$q_3 * R$	
q_3	$q_3 1R$	$q_4 1L$		
q_4	$q_4 1L$		$q_4 * L$	$q_1 0R$

Диаграмма переходов $T_{\text{коп}}$ дана на рис. 5.4. На этой диаграмме (а также последующих) приняты сокращения: 1) если из q_i в q_j ведут два ребра с одинаковой правой частью, то они объединяются в одно ребро, на котором левые части записаны через запятую; 2) если символ на ленте не изменяется, то он в правой части команды не пишется. На петле в q_4 использованы одновременно оба сокращения.

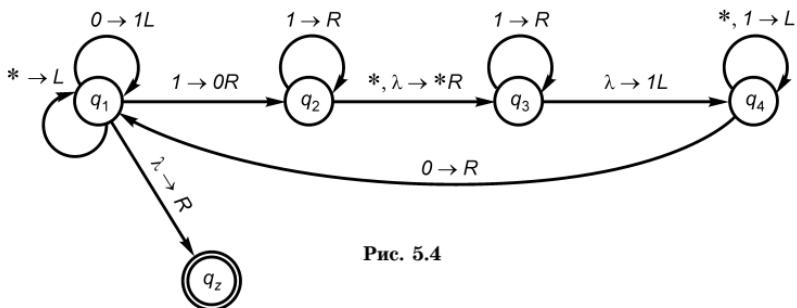


Рис. 5.4

Машине $T_{\text{коп}}$ при каждом проходе исходного числа 1^a заменяет левую из его единиц нулем и пишет (в состоянии q_3) одну единицу справа от 1^a в ближайшую пустую клетку. При первом проходе, кроме того, в состоянии q_2 ставится маркер. Таким образом, копия 1^a строится за a проходов. После записи очередной единицы машина переходит в состояние q_4 , которое передвигает головку влево от ближайшего нуля, после чего машина переходит в q_1 и цикл повторяется. Он прерывается, когда q_1 обнаруживает на ленте не единицу, а маркер. Это значит, что все единицы 1^a исчерпаны, т. е. сделано a проходов. Тогда головка возвращается влево в свое исходное положение, заменяя по дороге все нули единицами.

Некоторые операции над машинами Тьюринга. Работа машины Тьюринга полностью определяется исходными данными и системой команд. Однако для понимания того, как конкретная машина решает данную задачу, как правило, возникает потребность в содержательных пояснениях типа тех, которые приводились для машины $T_{\text{коп}}$. Эти пояснения часто можно сделать более формальными и точными, если использовать блок-схемы и некоторые операции над машинами Тьюринга.

Напомним, что композицией двух функций $f_1(x)$ и $f_2(y)$ называется функция $g_1(x) = f_2(f_1(x))$, которая получается при применении f_2 к результату вычисления f_1 . Для того чтобы $g(x)$ была определена при данном x , необходимо и достаточно, чтобы f_1 была определена на x и f_2 была определена на $f_1(x)$.

Теорема 5.1. Если $f_1(x)$ и $f_2(y)$ вычислимы по Тьюрингу, то их композиция $f_2(f_1(x))$ также вычислена по Тьюрингу.

Пусть T_1 — машина, вычисляющая f_1 , а T_2 — машина, вычисляющая f_2 , и множества их состояний соответственно $Q_1 = \{q_{11}, \dots, q_{1n_1}\}$ и $Q_2 = \{q_{21}, \dots, q_{2n_2}\}$. Построим диаграмму переходов машины T из диаграмм T_1 и T_2 следующим образом: отождествим начальную вершину q_{21} диаграммы машины T_2 с конечной вершиной q_{1z} диаграммы машины T_1 (для систем команд это равносильно тому, что систему команд T_2 приписываем к системе команд T_1 и при этом q_{1z} в командах T_1 заменяем на q_{21}). Получим диаграмму с $n_1 + n_2 - 1$ состояниями. Началь-

ным состоянием T объявим q_{11} , а заключительным — q_{2z} . Для простоты обозначений будем считать f_1 и f_2 числовыми функциями одной переменной.

Пусть $f_2(f_1(x))$ определена. Тогда

$$T_1(1^x) = 1^{f_1(x)} \text{ и } q_{11}1^x \xrightarrow{T_1} q_{1z}1^{f_1(x)}.$$

Машина T пройдет ту же последовательность конфигураций с той разницей, что вместо $q_{1z}1^{f_1(x)}$ она перейдет в $q_{21}1^{f_1(x)}$. Эта конфигурация является стандартной начальной конфигурацией для машины T_2 , поэтому

$$q_{21}1^{f_1(x)} \xrightarrow{T_2} q_{2z}1^{f_2(f_1(x))}.$$

Но так как все команды T_2 содержатся в T , то

$$q_{11}1^x \xrightarrow{T} q_{21}1^{f_1(x)} \xrightarrow{T_2} q_{2z}1^{f_2(f_1(x))}$$

и, следовательно, $T(1^*) = 1^{f_2(f_1(x))}$. Если же $f_2(f_1(x))$ не определена, то T_1 или T_2 не остановится и, следовательно, машина T не остановится. Итак, машина T вычисляет $f_2(f_1(x))$. \square

Построенную таким образом машину T будем называть *композицией* машин T_1 и T_2 и обозначать $T_2(T_1)$, а также изображать блок-схемой (рис. 5.5). Эта блок-схема имеет более точный смысл, чем изображенная на рис. 5.2, так как она всегда предполагает, что исходными данными машины T_2 являются результаты T_1 . При этом они уже «готовы к употреблению», так как благодаря правильной вычислимости (которая существенна при композиции) заключительная конфигурация T_1 легко превращается в стандартную начальную конфигурацию T_2 .

Поскольку машина Тьюринга вектор над $A_{\text{исх}}$ воспринимает как слово в алфавите $A_{\text{исх}} \cup \{\ast\}$, определение композиции и теорема 5.1 остаются в силе, если T_1 и T_2 вычисляют функции от нескольких переменных. Важно лишь, чтобы данные для T_2 были в обусловленном виде подготовлены машиной T_1 . Это видно на следующем примере.

Пример 5.5. Машина, диаграмма которой приведена на рис. 5.6, — это машина $T_+(T_{\text{коп}})$. Она вычисляет функцию $f(x) = 2x$ для $x \neq 0$; при этом машина $T_{\text{коп}}$ строит

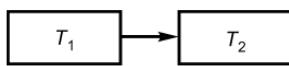


Рис. 5.5

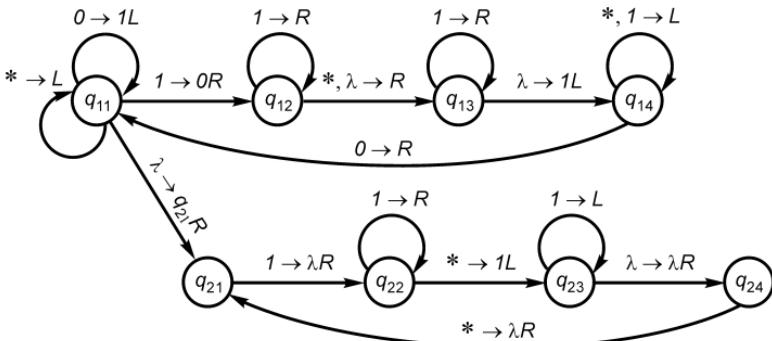


Рис. 5.6

двухкомпонентный вектор, а T_+ вычисляет функцию от двух переменных.

Для удобства последующих построений установим следующий важный факт. Оказывается, что для вычисления на машине Тьюринга достаточно, чтобы лента была бесконечной только в одну сторону, например вправо. Такая лента называется правой полулентой.

Теорема 5.2. Любая функция, вычислимая по Тьюрингу, вычислима на машине Тьюринга с правой полулентой; иначе говоря, для любой машины Тьюринга T существует эквивалентная ей машина T_R с правой полулентой. Аналогичная теорема формулируется для левой полуленты.

Можно предложить по крайней мере две идеи построения такой эквивалентной машины: 1) всякий раз, когда головке прежней машины надо зайти за левый край ленты, новая машина предварительно сдвигнет все написанное (вместе с головкой) на клетку вправо; б) лента «перегибается пополам»; при этом ячейки правой половины прежней ленты размещаются, скажем, в нечетных ячейках полуленты, а ячейки левой половины — в четных. Первая идея реализована в [32]; реализация второй предлагается как упражнение. □

Рассмотрим теперь вычисление предикатов на машинах Тьюринга. Машина T вычисляет предикат $P(\alpha)$ (α — слово в $A_{\text{исх}}$), если $T(\alpha) = \omega$, где $\omega = \text{И}$, когда $P(\alpha)$ истинно, и $\omega = \text{Л}$, когда $P(\alpha)$ ложно. Если же $P(\alpha)$ не определен, то машина T , как и при вычислении функций, не останавливается. При обычном вычислении предиката

уничтожается α , что может оказаться неудобным, если после T должна работать другая машина. Поэтому введем понятие вычисления с восстановлением: машина T вычисляет $P(\alpha)$ с восстановлением, если $T(\alpha) = \omega\alpha$. Если существует машина T , вычисляющая $P(\alpha)$, то существует и \tilde{T} , вычисляющая $P(\alpha)$ с восстановлением. Действительно, вычисление с восстановлением можно представить следующей последовательностью конфигураций: $q_1\alpha \Rightarrow q_{n_1}\alpha * \alpha \Rightarrow \alpha * q_{n_2}\alpha \Rightarrow \alpha * q_{n_3} \circledast \Rightarrow q_{n_4} \omega\alpha$. Первая часть этой последовательности реализуется машиной $T_{\text{коп}}$, вторая — простым сдвигом головки до маркера $*$, третья — машиной T_R , вычисляющей $P(\alpha)$ на правой полуленте (одного копирования мало для восстановления, так как копию может испортить основная машина T ; нужна машина, не заходящая влево от α), четвертая — переносом ω в крайнее левое положение. Машина T является композицией указанных четырех машин. Однако в конкретных случаях возможны и более простые способы восстановления α .

Пример 5.6. Машина с диаграммой на рис. 5.7 вычисляет предикат « α — четное число»: головка достигает конца числа в состоянии q_2 , если число единиц четно, и в состоянии q_3 , если число единиц нечетно, после чего она перемещается в исходное положение в состоянии q_4 либо q_5 и печатает И либо Л соответственно. Для того чтобы этот предикат вычислялся с восстановлением, достаточно в петлях q_4 и q_5 не стирать, а сохранять единицы, т. е. заменить команды $1 \rightarrow \lambda L$ на команды $1 \rightarrow L$.

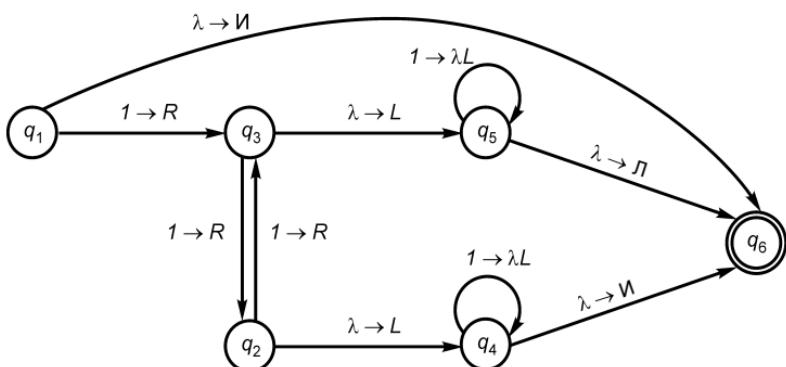


Рис. 5.7

Так как машина Тьюринга с алфавитом $A_{\text{исх}} = \{\text{И}, \text{Л}\}$ и командами $q_1\text{И} \rightarrow q_z\text{ЛЕ}$ и $q_1\text{Л} \rightarrow q_z\text{ИЕ}$ вычисляет отрицание логической переменной, то из вычислимости всюду определенного $P(\alpha)$ следует вычислимость $\bar{P}(\alpha)$.

Пусть функция $f(\alpha)$ задана описанием: «если $P(\alpha)$ истинно, то $f(\alpha) = g_1(\alpha)$, иначе $f(\alpha) = g_2(\alpha)$ » (под «иначе» имеется в виду «если $P(\alpha)$ ложно»; если же $P(\alpha)$ не определен, то $f(\alpha)$ также не определена). Функция $f(\alpha)$ называется *разветвлением* или *условным переходом к $g_1(\alpha)$ и $g_2(\alpha)$* по условию $P(\alpha)$.

Теорема 5.3. Если $g_1(\alpha)$, $g_2(\alpha)$ и $P(\alpha)$ вычислимы по Тьюрингу, то разветвление g_1 и g_2 по P также вычислимо.

Пусть T_1 — машина с состояниями $q_{11}, q_{12}, \dots, q_{1n_1}$ и системой команд Σ_1 , вычисляющая g_1 ; T_2 — машина с состояниями $q_{21}, q_{22}, \dots, q_{2n_2}$ и системой команд Σ_2 , вычисляющая g_2 ; T_P вычисляет с восстановлением $P(\alpha)$. Тогда машина T , вычисляющая разветвление g_1 и g_2 по P , — это композиция T_P и машины T_3 , система команд Σ_3 которой имеет следующий вид:

$$\begin{aligned}\Sigma_3 = \Sigma_1 \cup \Sigma_2 \cup \{ &q_{31}\text{И} \rightarrow \lambda q_{11}R, q_{31}\text{Л} \rightarrow \\ &\rightarrow \lambda q_{21}R, q_{1z} \rightarrow q_{2z}E \}.\end{aligned}$$

Первые две из новых команд передают управление системам команд Σ_1 или Σ_2 в зависимости от значения предиката $P(\alpha)$. Третья команда введена для того, чтобы T_3 имела одно заключительное состояние q_{2z} . Отсутствие символа ленты в этой команде означает, что она выполняется при любом символе. \square

Разветвление встречается в структуре диаграмм переходов многих машин Тьюринга.

Пример 5.7. а. В примере 5.4, а было описано построение машины T_+ для сложения двух чисел. На рис. 5.8 приведена диаграмма машины T_{++} для сложения n чисел ($n = 1, 2, \dots$). Цикл из состояний q_1, q_2, q_3 — это «зацикленная» машина T_+ , в которой заключительное состояние совмещено с начальным.

Сумма, полученная на очередном цикле, является первым слагаемым следующего цикла. Состояние q_4 реализует разветвление. В нем проверяется условие — есть ли второе слагаемое. Если да (о чём говорит наличие маркера *), то происходит переход к следующему циклу;

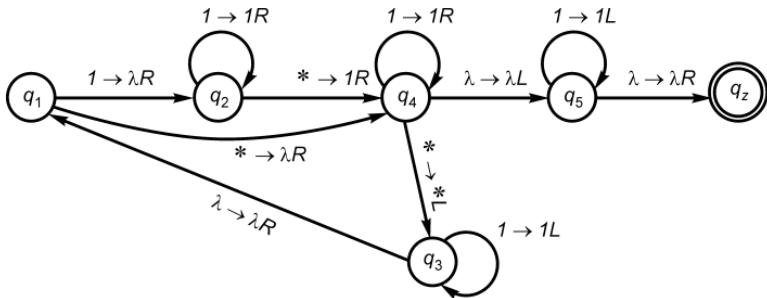


Рис. 5.8

если нет (о чём говорит λ после единиц), то машина выходит из цикла.

б. Пусть машина T с алфавитом A_T и с $A_{\text{исх}} = \{1\}$ не является правильно вычисляющей и ее заключительная конфигурация стандартна, но может содержать любые символы из A_T (кроме пробелов), а результат интерпретируется как число, равное числу единиц на ленте. Построим для T машину T'_{++} , которая работает как T_{++} , а все символы, кроме 1 и λ , она воспринимает как маркеры, т. е. команды для них — те же, что и для *. Тогда T'_{++} соберет все единицы в один массив и, следовательно, $T'_{++}(T(\alpha))$ правильно вычисляет функцию, вычисляемую машиной T .

Пример 5.7, а иллюстрирует важный частный случай разветвления — выход из цикла по условию, хорошо известный в программировании. В словесных описаниях (см. пример 5.1) и во многих языках программирования этот случай формулируется так: повторять вычисление f_1 до тех пор, пока истинно условие P ; если P ложно, перейти к вычислению f_2 .

Благодаря вычислимости композиции и разветвления словесные описания и язык блок-схем можно сделать вполне точным языком для описания работы машин Тьюринга. Каждый блок — это множество состояний, в котором выделены начальное и заключительное состояния, и система команд. Правильное вычисление для промежуточных блоков не обязательно, поэтому при стыковке блоков важно согласовывать всю конфигурацию. Переход к блоку — это обязательно переход в его начальное состояние. Машина Тьюринга, описываемая блок-схемой, — это объединение состояний и команд,

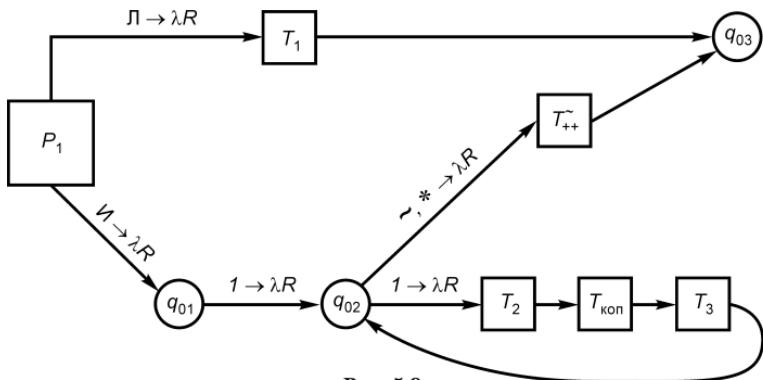


Рис. 5.9

содержащихся во всех блоках. В частности, блоком может быть одно состояние. Блоки, вычисляющие предикаты, обозначим буквой P .

Пример 5.8. На рис. 5.9 приведена блок-схема машины Тьюринга T_x , осуществляющей умножение двух чисел: $T_x(1^a * 1^b) = 1^{ab}$. Ее заключительное состояние — q_{03} . Блоки реализуют следующие указания и вычисления:

P_1 — вычислить с восстановлением предикат «оба слагаемых больше нуля»;

T_1 — стереть все непустые символы справа;

T_2 — установить головку у ячейки, следующей (вправо) за маркером *; маркер * заменить на ~;

$T_{\text{коп}}$ — см. пример 5.4, б, в котором команду $q_1 \lambda \rightarrow q_z R$ надо заменить на команду $q_1 \sim \rightarrow q_z R$;

$T_{\text{коп}}(a - 1)$ раз копирует 1^b ; после i -го цикла она останавливается в конфигурации $1^{a-i-1}(\sim 1^b)^{i-1} \sim q_1 b * 1^b$;

T_3 — вернуть головку к крайнему слева непустому символу. После $(a - 1)$ -го цикла этим символом окажется ~ (или *, если $a = 1$), и происходит выход из цикла и переход к T_{++} .

T_{++} работает как T_{++} (см. пример 5.7, а) с той разницей, что числа, которые она суммирует, разделены двумя видами маркеров: ~ и *; для этой цели в нее к командам с маркером * в левой части добавлены такие же команды для маркера ~.

Аналогично тому, как из машины T_+ была построена T_{++} , можно из T_x построить машину T_{xx} , которая перемножает несколько чисел, и по схеме, аналогичной приведенной на рис. 5.9, с использованием T_{xx} вместо T_{++} .

построить машину, осуществляющую возведение в степень. Другой важный пример, который читатель может либо построить, либо найти в [22, 32], — перевод унарного представления чисел в позиционное — двоичное или десятичное.

Универсальная машина Тьюринга. Систему команд машины Тьюринга можно интерпретировать и как описание работы конкретного механизма, и как программу, т. е. совокупность предписаний, однозначно приводящих к результату. При разборе примеров любой читатель невольно принимает вторую интерпретацию, выступая в роли механизма, который способен воспроизвести работу любой машины Тьюринга. Уверенность в том, что все это будут делать одинаково (если не наделяют ошибок, что, кстати, предполагается и при работе машины Тьюринга), — это по существу уверенность в существовании алгоритма воспроизведения работы машины Тьюринга по заданной программе, т. е. системе команд. Действительно, словесное описание такого алгоритма дать нетрудно. Его основное действие, циклически повторяющееся, состоит в следующем.

«Для текущей конфигурации $\alpha_1 a_k q_i a_j \alpha_2$ найти в системе команд команду с левой частью $q_i a_j$. Если правая часть этой команды имеет вид $q'_i a'_j R$, то заменить в текущей конфигурации $q_i a_j$ на $a'_i q'_j$ (получится конфигурация $\alpha_1 a_k a'_i q'_j \alpha_2$); если же правая часть имеет вид $q'_i a'_j L$, то заменить $a_k q_i a_j$ на $q'_i a'_j$ (как видно из примера 5.3, случай с E можно не рассматривать)».

Как уже говорилось в § 5.1, словесное описание алгоритма может быть неточным и нуждается в формализации. Поскольку в качестве такой формализации понятия алгоритма сейчас обсуждается машина Тьюринга, то естественно поставить задачу построения машины Тьюринга, реализующей описанный алгоритм воспроизведения. Для машин Тьюринга, вычисляющих функции от одной переменной, формулировка этой задачи такова: построить машину Тьюринга U , вычисляющую функцию от двух переменных, и такую, что для любой машины T с системой команд Σ_T $U(\Sigma_T, \alpha) = T(\alpha)$, если $T(\alpha)$ определена (т. е. если машина T останавливается при исходных данных α), и $U(\Sigma_T, \alpha)$ не останавливается, если

$T(\alpha)$ не останавливается. Любую машину U , обладающую указанным свойством, будем называть *универсальной машиной Тьюринга*. Нетрудно обобщить эту формулировку на любое число переменных.

Первая проблема, возникающая при построении универсальной машины U , связана с тем, что U , как и любая другая машина Тьюринга, должна иметь фиксированный алфавит A_U и фиксированное множество состояний Q_U . Поэтому систему команд Σ_T и исходные данные произвольной машины T нельзя просто переписать на ленту машины U (всегда найдется машина T , алфавиты A_T и Q_T которой превосходят по мощности A_U , Q_U или просто не совпадают с ними). Выход заключается в том, чтобы символы из A_T и Q_T кодировать словами в алфавите A_U . Пусть $|A_T| = m_T$, $|Q_T| = n_T$. Будем всегда считать, что $a_1 = 1$, $a_{m_T} = \lambda$ (эти два символа всегда есть в алфавите любой машины, работающей с числами). Обозначим коды q_i и a_j через $S(q_i)$ и $S(a_j)$ и определим их следующим образом: $S(\lambda) = \lambda^{m_T}$, для любого другого a_j $S(a_j) = a\lambda^{m_T - j - 1}1^j$, для заключительного состояния q_{Tz} $S(q_{Tz}) = q\lambda^{n_T - 1}$, $S(q_i) = q\lambda^{n_T - i - 1}1^i$, если $i \neq n_T$. Код $S(a_j)$ для данной машины T всегда имеет длину (формат) m_T , а код $S(q_i)$ — формат n_T . Символы R , L , \rightarrow введем в A_U , т. е. $S(R) = R$, $S(L) = L$, $S(\rightarrow) = \rightarrow$. Код слова α , образованный кодами символов, составляющих это слово, обозначим $S(\alpha)$. Таким образом, окончательное уточнение постановки задачи об универсальной машине U сводится к тому, что для любой машины T и слова α в алфавите A_T $U(S(\Sigma_T), S(\alpha)) = T(\alpha)$.

План построения машины U таков. Будем считать, что машина T всегда работает на правой полуленте*. Тогда ленту U можно разделить на две бесконечные полуленты с границей Z между ними: в правой полуленте записывается текущая конфигурация машины T (в силу нашего предположения конфигурация машины T при этой имитации никогда не зайдет на левую полу-

* Строго говоря, это предположение нарушает общность рассуждений, несмотря на теорему 5.2: ведь на произвольной системе команд Σ_T не написано, работает она с правой полулентой или нет. Однако теорема 5.2 содержит алгоритм преобразования в систему команд для работы с правой полулентой. Исходя из него, можно построить машину Тьюринга $T_{\text{пр}}$, реализующую этот алгоритм, и рассматривать универсальную машину $U(T_{\text{пр}}(\Sigma_T), \alpha)$.

ленту), а в левой полуленте записан код системы команд $S(\Sigma_T)$. В частности, начальная конфигурация U имеет вид $u_1 S(\Sigma_T) Z S(q_1) S(\alpha)$ (u_1 — начальное состояние машины U , q_1 — начальное состояние T , α — исходное слово T). Например, начальной конфигурации машины из примера 5.2, а с исходным словом 11 соответствует следующее слово на ленте машины U :

$$q1a1 \rightarrow q1a1Rq1\lambda\lambda \rightarrow q1a1RZq1a1a1.$$

Выполнению одной команды T соответствует цикл машины U , реализующей основное действие, описанное ранее, с той разницей, что оно будет осуществляться не над конфигурацией K машины T , а над ее кодом $S(K)$. Благодаря выбранному кодированию длины всех слов вида $S(q_i a_j)$ и $S(a_j q_i)$ равны, поэтому при заменах $S(q_i a_j)$ на $S(a'_j q_i)$ или $S(a_k q_i)$ на $S(q'_i a'_j)$, имитирующих движение головки машины T , окрестность заменяемых слов не затрагивается и никаких сдвигов или раздвигов не требуется. Сама длина заменяемого слова равна числу символов между \rightarrow и ближайшим символом движения (R или L) на левой полуленте.

Опишем более подробно работу машины U , разбив описание на шаги (блоки).

1. Для слова $S(q_i, a_j)$ на правой полуленте (его начало — единственный на правой полуленте символ q , а конец отстоит от q на число символов, равное формату $S(q_i a_j)$) выяснить, имеется ли в левой полуленте слово $S(q_i a_j) \rightarrow$. Если да, то перейти к шагу 2. Если такого слова нет, то перейти к шагу 10.

2. В найденном слове $S(q_i a_j)$ заменить \rightarrow на $*$.

3. Выяснить, равен ли R ближайший символ движения вправо от *. Если да, перейти к шагу 4; если нет, то к шагу 7.

4. В m_T ячеек правой полуленты, начинающихся с q , записать слово длины m_T , начинающееся с $(n_T + 1)$ -го символа правее * (это слово имеет вид $S(a'_j)$ и является кодом символа, который машина T печатает по команде $q_i a_j \rightarrow q'_i a'_j R$). После записанного слова напечатать метку $\|$.

5. В n_T ячеек правой полуленты, начинающихся с $\|$, записать слово длины n_T , начинающееся непосредственно

справа от A (это слово имеет вид $S(q'_i)$ и является кодом состояния, в которое машина T переходит по команде $q_i a_j \rightarrow q'_i a'_j R$).

В результате шагов 4 и 5 слово $S(q_i a_j)$ на правой полуленте заменяется словом $S(a'_j q'_i)$, что имитирует команду $q_i a_j \rightarrow q'_i a'_j R$.

6. Заменить в правой полуленте $*$ на \rightarrow и перейти к шагу 1.

7. Слово длины m_T , расположенное на правой полуленте непосредственно левее q , сдвинуть вправо на n_T клеток. После сдвинутого слова напечатать метку $\|$.

8. В m_T ячеек правой полуленты, начинающихся с $\|$, записать слово длины m_T , начинающееся с $(n_T + 1)$ -го символа правее $*$ (это слово имеет вид $S(a'_j)$ и является кодом символа, который машина T печатает по команде $q_i a_j \rightarrow q'_i a'_j L$). В клетке, расположенной на $m_T + n_T$ клеток левее начала записанного слова, напечатать $\|$.

9. В n_T ячеек правой полуленты, начинающихся с $\|$, записать слово длины n_T , начинающееся непосредственно справа от $*$ (это слово имеет вид $S(q'_i)$ и является кодом состояния, в которое машина T переходит по команде $q_i a_j \rightarrow q'_i a'_j L$). Перейти к шагу 6.

В результате шагов 7, 8, 9 слово $S(a_k q_i a_j)$ на правой полуленте заменяется словом $S(q'_i a_k a'_j)$, что имитирует команду $q_i a_j \rightarrow q'_i a'_j L$.

10. Стереть в правой полуленте код состояния (это состояние будет заключительным для T , так как переход к шагу 10 означает, что состояние машины T не встретилось в левых частях команд машины T) и остановиться. При этом заключительная конфигурация будет иметь вид $S(\Sigma_T)Z\lambda\dots\lambda u_z\beta$, где u_z — заключительное состояние U , а β — код результата T , т. е. $\beta = S(T(\alpha))$.

Блок-схема U , соответствующая этому описанию, приведена на рис. 5.10. Ее цикл реализует основное действие алгоритма воспроизведения работы машины T , причем ветвь цикла 3–4–5–6 соответствует сдвигу головки T вправо, а ветвь 3–7–8–9–6 — сдвигу головки влево.

Приведенное ранее описание работы машины U внешне ничем не отличается от словесного описания в примере 5.1, однако в действительности оно гораздо точнее. В нем полностью уточнены представление данных и их

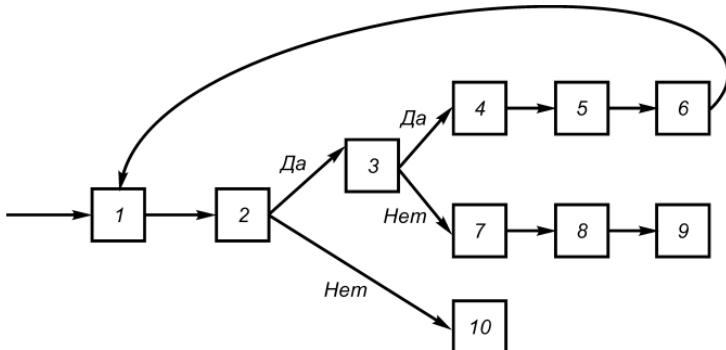


Рис. 5.10

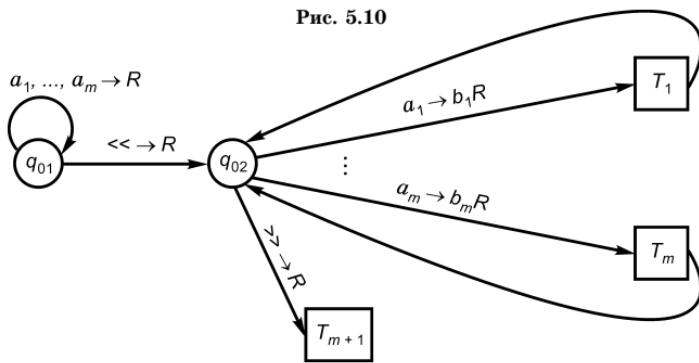


Рис. 5.11

расположение в памяти. Для того чтобы сделать это описание совершенно точным, т. е. превратить в систему команд машины U , остается показать, что блоки этого описания могут быть реализованы машинами Тьюринга. Для шагов 2, 3, 6, 10 это очевидно. Для реализации остальных шагов построим сначала машину $T_{\text{пер}}$, которая слово, расположенное между метками « и », переписывает на место, начинающееся с метки || (предполагается, что метки «, », || встречаются на ленте по одному разу, причем || лежит правее »). Блок-схема машины $T_{\text{пер}}$ с алфавитом $A_{\text{исх}} = \{a_1, \dots, a_m, «, », ||\}$ и m вспомогательными символами b_1, \dots, b_m приведена на рис. 5.11.

Все машины T_1, \dots, T_m имеют по пять состояний. В случае, если первый еще не переписанный символ левого слова равен a_i , он заменяется на b_i , т. е. помечается как переписанный, и управление передается машине T_i , которая идет вправо, проходит все уже выписанные символы (т. е. символы b_j) правого слова, справа (а на первом

	a_1, \dots, a_m	\gg	$ $	b_1, \dots, b_m
q_{i1}	$q_{i1}R$	$q_{i1}R$	$q_{i3}b_iL$	$q_{i2}R$
q_{i2}	$q_{i3}b_iL$	—	—	$q_{i2}R$
q_{i3}	$q_{i3}L$	$q_{i4}L$	—	$q_{i3}L$
q_{i4}	$q_{i4}L$	—	—	$q_{i2}R$

проходе — вместо метки $||$) от них пишет символ b_i и возвращается влево к первому еще не переписанному символу левого слова. Если этим символом оказывается метка $,$, то перезапись окончена и управление передается машине T_{m+1} , которая стирает все отметки, т. е. заменяет b_i на a_i , и останавливается. Система команд машины T_i ($i = 1, \dots, m$) приведена в табл. 5.2, содержащей $2m + 2$ столбца. Система команд T_{m+1} очевидна и здесь не приводится.

Шаги 4, 5, 8, 9 реализуются непосредственно с помощью машины $T_{\text{пер}}$, работе которой должна предшествовать расстановка меток $<$, $,$. Положение этих меток отсчитывается от исходной метки $*$, поставленной на шаге 2, с помощью форматов m_T и n_T , величина которых извлекается из кода системы команд, точнее — из левой части кода самой правой команды: состояние в ней не может быть заключительным и имеет код вида $q\lambda^{n_T-i-1}1^i$, поэтому число символов (клеток) между вторым справа символом движения и концом массива единиц равно n_T . Оставшееся число символов до стрелки \rightarrow равно m_T .

Шаг 1 реализуется с помощью зацикливания модифицированной машины $T'_{\text{пер}}$, в которой слово, отмеченное метками $<$, $,$, находится правее метки $||$, а машины T_i работают справа налево и при этом не переписывают, а сравнивают, т. е. проверяют, не равен ли a_i первый из неотмеченных символов левого слова. Работа $T'_{\text{пер}}$ продолжается до тех пор, пока либо этим символом окажется \rightarrow (это означает, что нужная команда найдена), либо до $a_j \neq a_i$, после чего метка $||$ передвигается в начало следующей влево команды и снова включается $T'_{\text{пер}}$.

Наконец, шаг 7 реализуется с помощью другой модификации $T''_{\text{пер}}$, в которой разрешается, чтобы метка $||$ находилась между $<$ и $,$. Ее построение предоставляется читателю.

На этом построение универсальной машины U заканчивается.

Отметим, что при построении U (как, впрочем, и для многих других машин, описанных в этом параграфе) мы не преследовали целей оптимизации и не жалели ни символов ленты, ни состояний, стремясь к наглядности построения. Нетрудно показать, — а инженер-дискретчик, привыкший к двоичному кодированию, легко в это поверит — что можно построить машину U всего с двумя символами на ленте. Шенон установил менее очевидный факт — он построил универсальную машину с двумя состояниями. В то же время показано (Бобру и Минский), что универсальная машина с двумя состояниями и двумя символами невозможна. Вообще в определенных пределах уменьшение числа символов U ведет к увеличению числа состояний, и наоборот. Подробная сводка результатов о минимальных универсальных машинах приведена в [22].

Существование универсальной машины Тьюринга означает, что систему команд Σ_T любой машины T можно интерпретировать двояким образом: либо как описание работы конкретного устройства машины T , либо как программу для универсальной машины U . Для современного инженера, проектирующего систему управления, это обстоятельство вполне естественно. Он хорошо знает, что любой алгоритм управления может быть реализован либо аппаратурно — построением соответствующей схемы, либо программно — написанием программы для универсального управляющего компьютера. Однако важно сознавать, что сама идея универсального алгоритмического устройства совершенно не связана с развитием современных технических средств его реализации (электроники, физики твердого тела и т. д.) и является не техническим, а математическим фактом, описываемым в абстрактных математических терминах, не зависящих от технических средств, и к тому же опирающимся на крайне малое количество весьма элементарных исходных понятий. Характерно, что основополагающие работы по теории алгоритмов (и, в частности, работы Тьюринга) появились в 30-х гг. XX в., до создания современных компьютеров.

Указанная двоякая интерпретация сохраняет на абстрактном уровне основные плюсы и минусы двух вариантов инженерной реализации. Конкретная машина T работает гораздо быстрее; кроме того, управляющее устройство машины U довольно громоздко (т. е. велико число состояний и команд). Однако его сложность постоянна, и, будучи раз построено, оно годится для реализации сколь угодно больших алгоритмов, понадобится лишь больший объем ленты, которую естественно считать более дешевой и более просто устроенной, чем управляющее устройство. Кроме того, при смене алгоритма не понадобится строить новых устройств; нужно лишь написать новую программу.

Тезис Тьюринга. До сих пор нам удавалось для всех процедур, претендующих на алгоритмичность, т. е. конструктивных процедур, строить реализующие их машины Тьюринга. Будет ли это удаваться всегда? Увердительный ответ на этот вопрос содержится в тезисе Тьюринга, который формулируется так: «Всякий алгоритм может быть реализован машиной Тьюринга».

Доказать тезис Тьюринга нельзя, поскольку само понятие алгоритма (или эффективной процедуры) является неточным. Это не теорема и не постулат математической теории, а утверждение, которое связывает теорию с теми объектами, для описания которых она создана. По своему характеру тезис Тьюринга напоминает гипотезы физики об адекватности математических моделей физическим явлениям и процессам. Подтверждением тезиса Тьюринга является, во-первых, математическая практика, а во-вторых, то обстоятельство (уже отмечавшееся в конце § 5.1), что описание алгоритма в терминах любой другой известной алгоритмической модели может быть сведено к его описанию в виде машины Тьюринга.

Тезис Тьюринга позволяет, с одной стороны, заменить неточные утверждения о существовании эффективных процедур (алгоритмов) точными утверждениями о существовании машин Тьюринга, а с другой стороны, утверждения о несуществовании машин Тьюринга истолковывать как утверждения о несуществовании алгоритмов вообще. Однако не следует понимать тезис Тьюринга в том смысле, что вся теория алгоритмов может

быть сведена к теории машин Тьюринга. Например, в быстро развивающейся сейчас теории сложности алгоритмов (которая рассматривает сравнительную сложность алгоритмов по памяти, числу действий и т. д.) результаты, верные в рамках одной алгоритмической модели (скажем, о числе действий, необходимых для вычисления данной функции), могут оказаться неверными в другой модели.

Проблема остановки. В числе общих требований, предъявляемых к алгоритмам (см. § 5.1), упоминалось требование результируемости. Наиболее радикальной формулировкой здесь было бы требование, чтобы по любому алгоритму A и данным α можно было определить, приведет ли работа A при исходных данных α к результату или нет. Иначе говоря, нужно построить алгоритм B , такой, что $B(A, \alpha) = I$, если $A(\alpha)$ дает результат, и $B(A, \alpha) = L$, если $A(\alpha)$ не дает результата. В силу тезиса Тьюринга эту задачу можно сформулировать как задачу о построении машины Тьюринга: построить машину T_0 , такую, что для любой машины Тьюринга T и любых исходных данных α для машины T $T_0(\Sigma_T, \alpha) = I$, если машина $T(\alpha)$ останавливается, и $T_0(\Sigma_T, \alpha) = L$, если машина $T(\alpha)$ не останавливается.

Эта задача называется проблемой остановки. Ее формулировка несколько напоминает задачу о построении универсальной машины и, в частности, также предполагает выбор подходящего кодирования Σ_T и α в алфавите машины T_0 . Однако в данном случае никакое кодирование не приводит к успеху.

Теорема 5.4. Не существует машины Тьюринга T_0 , решающей проблему остановки для произвольной машины Тьюринга T .

Предположим, что машина T_0 существует. Для определенности будем считать, что маркером между Σ_T и α на ленте машины T_0 служит *. Построим машину $T_1(\Sigma_T) = T_0(T_{\text{коп}}(\Sigma_T))$. Исходными данными машины T_1 являются системы команд (точнее, их коды) любой машины T . Запись Σ_T на ленте машина T_1 преобразует в $\Sigma_T * \Sigma_T$ (машина $T_{\text{коп}}$ для чисел описана в примере 5.4, б), а затем работает как машина T_0 . Таким образом, T_1 также решает проблему остановки для любой машины T , но только

в том случае, когда на ленте T в качестве данных α_T написана ее собственная система команд Σ_T . Иначе говоря, $T_1(\Sigma_T) = \text{И}$, если машина $T(\Sigma_T)$ останавливается, и $T_1(\Sigma_T) = \text{Л}$, если машина $T(\Sigma_T)$ не останавливается. Пусть q_{1n} — заключительное состояние T_1 . Добавим к системе команд T_1 одно состояние $q_{1,n+1}$, объявив его заключительным, и m команд (m — число символов T_1) $q_{1n}\text{Л} \rightarrow \rightarrow q_{1,n+1}E$, $q_{1n}a_j \rightarrow q_{1n}R$ для любого a_j (в том числе И), кроме Л. Получим машину $T'_1(\Sigma_T)$, которая останавливается, если T не останавливается, и не останавливается, если T останавливается. Запишем теперь на ленте машины T'_1 ее собственную систему команд $\Sigma_{T'_1}$. Тогда T'_1 останавливается, если она не останавливается, и не останавливается, если она останавливается. Очевидно, такая машина T'_1 невозможна. Но поскольку она получена из T_0 вполне конструктивными, не вызывающими сомнений средствами и при этом никак не связана с конкретной структурой машины T_0 , остается заключить, что никакая машина T_0 , решающая проблему остановки, невозможна. \square

В силу тезиса Тьюринга невозможность построения машины Тьюринга означает отсутствие алгоритма решения данной проблемы. Поэтому полученная теорема дает первый пример *алгоритмически неразрешимой проблемы*, а именно, алгоритмически неразрешимой оказывается проблема остановки для машин Тьюринга, т. е. проблема определения результативности алгоритмов.

При истолковании утверждений, связанных с алгоритмической неразрешимостью, следует иметь в виду следующее важное обстоятельство. В таких утверждениях речь идет об отсутствии единого алгоритма, решающего данную проблему; при этом вовсе не исключается возможность решения этой проблемы в частных случаях, но различными средствами для каждого случая. В частности, теорема 5.4 не исключает того, что для отдельных классов машин Тьюринга проблема остановки может быть решена.* Например, она решается для всех машин, приведенных в примерах 5.2–5.8. Поэтому неразрешимость общей проблемы остановки вовсе не снимает необходи-

* Однако существуют конкретные машины Тьюринга (например, любая универсальная машина) с неразрешимой проблемой остановки.

мости доказывать сходимость предлагаемых алгоритмов, а лишь показывает, что поиск таких доказательств нельзя полностью автоматизировать.

Неразрешимость проблемы остановки можно интерпретировать как несуществование общего алгоритма для отладки программ, точнее, алгоритма, который по тексту любой программы и данным для нее определял бы, зациклится ли программа на этих данных или нет. Если учесть сделанное ранее замечание, такая интерпретация не противоречит тому эмпирическому факту, что большинство программ в конце концов удается отладить, т. е. установить наличие зацикливания, найти его причину и устраниить ее. При этом решающую роль играют опыт и искусство программиста.

Другие модели абстрактных машин. В дальнейшем появились другие модели абстрактных вычислительных машин. Цель этих моделей — приблизиться к возможностям обычных компьютеров путем расширения возможностей доступа к памяти. Кратко опишем две типичные модели таких машин.

Многоленточная машина Тьюринга — это машина Тьюринга, в которой имеется несколько (конечное число) лент; управляющее устройство с головкой — по-прежнему одно. Головка читает и пишет сразу на всех лентах. Движения на разных лентах могут быть в разные стороны (но каждое — на одну ячейку), поэтому удобно считать, что движутся ленты, а не головка. Типичная команда такой машины имеет вид

$$q_i a_{j1} \dots a_{jn} \rightarrow q'_i a'_{j1} \dots a'_{jn} d_{j1} \dots d_{jn}.$$

Многие вычисления на такой машине приобретают более привычный вид, чем на одноленточной машине Тьюринга. Например, на четырехленточной машине легко организовать обычный алгоритм сложения столбиком для чисел в позиционной системе счисления: на первых двух лентах записываются слагаемые, на третьей ленте — перенос, на четвертой — результат. Команды реализуют строки таблицы сложения для всех пар цифр системы счисления, например, для десятичной системы $2 + 3 = 5$, $7 + 9 = 16$ (1 на третью ленту, 6 в результат) и т. д.

Машины с произвольным доступом к памяти. Существует несколько моделей таких машин. Все они реализуют доступ к памяти по адресу.

Одна из таких моделей — машина с неограниченными регистрами (МНР).

Ее память — бесконечная последовательность регистров $R_0, R_1, \dots, R_i, \dots$, каждый из которых может хранить произвольное целое число. Через r_i обозначается число, хранящееся в R_i .

Программа не хранится в памяти и поэтому не может изменять себя. Команды программы пронумерованы.

Команды (для любых m, n, q):

- 1) обнуление $Z(n)$: $r_n := 0$ (записать 0 в регистр R_n);
- 2) прибавление единицы $S(n)$: $r_n := r_n + 1$ (прибавить единицу к содержимому регистра R_n);
- 3) переадресация $T(m, n)$: $r_n := r_m$ (записать в регистр R_n содержимое регистра R_m);
- 4) условный переход $J(m, n, q)$: если $r_n = r_m$, то выполнять q -ю команду программы; иначе выполнять следующую команду.

Машина останавливается, если следующая (или указанная в условном переходе) команда отсутствует. Предполагаем, что результат вычисления находится в R_1 .

Как видим, за адресный доступ к бесконечной памяти приходится платить бесконечным объемом каждой ячейки (регистра) — так сказать, двумерной бесконечностью.

Некоторые другие модели описаны в книге [2].

5.3. РЕКУРСИВНЫЕ ФУНКЦИИ

Введение. Всякий алгоритм однозначно ставит в соответствие исходным данным (в случае, если он определен на них) результат. Поэтому с каждым алгоритмом однозначно связана функция, которую он вычисляет. Верно ли обратное: для всякой ли функции существует вычисляющий ее алгоритм? Исследование проблемы остановки для машин Тьюринга показывает, что нет: для предиката $P(T, \alpha)$, истинного, если и только если машина Тьюринга T останавливается при исходных данных α , алгоритма его вычисления не существует. Возникает вопрос: для каких функций алгоритмы существуют? Как

описать такие алгоритмические, эффективно вычислимые функции?

Исследование этих вопросов привело к созданию в 30-х гг. XX в. теории рекурсивных функций. В этой теории, как и вообще в теории алгоритмов, принят конструктивный, финитный подход (см. § 5.1), основной чертой которого является то, что все множество исследуемых объектов (в данном случае функций) строится из конечного числа исходных объектов — базиса — с помощью простых операций, эффективная выполнимость которых достаточно очевидна. Операции над функциями в дальнейшем будем называть операторами.

Примитивно-рекурсивные функции. Определение и примеры. Займемся теперь конкретным выбором средств, с помощью которых будут строиться вычислимые функции. Очевидно, что к вычислимым функциям следует отнести все константы, т. е. 0 и все натуральные числа 1, 2... Однако в прямом включении бесконечного множества констант в базис нет необходимости. Достаточно одной константы 0 и функции следования $f(x) = x + 1$, которую иногда обозначают x' , чтобы получить весь натуральный ряд. Кроме того, в базис включим семейство $\{I_m^n\}$ функций тождества (или введения фиктивных переменных):

$$I_m^n(x_1, \dots, x_n) = x_m (m \leq n).$$

Весьма мощным средством получения новых функций из уже имеющихся является суперпозиция, знакомая нам по гл. 1 и 3. В них суперпозицией называлась любая подстановка функций в функции. Здесь ей для большей обозримости удобно придать стандартный вид. *Оператором суперпозиции* S_m^n называется подстановка в функцию от m переменных m функций от n одних и тех же переменных. Она дает новую функцию от n переменных. Например, для функций $h(x_1, \dots, x_m)$, $g_1(x_1, \dots, x_n)$, \dots , $g_m(x_1, \dots, x_n)$

$$\begin{aligned} S_m^n(h, g_1, \dots, g_m) &= h(g_1(x_1, \dots, x_n), \dots, \\ &\quad g_m(x_1, \dots, x_n)) = f(x_1, \dots, x_n). \end{aligned}$$

Это определение порождает семейство операторов суперпозиции $\{S_m^n\}$. Благодаря функциям тождества стандартизация суперпозиции не уменьшает ее возможностей:

любую подстановку функций в функции можно выразить через S_m^n и I_m^n . Например, $f(x_1, x_2) = h(g_1(x_1, x_2), g_2(x_1))$ в стандартном виде запишется как $f(x_1, x_2) = S_2^2(h(x_1, x_2), g_1(x_1, x_2), S_2^1(I_1^2(x_1, x_2), g_2(x_1), g_3(x_1)))$, где g_3 — любая функция от x_1 . В свою очередь, используя подстановку и функции тождества, можно переставлять и отождествлять аргументы в функции:

$$\begin{aligned} f(x_2, x_1, x_3, \dots, x_n) &= f(I_2^2(x_1, x_2), I_1^2(x_1, x_2), x_3, \dots, x_n); \\ f(x_1, x_1, x_3, \dots, x_n) &= f(I_1^2(x_1, x_2), I_1^2(x_1, x_2), x_3, \dots, x_n). \end{aligned}$$

Таким образом, если заданы функции I_m^n и операторы S_m^n , то можно считать заданными всевозможные операторы подстановки функций в функции, а также переименования, перестановки и отождествления переменных.

Еще одно семейство операторов, которое здесь понадобится, — это операторы примитивной рекурсии. *Оператор примитивной рекурсии* R_n определяет $(n + 1)$ -местную функцию f через n -местную функцию g и $(n + 2)$ -местную функцию h следующим образом:

$$\left. \begin{aligned} f(x_1, \dots, x_n, 0) &= g(x_1, \dots, x_n); \\ f(x_1, \dots, x_n, y + 1) &= h(x_1, \dots, x_n, y, f(x_1, \dots, x_n, y)). \end{aligned} \right\} \quad (5.2)$$

Пара равенств (5.2) называется *схемой примитивной рекурсии*. Тот факт, что функция f определена схемой (5.2), выражается равенством $f(x_1, \dots, x_n, y) = R_n(g, h)$. В случае, когда $n = 0$, т. е. определяемая функция f является одноместной, схема (5.2) принимает более простой вид:

$$f(0) = c; \quad f(y + 1) = h(y, f(y)), \quad (5.3)$$

где c — константа.

Схемы (5.2) и (5.3) определяют f рекурсивно не только через другие функции g и h , но и через значения f в предшествующих точках: значение f в точке $y + 1$ зависит от значения f в точке y . Для вычисления $f(x_1, \dots, x_n, k)$ понадобится $k + 1$ вычислений по схеме (5.2) — для $y = 0, 1, \dots, k$.

Пример такого определения функции приводился в гл. 1 (для функции $n!$); здесь оно будет рассмотрено более подробно. Существенным в операторе примитивной рекурсии является то, что независимо от числа переменных в f рекурсия ведется только по одной переменной y ; остальные

n переменных x_1, \dots, x_n на момент применения схем (5.2), (5.3) зафиксированы и играют роль параметров.

Функция называется *примитивно-рекурсивной*, если она может быть получена из константы 0, функции x' и функций I_m^n с помощью конечного числа применений операторов суперпозиции и примитивной рекурсии. Этому определению можно придать более формальный индуктивный вид.

1. Функции 0, x' и I_m^n для всех натуральных n, m , где $m \leq n$, являются примитивно-рекурсивными.

2. Если $g_1(x_1, \dots, x_n), \dots, g_m(x_1, \dots, x_n), h(x_1, \dots, x_m)$ — примитивно-рекурсивные функции, то $S_m^n(h, g_1, \dots, g_m)$ — примитивно-рекурсивные функции для любых натуральных n, m .

3. Если $g(x_1, \dots, x_n)$ и $h(x_1, \dots, x_n, y, z)$ — примитивно-рекурсивные функции, то $R_n(g, h)$ — примитивно-рекурсивная функция.

4. Других примитивно-рекурсивных функций нет.

Из такого индуктивного описания нетрудно извлечь процедуру, порождающую все примитивно-рекурсивные функции.

Пример 5.9 (сложение, умножение, возведение в степень).

1. Сложение $f_+(x, y) = x + y$ примитивно-рекурсивно:

$$\begin{aligned} f_+(x, 0) &= x = I_1^1(x); \\ f_+(x, y + 1) &= f_+(x, y) + 1 = (f_+(x, y))'. \end{aligned}$$

Таким образом, $f_+(x, y) = R_1(I_1^1(x), h(x, y, z))$, где $h(x, y, z) = z' = z + 1$.

2. Умножение $f_\times(x, y) = xy$ примитивно-рекурсивно:

$$\begin{aligned} f_\times(x, 0) &= 0; \\ f_\times(x, y + 1) &= f_\times(x, y) + x = f_+(x, f_\times(x, y)). \end{aligned}$$

3. Возведение в степень $f_{\exp}(x, y) = x^y$ примитивно-рекурсивно:

$$\begin{aligned} f_{\exp}(x, 0) &= 1; \\ f_{\exp}(x, y + 1) &= x^y x = f_\times(x, f_{\exp}(x, y)). \end{aligned}$$

Определим функцию $x \dashv y$ (арифметическое или урезанное вычитание) следующим образом:

$$x \dashv y = \begin{cases} x - y, & \text{если } x > y; \\ 0 & \text{в противном случае.} \end{cases}$$

Пример 5.10. Примитивно-рекурсивными являются следующие функции:

1) $f(x) = x \div 1$, определяемая схемой:

$$f(0) = 0 \div 1 = 0;$$

$$f(y + 1) = y;$$

2) $f(x, y) = x \div y$, определяемая схемой:

$$f(x, 0) = x;$$

$$f(x, y + 1) = x \div (y + 1) = (x \div y) \div 1 = f(x, y) \div 1$$

(для определения функции из п. 2 использована функция из п. 1);

$$3) f(x, y) = |x - y| = (x \div y) + (y \div x);$$

$$4) sg(x) = \begin{cases} 0, & \text{если } x = 0; \\ 1, & \text{если } x \neq 0; \end{cases}$$

ее схема имеет вид:

$$sg(0) = 0;$$

$$sg(x + 1) = 1;$$

$$5) \min(x, y) = x \div (x \div y);$$

$$6) \max(x, y) = y + (x \div y).$$

С помощью функции sg (сигнум) из примера 5.10, г и ее отрицания $\overline{sg}(x) = 1 \div sgx$ построим примитивно-рекурсивное описание функций, связанных с делением.

Пример 5.11 (деление). а. Функция $r(x, y)$ — остаток от деления y на x :

$$r(x, 0) = 0;$$

$$r(x, y + 1) = (r(x, y) + 1)sg(|x - (r(x, y) + 1)|).$$

Смысл второй строки определения в следующем: если $y + 1$ не делится на x , то $sg(|x - (r(x, y) + 1)|) = 1$ и $r(x, y + 1) = r(x, y) + 1$; если же $y + 1$ делится на x , то $r(x, y + 1) = sg(|x - r(x, y) + 1|) = 0$.

б. Функция $g(x, y) = [y/x]$ — частное от деления y на x , т. е. целая часть дроби y/x :

$$g(x, 0) = 0;$$

$$g(x, y + 1) = g(x, y) + \overline{sg}(|x - (r(x, y) + 1)|).$$

Второе слагаемое, как и в случае $r(x, y)$, зависит от делимости $y + 1$ на x . Если $y + 1$ делится на x , то $g(x, y + 1) = g(x, y)$; если нет, то $g(x, y + 1) = g(x, y) + 1$.

В рекурсивных описаниях функций примера 5.11 отчетливо видны логические действия: проверка условия (делимости $y + 1$ на x) и выбор дальнейшего хода вычислений в зависимости от истинности условия. Займемся теперь более подробным выяснением того, какие возможности для логических операций имеются в рамках примитивно-рекурсивных функций.

Из функций примеров 5.10, б, д, е легко получается примитивная рекурсивность «арифметизированных» логических функций, т. е. числовых функций, которые на множестве $\{0, 1\}$ ведут себя как логические функции. Действительно, если $x, y \in \{0, 1\}$, то

$$\begin{aligned}\bar{x} &= 1 - x; \\ x \vee y &= \max(x, y); \\ x \& y &= \min(x, y).\end{aligned}\tag{5.4}$$

Из функциональной полноты (см. § 3.3) этого множества функций и того, что суперпозиция является примитивно-рекурсивным оператором (см. далее), следует примитивная рекурсивность всех логических функций.

Отношение $R(x_1, \dots, x_n)$ называется примитивно-рекурсивным, если примитивно-рекурсивна его характеристическая функция χ_R :

$$\chi_R(x_1, \dots, x_n) = \begin{cases} 1, & \text{если } R(x_1, \dots, x_n) \text{ выполняется;} \\ 0 & \text{в противном случае.} \end{cases}$$

Ввиду взаимно однозначного соответствия между отношениями и предикатами (см. § 3.4) χ_R будет характеристической функцией и для соответствующего предиката.

Напомним, что сам предикат принимает логические значения И и Л, с которыми нельзя производить арифметических действий, даже когда эти значения изображаются числами 0 и 1 (см. § 3.1). Поэтому следует проводить различие между предикатами и их характеристическими функциями. В алгоритмических языках предикат будет принимать значения true и false, а его характеристическая функция — значения 0 и 1.

Предикат называется примитивно-рекурсивным, если его характеристическая функция примитивно-рекурсивна. В силу соотношений (5.4), если предикаты P_1, \dots, P_k

примитивно-рекурсивны, то и любой предикат, полученный из них с помощью логических операций, примитивно-рекурсивен.

Пример 5.12 (предикаты и отношения). а. Предикат $Pd_n(x)$ « x делится на n » примитивно-рекурсивен для любого n :

$$\chi_{Pd_n}(x) = \overline{sg}(r(n, x)).$$

б. Предикат $Pd_{n, m}(x)$ « x делится на m и на n » примитивно-рекурсивен для любых m и n , так как $P_{m, n}(x) = P_m(x) \& P_n(x)$.

в. Отношение $x_1 > x_2$ примитивно-рекурсивно:

$$\chi_{>}(x_1, x_2) = sg(x_1 \dot{-} x_2).$$

г. Если $f(x)$ и $g(x)$ примитивно-рекурсивны, то предикат « $f(x) = g(x)$ » примитивно-рекурсивен, так как его функция χ имеет вид:

$$\chi(x) = \overline{sg}(|f(x) - g(x)|).$$

Примитивно-рекурсивные операторы. Оператор называется *примитивно-рекурсивным* (сокращенно ПР-оператором), если он сохраняет примитивную рекурсивность функций, т. е. если результат его применения к примитивно-рекурсивным функциям дает снова примитивно-рекурсивную функцию. Операторы S_m^n и R_n являются ПР-операторами по определению. Рассмотрим теперь другие ПР-операторы. Их использование позволяет существенно сократить примитивно-рекурсивные описания функций.

В § 5.2 мы встречались с оператором условного перехода (обозначим его здесь B), который по функциям $g_1(x_1, \dots, x_n)$, $g_2(x_1, \dots, x_n)$ и предикату $P(x_1, \dots, x_n)$ строит функцию $f(x_1, \dots, x_n) = B(g_1, g_2, P)$:

$$f(x_1, \dots, x_n) = \begin{cases} g_1(x_1, \dots, x_n), & \text{если} \\ & P(x_1, \dots, x_n) \text{ истинно;} \\ g_2(x_1, \dots, x_n), & \text{если} \\ & P(x_1, \dots, x_n) \text{ ложно.} \end{cases} \quad (5.5)$$

Теорема 5.3 утверждает, что B вычислим по Тьюрингу. Примитивная рекурсивность B видна из следующего соотношения, эквивалентного (5.5):

$$f(x_1, \dots, x_n) = g_1(x_1, \dots, x_n)\chi_P(x_1, \dots, x_n) + \\ + g_2(x_1, \dots, x_n)\chi_{\bar{P}}(x_1, \dots, x_n).$$

Обобщение оператора B на случай многозначного перехода по предикатам P_1, \dots, P_k , из которых истинен всегда один и только один предикат: $f(x_1, \dots, x_n) = B(g_1, \dots, g_k, P_1, \dots, P_k)$, также примитивно-рекурсивно, поскольку

$$f(x_1, \dots, x_n) = g_1\chi_{P_1} + \dots + g_k\chi_{P_k},$$

(отметим, что это соотношение определяет B и в том случае, когда ни один из P_1, \dots, P_k не истинен: B при этом равно нулю).

С помощью оператора B удобно задавать функции, определенные на конечных множествах. Правда, B , как и любой другой ПР-оператор, всегда определяет функцию на всем натуральном ряде, т. е. производит доопределение функции вне области задания. Например, функцию f , определенную на множестве 1, 2, 6, 17 равенствами $f(1) = 5, f(2) = 1, f(6) = 0, f(17) = 8$, с помощью оператора B можно описать так:

$$f(x) = \begin{cases} 5, & \text{если } x = 1; \\ 1, & \text{если } x = 2; \\ 0, & \text{если } x = 6; \\ 8 & \text{в остальных случаях.} \end{cases}$$

Такое описание полагает $f(x) = 8$ вне исходной области задания.

Пусть $f(x_1, \dots, x_n, y)$ — функция от $(n + 1)$ -й переменной. Хорошо известные операции суммирования $\sum_{y < z}$ и перемножения $\prod_{y < z}$ по переменной y с пределом z — это операторы, которые из функции $f(x_1, \dots, x_n, y)$ порождают новые функции

$$g(x_1, \dots, x_n, z) = \sum_{y < z} f(x_1, \dots, x_n, y) \text{ и}$$

$$h(x_1, \dots, x_n, z) = \prod_{y < z} f(x_1, \dots, x_n, y).$$

Покажем их примитивную рекурсивность:

$$g(x_1, \dots, x_n, 0) = 0 \text{ (по определению);}$$

$$g(x_1, \dots, x_n, z + 1) = g(x_1, \dots, x_n, z) + f(x_1, \dots, x_n, z);$$

$$g(x_1, \dots, x_n, 0) = 1; \\ h(x_1, \dots, x_n, z+1) = h(x_1, \dots, x_n, z)f(x_1, \dots, x_n, z).$$

Таким образом g и h могут быть определены по схеме примитивной рекурсии с использованием сложения и умножения, примитивная рекурсивность которых доказана, а также функции f . Следовательно, g и h примитивно-рекурсивны, если f примитивно-рекурсивна.

С помощью $\sum_{y < z}$ и $\prod_{y < z}$ нетрудно показать, что операторы $\sum_{y=k}^z$ и $\prod_{y=k}^z$ (т. е. суммирование и умножение от k до z) — также ПР-операторы.

Рассмотрим теперь оператор, играющий важную роль в теории рекурсивных функций, — *ограниченный оператор наименьшего числа (μ -оператор)*, называемый также *ограниченным оператором минимизации*, который применяется к предикатам и определяется так:

$$\mu y_{y \leq z} P(x_1, \dots, x_n, y) = \begin{cases} \text{наименьшему } y \leq z, \text{ такому,} \\ \text{что } P(x_1, \dots, x_n, y) \text{ истинно,} \\ \text{если такой } y \text{ существует;} \\ z \text{ в противном случае.} \end{cases}$$

Из предиката $P(x_1, \dots, x_n, y)$ с помощью оператора $\mu y_{y \leq z}$ получается функция $f(x_1, \dots, x_n, z)$. Второй случай в определении μ добавлен для того, чтобы f была всюду определена.

Пример 5.13. Пусть $P(x_1, x_2, y) = Pd_{x_1, x_2}(y)$ (пример 5.12, б). Тогда $\mu y_{y \leq z} P(x_1, x_2, y)$ равен наименьшему общему кратному (НОК) x_1 и x_2 , если $z \geq$ НОК, и равен z , если $z \leq$ НОК.

Ограниченнный μ -оператор примитивно-рекурсивен:

$$\mu y_{y \leq z} P(x_1, \dots, x_n, y) = \sum_{i=0}^z \prod_{j=0}^i (1 - \chi_P(x_1, \dots, x_n, j)).$$

Ограничение z в ограниченном μ -операторе дает гарантию окончания вычислений, поскольку оно оценивает сверху число вычислений предиката P . Возможность оценить сверху количество вычислений является существенной особенностью примитивно-рекурсивных функций. Уже отмечалось [см. (5.2), (5.3) и далее], что если $f(x_1, \dots, x_n, k) = R_n(g, h)$, то для вычисления $f(x_1, \dots, x_n, k)$

понадобится $k + 1$ вычислений по схеме (5.2): одно вычисление g и k вычислений h . Правда, каждое из них может, в свою очередь, состоять из некоторого количества вычислений функций, входящих в определение g и h ; но в силу конечности общего числа операторов S_m^n и R_n , использованных для построения f из базисных функций 0 , x' и I_m^n , для любого k можно оценить количество элементарных действий (т. е. вычислений базисных функций), необходимых для вычисления $f(x_1, \dots, x_n, k)$. В дальнейшем будет показано, что неограниченный μ -оператор не является примитивно-рекурсивным.

μ -оператор (как ограниченный, так и неограниченный) является удобным средством для построения обратных функций. Действительно, функция $g(x) = \mu y(f(y) = x)$ («наименьший y , такой, что $f(y) = x$ ») является обратной к функции $f(x)$. Поэтому в применении к одноместным функциям μ -оператор иногда называют оператором обращения.

Пример 5.14. а. С помощью ограниченного μ -оператора определим деление, точнее, функцию $[z/x]$ (частное от деления z на x — см. пример 5.11, б), как функцию, обратную* умножению:

$$[z/x] = \mu y_{y \leq z}(x(y + 1) > z).$$

б. Целая часть \sqrt{x} — функция $[\sqrt{x}]$ — примитивно-рекурсивная, так как

$$[\sqrt{x}] = \mu y_{y \leq x}((y + 1)^2 > x).$$

в. $[\log_k x] = \mu y_{y \leq x}(ky^{+1} > x)$, следовательно, целая часть логарифма по любому целому основанию k примитивно-рекурсивна.

Еще один ПР-оператор — это оператор совместной или одновременной рекурсии, точнее, целое семейство операторов $\{R_{nk}\}$. С помощью R_{nk} строится рекурсивное описание сразу нескольких функций f_1, \dots, f_k от $(n + 1)$ -й переменной, причем значение каждой функции $y + 1$ зависит от значения всех функций в точке y :

* Целая часть от деления — функция, «не совсем обратная» умножению; точнее, обратная ему только в тех точках, где z делится нацело на x . Поэтому в ее описании используется не предикат равенства (как в определении обратной функции), а предикат $>$.

$$\begin{aligned}
f_1(x_1, \dots, x_n, 0) &= g_1(x_1, \dots, x_n); \\
&\dots \\
f_k(x_1, \dots, x_n, 0) &= g_k(x_1, \dots, x_n); \\
f_1(x_1, \dots, x_n, y+1) &= h_1(x_1, \dots, x_n, y, f_1(x_1, \dots, x_n, y), \\
&\dots, f_k(x_1, \dots, x_n, y)); \\
&\dots \\
f_k(x_1, \dots, x_n, y+1) &= h_k(x_1, \dots, x_n, y, f_1(x_1, \dots, x_n, y), \\
&\dots, f_k(x_1, \dots, x_n, y)).
\end{aligned}$$

По существу совместная рекурсия дает рекурсивное описание функции-вектора (f_1, \dots, f_k) . Для того чтобы доказать примитивную рекурсивность совместной рекурсии, нужно закодировать этот вектор числом, причем так, чтобы существовала однозначная и примитивно-рекурсивная расшифровка этого кода (т. е. извлечение нужного разряда вектора). В качестве такого кода можно предложить число

$$\phi(x_1, \dots, x_n, y) = 2^{f_1(x_1, \dots, x_n, y)} \cdot 3^{f_2} \cdot 5^{f_3} \cdots p_{k-1}^{f_k(x_1, \dots, x_n, y)},$$

где p_i — i -е простое число (2 считается 0-м простым числом). Продемонстрируем эту идею на примере оператора R_{12} . Пусть $f_1(x, y), f_2(x, y)$ заданы совместной рекурсией:

$$\begin{aligned}
f_1(x, 0) &= g_1(x); \\
f_2(x, 0) &= g_2(x); \\
f_1(x, y+1) &= h_1(x, y, f_1(x, y), f_2(x, y)); \\
f_2(x, y+1) &= h_2(x, y, f_1(x, y), f_2(x, y)).
\end{aligned}$$

Определим кодирующую функцию F :

$$\begin{aligned}
F(x, 0) &= 2^{g_1(x)} \cdot 3^{g_2(x)}; \\
F(x, y+1) &= 2^{h_1(x, y, f_1(x, y), f_2(x, y))} \cdot 3^{h_2(x, y, f_1(x, y), f_2(x, y))}.
\end{aligned}$$

Тогда $f_1(x, y) = \mu z_{z \ll F(x, y)}(\overline{\text{Pd}}_{2^{z+1}}(F(x, y)))$, т. е. равна показателю при 2 в разложении $F(x, y)$ на простые множители;

$$f_2(x, y) = \mu z_{z \ll F(x, y)}(\text{Pd}_{3^{z+1}}(F(x, y))),$$

т. е. равна показателю при 3 в разложении $F(x, y)$ на простые множители. (Определение предиката Pd см. в примере 5.12, а).

Следует иметь в виду, что такая кодировка вовсе не предлагается в качестве метода рекурсивного вычисления функций-векторов. Наоборот, предлагается прямой

и более простой метод совместной рекурсии, а кодировка является лишь доказательством того, что этот метод относится к числу примитивно-рекурсивных средств вычисления.

Можно предложить полезную схемную интерпретацию примитивной рекурсии вообще и совместной рекурсии в частности. На рис. 5.12, *a* изображена схема, состоящая из устройства, вычисляющего за один такт функцию h от двух переменных, и элемента задержки на один такт; по каналам схемы могут передаваться натуральные числа. Время t будем считать дискретным: $t = 0, 1, 2\dots$. Схема имеет один вход x и выход f . Однако выход f зависит не только от значения x , но и от момента t , в который он рассматривается, т. е. представляет собой некоторую функцию $f(x, t)$. Рассмотрим эту зависимость подробнее. Будем считать значение x на все времена рассмотрения, начиная с $t = 0$, произвольным, но фиксированным. В начальный момент $t = 0$ значение второго входа h является константой c , зависящей от начального состояния схемы: $f(x, 0) = h(x, c) = g(x)$. В момент $t = 1$ $f(x, 1) = h(x, f(x, 0))$; в общем случае $f(x, t + 1) = h(x, f(x, t))$. Таким образом, схема рис. 5.12, *a* реализует примитивную рекурсию по переменной t , т. е. по времени. Нетрудно убедиться, например, что если h выполняет умножение, а $c = 1$, то $f(x, t) = x^{t+1}$. При аналогичной интерпретации (с начальными константами c_1, c_2) схема на рис. 5.12, *б* реализует

схему совместной рекурсии, причем рекурсия также ведется по времени, общему для устройств h_1 и h_2 . Доказательство того, что совместная рекурсия — это ПР-оператор, в терминах таких схем означает, что перекрестные обратные связи на рис. 5.12, *б* можно заменить простыми контурами обратной связи типа приведенных на рис. 5.12, *а*. Структура схемы при этом станет проще, однако понадобятся дополнительные вычислительные устройства, формирователь числового кода вектора (f_1, f_2) , передаваемого по одному каналу, и декодирующее устройство с двумя выходами f_1 и f_2 .

Описанная схемная интерпретация примитивно-рекурсивных функций проходит при одном предположении, что

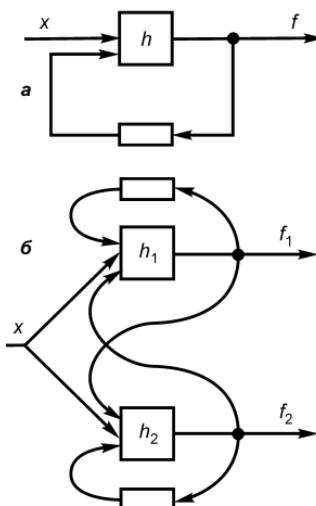


рис. 5.12

за один такт любой канал схемы способен передать, а вычислительное устройство — воспринять и переработать любое, сколь угодно большое натуральное число. Такое предположение физически нереализуемо, поэтому теория таких схем не будет представлять самостоятельного интереса. Если же наложить ограничения на возможности каналов и элементов схем, то это приведет к теории конечных автоматов и схем из автоматов, о которой будет идти речь в гл. 8.

Подведем некоторые итоги. Из простейших функций — константы 0, функции $x + 1$ и функций тождества I_m^n — с помощью операторов суперпозиции и примитивной рекурсии было получено огромное разнообразие функций, включающих основные функции арифметики, алгебры и анализа (с поправкой на целочисленность). Тем самым выяснено, что эти функции имеют примитивно-рекурсивное описание, которое однозначно определяет процедуру их вычисления; следовательно, их естественно отнести к классу вычислимых функций. Попутно сделаем два замечания. Во-первых, все примитивно-рекурсивные функции всюду определены. Это следует из того, что простейшие функции всюду определены, а операторы S_m^n и R_n это свойство сохраняют. Во-вторых, строго говоря, мы имеем дело не с функциями, а с их примитивно-рекурсивными описаниями. Различие здесь имеет тот же смысл, что и отмечавшееся неоднократно в гл. 3 различие между функциями и их представлениями в виде формул. Примитивно-рекурсивные описания также можно разбить на классы эквивалентности, отнеся в один класс все описания, задающие одну и ту же функцию. Однако задача распознавания эквивалентности примитивно-рекурсивных описаний, как будет показано, алгоритмически неразрешима.

Функции Аккермана. Пора уже задать вопрос: все ли функции являются примитивно-рекурсивными? Простые теоретико-множественные соображения показывают, что нет. В гл. 1 (пример 1.8, г) было показано, что множество всех функций типа $N \rightarrow N$ (т. е. одноместных целочисленных функций) несчетно, тем более это верно для функций типа $N^n \rightarrow N$. Каждая примитивно-рекурсивная функция имеет конечное описание, т. е. задается конечным словом в некотором (фиксированном для всех функций) алфавите. Мно-

жество всех конечных слов счетно, поэтому примитивно-рекурсивные функции образуют не более чем счетное подмножество несчетного множества функций типа $N^n \rightarrow N$. В действительности здесь рассматривается более узкий вопрос: все ли вычислимые функции можно описать как примитивно-рекурсивные? Чтобы показать, что ответ на этот вопрос также является отрицательным, построим пример вычислимой функции, не являющейся примитивно-рекурсивной. Идея примера в том, чтобы, построив последовательность функций, каждая из которых растет существенно быстрее предыдущей, сконструировать с ее помощью функцию, которая растет быстрее любой примитивно-рекурсивной функции.

Начнем с построения последовательности функций $\varphi_0, \varphi_1, \varphi_2$. Положим $\varphi_0(a, y) = a + y; \varphi_1(a, y) = ay; \varphi_2(a, y) = a^y$. Эти функции связаны между собой следующими рекурсивными соотношениями:

$$\begin{aligned}\varphi_1(a, y + 1) &= a + ay = \varphi_0(a_1, \varphi_1(a, y)); \varphi_1(a, 1) = a; \\ \varphi_2(a, y + 1) &= aa^y = \varphi_1(a, \varphi_2(a, y)); \varphi_2(a, 1) = a.\end{aligned}$$

Продолжим эту последовательность, положив по определению

$$\left. \begin{aligned}\varphi_{n+1}(a, 0) &= 1; \\ \varphi_{n+1}(a, 1) &= a; \\ \varphi_{n+1}(a, y + 1)\varphi_n &= (a_1, \varphi_{n+1}(a, y)).\end{aligned} \right\} \quad (5.6)$$

Эта схема имеет вид примитивной рекурсии, следовательно, все функции φ_n примитивно-рекурсивные. Растут они крайне быстро; например,

$$\begin{aligned}\varphi_3(a, 1) &= a; \varphi_3(a, 2) = \varphi_2(a, a) = \\ &= a^a; \varphi_3(a, 3) = a^{a^a}; \varphi_3(a, k) = a^{a \cdot \dots \cdot a} \}^k \text{ раз}.\end{aligned}$$

Зафиксируем теперь значение a : $a = 2$. Получим последовательность одноместных функций: $\varphi_0(2, y), \varphi_1(2, y) \dots$ Определим теперь функцию $B(x, y)$, которая перечисляет эту последовательность: $B(x, y) = \varphi_x(2, y)$, т. е. $B(0, y) = \varphi_0(2, y) = 2 + y; B(1, y) = \varphi_1(2, y) = 2y$ и т. д., а также диагональную функцию $A(x) = B(x, x)$. Эти функции называются функциями Аккермана. Из соотношений (5.6) следует, что

$$\left. \begin{aligned}B(0, y) &= 2 + y; \\ B(x + 1, 0) &= \text{sg } x; \\ B(x + 1, y + 1) &= B(x, B(x + 1, y)).\end{aligned} \right\} \quad (5.7)$$

Эти соотношения позволяют вычислять значения функций $B(x, y)$ и, следовательно, $A(x)$, причем для вычисления функции в данной точке нужно обратиться к значению функции в предшествующей точке — совсем как в схеме

примитивной рекурсии. Однако здесь рекурсия ведется сразу по двум переменным (такая рекурсия называется двойной, двукратной, или рекурсией 2-й ступени), и это существенно усложняет характер упорядочения точек, а следовательно, и понятие предшествования точек также усложняется. Например, $B(3, 3) = B(2, B(3, 2))$, а так как $B(3, 2) = \varphi_3(2, 2) = 2^2 = 4$, то вычислению B в точке $(3, 3)$ по схеме (5.7) должно предшествовать вычисление B в точке $(2, 4)$; читатель может убедиться, что вычислению B в точке $(3, 4)$ должно предшествовать вычисление в точке $(2, 16)$. Важно отметить, что это упорядочение не предопределено заранее, как в схеме примитивной рекурсии, где n всегда предшествует $n + 1$, а выясняется в ходе вычислений и для каждой схемы вида (5.7), вообще говоря, различно. Поэтому схему двойной рекурсии (в отличие от рассмотренной ранее одновременной рекурсии) не всегда удается свести к схеме примитивной рекурсии.

Теорема 5.5. Функция Аккермана $A(x)$ растет быстрее, чем любая примитивно-рекурсивная функция, и, следовательно, не является примитивно-рекурсивной. Более точно, для любой одноместной примитивно-рекурсивной функции $f(x)$ найдется такое n , что для любого $x \geq n$ $A(x) > f(x)$.

Ограничимся наброском доказательства, опустив некоторые выкладки.

1. Вначале доказываются два свойства функции $B(x, y)$

$$B(x, y + 1) > B(x, y) \quad (x, y = 1, 2, \dots); \quad (5.8)$$

$$B(x + 1, y) \geq B(x, y + 1) \quad (x \geq 1, y \geq 2). \quad (5.9)$$

2. Назовем функцию $f(x_1, \dots, x_n)$ B -мажорируемой, если существует натуральное m , такое, что для любых x_1, \dots, x_n , удовлетворяющих условию $\max(x_1, \dots, x_n) > 1$, $f(x_1, \dots, x_n) < B(m, \max(x_1, \dots, x_n))$. Покажем, что все примитивно-рекурсивные функции B -мажорирумы:

а) для простейших функций 0 , $x + 1$, I_m^n их B -мажорируемость очевидна;

б) рассмотрим оператор суперпозиции S_m^n , причем для простоты выкладок ограничимся случаем $n = 1$, $m = 2$, т. е. функцией $f(x) = h(g_1(x), g_2(x))$.

Пусть функции h , g_1 , g_2 B -мажорирумы с числами m_h , m_{g1} , m_{g2} соответственно. Положим $m = \max(m_h, m_{g1}, m_{g2})$. Тогда по определению B -мажорируемости и свойствам (5.8), (5.9)

$$h(x_1, x_2) < B(m, \max(x_1, x_2));$$

$$g_1(x) < B(m, x) < B(m + 1, x);$$

$$g_2(x) < B(m, x) < B(m + 1, x)$$

и, следовательно, с учетом (5.7) и (5.9)

$f(x) = h(g_1(x), g_2(x)) < B(m, \max(g_1(x), g_2(x))) <$
 $< B(m, B(m+1, x)) = B(m+1, x+1) \leq B(m+2, x),$
 т. е. $f(x)$ B -мажорируема;

в) рассмотрим теперь оператор примитивной рекурсии, ограничившись для простоты схемой (5.3):

$$f(0) = c_0;$$

$$f(x+1) = h(x, f(x)).$$

Пусть h B -мажорируема, т. е. $h(x_1, x_2) < B(m_h, \max(x_1, x_2))$ при $\max(x_1, x_2) > 1$. Докажем индукцией по x , что f B -мажорируема.

Учитывая условие в определении B -мажорируемости, индукцию надо начинать с $x = 2$. Пусть $f(2) = c_2$. Из (5.8), (5.9) следует, что $B(x+1, 2) > B(x, 2)$, поэтому найдется такая величина m_2 , что $B(m_2, 2) > c_2 = f(2)$.

Положим $m = \max(m_2, m_h) + 1$. Очевидно, что $f(2) < B(m, 2)$.

Пусть теперь $f(k) < B(m, k)$. Тогда

$$f(k+1) = h(k, f(k)) > B(m_h, \max(k, f(k))). \quad (5.10)$$

Если $\max(k, f(k)) = k$, то $f(k+1) < B(m_h, k) < B(m, k+1)$. Если же $\max(k, f(k)) = f(k)$, то, используя (5.7)–(5.10) и то, что $m_h \leq m-1$, имеем $f(k+1) < B(m_h, f(k)) < B(m_h, B(m, k)) \leq B(m-1, B(m, k)) = B(m, k+1)$, что и завершает индукцию.

Из пп. а–в следует, что все примитивно-рекурсивные функции B -мажорируемые.

3. Пусть $f(x)$ — примитивно-рекурсивная функция. В силу п. 2 для некоторого m и $x \geq 2$ $f(x) < B(m, x)$. Но тогда

$$A(m+x) = B(m+x, m+x) > B(m, m+x) > f(m+x),$$

т. е. $A(x) > f(x)$, начиная по крайней мере с $x = m+2$. \square

Общерекурсивные и частично-рекурсивные функции. Функция Аккермана $A(x)$ является примером вычислимой, но не примитивно-рекурсивной функции. Этот пример говорит о том, что средства построения вычислимых функций нуждаются в расширении. Операторы кратной рекурсии (т. е. по нескольким переменным одновременно) не дают желаемого замыкания класса всех вычислимых функций: было показано, что для любого n найдется функция, определимая с помощью n -кратной рекурсии (n -рекурсивная), но не $(n-1)$ -рекурсивная. Более подходящим для этой цели оказывается неограниченный μ -оператор (в дальнейшем прилагательное «неограниченный» будем опускать).

Функция называется *частично-рекурсивной*, если она может быть построена из простейших функций 0 , $x + 1$, I_m^n с помощью конечного числа применений суперпозиции, примитивной рекурсии и μ -оператора.

По определению μ -оператор применяется к предикатам. Поскольку в теории рекурсивных функций истинность предиката $P(x)$ всегда связана со справедливостью некоторого равенства [например, $\chi_P(x) = 1$] и, наоборот, всякое равенство является предикатом от содержащихся в нем переменных, то μ -оператору можно придать стандартную форму, например $f(x_1, \dots, x_n) = \mu y(g(x_1, \dots, x_{n-1}, y) = = x_n)$ или $f(x_1, \dots, x_n) = \mu y(g(x_1, \dots, x_n, y) = 0)$. Будучи применен к вычислимой функции, μ -оператор снова дает вычислимую функцию (т. е. сохраняет вычислимость). Действительно, для вычисления функции $f(x_1, \dots, x_n) = = \mu y(g(x_1, \dots, x_n, y) = 0)$ на наборе x_1, \dots, x_n существует следующая простая процедура: вычисляем g на наборах $(x_1, \dots, x_n, 0), (x_1, \dots, x_n, 1), \dots$ до тех пор, пока не получим нулевое значение. Однако в отличие от рассмотренных ранее процедур она может не привести к результату: это произойдет в случае, когда на данном наборе x_1, \dots, x_n уравнение $g(x_1, \dots, x_n, y) = 0$ не имеет решения. В таком случае функция $f(x_1, \dots, x_n)$ считается неопределенной. Например, обратная к $x + 1$ функция $x - 1 = \mu y(y + 1 = x)$ не определена при $x = 0$. Заметим, что механизм возникновения неопределенности здесь такой же, как и при вычислениях на машине Тьюринга: в случае неопределенности процесс вычисления не останавливается.

Таким образом, среди рекурсивных функций появляются не полностью определенные, т. е. частичные функции; операторы над частичными функциями порождают новые частичные функции. При этом характер неопределенности может оказаться довольно сложным, а именно: для данного набора значений x_1, \dots, x_n не найдется способа установить, определена ли функция f на этом наборе, и нам придется продолжать процесс вычисления неопределенное время, не зная, остановится он или нет.

В случае, когда функция g , к которой применяется μ -оператор, сама является частичной, функция $f(x_1, \dots, x_n) = = \mu y(g(x_1, \dots, x_n, y) = 0)$ вычисляется с учетом следующего условия: если для $y = 0, 1, \dots, i - 1$ $g(x_1, \dots, x_n, y) \neq 0$,

а $g(x_1, \dots, x_n, i)$ не определена, то и $f(x_1, \dots, x_n)$ не определена. Например, функция $f(x) = \mu y[y - x = 5]$ при $x = 1$ не определена (хотя уравнение $y - 1 = 5$ имеет решение $x = 6$), так как функция $y - 1$ не определена при $y = 0$.

Частично-рекурсивная функция называется *общерекурсивной*, если она всюду определена. Из сказанного ранее о неопределенности видно, что не всегда частично-рекурсивную функцию можно эффективным образом додопределить до общерекурсивной. Подробнее об этом — в следующем параграфе.

Тезис Черча. *Связь рекурсивных функций с машинами Тьюринга.* Понятие частично-рекурсивной функции оказалось исчерпывающей формализацией понятия вычислимой функции. (В частности, функция Аккермана общерекурсивна: доказательство этого факта можно найти, например, в [13], ч. III, § 1, задача 42, а.) Это обстоятельство выражено в виде *тезиса Черча*, являющегося аналогом* тезиса Тьюринга для рекурсивных функций: *всякая функция, вычислимая некоторым алгоритмом, частично-рекурсивна.*

Из сопоставления двух тезисов вытекает утверждение: *функция вычислена машиной Тьюринга тогда и только тогда, когда она частично-рекурсивна.* Это утверждение об эквивалентности двух алгоритмических моделей является (в отличие от тезисов) вполне точным математическим утверждением и, следовательно, должно быть доказано. Для доказательства разобьем его на две теоремы.

Теорема 5.6. Всякая частично-рекурсивная функция вычислена на машине Тьюринга.

План доказательства ясен: сначала доказывается, что простейшие функции вычислимы (это довольно очевидно), а затем — что операторы суперпозиции, примитивной рекурсии и μ -оператор сохраняют вычислимость.

Ограничимся построением машины Тьюринга для оператора примитивной рекурсии R_n . Для простоты обозначений рассмотрим случай $n = 1$. Пусть $f(x, y) = R_1(g, h)$, т. е. определена схемой

$$\begin{aligned} f(x, 0) &= g(x); \\ f(x, y + 1) &= h(x, y, f(x, y)), \end{aligned}$$

* Исторически тезис Черча был сформулирован раньше тезиса Тьюринга.

где функции g и h вычислимы машинами T_g и T_h соответственно, работающими с правой полулентой. Построим машину T_f , вычисляющую f . Машина T_f должна воспроизводить процесс вычисления по схеме примитивной рекурсии, т. е. последовательно вычислять $f(x, 0) = g(x)$; $f(x, 1) = h(x, 0, f(x, 0))$; ..., $f(x, i + 1) = h(x, i, f(x, i))$... до тех пор, пока не вычислит $h(x, y, f(x, y))$. Этот процесс, начинающийся с конфигурации $q1^x * 1^y$, разбивается на блоки, выполняющие следующие действия (для каждого блока указана заключительная конфигурация, служащая начальной для следующего блока; состояния не конкретизированы, а символ q указывает положение головки):

1. Подготовить данные для вычисления $g(x)$:

$$\circ 1^x * 1^y \circ q1^x.$$

2. Вычислить $g(x)$ на правой полуленте:

$$\circ 1^x * 1^y \circ q1^{g(x)}.$$

3. Проверить (с восстановлением), верно ли, что $y = 0$. Если да, то блок 7. Если нет, то блок 4.

4. Подготовить данные для вычисления $h(x, i, f(x, i))$ и записать 1 в крайнюю слева пустую ячейку (т. е. прибавить 1 к числу слева от исходных данных, выделенных маркерами \circ):

$$1^{i+1} \circ 1^x * 1^y \circ q1^x * 1^i * 1^{f(x, i)}.$$

5. Вычислить h на правой полуленте:

$$1^{i+1} \circ 1^x * 1^y \circ q1^{h(x, i, f(x, i))}.$$

6. Проверить, верно ли, что $y = i + 1$. Если да, то блок 7. Если нет, то блок 4.

7. Выдать результат (стереть все слева от q , вернуться обратно и остановиться).

Блок 2 реализуется машиной T_g , блок 5 — машиной T_h . Построение блоков 1, 3, 6, 7 очевидно. Блок 4 сдвигает число $1^{h(x, i - 1, f(x, i - 1))} = 1^{f(x, i)}$, полученное на предыдущем шаге блоком 5, на $x + i + 2$ ячейки вправо и записывает в освободившийся пробел слово $1^x * 1^i *$ (число 1^i равно числу, записанному в начале работы блока 4 слева от исходных данных). Похожий процесс осуществлялся с помощью машины $T_{\text{пер}}$ при построении универсальной машины T , поэтому на деталях останавливаться не будем.

Таким образом, машина T_f , реализующая оператор примитивной рекурсии для $n = 1$, построена. Для $n > 1$ все остается таким же, увеличивается лишь число пере-

менных и соответственно маркеров в векторах исходных данных f , g и h . Реализация трех рекурсивных операторов позволяет по рекурсивному описанию любой частично-рекурсивной функции (представлению ее в виде конечной последовательности применений операторов S_m^n , R_n и μ к простейшим функциям) построить реализующую ее машину Тьюринга. \square

Теорема 5.7. Всякая функция, вычислимая на машине Тьюринга, частично-рекурсивна.

Прежде чем доказывать эту теорему, несколько слов в пояснение. На первый взгляд — особенно после рассуждений о том, что алгоритмы могут иметь дело с нечисловыми объектами, и примеров машин Тьюринга, выполняющих нечисловые операции (сдвиг, запись, переписывание и т. д.), — такое утверждение может показаться слишком слабым для того, чтобы говорить об эквивалентности рекурсивных функций, имеющих дело с числами, и машин Тьюринга, которые могут не только вычислять. Однако от любых объектов можно перейти к числовым с помощью кодирования, причем это кодирование оказывается крайне простым. Дело в том, что с точки зрения машины, не вкладывающей в обозреваемые ею символы никакого «смысла», а лишь выполняющей с ними предписанные операции, нет особой разницы между числами и «нечислами». Свои элементарные действия: чтение, запись, сдвиг, смену состояния — машина может выполнять при наличии на ленте как цифр, так и других символов. Разница существует лишь для нас при интерпретации полученных результатов. В частности, ничто не мешает интерпретировать любой алфавит ленты $A = \{a_1, \dots, a_m\}$ как множество цифр m -ичной системы счисления, а слово, записанное на ленте, — как m -ичное число. Тогда любая деятельность машины Тьюринга рассматривается как преобразование чисел, т. е. как вычисление числовой функции. Именно такая интерпретация и подразумевается в теореме 5.7; доказательство теоремы заключается в том, чтобы точно описать эту интерпретацию (называемую арифметизацией) и показать, что любое преобразование, реализуемое машиной Тьюринга, если его интерпретировать как вычисление, является частично-рекурсивным.

Переходим к доказательству теоремы 5.7. Сначала опишем арифметизацию машин Тьюринга. Как уже указывалось, символы на ленте интерпретируются как m -ичные цифры (например, в порядке их перечисления в алфавите, т. е. a_i кодируется цифрой i). При этом символу

λ всегда ставится в соответствие 0. Поскольку непустых символов на ленте в любой момент — конечное число, то соглашение позволяет иметь дело только с конечными числами. Для определенности (и простоты обозначений) при иллюстрациях будем считать, что $A = \{1, \lambda\}$, а вместе λ будем писать 0. Состоянию q_i машины поставим в соответствие число i ; буквой z обозначим код заключительного состояния; сдвиги закодируем так: $R = 1, L = 0$ (как было показано в § 5.2, случай с E можно не рассматривать). Символы ленты, состояния и сдвиги не будем отличать от их кодов. Система команд машины T с множеством состояний Q и алфавитом A превращается при таком кодировании в тройку числовых функций $\varphi_q: Q \times A \rightarrow Q$ (функция следующего состояния), $\varphi_a: Q \times A \rightarrow A$ (функция печатаемого символа), $\varphi_d: Q \times A \rightarrow \{0, 1\}$ (функция сдвига). Если Σ_T содержит команду $q_i a_j \rightarrow q'_i a'_j d_k$, то $\varphi_q(q_i, a_j) = q'_i$; $\varphi_a(q_i, a_j) = a'_j$; $\varphi_d(a_i, a_j) = d_k$. Отметим, что все эти функции заданы на конечном множестве $Q \times A$ и, следовательно, примитивно-рекурсивны.

Каждой конфигурации $aq_i a_j \beta$ однозначно ставится в соответствие четверка чисел $(\tilde{\alpha}, \tilde{q}_i, \tilde{a}_j, \tilde{\beta})$, определяемых следующим образом: $\tilde{q}_i = i$; $\tilde{a}_j = j$; $\tilde{\alpha}$ — число, изображаемое цифрами, полученными кодированием символов слова α , $\tilde{\beta}$ аналогично получается из β , но при этом читается справа налево, т. е. крайняя слева ячейка β содержит самый младший разряд, а крайняя справа — самый старший (это сделано для того, чтобы нули справа от β не влияли на значение $\tilde{\beta}$). Например, конфигурации $10110q_511011$ соответствует четверка $(22, 5, 1, 13)$. Выполнение команды преобразует конфигурацию K в следующую конфигурацию K' ; при арифметизации этому соответствует преобразование четверки в новую четверку $(\tilde{\alpha}', \tilde{q}'_i, \tilde{a}'_j, \tilde{\beta}')$. Например, при команде $q_5 1 \rightarrow q_3 0 R$ ранее проведенная конфигурация перейдет в $K' = 101100q_31011$, которой соответствует четверка $(44, 3, 1, 6)$. Поэтому один шаг машины T однозначно определяет отображение множества конфигураций в себя, т. е. нечисловую функцию $\psi_T(K) = K'$. Назовем ее функцией следующего шага. При введенной арифметизации этой функции соответствует четверка числовых функций следующего шага; иначе говоря, $\tilde{\alpha}', \tilde{q}', \tilde{a}', \tilde{\beta}'$ — это числовые функции, каж-

дая из которых зависит от четырех числовых переменных $\tilde{\alpha}$, \tilde{q} , \tilde{a} , $\tilde{\beta}$. Эти функции довольно простым образом связаны с функциями системы команд φ_q , φ_a , φ_d . Действительно, $q' = \varphi_q$; другие функции зависят еще и от сдвига. Если головка сдвинулась вправо, то это означает сдвиг цифр на разряд влево, т. е. увеличение числа $\tilde{\alpha}$ в m раз (напомним, что m — мощность алфавита A), и сдвиг числа $\tilde{\beta}$ на разряд вправо (так как они читаются справа налево), т. е. уменьшение числа $\tilde{\beta}$ в m раз. Кроме того, в младший разряд $\tilde{\alpha}$ записывается символ, напечатанный на данном шаге, а младший разряд $\tilde{\beta}$ становится обозреваемым символом. При сдвиге головки влево роли $\tilde{\alpha}$ и $\tilde{\beta}$ меняются; $\tilde{\alpha}$ уменьшается, $\tilde{\beta}$ увеличивается и т. д. Поэтому

$$\tilde{\alpha}'(\tilde{\alpha}, \tilde{q}_i, \tilde{a}_j, \tilde{\beta}) = \begin{cases} m\tilde{\alpha} + \varphi_a(q_i, a_j), & \text{если } \varphi_d(q_i, a_j) = 1 \\ (\text{сдвиг вправо}); \\ [\tilde{\alpha}/m], & \text{если } \varphi_d(q_i, a_j) = 0 \\ (\text{сдвиг влево}). \end{cases} \quad (5.11a)$$

$$\tilde{q}_i = \varphi_q(q_i, a_j); \quad (5.11b)$$

$$\tilde{a}' = \begin{cases} r(m, \tilde{\beta}), & \text{если } \varphi_d(q_i, a_j) = 1; \\ r(m, \tilde{\alpha}), & \text{если } \varphi_d(q_i, a_j) = 0; \end{cases} \quad (5.11b)$$

$$\tilde{\beta}' = \begin{cases} m\tilde{\beta} + \varphi_d(q_i, a_j), & \text{если } \varphi_d(q_i, a_j) = 0; \\ [\tilde{\beta}/m], & \text{если } \varphi_d(q_i, a_j) = 1. \end{cases} \quad (5.11c)$$

Функции $[]$ и r , использованные здесь, определены в примере 5.11. Поскольку функции $\tilde{\alpha}'$, \tilde{a}' , $\tilde{\beta}'$ построены из примитивно-рекурсивных функций с помощью примитивно-рекурсивного оператора условного перехода, а функция \tilde{q}' просто совпадает с примитивно-рекурсивной функцией φ_q , то все они — примитивно-рекурсивные функции от четырех переменных $\tilde{\alpha}$, \tilde{q} , \tilde{a} , $\tilde{\beta}$. Можно убедиться, что применение соотношений (5.11) к приведенной ранее в качестве примера четверке (22, 5, 1, 13) и команде $q_5 1 \rightarrow q_3 0 R$ даст четверку (44, 3, 1, 6); напомним, что $m = 2$.

Итак, доказано, что на каждом шаге любая машина Тьюринга осуществляет примитивно-рекурсивное вычисление. Переходим теперь к арифметизации общего поведения машины.

Пусть задана начальная конфигурация $K(0) = (\tilde{\alpha}_0, \tilde{q}_0, \tilde{a}_0, \tilde{\beta}_0)$. Тогда конфигурация, возникающая на такте t , зависит от величины t и компонент $\tilde{\alpha}_0, \tilde{q}_0, \tilde{a}_0, \tilde{\beta}_0$ начальной конфигурации, иначе говоря, она является векторной функцией $K(t) = (K_\alpha(t), K_q(t), K_a(t), K_\beta(t))$, компоненты которой — векторные функции, зависящие от пяти переменных $t, \tilde{\alpha}_0, \tilde{q}_0, \tilde{a}_0, \tilde{\beta}_0$. Эти функции определяются следующим образом:

$$\begin{aligned} K_\alpha(0, \tilde{\alpha}_0, \tilde{q}_0, \tilde{a}_0, \tilde{\beta}_0) &= \tilde{\alpha}_0; \\ K_\alpha(t + 1, \tilde{\alpha}_0, \tilde{q}_0, \tilde{a}_0, \tilde{\beta}_0) &= \tilde{\alpha}'(K_\alpha(t, \tilde{\alpha}_0, \tilde{q}_0, \tilde{a}_0, \tilde{\beta}_0)), \quad (5.12a) \\ K_q(t, \tilde{\alpha}_0, \tilde{q}_0, \tilde{a}_0, \tilde{\beta}_0), K_a(t, \tilde{\alpha}_0, \tilde{q}_0, \tilde{a}_0, \tilde{\beta}_0), \\ K_\beta(t, \tilde{\alpha}_0, \tilde{q}_0, \tilde{a}_0, \tilde{\beta}_0)). \end{aligned}$$

В соотношениях (5.12б)–(5.12г) аргументы $\tilde{\alpha}_0, \tilde{q}_0, \tilde{a}_0, \tilde{\beta}_0$ в функциях K_i для краткости опустим:

$$\begin{aligned} K_q(0) &= \tilde{q}_0; \\ K_q(t + 1) &= \tilde{q}'(K_\alpha(t), K_q(t), K_a(t), K_\beta(t)); \quad (5.12б) \end{aligned}$$

$$\begin{aligned} K_a(0) &= \tilde{a}_0; \\ K_a(t + 1) &= \tilde{a}'(K_\alpha(t), K_q(t), K_a(t), K_\beta(t)); \quad (5.12в) \end{aligned}$$

$$\begin{aligned} K_\beta(0) &= \tilde{\beta}_0 \\ K(t + 1) &= \tilde{\beta}'(K_\alpha(t), K_q(t), K_a(t), K_\beta(t)). \quad (5.12г) \end{aligned}$$

Соотношения (5.12) формально выражают то очевидное обстоятельство, что координаты вектора $K(t + 1)$ однозначно определяются координатами вектора $K(t)$. Эти соотношения представляют собой схему примитивной рекурсии, определяющую четыре функции $K_\alpha, K_q, K_a, K_\beta$ (рекурсия ведется по переменной t) с помощью функций $\tilde{\alpha}', \tilde{q}', \tilde{a}', \tilde{\beta}'$, примитивная рекурсивность которых уже доказана. Следовательно, функции K_i также примитивно-рекурсивны.

Остался последний шаг доказательства: рекурсивное описание результата работы машины Тьюринга. Ограничимся для простоты правильно вычисляющими машинами (которые начинают с конфигураций вида $q_1 a_0 \beta_0$ и останавливаются в конфигурациях вида $q_z a_z \beta_z$). Для таких машин $K_\alpha(0) = 0, K_q(0) = 1$, исходное слово на ленте кодируется числом $m\beta_0 + a_0$, заключительное слово — числом $m\tilde{\beta}_z + a_z$. Поэтому результат работы машины —

это функция $f(m\tilde{\beta}_0 + a_0) = m\tilde{\beta}_z + a_z$. Заключительное слово — это слово, написанное на ленте в тот момент t_z , когда машина впервые перешла в заключительное состояние q_z , т. е. в момент $t_z = \mu t(K(t) = z)$. Поэтому $\tilde{\beta}_z = K_\beta(t_z)$, $\tilde{a}_z = K_a(t_z)$ и $f(m\tilde{\beta}_0 + \tilde{a}_0) = mK_\beta(t_z, 0, 1, \tilde{a}_0, \tilde{\beta}_0) + K_a(t_z, 0, 1, \tilde{a}_0, \tilde{\beta}_0)$.

Если придать f стандартный вид $f(x)$, учитывая при этом, что $\tilde{\beta}_0 = [x/m]$, $\tilde{a}_0 = r(m, x)$, и выписать все аргументы функций K , получаем:

$$\begin{aligned} f(x) &= mK_\beta(\mu t(K_q(t, 0, 1, r(m, x), [x/m]) = z), \\ &0, 1, r(m, x), [x/m]) + K_a(\mu t(K_q(t, 0, 1, r(m, x), [x/m]) = z), \\ &0, 1, r(m, x), [x/m]) \end{aligned} \quad (5.13)$$

(m, z — константы, зависящие от конкретной машины), откуда непосредственно видно, что функция f , описывающая результат работы машины Тьюринга, построена из примитивно-рекурсивных функций с помощью оператора μ и, следовательно, является частично-рекурсивной. \square

Из теорем 5.6, 5.7 и соотношения (5.13) следует теорема Клини о нормальной форме — теорема 5.8.

Теорема 5.8. Любая частично-рекурсивная функция $f(x)$ представима в виде

$$f(x) = F(\mu y[G(x, y) = 0]),$$

где F и G — примитивно-рекурсивные функции. \square

Таким образом, класс частично-рекурсивных функций оказался эквивалентным классу функций, вычисляемых машинами Тьюринга. Эквивалентность этих классов, в основе построения которых лежали совершенно различные подходы, косвенным образом подтверждает справедливость тезисов Черча и Тьюринга и позволяет объединить их в один тезис Черча—Тьюринга. Кроме того, она дает возможность формулировать основные результаты теории алгоритмов в более общем виде. Об этом — в § 5.4.

5.4. ВЫЧИСЛИМОСТЬ И РАЗРЕШИМОСТЬ

Итак, всякий алгоритм, описанный в терминах частично-рекурсивных функций, можно реализовать машиной Тьюринга, и наоборот. Отсюда следует, что любые утверждения о существовании или несуществовании алгоритмов, сделанные в одной из этих теорий,

верны и в другой. В сочетании с тезисом Черча–Тьюринга это означает, что такие утверждения можно формулировать для алгоритмов вообще, не фиксируя конкретную модель и используя результаты обеих теорий. Таким образом, возможно изложение теории алгоритмов, инвариантное по отношению к способу формализации понятия «алгоритм», — при любой формализации основные свойства алгоритмов остаются теми же самыми*. Основные понятия такой инвариантной теории (будем называть ее общей теорией алгоритмов) — это *алгоритм* (рекурсивное описание функции, система команд машины Тьюринга или описание в какой-либо другой модели; считается, что выбрана какая-то модель, но какая именно — неважно) и *вычислимая функция*. Функция называется вычислимой, если существует вычисляющий ее алгоритм. При этом несущественно, числовая функция это или нет. Термин «общерекурсивная функция», употребленный в инвариантном смысле, является синонимом термина «всюду определенная вычислимая функция».

Эквивалентность утверждений «функция f вычислима» и «существует алгоритм, вычисляющий функцию f » иногда приводит к смешению понятий алгоритма и вычислимой функции; в частности, говоря о рекурсивной функции, часто имеют в виду ее конкретное рекурсивное описание. В действительности же различие между вычислимой функцией и алгоритмом — это различие между функцией и способом ее задания. Без соблюдения этих различий невозможна корректная интерпретация некоторых важных результатов теории алгоритмов. В то же время традиция изложения теории алгоритмов позволяет не различать понятия алгоритма и функции в тех случаях, когда это не приводит к путанице.

В этом параграфе резюмируются уже полученные и формулируются некоторые новые понятия и результаты теории алгоритмов именно в их инвариантном виде. Они верны для любой формализации понятия «алгоритм», что не мешает использовать в каждом конкретном рас-

* Это верно, когда речь идет о существовании или несуществовании алгоритмов. Характеристики качества алгоритмов (их сложности в том или ином смысле), вообще говоря, неинвариантны по отношению к выбранной формализации.

суждении конкретную алгоритмическую модель, наиболее подходящую для данного случая.

Отступление 5.1. Для читателя этой книги, который в конечном счете интересуется прикладной стороной излагаемых здесь теорий, важно отметить следующее. Ввиду инвариантности основных результатов общей теории алгоритмов прикладное значение ее результатов никак не связано с тем, насколько близки к практике используемые в ней теоретические модели алгоритмов. Машины Тьюринга весьма далеки от современных компьютеров, а рекурсивные функции — от языков программирования прежде всего из-за предельной скромности используемых средств. Однако именно скромность средств, во-первых, делает эти модели чрезвычайно удобным языком доказательств, а во-вторых, позволяет понять, без чего обойтись нельзя, а без чего можно и какой ценой, т. е. отличать удобства от принципиальных возможностей. Иначе говоря, прикладное значение рассматриваемых здесь моделей заключается в том, что с их помощью удобно строить теорию, верную для любых алгоритмических моделей, в том числе и для сколь угодно близких к практике.

Нумерация алгоритмов. Множество всех алгоритмов счетно. Этот факт уже отмечался в § 5.3, однако он ясен и без обращения к конкретным алгоритмическим моделям: ведь любой алгоритм можно задать конечным описанием (например, в алфавите знаков, используемых при наборе математических книг), а множество всех конечных слов в фиксированном алфавите счетно. Счетность множества алгоритмов означает наличие взаимно однозначного соответствия между алгоритмами и числами натурального ряда, т. е. функции типа $\phi: N \rightarrow Al^*$, взаимно однозначно отображающей в слова в алфавите Al , выбранном для описания алгоритмов, числа натурального ряда. Такая функция $\phi(n) = A$ называется *нумерацией* алгоритмов, а ее аргумент n — номером алгоритма A при нумерации ϕ . Из взаимной однозначности отображения ϕ следует существование обратной функции $\phi^{-1}(A_n) = n$, восстанавливающей по описанию алгоритма A_n его номер n . Более общие определения нумераций см. в [17].

Фактически мы уже построили одну нумерацию: алфавит Al и способ представления алгоритмов в этом алфавите выбраны при построении универсальной машины U , а функция ϕ определяется методом числового кодирования

конфигураций, использованным при доказательстве теоремы 5.7. Правда, при таком определении нумерация ϕ некоторые натуральные числа декодирует в слова, не являющиеся записью никакого алгоритма. Эти числа можно считать номерами нигде не определенных алгоритмов, а соответствующую универсальную машину определить так, чтобы на этих словах она зацикливалась.

Приведенный пример показывает, что существуют вычислимые нумерации. Очевидно, что различных нумераций много; мы не будем заниматься их особенностями, а договоримся, что зафиксирована какая-то вычислимая нумерация. Основной интерес для приложений теории алгоритмов представляют инвариантные свойства номеров алгоритмов, не зависящие от особенностей выбранной нумерации. Нумерация всех алгоритмов является одновременно и нумерацией всех вычислимых функций в следующем смысле: номер функции f — это номер некоторого алгоритма, вычисляющего f . В любой нумерации всякая функция будет иметь бесконечное множество различных номеров. Это ясно из того, что к любой машине Тьюринга можно добавить любое количество недостижимых состояний (т. е. состояний, не встречающихся в правых частях команд); все полученные машины имеют различные номера, но вычисляют одну и ту же функцию. Говоря о нумерации, будем считать, что она выбрана так, что A_0 и вычисляемая им функция f_0 нигде не определены.

Существование нумераций позволяет работать с алгоритмами как с числами. Это особенно удобно при исследовании алгоритмов над алгоритмами. Такие алгоритмы уже рассматривались при построении универсальной машины Тьюринга и в связи с проблемой остановки. Полученные результаты теперь можно сформулировать в инвариантном виде.

Теорема 5.9. Существует универсальный алгоритм $U(x, y)$, такой, что для любого алгоритма A с номером $\phi^{-1}(A)$ $U(\phi^{-1}(A), y) = A(y)$; в других обозначениях $U(x, y) = A_x(y)$.

Для конкретной нумерации ϕ^* эта теорема доказана в § 5.2 построением универсальной машины Тьюринга. Для любой другой вычислимой нумерации ϕ можно выбрать один из двух путей: а) построить новый универсальный

алгоритм, работающий непосредственно с номерами, порождаемыми φ ; б) построить алгоритм перевода (взаимно однозначного отображения) φ в φ^* . \square

По существу вычислимая нумерация служит языком программирования для универсального алгоритма. Путь «а» — это построение новой машины для каждого нового языка, путь «б» — построение нового транслятора для прежней машины.

Теорема 5.4'. Не существует алгоритма, который по номеру x любого алгоритма и исходным данным y определял бы, остановится алгоритм при этих данных или нет; иначе говоря, не существует алгоритма $B(x, y)$, такого, что для любого алгоритма A_x (с номером $\varphi^{-1}(A) = x$)

$$B(x, y) = \begin{cases} 1, & \text{если } A_x(y) \text{ определен;} \\ 0, & \text{если } A_x(y) \text{ не определен.} \end{cases}$$

Эта теорема — переформулировка в инвариантном виде теоремы 5.4 о неразрешимости проблемы остановки. \square

Один частный случай проблемы остановки имеет вполне самостоятельную интерпретацию. При доказательстве теоремы 5.4 была рассмотрена машина T_1 , которая решала проблему остановки для машины T в случае, когда на ленте машины T написана ее собственная система команд. Такая проблема называется *проблемой самоприменимости* машин Тьюринга. Было показано, что такая машина невозможна. В инвариантном виде соответствующее утверждение формулируется так.

Теорема 5.10. Проблема самоприменимости алгоритмов алгоритмически неразрешима: не существует алгоритма $B_1(x)$, такого, что для любого алгоритма A_x

$$B_1(x) = \begin{cases} 1, & \text{если } A_x(x) \text{ определен;} \\ 0, & \text{если } A_x(x) \text{ не определен.} \end{cases} \quad (5.14)$$

Отметим, что самоприменимость — частный случай проблемы остановки, и именно поэтому теорему 5.10 нельзя получить из теоремы 5.4' простой подстановкой x вместо y в $B(x, y)$: *частный случай алгоритмически неразрешимой проблемы может оказаться и разрешимым*.

Теоремы 5.4' и 5.10 являются мощным средством для доказательства различных неразрешимостей.

Решение задачи перечисления всех алгоритмов (и, в частности, всех рекурсивных функций) достаточно ясно, по крайней мере в принципе. Могло бы показаться, что перечисление примитивно-рекурсивных или общерекурсивных функций окажется более легким делом. Однако это не так.

Теорема 5.11. Для любого перечисления любого множества Σ всюду определенных вычислимых (т. е. общерекурсивных) функций существует общерекурсивная функция, не входящая в это перечисление.

Пусть ψ — перечисление Σ , порождающее последовательность $A_0, A_1, A_2\dots$ всюду определенных функций.

Введем функцию $B(x) = A_x(x) + 1$. Так как A_x общерекурсивна, то и $B(x)$ общерекурсивна. Если $B \in \Sigma$, то B имеет номер x_B и, следовательно,

$$B(x) = A_{xB}(x). \quad (5.15)$$

Тогда в точке $x = x_B$ по определению $B(x_B) = A_{xB}(x_B) + 1$, а в силу (5.15) $B(x_B) = A_{xB}(x_B)$. Получаем противоречие, откуда следует, что B не входит в перечисление, порождаемое ψ . \square

Если же в перечислении допускаются частичные функции, то определение B не приводит к противоречию, а означает лишь, что в точке x_B B не определена.

Из теоремы 5.11 следует весьма важный результат.

Теорема 5.12. Проблема определения общерекурсивности алгоритмов неразрешима; не существует алгоритма $B(x)$, такого, что для любого алгоритма A_x

$$B(x) = \begin{cases} 1, & \text{если } A_x \text{ всюду определен;} \\ 0, & \text{если } A_x \text{ не всюду определен.} \end{cases}$$

Пусть алгоритм $B(x)$ существует; тогда он определяет общерекурсивную функцию $f(x)$. Определим функцию $g(x)$ следующим образом:

$$\begin{aligned} g(0) &= \mu y[f(y) = 1]; \\ g(x+1) &= \mu y[y > g(x) \& f(y) = 1]. \end{aligned}$$

Так как номеров всюду определенных функций (и, следовательно, точек y , в которых $f(y) = 1$) — бесконечное множество, то $g(x)$ всюду определена. Очевидно, что функция g из списка $A_0, A_1, A_2\dots$ всех алгоритмов отбирает все

всюду определенные алгоритмы, т. е. является перечислением множества всех общерекурсивных функций. Из теоремы 5.11 следует, что такое перечисление невозможно и, следовательно, алгоритма $B(x)$, определенного в условии теоремы 5.12, не существует. \square

Теоремы 5.4', 5.11 и 5.12 проливают свет на роль понятия частичной определенности в теории алгоритмов. Еще в § 5.1 среди требований к алгоритмам говорилось о желательности такого требования, как результативность алгоритма*. Первым ударом по этому требованию была неразрешимость проблемы остановки, означающая, что если алгоритм A может быть частичным, то по алгоритму A и данным x нельзя узнать, даст A результат на данных x или нет. Возникает естественное желание либо вообще убрать частичные алгоритмы из общей теории алгоритмов (скажем, не считать их алгоритмами), либо ввести стандартный метод доопределения частичных алгоритмов. Однако ни первое, ни второе эффективными методами сделать нельзя. Первая идея не годится из-за теоремы 5.12 — нет эффективного способа распознавать частичные алгоритмы среди множества всех алгоритмов, и, следовательно, предлагаемый отбор невозможен. Что же касается второй идеи, то для нее имеется не менее убедительное опровержение.

Теорема 5.13. Существует такая частично-рекурсивная функция f , что никакая общерекурсивная функция g не является ее доопределением.

Определим f следующим образом:

$$f(x) = \begin{cases} A_x(x) + 1, & \text{если } A_x(x) \text{ определен;} \\ \text{не определена}, & \text{если } A_x(x) \text{ не определен.} \end{cases}$$

Как и прежде, предполагается, что зафиксирована нумерация ϕ и $\phi^{-1}(A_x) = x$. Очевидно, что $f(x)$ вычислима. Пусть теперь g — произвольная общерекурсивная функция и x_g — ее номер: $\phi^{-1}(g) = x_g$. Так как g всюду определена, то $g(x_g) = A_{x_g}(x_g)$ определена и, следовательно, $f(x_g) = g(x_g) + 1$. Таким образом, для любой общерекурсивной функции g $f \neq g$ в точке x_g . \square

* Отметим, что существенность этого требования проявляется в понятии неразрешимости; неразрешимость проблемы истолковывается как отсутствие *всюду* определенного алгоритма, решающего эту проблему.

Следовательно, существуют частичные алгоритмы, которые нельзя доопределить до всюду определенных алгоритмов.

Наконец, еще одна идея: построить язык, описывающий все всюду определенные алгоритмы и только их, — не может осуществиться потому, что описания в этом языке можно упорядочить (например, лексикографически), и, следовательно, наличие такого языка означало бы существование полного перечисления всех всюду определенных функций, что противоречит теореме 5.11.

Таким образом, при формулировке общего понятия алгоритма неизбежно возникает дилемма — либо определение алгоритма должно быть достаточно общим, чтобы в число объектов, удовлетворяющих этому определению, заведомо вошли все объекты, которые естественно считать алгоритмами, либо требование об обязательной результативности алгоритма сохраняется. В первом случае этому определению будут удовлетворять частичные алгоритмы, и избавиться от них конструктивными методами нельзя; во втором случае никакую процедуру нельзя называть алгоритмом до тех пор, пока для нее не будет решена проблема остановки, а единого метода решения этой проблемы не существует. В общей теории алгоритмов (для того, чтобы она действительно была общей) используется первый вариант. Впрочем, еще раз отметим, что, когда речь идет об алгоритме, *решающем данную задачу*, теория алгоритмов обязательно требует сходимости (результативности), и только в случае, когда алгоритм решения всюду определен, соответствующая задача считается разрешимой. В этом же смысле используется термин «разрешимость» при введении одного из важнейших понятий теории алгоритмов — понятия разрешимого множества.

Разрешимые и перечислимые множества. Множество M называется *разрешимым* (или рекурсивным), если существует алгоритм A_M , который по любому объекту a дает ответ, принадлежит a множеству M или нет. Алгоритм A_M называется разрешающим алгоритмом для M .

Эквивалентное, но несколько более точное определение: множество M называется разрешимым, если оно

обладает общерекурсивной характеристической функцией, т. е. вычислимой всюду определенной функцией χ_M , такой, что

$$\chi_M(a) = \begin{cases} 1, & \text{если } a \in M; \\ 0, & \text{если } a \notin M. \end{cases}$$

Следующее определение удобнее дать сразу в функциональных терминах. Множество M называется *перечислимым* (или рекурсивно-перечислимым), если оно является областью значений некоторой общерекурсивной функции, т. е. существует общерекурсивная функция $\psi_M(x)$, такая, что $a \in M$, если и только если для некоторого x $a = \psi_M(x)$. Функция ψ_M называется *перечисляющей* для множества M ; соответственно алгоритм, вычисляющий ψ_M , называется перечисляющим или порождающим для M .

Пример 5.15. а. Множество квадратов натуральных чисел $M\{a \mid a = x^2\}$ перечислимо (оно просто задано перечисляющей функцией $a = x^2$) и разрешимо. В качестве разрешающей процедуры можно предложить, например, следующую: данное число a сравниваем последовательно с $0, 1^2, 2^2\dots$ и т. д. до тех пор, пока не появится такое число n^2 , что либо $a = n^2$, либо $a < n^2$. В первом случае $a \in M$, во втором $a \notin M$.

б. Множество M_π из гл. 1, т. е. множество всех троек цифр, встречающихся в десятичном разложении π , перечислимо. Перечисляющая процедура для него заключается в последовательном вычислении знаков разложения π (один из алгоритмов их вычисления известен всем еще из школьного курса геометрии) и выписывании троек из получающейся последовательности; для определенной таким образом функции ψ^π имеем: $\psi_\pi = 314$, $\psi_\pi(1) = 159$, $\psi_\pi(2) = 265$ и т. д. Некоторые тройки могут повторяться, поэтому ψ_π = это пример функции, перечисляющей с повторениями. Автору неизвестно (по крайней мере, к моменту написания этой книги) никакого разрешающего алгоритма для M_π , несмотря на то, что очевидна конечность M_π ! С другой стороны, это вовсе не означает, что M_π неразрешимо; оно может оказаться и разрешимым (например, через достаточно большое количество шагов перечисляющей процедуры окажется, что выписаны все возможные тройки от 000 до 999).

в. Из индуктивных определений, как правило, не трудно извлечь порождающую процедуру. Иногда она содержится в них явно, как, скажем, в определении $(n + 1)! = n!(n + 1)$, задающем и перечисляющей функцию $\psi(n) = n!$, и способ ее вычисления. Иногда, чтобы получить из такого определения порождающую функцию, нужно ввести дополнительные соглашения. Например, известное индуктивное определение «идентификатор — это либо буква, либо идентификатор, к которому справа приписана буква или цифра» порождает все возможные идентификаторы; однако для того, чтобы придать этому процессу порождения вид перечисляющей функции, нужно ввести порядок перечисления. Если в качестве такого порядка принять лексикографический и договориться, что буквы в этом порядке предшествуют цифрам, то в алфавите из 26 латинских букв и 10 цифр $\psi(0) = a$, $\psi(25) = z$, $\psi(51) = az$, $\psi(54) = a2$ и т. д. С помощью аналогичных соглашений на основе индуктивного определения формулы (см., например, § 3.1, 6.1) можно построить перечисляющие функции для различных множеств формул.

г. Из предыдущего примера видно, что перечисляющая функция, определенная на натуральном ряду, устанавливает соответствие (не обязательно взаимно однозначное) между числами и элементами порождаемого множества. Например, идентификатору az соответствует число 51. Функция ϕ , нумерующая алгоритмы (т. е. нумерация на стр. 202), устанавливает соответствие между описаниями алгоритмов и числами, причем в нашем случае соответствие взаимно однозначное. Поэтому можно сказать, что нумерация ϕ , определенная на стр. 202, является перечисляющей функцией без повторений для множества всех алгоритмов.

д. Множество всех тождественно-истинных булевых формул разрешимо (например, путем прямого вычисления на всех наборах).

Важность введенных понятий разрешимости и перечислимости — прежде всего в том, что благодаря им становятся точными такие понятия, как «конструктивный способ задания множества» и «множество, заданное эффективно», которые неформально обсуждались

еще в гл. 1. В частности, поскольку язык множеств является универсальным языком математики и всякому утверждению можно придать вид утверждения о множествах, язык разрешимых и перечислимых множеств является универсальным языком для утверждений о существовании (или отсутствии) алгоритмов решения математических проблем. Например, теорема 5.4 утверждает, что множество пар (x, y) , таких, что алгоритм A_x при данных y дает результат, неразрешимо, а теорема 5.12 — что неразрешимо множество всех обще рекурсивных функций.

Итак, эффективно заданное множество — это множество, обладающее разрешающей или перечисляющей функцией. Однако возникает вопрос, равносильны ли эти два типа задания. В одну сторону ответ почти очевиден.

Теорема 5.14. Если непустое множество M разрешимо, то оно перечислимо.

Для определенности будем считать, что M — это подмножество слов в алфавите A . Пусть $\alpha_1, \alpha_2, \dots, \alpha_n \dots$ — перечисление всех слов в алфавите A (например, в лексикографическом порядке), β — некоторое слово из M . Перечисляющую функцию $\psi_M(n)$ для M определим так:

$$\psi_M(0) = \beta;$$

$$\psi_M(n+1) = \begin{cases} \psi_M(n), & \text{если } \alpha_{n+1} \notin M; \\ \alpha_{n+1}, & \text{если } \alpha_{n+1} \in M. \end{cases}$$

Ясно, что $\psi_M(n)$ вычислима и всюду определена. \square

Обращение теоремы 5.14 неверно.

Теорема 5.15. Существует множество M , которое перечислимо, но неразрешимо.

Докажем, что таковым является множество $M = \{x \mid A_x(x) \text{ определен}\}$, т. е. множество номеров самоприменимых алгоритмов. Из теоремы 5.9 следует, что M неразрешимо. Построим перечисляющую функцию ψ_M для M . Будем считать для определенности, что алгоритмы — это машины Тьюринга. Введем предикат $St(x, m)$ — истинный, если машина с номером x при данных x останавливается через m шагов. Очевидно, что этот предикат вычислим и всюду определен; для его вычисления надо запустить машину T_x при данных x и дать ей проработать m шагов. Предикат $St(x, m)$ можно представить

x	m			
	0	1	2	...
0	St(0, 0)	St(0, 1)	St(0, 2)	...
1	St(1, 0)
2	St(2, 0)
...

в виде бесконечной квадратной матрицы (табл. 5.3), строки которой соответствуют первому аргументу x , столбцы — второму аргументу m , а в клетке (i, j) , т. е. на пересечении i -й строки и j -го столбца, стоит значение $St(i, j)$.

Упорядочим теперь клетки этой матрицы подобно тому, как упорядочивалось множество пар натуральных чисел в гл. 1 при доказательстве его счетности: $(0, 0)$, $(0, 1)$, $(1, 0)$, $(0, 2)$, $(1, 1)$, $(2, 0)$, $(0, 3)$..., т. е. сначала клетки с суммой координат 0, затем клетки с суммой 1 и т. д. Тогда каждая клетка получит номер, соответствующий ее порядку в этой последовательности. Выберем какую-нибудь заведомо самоприменимую машину Тьюринга (например, любую всюду определенную машину из примеров § 5.2), вычислим ее номер во введенной нумерации и обозначим его c . Определим теперь алгоритм вычисления функции $\psi_M(n)$: находим клетку (i, j) с номером n ; если $St(i, j) = И$, то $\psi_M(n) = i$; если $St(i, j) = Л$, то $\psi_M(n) = c$. Так как $\psi_M(n) = i$ — это номер машины T_i , останавливающейся при данных i через j шагов, то область значений ψ_M содержит только номера самоприменимых машин. С другой стороны, всякая самоприменимая машина T_x остановится через конечное число k шагов; но тогда $\psi_M(n') = x$, где n' — это номер клетки (x, k) . Таким образом, область значений ψ_M совпадает с множеством номеров самоприменимых алгоритмов; кроме того, ввиду общерекурсивности предиката St функция ψ_M также общерекурсивна. Следовательно, она является перечисляющей функцией для M . \square

Согласно теореме 5.15 перечислимость — более слабый вид эффективности; хотя перечисляющая процедура и задает эффективно список элементов множества M , однако поиск данного элемента a в этом списке (всегда

бесконечном, но, может быть, с повторяющимися элементами) может оказаться неэффективным: это неопределенно долгий процесс, который в конечном счете остановится, если $a \in M$, но не остановится, если $a \notin M$. Поэтому список элементов M , заданный перечисляющей функцией, сам по себе не гарантирует разрешающей процедуры для M ; теорема 5.15 дает пример, когда ее просто не существует.

Впрочем, в одном важном случае перечислимость гарантирует разрешимость.

Теорема 5.16. M разрешимо, если и только если M и \tilde{M} перечислимы.

Действительно, если M разрешимо, то \tilde{M} также разрешимо: $\psi_{\tilde{M}} = 1 - \chi_M$, но тогда перечислимость M и \tilde{M} следует из теоремы 5.14.

Пусть теперь M и \tilde{M} перечислимы с функциями ψ_M и $\psi_{\tilde{M}}$ соответственно. Но тогда для любого элемента a его поиск в обоих списках, точнее, в объединенном списке $\psi_M(0), \psi_{\tilde{M}}(0), \psi_M(1), \psi_{\tilde{M}}(1) \dots$ обязательно увенчается успехом, так как либо $a \in M$, либо $a \in \tilde{M}$. Такой поиск и есть разрешающая процедура для M . \square

Сопоставление теорем 5.15 и 5.16 показывает, что дополнение к перечислимому множеству может оказаться неперечислимым; в частности, множество несамоприменимых алгоритмов неперечислимо. Такая «несимметрия» самоприменимости и несамоприменимости объясняется отмеченной ранее несимметрией положительного и отрицательного ответа при поиске заданного элемента в бесконечном списке: если A_x самоприменим, то, вычисляя $A_x(x)$, мы это когда-нибудь узнаем; если же A_x несамоприменим, то об этом нельзя узнать, вычисляя $A_x(x)$.

Теорема Райса. Заключительный комментарий. Присматривая накопленный запас алгоритмически неразрешимых проблем, нетрудно заметить, что почти все они так или иначе связаны с самоприменимостью — довольно экзотической ситуацией, когда алгоритм работает с собственным описанием. Читатель, интересующийся прикладной теорией алгоритмов, может решить, что поскольку понятие самоприменимости далеко от алгоритмической практики, то неразрешимость в этой практике также никогда не встретится. Такого читателя ждет горькое разочарование.

Теорема 5.17 (Райс). Никакое нетривиальное свойство вычислимых функций не является алгоритмически разрешимым.

Для доказательства эту теорему удобнее сформулировать в менее эффектном, но более точном виде: пусть C — любой класс вычислимых функций одной переменной, нетривиальный в том смысле, что имеются как функции, принадлежащие C , так и функции, не принадлежащие C . Тогда не существует алгоритма, который бы по номеру x функции f_x определял бы, принадлежит f_x классу C или нет; иначе говоря, множество $\{x \mid f_x \in C\}$ неразрешимо.

Доказательство. Предположим, что множество $M = \{x \mid f_x \in C\}$ разрешимо; тогда оно обладает характеристической функцией

$$\chi_M(x) = \begin{cases} 1, & \text{если } f_x \in C \text{ (т. е. } x \in M); \\ 0, & \text{если } f_x \notin C. \end{cases}$$

Пусть нигде не определенная функция $f_0 \in \bar{M}$. Выберем какую-нибудь конкретную вычислимую функцию $f_a \in M$ и определим функцию $F(x, y)$:

$$F(x, y) = \begin{cases} f_a(y), & \text{если } f_x(x) \text{ определена;} \\ f_0(y), & \text{если } f_x(x) \text{ не определена.} \end{cases}$$

Функция $F(x, y)$ вычислена: для ее вычисления надо вычислять $f_x(x)$; если $f_x(x)$ определена, то этот процесс когда-нибудь остановится и тогда надо перейти к вычислению $f_a(y)$, если же $f_x(x)$ не определена, то процесс не остановится, что равносильно вычислению нигде не определенной функции $f_0(y)$. Если в $F(x, y)$ зафиксировать x , то F станет вычислимой функцией от одной переменной y . Номер этой функции зависит от значения x , т. е. является всюду определенной функцией $g(x)$; нетрудно показать, что $g(x)$ вычислена. Итак,

$$f_{g(x)}(y) = \begin{cases} f_a(y), & \text{если } f_x(x) \text{ определена;} \\ f_0(y), & \text{если } f_x(x) \text{ не определена.} \end{cases}$$

Следовательно, $f_{g(x)} \in M$, если и только если $f_x(x)$ определена (так как $f_a \in M$, а $f_0 \notin M$). Отсюда

$$\chi_M(g(x)) = \begin{cases} 1, & \text{если } f_x(x) \text{ определена;} \\ 0, & \text{если } f_x(x) \text{ не определена.} \end{cases}$$

т. е. построена разрешающая функция для проблемы самоприменимости, что невозможно.

Для случая, когда $f_0 \in M$, выбираем $f_a \in \bar{M}$; последующие рассуждения аналогичны, а разрешающая функция для проблемы самоприменимости будет иметь вид $1 - \chi_M(g(x))$. \square

Из теоремы Райса следует, что по номеру вычислимой функции нельзя узнать, является ли эта функция постоянной, периодической, ограниченной, содержит ли она среди своих значений данное число и т. д. Создается впечатление, что вообще ничего нельзя узнать и все на свете неразрешимо. С другой стороны, не противоречит ли теореме Райса тот очевидный факт, что по номеру машины T всегда можно узнать, например, содержит ли она больше десяти команд или нет?

Для того чтобы разобраться в смысле теоремы Райса, надо прежде всего вспомнить, что номер x функции f — это номер алгоритма A_x , вычисляющего f ; в свою очередь, по номеру алгоритма однозначно восстанавливается его описание, и разным номерам соответствуют разные алгоритмы. Для функций это неверно: при $x \neq y$ f_x и f_y могут быть одной и той же функцией (в ее классическом смысле — см. гл. 1). В теореме Райса участвуют и алгоритмы, и функции, и их следует четко различать. Класс C — это класс (или свойство) функций; в то же время « f_x » означает «функция, вычисляемая алгоритмом A_x ». Таким образом, смысл теоремы Райса в том, что по описанию алгоритма ничего нельзя узнать о свойствах функции, которую он вычисляет. В частности, оказывается неразрешимой *проблема эквивалентности алгоритмов*: по двум заданным алгоритмам нельзя узнать, вычисляют они одну и ту же функцию или нет. Количество же команд — это свойство не функции, а описания алгоритма; к теореме Райса оно отношения не имеет.

Опытного программиста теорема Райса не должна удивлять: он знает, что по тексту сколько-нибудь сложной программы, не запуская ее в работу, трудно понять, что она делает (т. е. какую функцию вычисляет), не имея гипотез о том, что она должна делать; если это понимание и приходит, то каждый раз по-своему; систематического метода здесь не существует. С другой стороны,

в этом тексте можно обнаружить алгоритмическим путем так называемые синтаксические ошибки (что и делают компиляторы с алгоритмических языков), т. е. выявлять те или иные свойства описания алгоритма. Здесь впервые стоит упомянуть (пока неформально) два понятия, о которых подробнее будет говориться в двух следующих главах, — *синтаксис* и *семантика*. Синтаксические свойства алгоритма — это свойства описывающих его текстов, т. е. свойства конечных слов в фиксированном алфавите. Семантические (или смысловые) свойства алгоритмов связаны с тем, что он делает; их естественно описывать в терминах функций и классов эквивалентности функций, вычисляемых алгоритмами. Хорошо известно, что в процессе отладки программ синтаксические ошибки отыскиваются довольно быстро; все неприятности связаны с анализом семантики неотлаженной программы, т. е. с попытками установить, что же она делает вместо того, чтобы делать задуманное. В несколько вольной и неформальной интерпретации теорема Райса могла бы выглядеть так: «по синтаксису алгоритма ничего нельзя узнать о его семантике».

Несколько раз повторенное выражение «ничего нельзя узнать» — это, строго говоря, преувеличение. Его точный эквивалент — «не существует единого алгоритма, позволяющего узнать». К тому же речь идет о невозможности распознавания свойств вычислимых функций, записанных в универсальном алгоритмическом языке (языке машин Тьюринга и др.). Свойства подклассов вычислимых функций, описанных в специальных языках, вполне могут оказаться разрешимыми. Например, в гл. 3 рассматривались алгоритмы распознавания свойств логических функций, описанных на языке формул в различных базисах.

До сих пор речь шла о неразрешимых проблемах внутри самой теории алгоритмов. Некоторые из них, такие как проблема остановки или теорема Райса, имеют вполне реальную интерпретацию в практике программирования. Неразрешимости возникают и в других областях: в теории автоматов, в теории языков и грамматик (см. гл. 7) и т. д. Постепенно они становятся бытом дискретной математики, и с их существованием приходится считаться.

С теоретической точки зрения, неразрешимость — не неудача, а научный факт. Знание основных неразрешимостей теории алгоритмов должно быть для специалиста по дискретной математике таким же элементом научной культуры, как для физика — знание о невозможности вечного двигателя. Если же важно иметь дело с разрешимой задачей (а для прикладных наук это стремление естественно), то следует четко представлять себе два обстоятельства. Во-первых (об этом уже говорилось при обсуждении проблемы остановки в § 5.2), отсутствие общего алгоритма, решающего данную проблему, не означает, что в каждом частном случае этой проблемы нельзя добиться успеха. Поэтому, если проблема неразрешима, надо искать ее разрешимые частные случаи. Во-вторых, появление неразрешимости — это, как правило, результат чрезмерной общности задачи (или языка, на котором описаны объекты задач). Задача в более общей постановке имеет больше шансов оказаться неразрешимой.

5.5. ВЫЧИСЛИТЕЛЬНАЯ СЛОЖНОСТЬ И NP-ТРУДНЫЕ ЗАДАЧИ

Предварительное обсуждение. В «классической» теории алгоритмов, основы которой были изложены в §§ 5.2–5.4, задача считается разрешимой, если существует решающий ее алгоритм. Однако существование алгоритма еще не гарантирует его практическую реализуемость. Алгоритмы, предлагаемые для решения реальных задач, нуждаются в количественных оценках, характеризующих их эффективность. Главные вопросы, которые возникают при оценке алгоритма, — это вопросы о требуемых вычислительных ресурсах: как долго работает алгоритм и каков объем памяти, который он занимает. Оценки расхода ресурсов при работе алгоритма принято называть оценками его *вычислительной сложности*.

Обсуждение этих вопросов требует уточнения основных понятий, в терминах которых формулируются точные постановки проблем и их решения. Как измерять ресурсы? Как оценивать расход ресурсов алгоритма, если при разных данных он работает разное время и использует разный объем памяти? Можно ли сравнивать — и если

да, то как — разные алгоритмы, решающие одну и ту же задачу, но реализованные в разных алгоритмических моделях? И, наконец, — как отделить собственные характеристики алгоритма от характеристик машины, на которой он реализован?

Эти проблемы удобнее обсуждать в терминах абстрактных машин, в которых основные компоненты алгоритмов, участвующие в оценках (единицы памяти и элементарные шаги), представлены явно — в отличие, скажем, от представлений в виде рекурсивных описаний. Поэтому в дальнейшем мы будем говорить на языке «машинных» алгоритмических моделей.

Проще всего ответить на вопрос об измерении ресурсов. Объем памяти определяется числом единиц памяти (ячеек ленты машины Тьюринга или машинных слов в современных компьютерах), которые используются в процессе работы алгоритма. Несколько сложнее обстоит дело со временем. С прикладной точки зрения кажется естественным измерять время работы алгоритма в обычных физических единицах — секундах и долях секунды, минутах и т. д. Это время можно грубо представить величиной at , где — число действий (элементарных шагов) алгоритма, а α — среднее физическое время выполнения одного действия. Время α определяется физическими характеристиками аппаратуры: скоростью передачи сигналов, временем записи в единицу памяти и считывания из нее. Эти параметры отсутствуют в моделях абстрактных машин и качество алгоритма никак не характеризуют. Поэтому время, затрачиваемое алгоритмом (*временную сложность алгоритма*), следует измерять числом действий.

Объем s памяти, используемой алгоритмом, по порядку никогда не превосходит числа его действий: если любое действие использует не более s единиц памяти, то $s \leq ct$. Однако каждая единица памяти может использоваться неограниченное число раз, и потому t может существенно превосходить s . Поэтому временная сложность алгоритма считается более важной его характеристикой, чем объем используемой памяти. Она более важна и в прикладном отношении: известно, что повышение быстродействия компьютеров — задача гораздо более слож-

ная и подверженная более серьезным физическим ограничениям, чем задача увеличения памяти.

Некоторые характеристики машины, на которой реализован алгоритм, — такие, как тип команд (т. е. элементарных шагов), возможности доступа к единицам памяти, — неотделимы от самого алгоритма. Их можно назвать характеристиками машинной архитектуры. Алгоритмы, решающие одну и ту же задачу на машинах разной архитектуры, всегда различны. Все, что их объединяет, — это задача, которую они решают. Это приводит к новой постановке проблемы — наряду со сложностью конкретных алгоритмов оценивать сложность самой задачи. Сложностью задачи естественно считать сложность наиболее эффективного алгоритма, ее решающего. Однако эта оценка сложности должна опираться, во-первых, на более точное определение понятия задачи, а во-вторых, на такие подходы к ее получению, которые давали бы оценку, максимально независимую от конкретных типов машин и конкретных представлений исходных данных задачи.

Все эти проблемы привели к возникновению в 60-х гг. XX в. нового направления теории алгоритмов — теории вычислительной сложности алгоритмов и задач.

Сложность массовых задач. Массовая задача Q определяется:

- 1) списком параметров;
- 2) формулировкой свойств ответа.

Индивидуальная задача I получается из Q , если всем параметрам Q присвоены конкретные значения.

Алгоритм A решает задачу Q , если он решает любую задачу I из Q .

Пример 5.16. Задача о коммивояжере. Дан полный взвешенный граф; требуется найти цикл минимальной длины, содержащий каждую вершину графа ровно один раз.

Параметры: множество вершин (городов) $V = \{v_1, \dots, v_m\}$ и множество весов ребер $D = d(v_i, v_j)$ (расстояний между городами).

Вид ответа: упорядоченный набор вершин $(v_{h(1)}, \dots, v_{h(m)})$, который минимизирует величину

$$\sum_{i=1}^{m-1} d(c_{h(i)}, c_{h(i+1)}) + d(c_{h(m)}, c_{h(1)})$$

Индивидуальная задача о коммивояжере определяется конкретной величиной n и конкретными весами ребер.

Время работы алгоритма зависит от конкретных входных данных и в первую очередь от их объема, т. е. от длины слова во входном алфавите, кодирующего данные. Поэтому *временная сложность алгоритма* определяется как функция $T_A(n)$, которая каждой длине n входного слова ставит в соответствие максимальное (по всем индивидуальным задачам длины n) число элементарных действий, затрачиваемое алгоритмом в процессе решения задачей этой длины.

О точной длине входного слова можно говорить только тогда, когда зафиксировано представление задачи, т. е. *схема кодирования индивидуальных задач*. Разные кодирования могут существенно изменить длину входного слова. Например, переход от десятичного представления чисел к двоичному увеличивает длину записи чисел в $\log_2 10$ раз. Длина представления дерева с n вершинами в виде списка ребер примерно равна $c_1 n$, где c_1 — длина описания одного ребра, а представление того же дерева матрицей смежности имеет длину $c_2 n^2$, т. е. увеличивается более чем линейно. Таким образом, сложность задачи как таковой, не учитывая конкретную схему кодирования данных, уже не может характеризоваться точной функцией. Речь может идти лишь о более или менее грубой оценке.

Наиболее распространенный способ грубой сравнительной оценки функций — это сравнение порядка функций, принятые в математическом анализе. Запись $f(n) \sim g(n)$ (читается « $f(n)$ имеет порядок $g(n)$ ») означает, что существуют константы c_1 и c_2 , такие, что $c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|$ для всех $n \geq 0$. В частности, два полинома имеют одинаковый порядок, если и только если степени их старших членов одинаковы.

Представление дерева списком ребер тоже можно закодировать по-разному — потратив разное число символов на описание одного ребра. Функции длины этих представлений будут иметь одинаковый порядок. Однако при переходе к матрице смежности порядок меняется. Поэтому, если мы хотим оценивать сложность задачи безот-

носительно к ее конкретному представлению, оценка по порядку оказывается недостаточно грубой.

Опыт оценки большого числа задач дискретной математики показывает, что достаточно грубой является полиномиальная оценка, т. е. оценка с точностью до степени полинома. Функцию $f(n)$ будем называть *полиномиальной*, если $f(n) \sim (p(n))$, где $p(n)$ — некоторый полином от n (неважно, какой степени).

Разные представления одной задачи могут отличаться и экспоненциально. Например, так отличаются представления целых чисел в унарном коде (который мы использовали при описании вычислений на машинах Тьюринга) и в любой позиционной системе счисления. Действительно, в унарном коде на запись числа m тратится m символов, а в позиционном — $\log_a m$, где a — основание системы счисления. Но это означает, что унарное представление чисел годится только для иллюстрации работы абстрактных машин и при решении реальных задач никогда использоваться не будет. В дальнейшем будем считать, что коды одной и той же задачи при различных разумных схемах кодирования различаются по длине не более чем полиномиально. Иначе говоря, если n — длина входного слова, кодирующего данные α при некоторой схеме кодирования, то при любой другой разумной схеме кодирования длина кода данных α ограничена некоторым полиномом от n .

Рассмотрим теперь вопрос о том, как зависит сложность от архитектуры машины, на которой она решается. Типичным примером является задача о распознавании палиндрома (слова, симметричного относительно середины). Идея решения ясна: надо последовательно сравнивать символы, одинаково отстоящие от краев слова: первый с последним, второй с предпоследним и т. д. На обычной (одноленточной) машине Тьюринга алгоритм, реализующий эту идею, имеет сложность $O(n^2)^*$, так как для i -го сравнения головка машины должна пройти $n - 2i$ ячеек ленты. Доказано, что понизить порядок этой сложности нельзя. Но уже на двухленточной машине эта

* Более точно — сумму арифметической прогрессии $n + (n - 2) + \dots + (n - 4) + \dots$

Моделируемая машина B	Моделирующая машина A		
	1МТ	k МТ	МДП
1-ленточная МТ (1МТ)	—	$O(f(n))$	$O(f(n))\log f(n)$
k -ленточная МТ (k МТ)	$O(f^2(n))$	—	$O(f(n))\log f(n)$
Машина произвольного доступа (МДП)	$O(f^3(n))$	$O(f^2(n))$	—

задача решается в три просмотра слова, т. е. со сложностью порядка $3n$: 1) исходное слово переписывается на вторую ленту; в конце этого прохода головка видит последние символы слова на обеих лентах; 2) одна из лент сдвигается к первому символу; в конце этого прохода головка видит последний символ слова на одной ленте и первый символ — на другой; 3) ленты сдвигаются навстречу друг другу; соответствующие символы сравниваются.

Один из способов сравнивать алгоритмы, реализованные на разных машинах M_1 и M_2 , — это моделировать на машине M_2 алгоритм, реализованный на машине M_1 и имеющий сложность $f(n)$. Оценки сложности такого моделирования приведены в табл. 5.4 [7].

Как видим, и здесь оценки сложности алгоритмов на разных машинах отличаются полиномиально. То же самое можно утверждать и для задач, поскольку сложность задачи — это сложность самого эффективного алгоритма, который ее решает. Обобщает эти утверждения тезис, лежащий в основе последующей теории:

*Перенос задачи на детерминированную машину любой архитектуры изменяет ее сложность не более чем полиномиально**.

Из этого тезиса вытекают очевидные следствия:

- любое улучшение архитектуры машины может лишь понизить степень полинома в оценке сложности решаемых задач;
- если задача имеет более чем полиномиальную сложность (например, сложность вида $n^{\log n}$ или экспоненциальную сложность вида a^n), то она сохранит эту сложность на машине любой архитектуры. Можно

* Такой перенос может повлечь за собой смену схемы кодирования. Однако мы уже договорились, что эта смена изменит длину слова не более чем полиномиально.

- сказать, что задача с такой сложностью объективно трудна — ее сложность неулучшаема;
- при получении оценок сложности, не зависящих от конкретных машин и конкретных схем кодирования исходных данных, степень полинома уточнить нельзя. Поэтому сложность задачи нельзя охарактеризовать конкретной функцией. Можно говорить лишь о типах, или классах сложности — например, о том, что сложность полиномиальна, экспоненциальна и т. д.
- Алгоритм A называется *полиномиальным*, если его времененная сложность $T_A(n)$ — полиномиальная функция. Из вышеописанного ясно, что эта характеристика алгоритма не зависит ни от архитектуры машины, ни от выбранной схемы кодирования задачи.

Классы P и NP. Среди концепций и разделов теории вычислительной сложности наибольший интерес представляет NP-теория, или теория NP-сложности. Это объясняется двумя причинами. Во-первых, класс задач, рассматриваемых в этой теории, включает большинство известных задач из различных разделов дискретной математики — теории графов, теории автоматов, теории формальных языков и др. Задачи этого класса часто называют комбинаторными. Во-вторых, в ней удалось сформировать систему понятий и теорем, которые создают стройную картину сравнительной сложности алгоритмов и задач.

На полуформальном уровне комбинаторную задачу можно определить как задачу, характеризующуюся двумя свойствами.

1. Длина решения ограничена функцией $g(n)$, полиномиально зависящей от длины n исходных данных и определяемой содержанием задачи. Поэтому текст, описывающий решение, находится среди конечного множества M слов во входном алфавите, имеющих длину, не превосходящую $g(n)$. Само же множество M ввиду его ограниченности может быть порождено чисто комбинаторным способом — просто генерацией всех слов длины, не большей $g(n)$.

2. Сложность проверки правильности решения не более чем полиномиальна. В частности, она может оказаться существенно меньшей, чем сложность получения решения.

Пример 5.17. Типичный пример комбинаторной задачи — задача о гамильтоновом цикле (§ 4.2). Ее решением для графа с n вершинами является либо цикл, содержащий ровно n ребер и проходящий через каждую вершину один раз, либо утверждение, что такого цикла не существует. Из формулировки задачи ясно, что длина описания такого цикла не превосходит cn , где c — длина описания одного ребра. Поэтому решение задачи находится среди множества всех слов длины $\leq cn$. Проверка правильности решения весьма проста: нужно проверить, является ли предъявленное решение описанием цикла, не повторяются ли в нем вершины и посчитать число ребер.

Используя указанные два свойства, комбинаторную задачу всегда можно решить простым *перебором*: последовательно порождать все слова длины $\leq g(m)$ и каждое из них проверять — не является ли оно решением задачи. Поэтому комбинаторные задачи называют также переборными. Однако перебор имеет экспоненциальную сложность, поскольку число всех слов длины $g(m)$ в алфавите мощности k равно $k^{g(m)}$.

Не все задачи — комбинаторные. Например, задачи, связанные с поиском вывода слов в формальных исчислениях (см. § 6.4), не являются комбинаторными, так как функция $g(n)$, ограничивающая длину ответа (т. е. вывода) для данных длины n , может не быть полиномиальной.

Точным эквивалентом понятия комбинаторной задачи является понятие «задача класса NP», формализации которого мы сейчас и займемся. В это понятие будут включаться только задачи распознавания свойств, т. е. задачи, решение которых — это ответ «да» или «нет». Например, такова задача о гамильтоновом цикле (пример 5.17), поскольку по предъявленному графу надо дать ответ — есть гамильтонов цикл или нет. Оптимизационная задача «найти минимум (или максимум) величины x » формулируется как задача распознавания путем перевода ее в форму «верно ли, что величина x не больше (не меньше) числа $B?$ » Например, задача о коммивояжере (пример 5.16) в этом виде формулируется так: «Существует ли путь, проходящий через все города, длина ко-

торого не превосходит B ?» Для задач с целочисленными параметрами решение конечного числа таких задач распознавания даст решение задачи оптимизации.

Задача распознавания Q характеризуется двумя множествами: множеством D_Q всех индивидуальных задач и множеством индивидуальных задач Y_Q с ответом «да». При фиксированной схеме кодирования Σ каждая индивидуальная задача становится конкретным текстом, т. е. словом в алфавите K схемы кодирования. (Множество всех слов α в алфавите K обозначается как K^* — см. пример 1.5, г и § 7.1.) Множество всех слов, кодирующих задачи Y_Q в схеме кодирования Σ , образует язык^{*} $L(Q, \Sigma)$. Таким образом, задача распознавания Q формально является задачей распознавания языка $L(Q, \Sigma)$.

Язык $L(Q, \Sigma)$ принадлежит сложностному классу P , если он распознается за полиномиальное время, т. е. существует алгоритм A , такой, что для любого слова $\alpha \in K^*$ длины n $A(\alpha) = 1$, если и только если $\alpha \in L(Q, \Sigma)$, причем время работы A не больше $O(n^k)$ для некоторого k .

Задача распознавания Q принадлежит классу P , если для некоторой схемы кодирования Σ язык $L(Q, \Sigma)$ принадлежит классу P . В силу наших соглашений в этом случае $L(Q, \Sigma') \in P$ для любой разумной схемы кодирования Σ' задачи Q .

Задача распознавания Q принадлежит сложностному классу NP , если выполнены следующие условия: длина решения β любой индивидуальной задачи из Y_Q ограничена полиномом от длины исходных данных; иначе говоря, для любого слова $\alpha \in L(Q, \Sigma)$ длины n длина его решения не больше $O(n^k)$ для некоторого k ; решение β проверяется за полиномиальное время, т. е. существует полиномиальный алгоритм $A(\alpha, \beta)$, такой, что $A(\alpha, \beta) = 1$, если и только если $\alpha \in L(Q, \Sigma)$.

Понятие класса NP — центральное в теории сложности. Оно содержит ряд подводных камней, требующих уточнения. Однако попытки такого уточнения приводят к проблемам, которые до сих пор не удалось решить.

Первая — и главная — проблема формулируется как ответ на вопрос: верно ли, что $P = NP$? Дело в том, что

* Общее понятие формального языка определяется в гл. 7.

из п. 1 определения неясно, какова сложность получения решения. Ведь число всех слов, ограниченных полиномом от n , экспоненциально зависит от n . Поэтому перебор всех слов этой длины с последующей полиномиальной проверкой не является полиномиальным алгоритмом. Показать, что для любой задачи класса **NP** существует полиномиальный алгоритм поиска ее решения, т. е. что $P = NP$, не удается. Большинство специалистов считает, что по крайней мере для некоторых задач класса **NP** таких алгоритмов нет, и, следовательно, $P \neq NP$. Однако и это утверждение до сих пор не доказано.

Вторая проблема связана с п. 2 определения. Всякая задача распознавания содержит логическое утверждение, а решение задачи — это ответ на вопрос, истинно ли это утверждение или ложно. Если решение β получено и предъявлено, т. е. ответ на задачу положителен, то он проверяется за полиномиальное время. Но если ответ отрицателен, то нет предмета проверки. Как например, проверить утверждение, что данный граф — не гамильтонов? Можно перейти к решению задачи $\neg Q$, т. е. задачи, положительный ответ которой совпадает с отрицательным ответом исходной задачи Q . Однако это не решает проблемы: в этом случае неясно, удовлетворяет ли $\neg Q$ определению задач класса **NP**. Непонятно, например, как описать положительное решение задачи «верно ли, что данный граф не является гамильтоновым» в терминах этого определения. Определим еще один *сложностной класс со-NP* как класс всех задач $\neg Q$, таких, что $Q \in NP$. Тогда формально наша проблема выглядит так: верно ли, что $co\text{-}NP = NP$? Эта проблема также не решена.

Неконструктивность процесса получения решения в определении класса **NP** привела к попыткам описать этот процесс с помощью «недетерминированной конструктивности» — псевдо-машины, названной *недетерминированной машиной Тьюринга* (НДТМ), которая фигурирует в большинстве изложений **NP**-теории*. Используются два эквивалентных варианта описания НДТМ. Первый вариант [2, 35] предполагает, что НДТМ, грубо говоря, по-

* Использованием недетерминированных вычислений объясняется аббревиатура **NP**: Nondeterministic Polinomial. **P** означает просто Polinomial.

рождает дерево всех возможных слов в алфавите решения, ограниченных полиномом от длины n входного слова, причем просматривает параллельно (и следовательно, за полиномиальное время) каждую ветвь дерева и с полиномиальной сложностью проверяет, не является ли слово на этой ветви решением. Как только решение найдено, вычисление останавливается. Второй вариант [7] содержит недетерминированный модуль, угадывающий решение за полиномиальное время, и детерминированный блок проверки решения. Предполагается, опять-таки, что все угадывания происходят параллельно. Поэтому определение класса **NP** с помощью НДТМ выглядит так: *задача принадлежит классу NP, если существует НДТМ, которая вычисляет его за полиномиальное время.*

Поскольку НДТМ не является машиной в обычном смысле слова, и, следовательно, реализуемые на ней процедуры не являются обычными детерминированными алгоритмами, использование НДТМ в определении класса **NP**, на наш взгляд, не добавляет ему конструктивности и затрудняет понимание содержательного смысла этого понятия. Уже имеются современные изложения **NP**-теории [12, 43], которые обходятся без него.

Полиномиальная сводимость и NP-полнота. Язык L_1 полиномиально сводим к языку L_2 (обозначение $L_1 \leq L_2$), если существует функция f , удовлетворяющая двум условиям:

- 1) f имеет полиномиальную сложность;
- 2) для любого $\alpha \in L_1$, если и только если $f(\alpha) \in L_2$.

Функция f — это функция, преобразующая условия задачи в коде L_1 в условия задачи (быть может, содержательно другой) в коде L_2 .

Теорема 5.18. Если $L_1 \leq L_2$, то из $L_2 \in P$ следует, что $L_1 \in P$.

Доказательство очевидно: если $p_1(n)$ — полином, ограничивающий сложность распознавания языка L_2 , а $p_2(n)$ — полином, ограничивающий сложность функции сведения f , то L_1 распознается за время порядка $p_1(n) + p_2(n)$. \square

Столь же очевидно, что отношение \leq транзитивно.

Сводимость задачи Q_1 к задаче Q_2 содержательно означает, что Q_2 не проще, чем Q_1 . Понятно, что легкую задачу можно свести к более трудной. Если же «трудная»

задача сведена к легкой, значит, она на самом деле трудной не является.

Пример 5.18. Сведение задачи о существовании гамильтонова цикла к задаче о коммивояжере.

Пусть в индивидуальной задаче о гамильтоновом цикле $G = (V, E)$, $|V| = m$.

Соответствующая задача о коммивояжере строится так:
 $d(v_i, v_j) = 1$, если $(v_i, v_j) \in E$, и 2 в противном случае;
 $B = m$.

Функция f , осуществляющая это сведение, полиномиальна, поскольку вычисление $d(v_i, v_j)$ сводится к выяснению вхождения (v_i, v_j) в E ; всего таких выяснений $m(m - 1)/2$.

Если G содержит гамильтонов цикл, то этот цикл является путем в $f(G)$ длины m . Он минимален, так как все веса на ребрах этого пути равны 1. Следовательно, в этом случае задача $f(G)$ дает положительный ответ. Если же G не содержит гамильтонов цикл, то длина минимального пути в $f(G)$ больше m , и ответ в задаче $f(G)$ — отрицательный. \square

Язык L называется **NP-полным**, если $L \in \text{NP}$ и любой другой язык $L' \in \text{NP}$ сводится к L . Тем самым NP-полные задачи — самые трудные в классе NP, и если хотя бы одна NP-полнная задача содержится в P, то и все NP-полные задачи содержатся в P.

Теорема 5.19. Если $L_1, L_2 \in \text{NP}$, L_1 — NP-полный и $L_1 \propto L_2$, то L_2 — также NP-полный.

Действительно, по определению, для любого $L' \in \text{NP}$ $L' \propto L_1$ и, следовательно, в силу транзитивности отношения полиномиальной сводимости $L' \propto L_2$. \square

Из определения NP-полней задачи не следует, что существует хотя бы одна такая задача. Следующая теорема, доказанная С. Куком в начале 70-х гг. XX в., содержит конструктивное доказательство существования таких задач.

Теорема 5.20 (Кук). Задача выполнимости конъюнктивной нормальной формы (КНФ) является NP-полней.

Доказательство этой теоремы довольно сложно и здесь опускается. Его можно найти в книгах [7, 12]. \square

После появления этой теоремы довольно быстро появились доказательства NP-полноты многих дискретных

задач. Всего же в настоящее время известно несколько сотен **NP**-полных задач из теорий графов, грамматик, автоматов и т. д. Для всех них **NP**-полнота доказывается сведением к какой-нибудь другой **NP**-полной задаче. Важно отметить, что, как правило, эти задачи не специально сконструированы для иллюстрации понятий и результатов **NP**-теории, а были известны еще до ее возникновения.

В книге [7] содержатся доказательства (сведением к задаче Кука) **NP**-полноты 6 известных задач.

1. *З-выполнимость* — выполнимость для КНФ, к которой каждая дизъюнкция содержит ровно 3 литерала (литерал — это либо переменная, либо переменная с отрицанием).

2. *Вершинное покрытие* — имеется ли в графе $G(V, E)$ вершинное покрытие не более чем из k элементов, т. е. такое $V' \subseteq V$, что $|V'| \leq k$ и хотя бы один конец любого ребра принадлежит V' ?

3. *Клика* — верно ли, что граф содержит клику мощности $\geq k$?

4. *Гамильтонов цикл*.

5. *k-раскрашиваемость* — можно ли раскрасить граф $G(V, E)$ в k цветов, где $k < |V|$?

6. *Изоморфизм подграфу* — изоморфен ли граф G_1 какому-нибудь подграфу графа G_2 ?

ФОРМАЛЬНЫЕ СИСТЕМЫ

Формальные системы — это системы операций над объектами, понимаемыми как последовательности символов (т. е. как слова в фиксированных алфавитах); сами операции также являются операциями над символами. Термин «формальный» подчеркивает, что объекты и операции над ними рассматриваются чисто формально, без каких бы то ни было содержательных интерпретаций символов. Предполагается, что между символами не существует никаких связей и отношений, кроме тех, которые явно описаны средствами самой формальной системы.

Если предложить читателю упорядочить объекты 53, 109, 3, то, скорее всего, он без всяких дополнительных вопросов расположит их в порядке 3, 53, 109. Иначе говоря, этой задаче будет дана обычная арифметическая интерпретация: последовательности цифр рассматриваются как изображения чисел в десятичной системе, упорядочение этих последовательностей есть расположение изображаемых ими чисел по возрастанию, а правила сравнения таких изображений чисел известны настолько хорошо, что обычно о них никто не задумывается. В действительности же такое истолкование задачи, вообще говоря, не вытекает из текста «упорядочить объекты 53, 109, 3»; его можно понимать как задачу лексикографического упорядочения (что приведет к результату 109, 3, 53), как задачу распределения бегунов с номерами 53, 109, 3 по дорожкам (решение которой зависит от процедуры распределения и заведомо не связано с числовой

интерпретацией объектов) и т. д. Возможность неоднозначного извлечения задач из текста означает, что этот текст не содержит формального определения задачи. Для такого определения нужно четко описать класс объектов, для которых задача решается, и явно ввести для них понятие упорядочения, описав его как систему локальных операций над символами, из которых эти объекты состоят.

По существу при таком понимании «формальное описание» задачи означает ее точное, явное описание — все, что существенно для решения задачи, выписано явно. Поэтому уточнение задачи принято называть ее формализацией.

Сходные соображения по поводу того, что такое «точное описание», довольно подробно рассматривались при обсуждении понятия «алгоритм» и таких его элементов, как «данные», «элементарный шаг» (§ 5.1), а также в связи с понятием «финитного подхода» к математике (конец § 3.4). В определенном смысле проблему точного описания некоторого множества можно рассматривать как проблему построения алгоритма, перечисляющего или порождающего это множество. Однако иногда акценты здесь несколько смещаются. Чтобы было ясно, о чем идет речь, рассмотрим формализацию понятия формулы (см. гл. 3 и далее §§ 6.1, 6.2), которая дается индуктивным определением формулы. Нетрудно показать, что множество формул, определенных таким образом, перечислимо (и даже разрешимо) и, следовательно, существует алгоритм, порождающий это множество. Однако конкретный порядок порождения формул, который определил бы номер формулы в списке этих формул, т. е. конкретный перечисляющий алгоритм, этим определением не фиксируется. Зафиксировать такой алгоритм можно различными способами, но для точного определения формулы этого не требуется.

Индуктивное определение формулы — простой пример описания перечислимого множества, в котором использованы все существенные составные части понятия «алгоритм», кроме одного — детерминированности. Отбрасывая несущественный здесь порядок перечисления элементов множества, мы выигрываем в компактности

описания, которое при этом не становится менее точным. Такое описание, не являясь алгоритмом, представляет собой формальную систему, однозначно описывающую множество формул.

Исторически теория формальных систем, так же как и теория алгоритмов, возникла в рамках оснований математики при исследовании строения аксиоматических теорий и методов доказательства в таких теориях. С их изучения и начнется знакомство с формальными системами.

6.1. ФОРМАЛЬНЫЕ ТЕОРИИ (ЛОГИЧЕСКИЕ ИСЧИСЛЕНИЯ). ИСЧИСЛЕНИЕ ВЫСКАЗЫВАНИЙ

Принципы построения формальных теорий. Всякая точная теория определяется, во-первых, языком, т. е. некоторым множеством высказываний, имеющих смысл с точки зрения этой теории, и, во-вторых, совокупностью теорем — подмножеством языка, состоящим из высказываний, истинных в данной теории.

Каким образом теория получает свои теоремы?

В математике с античных времен существовал образец систематического построения теории — геометрия Евклида, в которой все исходные предпосылки сформулированы явно, в виде аксиом, а теоремы выводятся из этих аксиом с помощью цепочек логических рассуждений, называемых доказательствами. Однако до середины XIX в. математические теории, как правило, не считали нужным явно выделять действительно все исходные принципы; критерии же строгости доказательств и очевидности утверждений в математике в разные времена были различны и также явно не формулировались. Время от времени это приводило к необходимости пересмотра основ той или иной теории. Известно, например, что основания дифференциального и интегрального исчислений, разработанных в XVIII в. Ньютона и Лейбницем, в XIX в. подверглись серьезному пересмотру; математический анализ в его современном виде опирается на работы Коши, Больцано, Вейерштрасса по теории пределов. В конце XIX в. такой пересмотр затронул общие принципы организации математических

теорий. Это привело к созданию новой отрасли математики — оснований математики, предметом которой стало как раз строение математических утверждений и теорий и которая поставила своей целью ответить на вопросы типа: «как должна быть построена теория, чтобы в ней не возникало противоречий?», «какими свойствами должны обладать методы доказательств, чтобы считаться достаточно строгими?» и т. д. Одной из фундаментальных идей, на которые опираются исследования по основаниям математики, является идея формализации теорий, т. е. последовательного проведения аксиоматического метода построения теорий. При этом не допускается пользоваться какими-либо предположениями об объектах теории, кроме тех, которые выражены явно в виде аксиом; аксиомы рассматриваются как формальные последовательности символов (выражения), а методы доказательств — как методы получения одних выражений из других с помощью операций над символами. Такой подход гарантирует четкость исходных утверждений и однозначность выводов, однако может создаться впечатление, что осмысленность и истинность в формализованной теории не играют никакой роли. Внешне это так; однако в действительности и аксиомы, и правила вывода стремятся выбирать таким образом, чтобы построенной с их помощью формальной теории можно было придать содержательный смысл.

Более конкретно *формальная теория* (или *исчисление*) строится следующим образом.

1. Определяется конечный алфавит теории. Из символов алфавита строится множество *формул*, или правильно построенных выражений, образующее язык теории. Это множество задается конструктивными средствами (как правило, индуктивным определением) и, следовательно, перечислимо. Обычно оно и разрешимо.

2. Выделяется подмножество формул, называемых *аксиомами* теории. Множество может быть и бесконечным; во всяком случае оно должно быть разрешимо.

3. Задаются *правила вывода* теории. Правило вывода $R(F_1, \dots, F_n, G)$ — это вычислимое отношение на множестве формул. Если формулы F_1, \dots, F_n, G находятся в отношении R , то формула G называется *непосредственно*

выводимой из F_1, \dots, F_n по правилу R . Часто правило $R(F_1, \dots, F_n, G)$ записывается в виде

$$\frac{F_1, \dots, F_n}{G}.$$

Формулы F_1, \dots, F_n называются *посылками* правила R , а G — его *следствием* или *заключением*. Примеры аксиом и правил вывода будут приведены несколько позднее.

Выводом формулы B из формул A_1, \dots, A_n называется последовательность формул F_1, \dots, F_m , такая, что $F_m = B$, а любая F_i ($i = 1, \dots, m$) есть либо аксиома, либо одна из исходных формул A_1, \dots, A_n , либо непосредственно выводима из формул F_1, \dots, F_{i-1} (или какого-то их подмножества) по одному из правил вывода. Если существует вывод B из A_1, \dots, A_n , то говорят, что B *выводима* из A_1, \dots, A_n . Этот факт обозначается так: $A_1, \dots, A_n \vdash B$. Формулы A_1, \dots, A_n называются *гипотезами* или *посылками вывода*. Переход в выводе от F_{i-1} к F_i называется *i-м шагом вывода*.

Доказательством формулы B в теории T называется вывод B из пустого множества формул, т. е. вывод, в котором в качестве исходных формул используются только аксиомы. Формула B , для которой существует доказательство, называется *формулой, доказуемой* в теории T , или *теоремой* теории T ; факт доказуемости B обозначается $\vdash B$.

Очевидно, что присоединение формул к гипотезам не нарушает выводимости. Поэтому если $\vdash B$, то $A \vdash B$, и если $\vdash B$, то $A \vdash B$, и если $A_1, \dots, A_n \vdash B$, то $A_1, \dots, A_n, A_{n+1} \vdash B$ для любых A и A_{n+1} . Порядок гипотез в списке несуществен.

При изучении формальных теорий мы имеем дело с двумя типами высказываний: во-первых, с высказываниями самой теории (теоремами), которые рассматриваются как чисто формальные объекты, определенные ранее, а во-вторых, с высказываниями о теории (о свойствах ее теорем, доказательств и т. д.), которые формулируются на языке, внешнем по отношению к теории, — метаязыке и называются *метатеоремами*. Различие между теоремами и метатеоремами не всегда будет проводиться явно, но его обязательно надо иметь в виду.

Например, если удалось построить вывод B из A_1, \dots, A_n , то « $A_1, \dots, A_n \vdash B$ » является дополнительным («производным») правилом вывода, которое можно присоединить к исходным и использовать в дальнейшем.

Ясно, что общезначимые (тождественно-истинные) высказывания типа $A \vee \bar{A}$ или $\forall x P(x) \rightarrow P(y)$, имеющие силу общих логических законов, должны содержаться в любой теории, претендующей на логический смысл. Поэтому изучение конкретных формальных теорий начнем с исчислений, порождающих все общезначимые формулы.

Исчисление высказываний. Аксиомы и правила вывода. В исчислении высказываний мы вновь встречаемся с объектами, с которыми однажды уже имели дело, — с формулами алгебры логики. Однако здесь формулы рассматриваются не как способ представления функций, а как составные высказывания, образованные из элементарных высказываний (переменных) с помощью логических операций или, как говорят в логике, связок \vee , $\&$, \neg , \rightarrow . При этом особое внимание уделяется тождественно-истинным высказываниям, поскольку, как уже отмечалось, они должны входить в любую теорию в качестве общелогических законов. Их порождение и является основной задачей исчисления высказываний. Исчисление высказываний определяется следующим образом.

1. А л ф а и т исчисления высказываний состоит из переменных высказываний (пропозициональных букв): A , B , $C\dots$, знаков логических связок \vee , $\&$, \neg , \rightarrow и скобок $(,)$.

2. Ф о р м у л ы:

- переменное высказывание есть формула;
- если \mathfrak{A} и \mathfrak{B} — формулы, то $(\mathfrak{A} \vee \mathfrak{B})$, $(\mathfrak{A} \& \mathfrak{B})$, $(\mathfrak{A} \rightarrow \mathfrak{B})$ и $\neg \mathfrak{A}$ — формулы;
- других формул нет.

Внешние скобки у формул обычно договариваются опускать: например, вместо $(\mathfrak{A} \vee \mathfrak{B})$ пишут $\mathfrak{A} \vee \mathfrak{B}$. Вместо синтаксически более удобного знака \neg часто употребляется черта над формулой (она использовалась в гл. 3).

3. А к с и о м ы. Приведем здесь две системы аксиом. Первая из них непосредственно использует все логические связки:

Система аксиом I.

- I 1. $A \rightarrow (B \rightarrow A)$;
- I 2. $(A \rightarrow B) \rightarrow ((A \rightarrow (B \rightarrow C)) \rightarrow (A \rightarrow C))$;
- I 3. $(A \& B) \rightarrow A$;
- I 4. $(A \& B) \rightarrow B$;
- I 5. $A \rightarrow (B \rightarrow (A \& B))$;
- I 6. $A \rightarrow (A \vee B)$;
- I 7. $B \rightarrow (A \vee B)$;
- I 8. $(A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow ((A \vee B) \rightarrow C))$;
- I 9. $(A \rightarrow B) \rightarrow ((A \rightarrow \neg B) \rightarrow \neg A)$;
- I 10. $\neg \neg A \rightarrow A$.

Другая система использует только две связки \neg и \rightarrow ; при этом сокращается алфавит исчисления (выбрасываются знаки \vee , $\&$) и, соответственно, определение формулы. Операции \vee , $\&$ рассматриваются не как связки исчисления высказываний, а как сокращения (употреблять которые удобно, но не обязательно) для некоторых его формул: $A \vee B$ заменяет $\neg A \rightarrow B$, $A \& B$ заменяет $\neg(A \rightarrow \neg B)$. В результате система аксиом становится намного компактнее.

Система аксиом II.

- II 1. $A \rightarrow (B \rightarrow A)$;
- II 2. $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$;
- II 3. $(\neg A \rightarrow \neg B) \rightarrow ((\neg A \rightarrow B) \rightarrow A)$.

Приведенные системы аксиом равносильны в том смысле, что порождают одно и то же множество формул. Разумеется, такое утверждение нуждается в доказательстве, которое заключается в том, что показывается выводимость всех аксиом системы II из аксиом системы I и, наоборот, системы I из системы II (с учетом замечаний относительно \vee и $\&$). Доказательство этих выводимостей предоставляется читателю после того, как он познакомится с примерами вывода в исчислении высказываний (см. также пример 6.2, а).

Возможны и другие системы аксиом, равносильные первым двум системам.

Какая из систем лучше? Это зависит от точки зрения. Система II компактнее и обозримее; соответственно более компактны и доказательства различных ее свойств. С другой стороны, в более богатой системе I короче выводы различных формул.

4. Правила вывода:

1) правило подстановки. Если \mathfrak{A} — выводимая формула, содержащая букву A (обозначим этот факт $\mathfrak{A}(A)$), то выводима формула $\mathfrak{A}(\mathfrak{B})$, получающаяся из \mathfrak{A} заменой *всех* вхождений A на произвольную формулу \mathfrak{B} :

$$\frac{\mathfrak{A}(A)}{\mathfrak{A}(\mathfrak{B})};$$

2) правило отделения (Modus Ponens, или сокращенно MP). Если \mathfrak{A} и $\mathfrak{A} \rightarrow \mathfrak{B}$ — выводимые формулы, то \mathfrak{B} выводима:

$$\frac{\mathfrak{A}, \mathfrak{A} \rightarrow \mathfrak{B}}{\mathfrak{B}}.$$

В этом описании исчисления высказываний аксиомы являются формулами исчисления (соответствующими определению формулы); формулы же, использованные в правилах вывода (\mathfrak{A} , $\mathfrak{A} \rightarrow \mathfrak{B}$ и т. д.), это «метаформулы», или *схемы формул*. Схема формул, например $\mathfrak{A} \rightarrow \mathfrak{B}$, обозначает множество всех тех формул исчисления, которые получаются, если ее метапеременные заменить формулами исчисления: скажем, если \mathfrak{A} заменить на A , а \mathfrak{B} — на $A \& B$, то из схемы формул $\mathfrak{A} \rightarrow \mathfrak{B}$ получим формулу $A \rightarrow A \& B$.

Использование схем формул можно распространить и на аксиомы. Например, если в системе II метапеременные (пропозициональные буквы) A , B , C заменить метапеременными \mathfrak{A} , \mathfrak{B} , \mathfrak{C} , то получаются три схемы аксиом, задающие три бесконечных множества аксиом. В результате возникает другой способ построения исчисления высказываний: с бесконечным множеством аксиом (задаваемым конечным числом схем аксиом), но без *правила подстановки*, поскольку оно неявно содержится в истолковании схем аксиом. Первый способ более последовательно конструктивен: все его средства явно зафиксированы и конечны; при вводе исчисления в ЭВМ (например, при автоматизации доказательства теорем) он выглядит более естественным. С другой стороны, второй способ больше соответствует математической традиции истолкования формул: например, алгебраическое тождество $(a + b)^2 = a^2 + 2ab + b^2$ или тождества булевой алгебры истолковываются именно как схемы тождеств,

а не конкретные тождества, верные лишь для конкретных букв. Правило подстановки при этом подразумевается (см. гл. 3, § 3.2). Впрочем, достаточно очевидно, что переход от одного способа построения исчислений к другому не представляет труда.

Рассмотрим теперь примеры вывода в исчислении высказываний.

Пример 6.1. а. Покажем, что формула $A \rightarrow A$ выводима из системы аксиом П:

$$\vdash A \rightarrow A.$$

1. $(A \rightarrow ((A \rightarrow A) \rightarrow A)) \rightarrow ((A \rightarrow (A \rightarrow A)) \rightarrow (A \rightarrow A))$ (подстановка в аксиому П 2 $A \rightarrow A$ вместо B и A вместо C).

2. $A \rightarrow ((A \rightarrow A) \rightarrow A)$ (подстановка в П 1 $A \rightarrow A$ вместо B).

3. $(A \rightarrow (A \rightarrow A)) \rightarrow (A \rightarrow A)$ (из шагов 2, 1 по правилу MP).

4. $A \rightarrow (A \rightarrow A)$ (подстановка в П 1 A вместо B).

5. $A \rightarrow A$ (из шагов 4, 3 по правилу MP).

6. $A \vdash B \rightarrow A$.

Пусть A выводима. Тогда из A и аксиомы П 1 по правилу заключения получаем

$$\frac{A, A \rightarrow (B \rightarrow A)}{B \rightarrow A},$$

что и доказывает искомую выводимость.

Как уже отмечалось ранее, всякую доказанную в исчислении выводимость вида $\Gamma \vdash \mathfrak{A}$, где Γ — список формул, \mathfrak{A} — формула, можно рассматривать как правило вывода

$$\frac{\Gamma}{\mathfrak{A}},$$

которое можно присоединить к уже имеющимся. Полученный нами вывод $A \vdash B \rightarrow A$ вместе с правилом подстановки можно рассматривать как правило

$$\frac{\mathfrak{A}}{\mathfrak{B} \rightarrow \mathfrak{A}} :$$

«если формула \mathfrak{A} выводима, то выводима и формула $\mathfrak{B} \rightarrow \mathfrak{A}$, где \mathfrak{B} — любая формула». Воспользуемся этим правилом в следующем примере.

в. $A \rightarrow B, B \rightarrow C \vdash A \rightarrow C$.

1. $B \rightarrow C \vdash A \rightarrow (B \rightarrow C)$, по новому правилу

$$\frac{\mathfrak{A}}{\mathfrak{B} \rightarrow \mathfrak{A}}.$$

2. Из $A \rightarrow (B \rightarrow C)$ и аксиомы II 2 по правилу MP следует $(A \rightarrow B) \rightarrow (A \rightarrow C)$; следовательно,
 $B \rightarrow C \vdash (A \rightarrow B) \rightarrow (A \rightarrow C)$.

3. Из $(A \rightarrow B)$ и $(A \rightarrow B) \rightarrow (A \rightarrow C)$ по правилу MP следует $A \rightarrow C$; учитывая 2, получаем искомую выводимость.

При переходе от 1 к 2 неявно использовалось следующее свойство выводимости: если $\Gamma \vdash \mathfrak{A}$ (Γ — список формул), а $\mathfrak{A} \vdash \mathfrak{B}$, то $\Gamma \vdash \mathfrak{B}$. Это свойство (*транзитивность отношения выводимости*) непосредственно следует из определения выводимости.

Основные метатеоремы исчисления высказываний. Для получения выводов в исчислении высказываний оказывается крайне полезной следующая метатеорема.

Теорема 6.1а (теорема дедукции).

Если $\Gamma, \mathfrak{A} \vdash \mathfrak{B}$, то $\Gamma \vdash \mathfrak{A} \rightarrow \mathfrak{B}$ (Γ — множество формул, $\mathfrak{A}, \mathfrak{B}$ — формулы).

Будем исходить из систем аксиом II и рассматривать их как схемы аксиом (т. е. не пользоваться правилом подстановки).

Пусть $\Gamma, \mathfrak{A} \vdash \mathfrak{B}$. Тогда существует вывод $\mathfrak{B}_1, \dots, \mathfrak{B}_n$ из Γ, \mathfrak{A} , такой, что $\mathfrak{B}_n = \mathfrak{B}$. Докажем по индукции, что для любого $k \leq n$ $\Gamma \vdash \mathfrak{A} \rightarrow \mathfrak{B}_k$.

Рассмотрим сначала \mathfrak{B}_1 . \mathfrak{B}_1 , как первая формула вывода, должна либо быть аксиомой, либо содержаться в Γ , либо совпадать с \mathfrak{A} . Из схемы аксиом II 1 следует, что $\mathfrak{B}_1 \rightarrow (\mathfrak{A} \rightarrow \mathfrak{B}_1)$ является аксиомой. Если \mathfrak{B}_1 — аксиома или содержится в Γ , то по правилу MP $\mathfrak{A} \rightarrow \mathfrak{B}_1$ выводима из Γ . Если же $\mathfrak{B}_1 = \mathfrak{A}$, то из примера 6.7, а имеем $\mathfrak{A} \rightarrow \mathfrak{A}$, т. е. $\mathfrak{A} \rightarrow \mathfrak{B}_1$. В любом случае получаем $\Gamma \vdash \mathfrak{A} \rightarrow \mathfrak{B}_1$.

Предположим теперь, что $\Gamma \vdash \mathfrak{A} \rightarrow \mathfrak{B}_i$ для любого $i < k$, и рассмотрим \mathfrak{B}_k . Возможны четыре случая: а) \mathfrak{B}_k — аксиома; б) $\mathfrak{B}_k \in \Gamma$; в) $\mathfrak{B}_k = \mathfrak{A}$; г) \mathfrak{B}_k выводимо из некоторых предшествующих формул $\mathfrak{B}_j, \mathfrak{B}_l$ по правилу MP; но тогда \mathfrak{B}_l должно иметь вид $\mathfrak{B}_j \rightarrow \mathfrak{B}_k$. В случаях «а»—«в» доказательство точно такое же, как и для \mathfrak{B}_1 (случаи «а», «б» — с помощью аксиомы II 1; случай «в» — с помощью примера 6.1, а). В случае «г» по индуктивному предположению имеем:

$$\Gamma \vdash \mathfrak{A} \rightarrow \mathfrak{B}_j \quad (6.1)$$

и $\Gamma \vdash \mathfrak{A} \rightarrow \mathfrak{B}_l$, т. е.

$$\Gamma \vdash \mathfrak{A} \rightarrow (\mathfrak{B}_j \rightarrow \mathfrak{B}_k). \quad (6.2)$$

Подставив в схему аксиом II 2 \mathfrak{B}_j вместо \mathfrak{B} и \mathfrak{B}_k вместо \mathfrak{C} , получим:

$$(\mathfrak{A} \rightarrow (\mathfrak{B}_j \rightarrow \mathfrak{B}_k)) \rightarrow ((\mathfrak{A} \rightarrow \mathfrak{B}_j) \rightarrow (\mathfrak{A} \rightarrow \mathfrak{B}_k)). \quad (6.3)$$

Применив правило MP к (6.2) и (6.3), получим:

$$\Gamma \vdash (\mathfrak{A} \rightarrow \mathfrak{B}_j) \rightarrow (\mathfrak{A} \rightarrow \mathfrak{B}_k), \quad (6.4)$$

а применив то же правило к (6.1) и (6.4), имеем: $\Gamma \vdash \mathfrak{A} \rightarrow \mathfrak{B}_k$. Остается положить $k = n$. \square

Отметим, что при построении выводов использовались только аксиомы II 1 и II 2, которые содержатся и в системе аксиом I. Поэтому приведенное доказательство теоремы дедукции справедливо и для исчисления высказываний, основанного на системе I.

Теорема 6.16. (обратная теорема о дедукции). Если существует вывод $\Gamma \vdash A \rightarrow B$, то формула B выводима из Γ и A , т. е. если $\Gamma \vdash A \rightarrow B$, то $\Gamma, A \vdash B$.

Доказательство. Пусть вывод формулы $A \rightarrow B$ имеет вид: $B_1, \dots, B_{n-1}, A \rightarrow B$, где B_1, \dots, B_{n-1} — формулы из множества Γ . Тогда вывод формулы B из Γ и A будет иметь вид: $B_1, \dots, B_{n-1}, A \rightarrow B, A, B$, так как B следует из $A \rightarrow B$ и A по правилу MP. \square

Пример 6.2. а. В качестве первого применения теоремы дедукции покажем, что аксиома II 3 выводима из системы аксиом I.

1. Подставим в I 9 $\neg A$ вместо A . Получим:

$$(\neg A \rightarrow B) \rightarrow ((\neg A \rightarrow \neg B) \rightarrow \neg \neg A).$$

2. Двойное применение теоремы 6.16 к шагу 1 дает:

$$\neg A \rightarrow B, \neg A \rightarrow \neg B \vdash \neg \neg A.$$

3. Так как из аксиомы I 10 следует по правилу MP, что $\neg \neg A \vdash A$, то по транзитивности выводимости (см. замечание к примеру 6.1, в) получим $\neg A \rightarrow B; \neg A \rightarrow \neg B \vdash A$.

4. Переставим гипотезы в полученной выводимости (их порядок не важен, как видно из определения выводимости):

$$\neg A \rightarrow \neg B; \neg A \rightarrow B \vdash A.$$

5. Применив 2 раза к шагу 4 теорему дедукции, получим аксиому II 3:

$$(\neg A \rightarrow \neg B) \rightarrow ((\neg A \rightarrow B) \rightarrow A).$$

Отметим, что при доказательстве выводимости системы II из системы I были использованы аксиомы системы I, не содержащие дизъюнкции и конъюнкции.

б. Очень распространенным методом математических доказательств является метод *доказательства от противного*: предполагаем, что A верно, и показываем, что, во-первых, из A выводится B , а во-вторых, что из A выводится $\neg B$, что невозможно, и, следовательно, A неверно, т. е. верно $\neg A$. Этот метод формулируется как правило: «если $\Gamma, A \vdash B$ и $\Gamma, A \vdash \neg B$, то $\Gamma \vdash \neg A$ ». Докажем, что оно в исчислении высказываний выполняется. Действительно, по теореме дедукции, если $\Gamma, A \vdash B$ и $\Gamma, A \vdash \neg B$, то $\Gamma \vdash A \rightarrow B$ и $\Gamma \vdash A \rightarrow \neg B$. Из этих импликаций и аксиомы I 9 двойным применением правила MP получаем $\Gamma \vdash \neg A$. Доказанное правило называется также правилом *введения отрицания*.

в. Докажем теперь закон исключенного третьего: $\vdash A \vee \neg A$.

1. $\neg(A \vee \neg A), A \vdash A \vee \neg A$ (аксиома I 6 при $B = \neg A$ и правило MP).

2. $\neg(A \vee \neg A), A \vdash \neg(A \vee \neg A)$ (очевидно).

3. Применяя к шагам 1 и 2 только что доказанное правило введения отрицания, получаем:

$$\neg(A \vee \neg A) \vdash \neg A.$$

4. Аналогично доказывается $\neg(A \vee \neg A) \vdash \neg \neg A$.

5. Применяя к шагам 3 и 4 введение отрицания, получаем:

$$\vdash \neg \neg(A \vee \neg A).$$

6. С помощью аксиомы I 10 и правила MP снимаем двойное отрицание в шаге 5 и получаем $\vdash A \vee \neg A$.

Исчисление высказываний и алгебра логики. Формула F исчисления высказываний содержательно интерпретируется как составное высказывание, истинность которого зависит от истинности входящих в него элементарных высказываний. Эта зависимость в точности соответствует зависимости значения логической функции, представляемой формулой F , от значений переменных этой функции. Иначе говоря, если задана формула $F(A_1, \dots, A_n)$ и распределение истинностей входящих в нее элементарных высказываний A_1, \dots, A_n , то для выяснения

истинности ее нужно вычислить методами, приведенными в гл. 3, как логическую функцию на наборе $(\sigma_1, \dots, \sigma_n)$, где $\sigma_i = 1$, если A_i истинно, и $\sigma_i = 0$, если A_i ложно. (В этом смысле можно считать, что набор $(\sigma_1, \dots, \sigma_n)$ задает распределение истинностей.) Если $F(\sigma_1, \dots, \sigma_n) = 1$, то высказывание F истинно при данном распределении истинностей A_1, \dots, A_n , если $F(\sigma_1, \dots, \sigma_n) = 0$, то F ложно.

Возникает вопрос: как связано такое содержательное, «истинностное» истолкование формул с их выводимостью в исчислении высказываний? Для описания этой связи введем обозначения, аналогичные введенным в § 3.2 обозначениям вида x^σ . Пусть для элементарных высказываний A_1, \dots, A_n задано распределение истинностей $(\sigma_1, \dots, \sigma_n)$. Обозначим

$$A_i^{\sigma_i} = \begin{cases} A_i, & \text{если } \sigma_i = 1, \text{ т. е. если } A_i \text{ истинно;} \\ \neg A_i, & \text{если } \sigma_i = 0, \text{ т. е. если } A_i \text{ ложно.} \end{cases}$$

Теорема 6.2. Пусть формула $\mathfrak{A}(A_1, \dots, A_n)$ определяет логическую функцию f от n переменных. Тогда, если $f(\sigma_1, \dots, \sigma_n) = \sigma$, то в исчислении высказываний $A_n^{\sigma_1}, \dots, A_n^{\sigma_n} \vdash \mathfrak{A}^\sigma$.

Например, формула $\mathfrak{A}(A_1, A_2, A_3) = (A_1 \rightarrow A_2) \rightarrow A_3$ на наборе $(0, 1, 0)$ равна нулю. Тогда теорема утверждает, что

$$\neg A_1, A_2, \neg A_3 \vdash \neg((A_1 \rightarrow A_2) \rightarrow A_3).$$

Доказательство теоремы проводится индукцией по построению формулы (см. определение формулы).

Если \mathfrak{A} — буква A_i , то утверждение теоремы сводится к $A_i \vdash A_i$ и $\neg A_i \vdash \neg A_i$, что тривиально верно.

Пусть теорема верна для некоторых формул \mathfrak{B} и \mathfrak{C} . Тогда нужно доказать, что она верна для формул \mathfrak{A} , имеющих вид $\neg \mathfrak{B}$, $\mathfrak{B} \rightarrow \mathfrak{C}$, $\mathfrak{B} \& \mathfrak{C}$ и $\mathfrak{B} \vee \mathfrak{C}$. Перебор всех случаев опускаем; приведем лишь некоторые.

Пусть $\mathfrak{A} = \neg \mathfrak{B}$ и при заданном $(\sigma_1, \dots, \sigma_n) \mathfrak{B}$ ложно. По индуктивному предположению $A^{\sigma_1}, \dots, A^{\sigma_n} \vdash \neg \mathfrak{B}$ и, следовательно, $A^{\sigma_1}, \dots, A^{\sigma_n} \vdash \mathfrak{A}$, что и требовалось.

Пусть $\mathfrak{A} = \mathfrak{B} \rightarrow \mathfrak{C}$ и \mathfrak{C} истинно. Тогда (в силу свойств импликации) A тоже истинно и $A^\sigma = A$. По индуктивному предположению $A^{\sigma_1}, \dots, A^{\sigma_n} \vdash \mathfrak{C}$. Но тогда по производному правилу из примера 6.1, б $A^{\sigma_1}, \dots, A^{\sigma_n} \vdash \mathfrak{B} \rightarrow \mathfrak{C}$, т. е. $A^{\sigma_1}, \dots, A^{\sigma_n} \vdash \mathfrak{A}$, что и требовалось.

Полный перебор всех случаев для системы аксиом II можно найти в [20]. \square

Теоремы исчисления высказываний в терминах истинности характеризуются довольно просто.

Теорема 6.3. Всякая теорема исчисления высказываний является тождественно-истинным высказыванием.

Тождественная истинность аксиом проверяется либо прямым вычислением на всех наборах, либо приведением их к константе 1 путем тождественных преобразований булевой алгебры. Очевидно, что любая подстановка в тождественно-истинную формулу также даст тождественно-истинную формулу.

Остается показать, что правило МР сохраняет тождественную истинность. Пусть формулы \mathfrak{A} и $\mathfrak{A} \rightarrow \mathfrak{B}$ тождественно-истинны, т. е. $\mathfrak{A} \equiv 1$, $\mathfrak{A} \rightarrow \mathfrak{B} \equiv 1$.

Так как $A \rightarrow B \equiv \neg A \vee B$, то $0 \vee \mathfrak{B} \equiv 1$ и $\mathfrak{B} \equiv 1$, т. е. формула \mathfrak{B} , выводимая из \mathfrak{A} и $\mathfrak{A} \rightarrow \mathfrak{B}$ по правилу МР, также тождественно-истинна.

Итак, аксиомы тождественно-истинны, а правила вывода сохраняют тождественную истинность; поэтому любая доказуемая формула тождественно-истинна. \square

Справедлива и обратная теорема.

Теорема 6.4. Всякая тождественно-истинная формула является теоремой исчисления высказываний.

Пусть $\mathfrak{A}(A_1, \dots, A_n)$ — тождественно-истинная формула. Тогда по теореме 6.2 $A^{\sigma_1}, \dots, A^{\sigma_n} \vdash \mathfrak{A}$ для всех 2^n наборов $(\sigma_1, \dots, \sigma_n)$. Применив n раз теорему дедукции, получим 2^n выводимостей

$$\vdash A_1^{\sigma_1} \rightarrow (A_2^{\sigma_2} \rightarrow (\dots \rightarrow (A_n^{\sigma_n} \rightarrow \mathfrak{A}))\dots).$$

Подставим в аксиому I 8 $\neg A$ вместо B . Получим $(A \rightarrow C) \rightarrow ((\neg A \rightarrow C) \rightarrow ((A \vee \neg A) \rightarrow C))$. Из этой формулы и правила МР (учитывая, что $A \vee \neg A$ — теорема: см. пример 6.2, в) получаем производное правило: если $\vdash A \rightarrow C$ и $\vdash \neg A \rightarrow C$, то $\vdash C$. Для любого набора $(\sigma_2, \dots, \sigma_n)$ имеем при $\sigma_1 = 1 \vdash A_1 \rightarrow (A_2^{\sigma_2} \rightarrow (\dots \rightarrow (A_n^{\sigma_n} \rightarrow \mathfrak{A}))\dots)$, а при $\sigma_1 = 0 \vdash \neg A_1 \rightarrow (A_2^{\sigma_2} \rightarrow (\dots \rightarrow (A_n^{\sigma_n} \rightarrow \mathfrak{A}))\dots)$. Применяя к этим формулам только что доказанное правило, получаем $\vdash (A_2^{\sigma_2} \rightarrow \dots \rightarrow (A_n^{\sigma_n} \rightarrow \mathfrak{A}))\dots$, т. е. первая посылка «длинной импликации» удалена. Применяя это удаление еще $n - 1$ раз, получаем $\vdash \mathfrak{A}$. \square

Таким образом, исчисление высказываний действительно выполняет задачу порождения общелогических законов — тождественно-истинных высказываний.

В заключение отметим следующее. Эквивалентные соотношения булевой алгебры соединяют знаком = формулы, которые одновременно равны нулю или единице. В логике такая равнозначность выражается функцией \sim (см. § 3.1); если $f_1 = f_2$ — эквивалентное соотношение, то формула $f_1 \sim f_2$ является тождественно-истинным высказыванием. В исчислении высказываний формула $f_1 \sim f_2$ является сокращением формулы $(f_1 \rightarrow f_2) \& (f_2 \rightarrow f_1)$. В силу теоремы 6.4 все такие эквивалентности, например $A \vee B \sim \bar{A} \& \bar{B}$, являются теоремами исчисления высказываний.

Некоторые общие свойства исчисления высказываний будут рассмотрены в § 6.3.

6.2. ИСЧИСЛЕНИЕ ПРЕДИКАТОВ И ТЕОРИИ ПЕРВОГО ПОРЯДКА

Аксиомы и правила вывода. 1. Алфавит исчисления предикатов состоит из предметных переменных x_1, x_2, \dots , предметных констант a_1, a_2, \dots , предикатных букв $P_1^1, P_1^2, \dots, P_k^i$ и функциональных букв $f_1^1, f_2^1, \dots, f_k^i$, ..., а также знаков логических связок $\vee, \&, \neg, \rightarrow$, кванторов \forall, \exists и скобок $(,)$.

Верхние индексы предикатных и функциональных букв указывают число аргументов, их нижние индексы служат для обычной нумерации букв. Переменные высказывания в исчисление предикатов вводятся либо непосредственно как пропозициональные буквы A_1, A_2, \dots , либо как 0-местные предикаты P_1^0, P_2^0, \dots , т. е. как предикаты без предметных переменных.

2. Формулы. Понятие формулы определяется в два этапа.

1) Термы:

- а) предметные переменные и константы являются термами;
- б) если f^n — функциональная буква, а t_1, \dots, t_n — термы, то $f^n(t_1, \dots, t_n)$ — терм.

2) Ф о р м у л ы:

а) если P^n — предикатная буква, а t_1, \dots, t_n — термы, то $P^n(t_1, \dots, t_n)$ — формула; все вхождения предметных переменных в формулу вида $P^n(t_1, \dots, t_n)$ называются свободными;

б) если F_1, F_2 — формулы, то формулами являются $\neg F_1, (F_1 \rightarrow F_2), (F_1 \vee F_2), (F_1 \& F_2)$; все вхождения переменных, свободные в F_1, F_2 , являются свободными и в указанных четырех видах формул;

в) если $F(x)$ — формула, содержащая свободные вхождения переменной x , то $\forall x F(x)$ и $\exists x F(x)$ — формулы; в этих формулах все вхождения переменной x называются связанными; свободные вхождения остальных переменных в F остаются свободными.

Функциональные буквы и термы введены «впрок» для целей различных прикладных исчислений предикатов. Числовое исчисление предикатов строится для произвольной предметной области; структура этой области и связи между ее элементами не имеют значения, поэтому в нем функциональные буквы и термы не обязательны*. В прикладных исчислениях (например, в формальной арифметике) структура предметной области оказывается существенной, поэтому в исчислении необходимо иметь средства для описания связей между элементами, т. е. функций и отношений, определенных на области. Отношениям соответствуют предикатные буквы, функциям — функциональные буквы. Термы — это имена элементов предметной области, построенные с помощью функций. Они могут быть постоянными (если они построены из предметных констант) и переменными. Формулы — это высказывания о термах. Например, $4 + 5 \cdot 3$ — постоянный терм любого исчисления, содержащего функциональные буквы, $+$ и \cdot , а $x + 7$ — переменный терм этого же исчисления. Выражение $4 + 5 \cdot 3 = x + 7$ — это переменное высказывание, полученное подстановкой двух термов в двухместный предикат равенства; его истинность зависит от значения переменной x .

3. А к с и о м ы и с ч и с л е н и я п р е д и к а т о в д е л я ю т с я на две группы:

1) аксиомы исчисления высказываний (можно взять любую из систем I или II);

* В частности, приводимые далее аксиомы Р1 и Р2 не учитывают наличия термов в формулах. Для исчисления с термами и функциональными буквами эти аксиомы имеют вид Р1' и Р2'.

2) две предикатные аксиомы:

$$P1. \forall x F(x) \rightarrow F(y);$$

$$P2. F(y) \rightarrow \exists x F(x).$$

В этих аксиомах $F(x)$ — любая формула, содержащая свободные вхождения x , причем ни одно из них не находится в области действия квантора по y ; формула $F(y)$ получена из $F(x)$ заменой всех *свободных* вхождений x на y .

Чтобы пояснить существенность требования к вхождениям x в F , рассмотрим в качестве $F(x)$ формулу $\exists y P(y, x)$, где это требование нарушено: свободное вхождение x находится в области действия $\exists y$. Подстановка этой формулы в аксиому P1 дает формулу $\forall x \exists y P(y, x) \rightarrow \exists y P(y, y)$. Если ее проинтерпретировать на множестве N натуральных чисел с предикатом P «быть больше», то получим высказывание: «если для всякого x найдется y , больший x , то найдется y , больший самого себя». Посылка этой импликации истинна на N , а ее заключение ложно, и, следовательно, само высказывание ложно.

4. Правила вывода:

1) правило Modus Ponens — то же, что и в исчислении высказываний;

2) правило обобщения (\forall -введения):

$$\frac{F \rightarrow G(x)}{F \rightarrow \forall x G(x)},$$

где $G(x)$ содержит свободные вхождения x , а F их не содержит;

3) правило \exists -введения:

$$\frac{G(x) \rightarrow F}{\exists x G(x) \rightarrow F}$$

при тех же требованиях к F и G , что и в предыдущем правиле.

Нарушения этих требований могут привести к ложным выводам из истинных высказываний. Пусть, например, $P(x)$ — предикат « x делится на 6», $Q(x)$ — предикат « x делится на 3». Высказывание $P(x) \rightarrow Q(x)$, очевидно, истинно для любого x , однако применение к нему правила обобщения дает высказывание $P(x) \rightarrow \forall x Q(x)$, не являющееся всегда истинным. Если же к $P(x) \rightarrow Q(x)$ при-

менить правило \exists -введения, то получим $\exists xP(x) \rightarrow Q(x)$, из которого путем (уже корректного!) применения правила обобщения получим высказывание $\exists xP(x) \rightarrow \forall xQ(x)$, ложное на множестве натуральных чисел.

Приведенные здесь аксиомы и правила вывода содержатся в [11]. Возможны и другие системы аксиом и правил (см., например, [14, 20]). В частности, в [20] последовательно проводится принцип минимизации числа логических операторов, который в исчислении высказываний оставляет лишь связки \neg и \rightarrow (что отражено в системе аксиом II). В исчислении предикатов он выражается в том, что квантор $\exists x$ не считается самостоятельным символом, а рассматривается как сокращение выражения $\neg \forall x \neg$: например, выражение $\exists xP(x)$ эквивалентно выражению $\neg \forall x \neg P(x)$.

Читатель, вероятно, уже заметил, что правило подстановки окончательно исчезло из изложения; тем самым из двух возможных истолкований системы аксиом (о чём шла речь в исчислении высказываний) выбрано второе, при котором правило подстановки отсутствует, а вместо аксиом рассматриваются схемы аксиом. Фактически этот выбор произошел уже тогда, когда аксиомы Р1 и Р2 были сопровождены словесным описанием ограничений на вхождения переменных. Тем самым аксиомы перестали быть выражениями исчисления, а вместе с этим словесным текстом превратились в метаописания множества формул, являющихся аксиомами, т. е. в схемы аксиом.

Построение исчисления предикатов с правилом подстановки существенно более громоздко из-за необходимости различать свободные и связанные вхождения предметных переменных (см., например, [23]). Поэтому в большинстве современных книг по логике используется подход со схемами аксиом. Предполагая, что после знакомства с исчислением высказываний разница между переменными и метапеременными уже усвоена, не будем больше употреблять готические буквы; в качестве метапеременных, обозначающих формулы, в этом разделе будут использоваться буквы F и G .

Приведем теперь примеры вывода в исчислении предикатов.

Пример 6.3. а. Покажем, что в исчислении предикатов из выводимости формулы $F(x)$, содержащей свободные вхождения x , ни одно из которых не находится в области действия квантора по y , следует выводимость $F(y)$. Это утверждение представляет собой *правило переименования свободных переменных*.

1. $\vdash F(x)$ (по условию).
2. $F(x) \rightarrow (G \rightarrow F(x))$ (аксиома II 1; в качестве G выбираем любую доказуемую формулу, не содержащую свободных вхождений x : ее доказуемость понадобится на шаге 5, а ограничение на x — на шаге 4).
3. $G \rightarrow F(x)$ (правило MP к шагам 1 и 2).
4. $G \rightarrow \forall x F(x)$ (правило обобщения к шагу 3).
5. $\forall x F(x)$ правило MP к G и шагу 4).
6. $F(y)$ (правило MP к шагу 5 и аксиоме P1).
6. В исчислении предикатов из выводимости $\forall x F(x)$ следует выводимость $\forall y F(y)$, а из выводимости $\exists x F(x)$ — выводимость $\exists y F(y)$ при условии, что $F(x)$ не содержит свободных вхождений y и содержит свободные вхождения x , ни одно из которых не входит в область действия квантора по y (*правило переименования связанных переменных*).

Докажем это правило для квантора общности.

1. $\vdash \forall x F(x)$ (по предположению).
2. $\forall x F(x) \rightarrow F(y)$ (аксиома P1).
3. $\forall x F(x) \rightarrow \forall y F(y)$ (правило обобщения к шагу 2).
4. $\forall y F(y)$ (правило заключения к шагам 1 и 3).

Доказательство для \exists совершенно аналогично, но использует аксиому P2 и правило \exists -введения.

Выводимость и истинность. Эквивалентные преобразования.

Теорема 6.5. Всякая доказуемая формула исчисления предикатов тождественно-истинна (общезначима — см. § 3.4).

Эта теорема доказывается аналогично теореме 6.3: непосредственно проверяется общезначимость аксиом и показывается, что правила вывода сохраняют общезначимость, т. е. их применение к общезначимым формулам снова дает общезначимые формулы. \square

Теорема 6.6. Всякая общезначимая предикатная формула доказуема в исчислении предикатов.

Доказательство этой теоремы намного более сложно; здесь оно опускается.

В конце раздела об исчислении высказываний вскользь было упомянуто о том, что всякому эквивалентному соотношению $F = G$ в булевой алгебре соответствует доказуемая эквивалентность $\vdash F \sim G$ в исчислении высказываний. Из теорем 6.5 и 6.6 следует, что между соотношениями содержательной логики предикатов (см. § 3.4) и формальными эквивалентностями в исчислении предикатов имеется аналогичное соответствие (напомним, что $F \sim G$ рассматривается как сокращение $(F \rightarrow G) \& (G \rightarrow F)$). На нем имеет смысл остановиться подробнее. Дело в том, что доказательство общезначимости в логике предикатов существенно сложнее, чем в логике высказываний (об этом уже говорилось в гл. 3), и поэтому формальный вывод эквивалентностей становится важным способом их получения.

Теорема 6.7. Пусть $F(A)$ — формула, в которой выделено вхождение формулы A ; $F(B)$ — формула, полученная из $F(A)$ заменой этого вхождения A формулой B . Тогда если $\vdash A \sim B$, то $\vdash F(A) \sim F(B)$.

Эта теорема формулирует для исчисления предикатов правило, аналогичное правилу замены эквивалентных подформул в алгебрах (см. гл. 3). Благодаря ему можно получать доказуемые эквивалентности в исчислении, не строя их непосредственного вывода.

При доказательстве теоремы используются следующие производные правила:

- 1) если $\vdash A \sim B$, то $\vdash A \rightarrow C \sim B \rightarrow C$ и $C \rightarrow A \sim C \rightarrow B$;
- 2) если $\vdash A \sim B$, то $\vdash A \vee C \sim B \vee C$ и $\vdash C \vee A \sim C \vee B$;
- 3) если $\vdash A \sim B$, то $\vdash A \& C \sim B \& C$ и $\vdash C \& A \sim C \& B$;
- 4) если $\vdash A \sim B$, то $\vdash \neg A \sim \neg B$;
- 5) если $\vdash F(x) \sim G(x)$, то $\forall x F(x) \sim \forall x G(x)$;
- 6) если $\vdash F(x) \sim G(x)$, то $\exists x F(x) \sim \exists x G(x)$.

Первые четыре правила легко проверяются с помощью таблиц истинности. Доказательства последних двух правил можно найти, например, в [20].

С помощью этих шести правил теорема 6.7 доказывается индукцией по построению формулы $F(A)$: показывается, что если $A \sim B$, то эта эквивалентность сохраняется на всех шагах построения $F(A)$ из A и $F(B)$ из B . \square

Продемонстрируем это доказательство на примере.

Пусть $F(A) = \forall y(P_1(y) \vee \neg \exists x P_2(x, y))$, $A = \neg \exists x P_2(x, y)$.

В качестве эквивалентности $A \sim B$ возьмем эквивалентность $\neg \exists x P_2(x, y) \sim \forall x \neg P_2(x, y)$, верную в логике предикатов [см. соотношение (3.31)] и, следовательно, в силу теоремы 6.7 доказуемую в исчислении предикатов.

1. $\neg \exists P_2(x, y) \sim \forall x \neg P_2(x, y)$ (исходная эквивалентность $A \sim B$).

2. $(P_1(y) \vee \neg \exists x P_2(x, y)) \sim (P_1(y) \vee \forall x \neg P_2(x, y))$ (правило 2).

3. $\forall y(P_1(y) \vee \neg \exists x P_2(x, y)) \sim \forall y(P_1(y) \vee \forall x \neg P_2(x, y))$ (правило 5).

Формула из шага 3 и есть искомая эквивалентность $F(A) \sim F(B)$.

Приведем теперь без доказательства некоторые важные эквивалентности, выводимые в исчислении предикатов (в них A и B — формулы, не содержащие свободных вхождений x):

$$A \& \forall x F(x) \sim \forall x (A \& F(x)); \quad (6.5)$$

$$A \vee \exists x F(x) \sim \exists x (A \vee F(x)); \quad (6.6)$$

$$A \& \exists x F(x) \sim \exists x (A \& F(x)); \quad (6.7)$$

$$A \vee \forall x F(x) \sim \forall x (A \vee F(x)); \quad (6.8)$$

$$A \rightarrow \forall x F(x) \sim \forall x (A \rightarrow F(x)); \quad (6.9)$$

$$A \rightarrow \exists x F(x) \sim \exists x (A \rightarrow F(x)); \quad (6.10)$$

$$\forall x F(x) \rightarrow B \sim \exists x (F(x) \rightarrow B); \quad (6.11)$$

$$\exists x F(x) \rightarrow B \sim \forall x (F(x) \rightarrow B). \quad (6.12)$$

Эквивалентности (6.5)–(6.12), а также полученные ранее эквивалентности (3.33) и (3.34) позволяют выносить кванторы вперед. Используя при этом соотношения (3.31) и (3.32), позволяющие заменять один квантор другим и «спускать» отрицание внутрь области действия квантора, а также правила переименования переменных (примеры 6.3, а, б), кванторы можно вынести вперед для любой формулы. Формула, имеющая вид $Q_1 x_1 Q_2 x_2 \dots Q_n x_n F$, где Q_1, \dots, Q_n — кванторы, а F — формула, не имеющая кванторов (и являющаяся областью действия всех n кванторов), называется *предваренной формой*, или формулой в предваренной форме.

В исчислении предикатов для любой формулы F существует эквивалентная ей предваренная форма F' , т. е. $\vdash F \sim F'$.

Пример 6.4. а. Приведем к предваренной форме формулу, которой была проиллюстрирована теорема 6.7:

1. $\forall y(P_1(y) \vee \neg \exists x P_2(x, y))$;
2. $\forall y(P_1(y) \vee \forall x \neg P_2(x, y))$ [по соотношению (3.31)];
3. $\forall y \forall x(P_1(y) \vee \neg P_2(x, y))$ [по соотношению (6.8)].

6. Проделаем то же самое с несколько более сложной формулой:

1. $\forall x P_1(x) \rightarrow \neg \forall x(P_2(y) \vee \exists y P_3(x, y))$;
2. $\forall x P_1(x) \rightarrow \neg \forall x(P_2(z) \vee \exists y P_3(x, y))$ (переименование свободной переменной);
3. $\forall x P_1(x) \rightarrow \neg \forall x \exists y(P_2(z) \vee P_3(x, y))$ [соотношение (6.6)];
- 4, 5. $\forall x P_1(x) \rightarrow \exists x \forall y \neg(P_2(z) \vee P_3(x, y))$ [здесь объединены два шага вывода: применение (3.31), а затем (3.32)];
6. $\forall u P_1(u) \rightarrow \exists x \forall y \neg(P_2(z) \vee P_3(x, y))$ (переименование связанной переменной);
- 7, 8, 9. $\exists x \exists u \forall y(P_1(u) \rightarrow \neg(P_2(z) \vee P_3(x, y)))$ [здесь объединены три шага вывода: применения (6.10), (6.11), (6.9)].

Теории первого порядка (прикладные исчисления предикатов). Исчисления с равенством. Формальная арифметика. Построенное ранее исчисление предикатов называется исчислением предикатов первого порядка. В исчислениях второго порядка возможны кванторы по предикатам, т. е. выражения вида $\forall P(P(x))$. Приложения таких исчислений встречаются гораздо реже; в этой книге исчисления второго порядка рассматриваться не будут.

Исчисление предикатов, не содержащее функциональных букв и предметных констант, называется чистым исчислением предикатов. По существу до сих пор рассматривалось именно чистое исчисление предикатов, хотя язык исчисления (т. е. формулы) был определен с учетом его использования в прикладных исчислениях.

Прикладные исчисления (теории первого порядка) характеризуются тем, что в них к чисто логическим аксиомам добавляются собственные аксиомы, в которых, как правило, участвуют конкретные (индивидуальные) предикатные буквы и предметные константы. Типичные примеры индивидуальных предикатных букв — предикаты $=$, $<$, функциональных букв — знаки арифметических

операций, предметных констант — натуральные числа, единица в теории групп, пустое множество в теории множеств.

Другая важная особенность прикладных исчислений заключается в том, что в схемах аксиом P1 и P2 участвуют уже не предметные переменные, а произвольные термы. Более точно эти схемы аксиом принимают следующий вид:

$$\begin{aligned} \text{P1'}. \forall x F(x) \rightarrow F(t); \\ \text{P2'}. F(t) \rightarrow \exists x F(x), \end{aligned}$$

где $F(t)$ — результат подстановки терма t в $F(x)$ вместо всех свободных вхождений x , причем все переменные t должны быть свободными в $F(t)$.

Большинство прикладных исчислений содержит предикат равенства $=$ и определяющие его аксиомы. Аксиомами для равенства могут служить следующие.

- E1. $\forall x (x = x)$ (конкретная аксиома);
- E2. $(x = y) \rightarrow (F(x, x) \rightarrow F(x, y))$ (схема аксиом),

где $F(x, y)$ получается из $F(x, x)$ заменой некоторых (не обязательно всех) вхождений x на y при условии, что y в этих вхождениях также остается свободным. Всякая теория, в которой E1 и E2 являются теоремами или аксиомами, называется *теорией* (или исчислением) с равенством. Дело в том, что из E1 и E2 выводимы основные свойства равенства — рефлексивность, симметричность и транзитивность.

Теорема 6.8. В любой теории с равенством:

- 1) $\vdash t = t$ для любого терма t ;
- 2) $\vdash x = y \rightarrow y = x$;
- 3) $\vdash x = y \rightarrow (y = z \rightarrow x = z)$.

Доказательство.

1. Непосредственно следует из аксиом E1 и P1' (где $F(x)$ имеет вид $x = x$) по правилу MP.

2. Из E2 имеем $(x = y) \rightarrow (x = x \rightarrow y = x)$. Отсюда $x = y, x = x \vdash y = x$ (двойное применение правила MP). Но так как $\vdash x = x$, то $x = y \vdash y = x$ и, следовательно, по теореме дедукции* $x = y \rightarrow y = x$.

* Теорема дедукции доказывалась ранее только для исчисления высказываний (для исчисления предикатов она верна лишь при некоторых ограничениях). Но данные формулы не содержат кванторов, поэтому эта теорема применима.

3. Поменяем местами в Е2 x и y , в качестве $F(y, y)$ возьмем $y = z$, а в качестве $F(y, x)$ возьмем $x = z$. Получим $y = x \rightarrow (y = z \rightarrow x = z)$ и по правилу МР $y = x \vdash (y = z \rightarrow x = z)$. Но так как в силу п. 2 $x = y \vdash y = x$, то по транзитивности выводимости (см. § 6.1) получаем $x = y \vdash (y = z \rightarrow x = z)$ и по теореме дедукции $x = y \rightarrow (y = z \rightarrow x = z)$. \square

Три свойства равенства, полученные этой теоремой, верны, как известно, для любого отношения (и, следовательно, соответствующего предиката) эквивалентности (см. гл. 1). Схема аксиом Е2 выражает более сильное свойство, присущее лишь равенству: неотличимость элементов, для которых выполняется равенство.

Другой пример прикладного исчисления — теория частичного строгого порядка, содержащая две конкретные аксиомы для предиката $<$:

$$\text{NE1. } \forall x \neg(x < x);$$

$$\text{NE2. } \forall x \forall y \forall z (x < y \rightarrow (y < z \rightarrow x < z)).$$

Читателя может удивить, что два сходных утверждения о транзитивности даются в разной форме: п. 3 теоремы 6.8 без кванторов, в открытой форме, а NE2 — с кванторами, в замкнутой форме. В действительности эти два способа задания аксиом равносильны: от первого ко второму переходим по правилу обобщения, а от второго к первому — по аксиоме Р1' и правилу МР.

Если к NE1 и NE2 добавить аксиому с предметной константой NE3 $\forall x \neg(x < a)$, то получим теорию частичного порядка с минимальными элементом a .

Пожалуй, наиболее изученной формальной теорией, которая играет фундаментальную роль в основаниях математики, является *формальная арифметика*. Ее схемы аксиом (см. [11]):

A1. $F(0) \& \forall x(F(x) \rightarrow F(x')) \rightarrow \forall x F(x)$ (принцип индукции);

$$\text{A2. } t'_1 = t'_2 \rightarrow t_1 = t_2;$$

$$\text{A3. } \neg(t'_1 = 0);$$

$$\text{A4. } t_1 = t_2 \rightarrow (t_1 = t_3 \rightarrow t_2 = t_3);$$

$$\text{A5. } t_1 = t_2 \rightarrow t'_1 = t'_2;$$

$$\text{A6. } t_1 + 0 = t_1;$$

$$\text{A7. } t_1 + t'_2 = (t_1 + t_2');$$

A8. $t_1 \cdot 0 = 0$;

A9. $t_1 \cdot t'_2 = t_1 \cdot t_2 = t_1$.

В этих аксиомах использованы три функциональные символы $+$, \cdot , $'$, один индивидуальный предикат (предикатная буква) $=$ и одна предметная константа 0. Они придают схемам A2–A9 вполне конкретный вид: все предикатные и функциональные буквы в них зафиксированы, и единственный способ их варьировать — это подставлять различные термы вместо метапеременных t_1 , t_2 . В частности, схемы A6–A9 — это просто предикат равенства, в который подставлены термы определенного вида.

Схема A1 имеет обычный вид; $F(x)$ — метаобозначение, употребляемое в том же смысле, в каком оно использовалось в чистом исчислении предикатов. Впрочем, схемы A2–A9 можно заменить конкретными аксиомами (т. е. формулами самой арифметики): A2'. $x'_1 = x'_2 \rightarrow x_1 = x_2$ и т. д., из которых любые формулы вида A2–A9 можно получить с помощью правила обобщения и схемы аксиом P1.

В заключение суммируем особенности построения прикладных исчислений предикатов и, в частности, посмотрим, в чем здесь проявляется различие между «схемно-аксиомным» и «подстановочным» подходами к построению исчислений.

1. При «схемно-аксиомном» подходе общелогические схемы аксиом, как уже отмечалось, описываются формулами в метапеременных; собственные аксиомы прикладных исчислений могут задаваться либо также схемами (например, A1), либо конкретными аксиомами, т. е. формулами самого исчисления (например, E1). Однако в любом случае собственные аксиомы, как правило, содержат фиксированные функциональные и предикатные буквы, которые тем самым наделяются некоторыми свойствами, отличающими их от других букв исчисления (например, свойства предикатной буквы $=$ определяются аксиомами E1 и E2).

2. При «подстановочном» подходе системы аксиом I (или II) и P1, P2 являются конечной системой формул самого исчисления. Буква F в аксиомах P1 и P2 — это предикатная буква исчисления (а не метапеременная,

как в случае 1), являющаяся переменным предикатом, вместо которого по правилу подстановки подставляются формулы исчисления. В собственных аксиомах прикладных исчислений появляются постоянные предикаты (например, равенство), вместо которых подставлять ничего нельзя. Поэтому в языке исчисления предикатные и функциональные буквы приходится делить на две группы — переменные и постоянные, что не обязательно при первом подходе.

6.3. МЕТАТЕОРИЯ ЛОГИЧЕСКИХ ИСЧИСЛЕНИЙ

Под метатеорией логических исследований понимается изучение их общих свойств и соответства этих свойств целям, ради которых исчисления создавались. Некоторые задачи такого рода (связь между доказуемостью и истинностью) уже рассматривались. Здесь они будут систематизированы и изучены более подробно.

Интерпретация и модели. Ценность всякой формальной теории в конечном счете определяется ее способностью описывать какие-то объекты и связи между ними. Поэтому один из первых для любой теории вопросов — это вопрос о том, для описания каких объектов пригодна данная теория. Конечно, если ставить его как общую проблему соответства научных знаний о мире самому миру, то он не может обсуждаться средствами лишь математики (поскольку математика в отличие от естественных наук имеет дело не непосредственно с миром, а с формальными описаниями его различных фрагментов) и становится фундаментальной проблемой философии и методологии науки. Такого рода проблемы лежат за пределами этой книги. Здесь речь пойдет о более простом случае, когда множество объектов, для которого строится формальная теория, само по себе представляет достаточно строго описанный предмет исследований. Более точно проблема адекватности формальной теории и описываемых ею объектов будет рассматриваться как математическая задача о соответствии между содержательно построенной теорией, рассматриваемой как множество объектов с операциями и отношениями на нем (т. е. как

алгебраическая система — см. гл. 2), и множеством высказываний об этой теории, построенном как формальное исчисление. При такой постановке задачи интуитивно ясное понятие интерпретации приобретает точный математический смысл.

Интерпретация формальной теории состоит из множества M и однозначного отображения, которое каждой предикатной букве P_i^n ставит в соответствие n -местное отношение на M (интерпретирует ее как отношение на M), каждой функциональной букве f_j^n — n -местную операцию на M , каждой предметной константе — элемент M . Постоянные термы исчисления (не содержащие предметных переменных) при таком определении также отображаются в элементы M . Таким образом, множество M (называемое областью интерпретации) рассматривается как основное множество алгебраической системы (см. гл. 2).

Всякая замкнутая, т. е. не содержащая свободных переменных, формула теории представляет собой высказывание об элементах, отношениях и функциях M , которое может быть истинным или ложным. Значения истинности составных формул вычисляются в соответствии с входящими в них логическими операциями. Открытая формула соответствует некоторому отношению на M , при подстановке предметных констант она превращается в высказывание о том, что между элементами M , соответствующими подставленным константам, выполняется данное отношение. Открытая формула называется *выполнимой** в данной интерпретации, если существует такая подстановка предметных констант, при которой она превращается в истинное высказывание. Формула называется *истинной в данной интерпретации*, если она выполняется (т. е. превращается в истинное высказывание) при любой подстановке констант. Формула называется ложной в данной интерпретации, если она невыполнима.

Интерпретация (а иногда область интерпретации M) называется *моделью* для множества формул Γ , если лю-

* Некоторые из определяемых здесь понятий уже использовались в § 3.4.

бая формула Γ истинна в данной интерпретации. Интерпретация называется *моделью теории T* , если она является моделью множества всех теорем теории T , т. е. если всякая формула, доказуемая в T , истинна в данной интерпретации. Если в T доказуема некоторая открытая формула F (которая, строго говоря, высказыванием не является), то в модели теории T должны быть истинными все высказывания, получающиеся из F всеми возможными подстановками констант на место свободных переменных формулы F , и, следовательно, должно быть истинно высказывание $\forall x_1 \dots \forall x_n F$, где x_1, \dots, x_n — свободные переменные формулы F . Это обстоятельство вполне соответствует правилу обобщения в исчислении предикатов и использованию открытых аксиом (например, E2) в прикладных исчислениях.

Пример 6.5. а. Для теории с равенством, очевидно, моделью является любая интерпретация, при которой предикатной букве $=$ поставлено в соответствие отношение равенства. Возможны и менее тривиальные интерпретации. Возьмем в качестве области интерпретации натуральный ряд N , в качестве интерпретации символа $=$ — некоторое отношение R эквивалентности на N (например, сравнимость по mod11), а все предикатные буквы теории (будем считать, что их конечное число) проинтерпретируем отношениями, которые не различают эквивалентные числа, т. е. по существу являются отношениями между классами эквивалентности. В данном конкретном случае такими отношениями будут отношения, сформулированные в терминах остатков от деления на 11, например: « aR_1b , если остаток от деления a на 11 меньше остатка от деления b на 11», « a обладает свойством R_2 , если остаток от деления a на 11 не превосходит 5» и т. д. Значения истинности высказываний, содержащих только такие отношения, не будут меняться, если в них одно число заменить числом из того же класса эквивалентности, т. е. имеющим тот же остаток от деления на 11. Поэтому аксиома E2 в такой интерпретации будет истинна, хотя интерпретация символа $=$ не совпадает с обычным отношением равенства.

б. В теории строгого частичного порядка (с аксиомами NE1 и NE2) моделью будет любая интерпретация,

при которой предикатная буква $<$ интерпретируется отношением «быть меньше». Однако почти столь же очевидно из аксиом NE1 и NE2 (хотя и выглядит парадоксально), что моделью этой теории будет и интерпретация, при которой символ $<$ интерпретируется как отношение «быть больше»!

Последний пример иллюстрирует существование формального подхода; в символы исчисления (даже самые привычные) не вкладывается никакого смысла, пока не введена их явная интерпретация. Но и введенная интерпретация, вообще говоря, не относится к числу средств самого исчисления: она позволяет осмысливать формулы исчисления, но не участвует в формальном выводе теорем. О формальных свойствах самого исчисления, его формул и формальных преобразований над ними принято говорить как о *синтаксисе исчисления*; свойства исчисления, описанные в терминах его интерпретаций, — это *семантика исчисления*. Например, метатеоремы 6.1, 6.7, 6.8 являются теоремами о синтаксисе, а метатеоремы 6.2–6.6 — теоремами о семантике.

Непротиворечивость. Напомним, что формула называется *общезначимой*, если она истинна в любой интерпретации, и *противоречивой*, если она ложна в любой интерпретации, т. е. если ее отрицание общезначимо. Для любого множества общезначимых формул и, в частности, для чистого исчисления предикатов (по теореме 6.5) любая алгебраическая система и вообще любое множество является моделью. Напротив, для любого множества формул, содержащего хотя бы одну противоречивую формулу, моделей не существует.

Аксиома E1 теории с равенством не является общезначимой: существуют интерпретации (в смысле, указанном в начале этого параграфа), в которых она ложна. Примером такой интерпретации может служить всякое отображение, которое предикатной букве $=$ ставит в соответствие отношение $<$. (Здесь символы $=$ и $<$ используются на разных уровнях: символ $=$ — как формальная предикатная буква, не имеющая смысла до ее интерпретации, а символ $<$ — как символ отношения на множестве, имеющий всем известный смысл «быть меньше».) Собственные аксиомы прикладных исчислений вообще

не общезначимы — иначе по теореме 6.6 они были бы доказуемы в чистом исчислении предикатов.

Введенное ранее определение противоречивой формулы является семантическим, т. е. связывающим непротиворечивость с истинностью. Исходя из него, можно сформулировать понятие семантически непротиворечивой теории: теория *семантически непротиворечива*, если ни одна из ее теорем не является противоречивой, т. е. ложной в любой интерпретации. Исчисление высказываний и исчисление предикатов семантически непротиворечивы в силу теорем 6.3 и 6.5: все их теоремы общезначимы и, следовательно, непротиворечивы.

С помощью введенных понятий ответ на общий вопрос, поставленный в начале параграфа, формулируется так: теория T пригодна для описания тех множеств, которые являются ее моделями; модель для теории T существует тогда и только тогда, когда T семантически непротиворечива. Чисто логические теории — исчисление высказываний и исчисление предикатов — в силу теорем 6.3 и 6.5 пригодны для описания любых множеств, что соответствует общенаучному принципу универсальности законов логики (Лейбниц формулировал его как выполнимость логических законов «во всех мыслимых мирах»). Такой критерий пригодности теорий по существу известен уже давно; отыскание модели для теории до возникновения оснований математики было единственным общепризнанным методом доказательства законности теории. Одно из важных достижений оснований математики заключается в том, что аналог этого критерия сформулирован в терминах самих формальных теорий, без привлечения семантических понятий. Таким аналогом является понятие формальной, или дедуктивной, непротиворечивости.

Теория T называется *формально непротиворечивой*, если не существует формулы F , такой, что F и $\neg F$ являются теоремами теории T , т. е. в T не выводимы одновременно формула и ее отрицание. Аналогичное определение можно сформулировать и для произвольного множества формул.

Для любой теории, содержащей исчисление высказываний, из определения непосредственно следует, что если

она формально противоречива, то в ней доказуема тождественно-ложная формула и, следовательно, она семантически противоречива. Действительно, если F и $\neg F$ доказуемы, то, подставив в аксиому I 5 F вместо A , $\neg F$ вместо B и дважды применив правило заключения, получим $\vdash A \& \neg A$. Более того, никакое множество формул, содержащее F и $\neg F$ (т. е. формально противоречивое), не может иметь модели, поскольку F и $\neg F$ не могут быть одновременно истинными ни в какой интерпретации. Тем самым доказано (от противного), что если множество формул семантически непротиворечиво, то оно формально непротиворечиво.

Обратное утверждение (которое также оказывается верным), если понимать его конструктивно, гораздо глубже. Смысл его в том, что по всякому формально непротиворечивому множеству формул можно построить его модель. Доказательство этого факта — его можно найти, например, в [14] — довольно сложно и заключается в изложении метода такого построения. Вместе оба утверждения образуют важную метатеорему.

Теорема 6.9. Множество формул формально непротиворечиво, если и только если оно семантически непротиворечиво (т. е. имеет модель). \square

Таким образом, понятие формальной непротиворечивости оказывается эквивалентным более привычному в математике понятию семантической непротиворечивости; однако оно сформулировано в синтаксических терминах и с конструктивной точки зрения более надежно (вспомним, что именно семантические трудности — парадоксы и привели к возникновению науки об основаниях математики и к концепциям формального подхода). Привыкнуть же к эквивалентности этих двух понятий в математике было нелегко. Неприятие современниками неевклидовых геометрий Лобачевского—Бойаи объясняется именно тем, что законность этих теорий обосновывалась отсутствием в них противоречий — аргументом, по существу совпадающим с современным понятием формальной непротиворечивости. Геометрические модели для этих теорий, доказывающие их семантическую непротиворечивость, были найдены позднее.

Полнота, разрешимость, аксиоматизируемость. Теперь можно приступить к обсуждению вопросов об адекватности формальных теорий, т. е. соответствии тем целям, ради которых они создавались. Пусть имеется, с одной стороны, содержательная теория S , сформулированная в семантических терминах, т. е. совокупность истинных высказываний о некоторой алгебраической системе M ; с другой стороны — формальная теория T , т. е. множество выражений, выводимых из аксиом теории T с помощью правил вывода теории T . В каких случаях можно утверждать, что T является удовлетворительной формализацией S ? Каковы признаки удовлетворительности формализации?

Очевидно, необходимым признаком является условие, чтобы S была моделью теории T , т. е. чтобы существовало отображение, при котором всякая теорема теории T отображается в истинное высказывание из S . Однако само по себе это условие явно недостаточно, иначе для формализации любой теории S хватило бы чистого исчисления предикатов: ведь для него любое множество является моделью. Исчерпывающей формализацией S будет служить такая теория T , для которой выполняется и обратное соответствие: каждое истинное высказывание теории S отображается в некоторую теорему теории T . Теория T с таким свойством называется *полной* относительно S , а иногда (например, в [14]) — *адекватной* S .

Из теорем 6.3 и 6.4 следует, что исчисление высказываний полно относительно алгебры высказываний, а теоремы 6.5 и 6.6 образуют известную *теорему Геделя о полноте исчисления предикатов* относительно логики предикатов, т. е. множества всех общезначимых предикатных формул.

Если для семантической (содержательной) теории S удается построить непротиворечивую и полную формальную теорию T , то S естественно называть *аксиоматизируемой*, или *формализуемой*, теорией. Из предыдущих результатов следует, что логика высказываний и логика предикатов аксиоматизуемы с помощью соответствующих исчислений.

Еще одна важная характеристика формальной теории — это ее *разрешимость*. Формальная теория T называется *разрешимой*, если существует алгоритм, который

для любой формулы языка определяет, является она теоремой в T или нет, иначе говоря, если предикат «быть теоремой в теории T » общерекурсивен.

Теорема 6.10. Исчисление высказываний разрешимо.

Разрешающий алгоритм для формулы F исчисления высказываний заключается в вычислении значений F на всех наборах значений ее переменных. Ввиду полноты исчисления высказываний F является его теоремой, если и только если она истинна на всех наборах. \square

Теорема 6.11 (Черч). Исчисление предикатов неразрешимо.

Несмотря на полноту исчисления предикатов, разрешающий алгоритм, связанный с вычислением значений истинности предикатных формул, построить не удается по причине, отмеченной в гл. 3, — из-за бесконечности предметной области, которая приводит в общем случае к бесконечным таблицам истинности. Идея доказательства теоремы Черча (которое здесь не приводится; его можно найти в [11]) состоит в том, чтобы в чистом исчислении предикатов описать предикат $Q(i, a, x)$: «машина Тьюринга с номером i , будучи применена к исходным данным a , закончит вычисление в момент x ». Предикат $\exists x Q(i, a, x)$ — это предикат остановки, а $\exists x Q(a, a, x)$ — предикат самоприменимости; неразрешимость обоих предикатов была доказана в гл. 5. \square

Отметим, что важный для приложений фрагмент исчисления предикатов разрешим: исчисление одноместных предикатов (т. е. исчисление, допускающее в формулах только одноместные предикатные буквы) разрешимо.

Еще тяжелее обстоит дело с формальной арифметикой, система аксиом которой приведена в конце § 6.2.

Теорема 6.12 (первая теорема Геделя о неполноте в форме Клини). Любая формальная теория T , содержащая формальную арифметику, неполна: в ней существует (и может быть эффективно построена) замкнутая формула F , такая, что $\neg F$ истинно, но ни F , ни $\neg F$ не выводимы в T .

Теорема 6.13 (вторая теорема Геделя о неполноте). Для любой непротиворечивой формальной теории T , содержащей формальную арифметику, формула, выражющая непротиворечивость T , недоказуема в T . \square

Таким образом, арифметика и теория чисел оказываются неаксиоматизируемыми теориями. В терминах теории алгоритмов сформулированные ранее результаты можно резюмировать так: 1) множество всех истинных высказываний логики высказываний перечислимо и разрешимо; 2) множество всех истинных высказываний логики предикатов перечислимо (ввиду его полноты), но неразрешимо; 3) множество всех истинных высказываний арифметики неперечислимо: если бы для него существовала перечисляющая процедура (и, следовательно, соответствующая машина Тьюринга), то по ней можно было бы построить полную формальную теорию, рассматривая правила перехода машины от одной конфигурации к другой как правила вывода, процесс вычисления — как вывод, а результаты вычисления — как теоремы.

Две знаменитые теоремы Геделя имеют важное методологическое значение. Из первой теоремы следует, что для достаточно богатых математических теорий не существует адекватных формализаций. Правда, любую неполную теорию T можно расширить, добавив, например, к ней в качестве новой аксиомы истинную, но не выводимую в T формулу. Однако по первой теореме Геделя новая теория T также будет неполна. Вторая теорема может быть истолкована как невозможность исследований метасвойств теории средствами самой формальной теории (опять невозможность самоприменимости!); иначе говоря, метатеория теории T для того, чтобы иметь возможность доказывать хотя бы непротиворечивость теории T , должна быть богаче T . По существу при этом ставится под сомнение первоначальная, «максималистская» программа финитного подхода: нельзя построить математику как некоторую фиксированную совокупность средств, которые можно было бы объявить единственными законными и с их помощью строить метатеории любых теорий.

С другой стороны, не следует истолковывать результаты Геделя (как это иногда делается, в основном среди непрофессионалов) как крах формального подхода. Наличие алгоритмически неразрешимых проблем вовсе не бросает тень на теорию алгоритмов, а лишь сообщает «суровую правду» об устройстве мира, изучаемого этой

теорией. Из этой правды не вытекает, что алгоритмический, конструктивный подход к решению проблемы непригоден; хотя он чего-то и не может, но лишь потому, что этого не может никто. Точно так же невозможность полной формализации содержательно определенных теорий — это не недостаток подхода или концепции, а объективный факт, неустранимый никакой концепцией. Формальный подход остается основным конструктивным средством изучения множеств высказываний. Невозможность адекватной формализации теории означает, что надо либо искать формализуемые ее фрагменты, либо строить какую-то более сильную формальную теорию, которая, правда, снова будет неполна, но, быть может, будет содержать всю исходную теорию. В частности, методами, не формализуемыми в формальной арифметике, Генцен доказал непротиворечивость формальной арифметики.

6.4. АБСТРАКТНЫЕ ФОРМАЛЬНЫЕ СИСТЕМЫ

Дальнейшие примеры формальных систем. Исторически формальные системы создавались с конкретной целью более точного обоснования методов построения математических теорий. Однако постепенно стало ясно, что на основе тех же принципов — исходного набора аксиом, правил вывода и понятия выводимости — можно описывать не только множества выражений, интерпретируемых как высказывания, но и перечислимые множества объектов произвольного вида. Основы теории* таких формальных систем были заложены Э. Постом. Этую теорию можно назвать абстрактной (хотя этот термин и не является общепринятым) или общей, так как она — в отличие от метатеории логических исчислений — не рассматривает свойства формальных систем относительно их конкретных интерпретаций, а изучает лишь их внутренние, синтаксические свойства.

* Начиная с этого момента понятие «теория» употребляется в обычном интуитивном смысле и не имеет прямого отношения к точному понятию «формальной теории», рассмотренному в предыдущих параграфах.

Прежде чем перейти к самой теории, рассмотрим некоторые примеры формальных систем, не связанных с логическими интерпретациями.

Пример 6.6. а. Множество допустимых шахматных позиций можно описать как формальную систему, в которой единственной аксиомой является начальная позиция, правилами вывода — правила игры, а теоремами — позиции, полученные по правилам игры из начальной. Однако эта идея требует уточнения. Пусть дана позиция, скажем, ее диаграмма или описание в шахматной нотации. Для того чтобы однозначно получить все позиции, достижимые из данной за один ход, недостаточно знать правила хода всех фигур, в том числе правила взятия и правило превращения пешки; может понадобиться информация, не содержащаяся явно в позиции, нужно знать: 1) чей ход; 2) ходил ли раньше король (если он стоит на своем начальном поле, позиция на этот вопрос не отвечает) — это нужно для определения допустимости рокировки; 3) не был ли последний ход ходом пешки через два поля — это нужно для определения взятия пешки на проходе. Чтобы превратить множество шахматных позиций в формальную систему, нужно за позицию принять такое ее описание, которое в явном виде содержит ответы на все три вопроса. Для исключения позиций, получающихся после взятия королей, надо ввести правила, запрещающие игнорировать шах, и понятие заключительных позиций (матовых и патовых), после которых никакие ходы не разрешаются.

б. Многие индуктивные определения можно превратить в формальные системы, аксиомы которых перечислены в базисе (первом пункте) определения, а правила вывода — в индуктивном шаге. Необходимым условием перехода к формальной системе является конструктивность задания аксиом и правил вывода, точнее, разрешимость множества аксиом и отношения непосредственной выводимости.

Рассмотрим, например, индуктивное определение абстрактной ориентированной двухполюсной схемы.

1. Пусть зафиксировано конечное число объектов, которые назовем элементами; в каждом элементе выделены два полюса: вход и выход. Элемент является схемой, полюсы которой совпадают с полюсами элемента.

2. Пусть S_1 и S_2 — схемы. Тогда объекты, получающиеся:

- а) путем отождествления выхода S_1 со входом S_2 (правило последовательного соединения);
- б) путем отождествления входа S_1 со входом S_2 и выхода S_1 с выходом S_2 (правило параллельного соединения), также являются схемами. В случае 2а входом схемы является вход S_1 , выходом — выход S_2 . В случае 2б входом является объединенный вход S_1, S_2 , выходом — объединенный выход S_1, S_2 .

Это определение описывает формальную систему, в которой аксиомами являются элементы с выделенными полюсами, а правилами вывода — правила соединения схем. Любой инженер даст этой системе обычную схемную (в инженерном смысле слова, как правило, электротехническую) интерпретацию; однако такая интерпретация вовсе не вытекает из определения. В понятие элемента не вкладывается никакого содержания; единственное, что от него требуется, — возможность эффективно распознавать полюсы. Элементом может быть ориентированное ребро, полюсы которого — вершины; тогда схемы — это просто графы с выделенным входом (источником) и выходом (стоком). (Вопрос к читателю: все ли такие графы порождаются данной формальной системой?) Другая возможная интерпретация: элементы — одномерные функции; последовательное соединение — композиция функций, т. е. их последовательное вычисление, а параллельное соединение — параллельное вычисление двух функций с одинаковыми исходными данными и суммированием результатов в конце.

При описании формальных систем в примере 6.6 была допущена одна вольность, которая, строго говоря, недопустима. Дело в том, что ни в одном из этих описаний не зафиксирован алфавит; соответственно и правила вывода не сформулированы как операции над символами в словарях. Можно, конечно, сказать, что «примерно понятно, как это сделать; детали не уточняются», но способы уточнения могут быть различными и приведут к различным формальным системам. Это обстоятельство известно всем, кто занимался программированием игры в шахматы или машинными преобразованиями схем. Да и вообще лю-

бой, кто программировал содержательно сформулированные алгоритмы, знает, что начинать приходится с выбора *формы представления данных*, т. е. символического кодирования объектов в терминах языка программирования (а в случае автокода — просто машинными словами), причем различные выборы приводят к формальным системам, совершенно различным по своему виду и качествам. Этот выбор — предыстория формальной системы, возникающей лишь тогда, когда он уже сделан.

Общая теория формальных систем не рассматривает все возможные свойства конкретных систем подобно тому, как теория алгоритмов не учит строить конкретные алгоритмы. Одной из ее главных целей является цель, в некотором смысле противоположная: выяснить, каков необходимый минимум средств, с помощью которых можно описать любую формальную систему. Сюда же примыкает вопрос, центральный для любой математической теории: каковы возможности объектов, изучаемых в данной теории? Для теории формальных систем он выглядит так: какие множества могут порождаться формальными системами? Опыт изучения теории алгоритмов говорит о том, что такого рода задачи решаются путем выбора конкретных средств, приводящих к конкретным моделям, общность которых показывается затем путем сравнения их с другими моделями. Средства формальных систем — это аксиомы и правила вывода. В абстрактных формальных системах в отличие от логических исчислений не выделяется понятие формулы («осмысленного выражения»); их объекты — произвольные слова в фиксированном алфавите. Итак, *формальная система FS* определяется: 1) алфавитом A (множество всех слов в алфавите A , как и в примере 1.5, обозначим A^*); 2) разрешимым множеством $A_1 \subseteq A^*$, элементы которого называются аксиомами; 3) конечным множеством вычислимых отношений R_i ($\alpha_1, \dots, \alpha_n, \beta$) на множестве A^* , называемых правилами вывода*; слово β называется выводимым из $\alpha_1, \dots, \alpha_n$ по правилу R_i . Понятия вывода, выводимости и доказательства — те же, что и для формальных теорий (см. § 6.1).

* Аксиомы также можно задать правилом вывода (одноместным отношением): $R_i(\beta) \sim \langle \beta \text{ — аксиома} \rangle$.

Иногда, впрочем, конкретное множество аксиом в системе не фиксируется, а рассматривается выводимость из произвольных разрешимых множеств слов; в этом случае понятия вывода и доказательства не различаются. Конкретные виды формальных систем определяются в основном видом правил вывода. Здесь будут рассмотрены два способа представления формальных систем: системы подстановок и системы продукции Поста. Однако уже общего определения формальной системы оказывается достаточно, чтобы получить простой, но важный факт.

Теорема 6.14. Для любой формальной системы FS множество всех доказуемых в ней слов перечислимо.

Рассмотрим множество A^{**} всех конечных последовательностей $\alpha_1, \dots, \alpha_n$, где α_i — слова в алфавите A . Множество A^{**} , очевидно, перечислимо. Ввиду разрешимости множества аксиом и вычислимости правил вывода по любой последовательности $\alpha_1, \dots, \alpha_n$ можно эффективно узнать, является она выводом в FS или нет. Поэтому если в процессе перечисления A^{**} выбрасывать все последовательности $\alpha_1, \dots, \alpha_n$, не являющиеся выводами, то получим перечисление множества выводов. Следовательно, множество последних слов выводов, т. е. слов, выводимых в FS , перечислимо. \square

Отметим, что описанная процедура перечисляет выводимые слова, вообще говоря, с повторениями, поскольку одно и то же выводимое слово может иметь много выводов (например, любая последовательность, заканчивающаяся аксиомой, есть вывод этой аксиомы).

Верно ли обратное — можно ли для перечислимого множества M построить перечисляющую его формальную систему, т. е. систему, в которой множество выводимых слов совпадает с M ? Из предварительных соображений можно ответить утвердительно: представляется, например, правдоподобным, что машину Тьюринга можно представить в виде формальной системы. Точный ответ на этот вопрос будет дан при рассмотрении конкретных видов формальных систем.

Системы подстановок. Система подстановок, или полусистема Туэ, — это формальная система, определяемая алфавитом A и конечным множеством подстановок

вида $\alpha_i \rightarrow \beta_i$, где α_i, β_i — слова, возможно пустые, в A . Подстановка $\alpha_i \rightarrow \beta_i$ интерпретируется как правило вывода R_i следующим образом: $\gamma \models \delta$ по правилу R_i , если слово δ получается из γ путем подстановки слова β_i вместо какого-нибудь вхождения α_i в слово γ . Выводом β из α в системе подстановок называется цепочка $\alpha \models \varepsilon_1 \models \varepsilon_2 \models \dots \models \beta$, где каждое ε_j получается из ε_{j-1} некоторой подстановкой; как и обычно, наличие выводимости обозначаем $\alpha \vdash \beta$. Такое определение вывода отличается от определения в начале § 6.1; предоставляем читателю убедиться, что для любой формальной системы, где все правила — бинарные и унарные отношения, эти два определения совпадают, т. е. $\alpha \vdash \beta$ в первом смысле, если и только если $\alpha \vdash \beta$ во втором смысле. (Заметим, что правило заключения в логических исчислениях — тернарное отношение!)

Зафиксируем множество аксиом системы и назовем слово δ заключительным, если оно доказуемо в системе и к нему неприменима ни одна из подстановок, т. е. если никакое его подслово не является левой частью какой подстановки. Системе команд Σ_T машины Тьюринга T нетрудно поставить в соответствие систему подстановок S_T над конфигурациями; фактически это было уже сделано при построении универсальной машины Тьюринга (§ 5.2). Если в качестве аксиом выбрать слова $q_1\alpha$, то заключительными словами в системе S_T будут слова $q_z\beta$ (α, β — слова в алфавите ленты машины T). Если же к S_T добавить подстановку $q_z \rightarrow \lambda$, то в полученной системе S'_T множество заключительных слов совпадает с множеством значений функции, вычисляемой машиной T . Это дает для систем подстановок теорему, обратную теореме 6.14.

Теорема 6.15. Для любого перечислимого множества M существует система подстановок, множество заключительных слов которой совпадает с M . \square

Ассоциативное исчисление, или *система Туз*, — это формальная система, определяемая алфавитом A и конечным множеством соотношений $\alpha_i \leftrightarrow \beta_i$ (левая подстановка) и $\beta_i \rightarrow \alpha_i$ (правая подстановка). Таким образом, ассоциативное исчисление всегда есть система подстановок; обратное, вообще говоря, неверно. При наличии

таких парных правил вывода, если $\alpha \vdash \beta$, то и $\beta \vdash \alpha$; ввиду отмеченной ранее транзитивности выводимости получаем, что в ассоциативном исчислении выводимость является отношением эквивалентности и разбивает множество всех слов в A на классы эквивалентности. Заключительные слова в ассоциативных исчислениях возможны, однако они соответствуют классам эквивалентности, состоящим из одного слова, и особого интереса не представляют.

Аналогично тому, как это делалось для систем подстановок, поставим в соответствие каждой машине Тьюринга T (будем считать, что T работает на правой полуленте) ассоциативное исчисление $S(T)$. Опишем это соответствие более подробно. Если A_T — алфавит ленты машины T , то $A_S = A_T \cup \{q_1, \dots, q_z\}$. Системе команд машины T соответствует система соотношений в $S(T)$ (слева — команды, справа — соотношения):

$$q_i a_j \rightarrow q_k a_l R; q_i a_j \leftrightarrow a_l q_k; \quad (6.13)$$

$$q_i a_j \rightarrow q_k a_l L; a_t q_i a_j \leftrightarrow q_k a_t a_l \text{ для всех } a_t \in A_T \quad (6.14)$$

(число соотношений, соответствующих одной команде с движением влево, равно $|A_T|$).

Теорема 6.16. В исчислении $S(T)$ слова $\alpha q_i a_j \beta$ и $\gamma q_z a_k \delta$ эквивалентны, если и только если машина T из конфигурации $\alpha q_i a_j \beta$ за конечное число тактов переходит в конфигурацию $\gamma q_z a_k \delta$.

Одна половина теоремы («если») непосредственно следует из доказательства теоремы 6.15. Вторая половина («только если») может показаться несколько неожиданной: ведь в $S(T)$ допускаются подстановки в обе стороны, а в T — в одну и, следовательно, возможности $S(T)$ выглядят более сильными. Тем не менее покажем, что если существует вывод $\alpha q_i a_j \beta \vdash \gamma q_z a_k \delta$, то машина T из конфигурации $\alpha q_i a_j \beta$ переходит в конфигурацию $\gamma q_z a_k \delta$. Рассмотрим этот вывод, т. е. цепочку $\alpha q_i a_j \beta \vdash \varepsilon_1 \vdash \dots \vdash \gamma q_z a_k \delta$. Каждое слово в этой цепочке получено из предыдущего либо левой, либо правой подстановкой. Кроме того, символ q в каждом слове — единственный. Последнее слово не может быть получено правой подстановкой, так как символ q_z отсутствует в левых частях команд машины T . Пусть ε_l — последнее слово в

цепочке, полученное правой подстановкой: $\varepsilon_l = \alpha_l q_r a_s \beta_l$. Слово ε_l получено из ε_{l-1} правой подстановкой, т. е. применением одного из соотношений (6.13), (6.14), содержащих $q_r a_s$ в левой части. Но таких соотношений столько, сколько команд T с $q_r a_s$ в левой части, т. е. ровно одно; обозначим его E_{rs} . Слово ε_{l+1} получено из ε_l левой подстановкой (по условию для ε_l); но единственная левая подстановка, применимая к ε_l , — это то же соотношение E_{rs} (точнее, его левая половина). Итак, к ε_{l-1} применено E_{rs} слева направо, а к результату этого применения ε_l применено то же E_{rs} справа налево. Так как все подстановки (6.13), (6.14) содержат q , а в любом слове цепочки q единственны, то любая подстановка $\alpha_i \rightarrow \beta_i$ к любому слову ε может быть применена единственным образом, т. е. в ε найдется не более чем одно вхождение α_i . Отсюда $\varepsilon_{l-1} = \varepsilon_{l+1}$; поэтому ε_l и ε_{l+1} из цепочки можно выбросить и построить вывод $\alpha q_i a_j \beta \vdash \dots \vdash \varepsilon_{l-1} \vdash \varepsilon_{l+2} \vdash \dots \gamma q_z a_k \delta$, в котором слов, полученных правой подстановкой, на единицу меньше, чем в прежнем выводе. Применяя к новому выводу те же рассуждения, из него также можно удалить слово, полученное правой подстановкой; в конечном счете будет построен вывод $\alpha q_i a_j \beta \rightarrow \dots \rightarrow \gamma q_z a_k \delta$, содержащий только левые подстановки, т. е. в точности воспроизводящий последовательность конфигураций машины T . \square

Теорема 6.17 (Марков, Пост). Существует ассоциативное исчисление, в котором проблема распознавания эквивалентности слов алгоритмически неразрешима.

Возьмем какую-нибудь универсальную машину Тьюринга U , которая является правильно вычисляющей, т. е. начинает с конфигураций вида $q_1 \alpha$ и заканчивает работу конфигурациями вида $q_z \beta$. Для машины U проблема остановки неразрешима (иначе ввиду универсальности U была бы разрешима общая проблема остановки для машин Тьюринга). Построим ассоциативное исчисление $S(U)$ и присоединим к нему систему соотношений вида $q_z a_i \leftrightarrow q_z$; получим новое исчисление $S'(U)$. В этом исчислении, так же как и в $S(U)$, можно имитировать вычислительный процесс U , однако благодаря новым соотношениям все заключительные конфигурации U в $S'(U)$ эквивалентны q_z . Поэтому теорема 6.16 для $S'(U)$ имеет следующий вид:

в $S(U)$ слова $q_1\alpha$ и q_z эквивалентны, если и только если U , начав с $q_1\alpha$, остановится. Ввиду неразрешимости проблемы остановки для U проблема эквивалентности $q_1\alpha$ и q_z также неразрешима. \square

Ассоциативные исчисления имеют важную алгебраическую интерпретацию. В гл. 2 говорилось о том, что всякую полугруппу можно получить из свободной полугруппы (т. е. просто из множества A^* всех слов в алфавите A) введением определяющих соотношений, т. е. равенств $\alpha_i = \beta_i$. Замена подслова α_i в слове α равным ему подсловом β_i , т. е. переход от слова $\alpha'\alpha_i\alpha''$ к слову $\alpha'\beta_i\alpha''$, называется эквивалентным преобразованием слова α .

Слова считаются равными (или эквивалентными), если они могут быть получены друг из друга цепочкой эквивалентных преобразований.

Содержательно эквивалентность слов означает, что они задают один и тот же элемент полугруппы; иначе говоря, элементами полугруппы, заданной определяющими соотношениями, являются классы эквивалентности слов, порождаемые этими соотношениями. Формально такое описание полугруппы — это просто ассоциативное исчисление с соотношениями $\alpha_i \leftrightarrow \beta_i$; эквивалентные преобразования в полугруппе — это выводы в исчислении. В гл. 2 была сформулирована *проблема эквивалентности слов в полугруппе*. Теперь становится ясным, что ответ на нее — это просто переформулировка теоремы 6.17.

Теорема 6.17'. Существует полугруппа, заданная определяющими соотношениями, в которой проблема распознавания эквивалентности (равенства) слов алгоритмически неразрешима.

Г. С. Цейтин нашел очень простое ассоциативное исчисление с неразрешимой проблемой равенства — с алфавитом из пяти букв $\{a, b, c, d, e\}$ и семью соотношениями:

$$\begin{aligned} ac &\leftrightarrow ca; ad \leftrightarrow da; bc \leftrightarrow cb; bd \leftrightarrow db; \\ abac &\leftrightarrow abace; eca \leftrightarrow ae; edb \leftrightarrow be. \end{aligned}$$

Теорема 6.17' явилась исторически первым примером доказательства алгоритмической неразрешимости проблемы, не связанной с логикой и основаниями математики.

Системы продукции Поста (канонические системы). Каноническая система определяется собственным алфавитом $A = \{a_1, \dots, a_m\}$ (алфавитом констант), алфавитом $X = \{x_1, \dots, x_n\}$ переменных, конечным множеством аксиом и конечным множеством правил вывода, имеющих

вид $\gamma_1, \dots, \gamma_k \Rightarrow \delta$ и называемых *продукциями* ($\gamma_1, \dots, \gamma_k$, как обычно, называются посылками, δ — следствием). Аксиомы, а также посылки и следствия продукции — это слова в алфавите $A \cup X$; иначе говоря, они содержат, кроме собственных букв системы, еще и переменные. В дальнейшем слова в алфавите $A \cup X$ будем называть *термами*, а слова в A — просто словами. Переменные канонической системы аналогичны метапеременным в логических исчислениях; аксиомы канонической системы — это по существу схемы аксиом логических исчислений. Говоря о подстановке в термы слов вместо переменных, мы всегда будем иметь в виду, что сохраняется обычное ограничение: вместо всех вхождений одной и той же переменной подставляется одно и то же слово (быть может, пустое).

Слово α называется применением аксиомы ω , если α получается подстановкой в ω слов вместо переменных. Слово α непосредственно выводимо из слова $\alpha_1, \dots, \alpha_n$ применением продукции P с n посылками, если найдется такая подстановка слов вместо переменных P , которая посылки P превратит в слова $\alpha_1, \dots, \alpha_n$, а заключение P — в слово α . Например, слово bb непосредственно выводимо из слов $acab$, $cabb$ применением продукции ax_1b , $x_1bx_2 \Rightarrow bx_2$ при подстановке $x_1 = ca$, $x_2 = b$. Последовательность слов называется доказательством в канонической системе, если каждое слово этой последовательности — либо применение аксиомы, либо непосредственно выводимо из предыдущих слов последовательности применением некоторой продукции системы. Слово называется доказуемым (или теоремой), если оно является последним словом некоторого доказательства.

Пример 6.7. а. Множество всех нечетных чисел в унарном представлении — это множество всех теорем канонической системы с алфавитами $\{\}\}, \{x\}$, аксиомой $|$ и продукцией $x \rightarrow x\|$. Если эту продукцию заменить продукцией $x \rightarrow xx$, то получим каноническую систему, порождающую степени двойки (в унарном представлении: $|, \|, \||, \|\|, \dots$

б. Множество всех формул исчисления высказываний порождается канонической системой с алфавитами

$\{a_1, \dots, a_n, \vee, \&, \neg, \rightarrow, (\cdot)\}, \{x_1, x_2\}$, аксиомами a_1, \dots, a_n и продукциями

$$\left. \begin{array}{l} x_1, x_2 \Rightarrow (x_1 \vee x_2); \\ x_1, x_2 \Rightarrow (x_1 \& x_2); \\ x_1, x_2 \Rightarrow (x_1 \rightarrow x_2); \\ x_1 \Rightarrow \neg x_1. \end{array} \right\} \quad (6.15)$$

Отметим, что здесь алфавит пропозициональных букв конечен. Для построения исчисления с бесконечным множеством пропозициональных букв сначала нужно построить формальную систему, порождающую это множество из конечного алфавита символов. Именно с этой целью в языках программирования вводится индуктивное определение идентификатора; оно представляет собой формальную систему, порождающую бесконечное множество переменных языка из конечного алфавита букв и цифр.

При построении машин Тьюринга часто оказывалось удобным (а иногда необходимым) вводить вспомогательные символы, которые участвуют в процессе вычисления, но не присутствуют ни в исходных данных, ни в результате. Аналогичные средства вводятся и в формальных системах.

Пусть дана каноническая система S с собственным алфавитом A' , в котором выделен подалфавит A . Если нас интересуют только те теоремы S , которые являются словами в A , то будем говорить, что система S является *канонической системой над A* , A назовем основным алфавитом, A' — расширением A , а символы из $A' \setminus A$ — вспомогательными.

Пример 6.8. а. Последовательность чисел $0, 1, 1, 2, 3, 5, 8\dots$, в которой каждый член, начиная с третьего, равен сумме двух предыдущих, называется последовательностью Фибоначчи, а ее элементы — числами Фибоначчи. Числа Фибоначчи в унарном представлении порождаются канонической системой над $\{\}$ со вспомогательным символом $*$, аксиомой $*1$ и двумя продукциями:

$$x_1 * x_2 \Rightarrow x_2 * x_1 x_2; \quad x_1 * x_2 \Rightarrow x_1.$$

В этой системе символ $*$ служит маркером, разделяющим два числа, аксиома задает первые два числа последовательности.

довательности (0 изображен пустым словом слева от маркера), первая продукция из n -го и $(n + 1)$ -го члена получает $(n + 1)$ -й и $(n + 2)$ -й члены последовательности, вторая продукция из пары чисел выделяет первое; только применение второй продукции дает слова в алфавите $\{\}$.

б. Реализуем изложенный в примере 6.7, б план построения формул исчисления высказываний с бесконечным множеством пропозициональных букв. Эти «буквы» будут обозначаться символом a с числовым индексом, т. е. будут представлять собой слова вида a , $a01$, $a523$ и т. д., играющие роль идентификаторов (имен переменных). Система содержит основной алфавит $A = \{a, 0, 1, 2, \dots, 9, \vee, \&, \neg, \rightarrow, (,)\}$, вспомогательные символы $\{i, f\}$, аксиому ai и 16 продуктов:

$$\begin{aligned} x_1i &\Rightarrow x_10i; \\ x_1i &\Rightarrow x_11i; \\ &\dots \\ x_1i &\Rightarrow x_19i; \\ x_1i &\Rightarrow x_1f; \\ x_1f, x_2f &\Rightarrow (x_1 \vee x_2)f; \\ x_1f, x_2f &\Rightarrow (x_1 \& x_2)f; \\ x_1f, x_2f &\Rightarrow (x_1 \rightarrow x_2)f; \\ x_1f &\Rightarrow \neg x_1f; \\ x_1f &\Rightarrow x_1. \end{aligned}$$

Первые десять продуктов порождают идентификаторы; следующие пять продуктов формализуют индуктивное определение формулы (из этих пяти четыре последние отличаются от продуктов (6.15) только символом f), последняя продукция служит для удаления символа f . Вспомогательные символы играют здесь несколько непривычную роль — они по существу являются признаками (или метками) определенного класса слов: i — метка класса идентификаторов, f — метка класса формул. Благодаря этому формальный вид продукции очень близок к тексту соответствующей части индуктивного определения: 11-я продукция означает «всякий идентификатор есть формула», а 12-я: «если x_1 и x_2 — формулы, то $(x_1 \vee x_2)$ — формула». Прием использования в формальных системах специальных символов как

признаков классов широко используется в формальных системах Смальяна [30].

Связь канонических систем с системами подстановок довольно ясна, по крайней мере в одну сторону; очевидно, что всякой подстановке $\alpha_i \rightarrow \beta_i$ соответствует продукция $x_1\alpha_i x_2 \Rightarrow x_1\beta_i x_2$, поэтому для всякой системы подстановок легко построить эквивалентную ей каноническую систему. Соответствие в обратную сторону менее очевидно; однако о том, что оно существует, можно заключить, сопоставив теорему 6.14 (из которой следует, что множество теорем канонической системы перечислим) с теоремой 6.15 (любое перечислимое множество можно представить как множество заключительных слов в некоторой системе подстановок). Правда, понятие заключительного слова может показаться несколько искусственным и слишком уже приближающим формальные системы к машинам Тьюринга; если рассматривать формальную систему как генератор элементов перечислимого множества, более естественным было бы описание перечислимого множества как множества теорем некоторой формальной системы. Это нетрудно сделать, используя введенное ранее понятие канонической системы с расширенным алфавитом.

Теорема 6.18. Для любого перечислимого множества M слов в алфавите A существует каноническая система над A , множество теорем которой совпадает с M .

Пусть M перечисляется машиной Тьюринга T с алфавитом ленты $A_T = \{a_1, \dots, a_n\}$, первые m букв которого образуют алфавит исходных данных ($m \leq n$), множеством состояний $Q = \{q_1, \dots, q_z\}$ и системой команд Σ_T . Определим каноническую систему S над A следующим образом: $A = A_T$, $A' = A_T \cup Q_T$; аксиома системы *; командам машины T поставим в соответствие продукцию аналогично тому, как это делалось при доказательстве теорем 6.15 и 6.16: например, команде $q_i a_j \rightarrow q_k a_l R$ соответствует продукция $x_1 q_i a_j x_2 \Rightarrow x_1 a_l q_k x_2$ [ср. (6.13)]. Кроме того, введем следующие $m + 2$ продукций:

$$\begin{aligned} x_1 * &\Rightarrow x_1 a_1 *; \\ &\dots \\ x_1 * &\Rightarrow x_1 a_m *; \\ x_1 * &\Rightarrow q_1 x_1; \\ q_z x_1 &\Rightarrow x_1. \end{aligned}$$

Первые m продукции порождают из аксиомы все множество слов в алфавите исходных данных с маркером в конце: ($m + 1$)-я продукция (стартовая) порождает из любого исходного слова начальную конфигурацию; после этого работают продукции, соответствующие командам машины T ; наконец, в заключительных конфигурациях (и только в них!) символ состояния удаляется и получаются искомые теоремы в алфавите A . \square

Итак, в терминах канонических систем можно описать любые перечислимые множества. Правда, используемое определение канонической системы (в частности, вид ее продукции) является слишком общим: применение продукции трудно назвать элементарным шагом (хотя свойство применимости продукции, как нетрудно видеть, разрешимо). Поэтому возникает задача нормализации канонических систем, т. е. приданье им более простого, «нормального» вида.

Каноническая система называется *нормальной*, если она имеет одну аксиому, а все ее продукции имеют вид $\alpha x \rightarrow x\beta$. Применение такой продукции к слову (если оно возможно) просто и стандартно: у слова вычеркивается начальный отрезок α и к концу приписывается β .

Теорема 6.19 (теорема Поста о нормальной форме). Для любой канонической системы CS с алфавитом A существует нормальная каноническая система NS над A , эквивалентная CS (т. е. множество теорем NS над A и множество теорем CS совпадают).

Прямое доказательство этой теоремы, содержащее метод построения NC по CS , довольно сложно. Его можно найти, например, в [22], где оно занимает целую главу. Ограничимся косвенным доказательством существования NS , а именно: покажем, что для любой системы подстановок S в алфавите A существует нормальная каноническая система NS над A , эквивалентная S .

Пусть $A = \{a_1, \dots, a_n\}$ и S содержит k подстановок $\alpha_i \rightarrow \beta_i$. Определим NS как систему с расширенным алфавитом $A' = \{a_1, \dots, a_n, a'_1, \dots, a'_n\}$ и $k + 2n$ продукциями:

$$\alpha_i x \Rightarrow x\beta'_i \quad (i = 1, \dots, k); \quad (6.16)$$

$$a_j(x) \Rightarrow x a'_j; \quad (6.17)$$

$$a'_j(x) \Rightarrow x a_j, \quad (6.18)$$

где β'_i здесь и в дальнейшем обозначает слово β_i , в котором все буквы из A заменены соответствующими буквами со штрихом.

Пусть к слову γ в S применима подстановка $\alpha_i \rightarrow \beta_i : \gamma = \gamma_1\alpha_i\gamma_2$ и в S $\gamma_1\alpha_i\gamma_2 \vdash \gamma_1\beta_i\gamma_2$. Но тогда в NS $\gamma_1\alpha_i\gamma_2 \vdash \alpha_i\gamma_2\gamma'_1 \vdash \gamma_2\gamma'_1\beta'_i \vdash \gamma'_1\beta'_i\gamma_2 \vdash \gamma_1\beta_i\gamma_2$. Так как любой вывод $\gamma \vdash \delta$ в S есть цепочка подстановок, то по выводу $\gamma \vdash \delta$ в S можно построить вывод $\gamma \vdash \delta$ в NS .

Пусть теперь в NS имеется вывод $\gamma \vdash \delta$, где γ, δ — слова в A . Разобьем этот вывод на отрезки $\gamma \vdash \varepsilon_{i_1} \vdash \varepsilon_{i_2} \vdash \dots \vdash \delta$, такие, что ε_{i_j} и $\varepsilon_{i_{j+1}}$ (для всех j) — слова в A , а все слова между ними содержат вспомогательные буквы, и рассмотрим для определенности первый из них $\gamma \vdash \varepsilon_{i_1}$. Если γ — слово в A , то продукциями (6.17), (6.18) из него можно получить лишь слова вида $\gamma_2\gamma'_1, \gamma'_1\gamma_1, \gamma'$, где слова γ_1, γ_2 таковы, что $\gamma_1\gamma_2 = \gamma$. Так как ε_{i_1} — слово в A , то либо $\varepsilon_{i_1} = \gamma$ (но тогда этот отрезок из вывода можно удалить), либо на отрезке $\gamma \vdash \varepsilon_{i_1}$ была применена по крайней мере одна из продуктов (6.16). Для любого вывода, содержащего между словами из A несколько применений продуктов (6.16), в NS можно построить эквивалентный вывод, в котором между этими применениями вставлены слова из A . Строгое доказательство этого утверждения опускаем; приведем лишь пример: выводу $\alpha_1\alpha_2 \vdash \alpha_2\beta'_1 \vdash \beta'_1\beta'_2 \vdash \beta'_2\beta_1 \vdash \beta_1\beta_2$, в котором применены продукции $\alpha_1x \Rightarrow x\beta_1$ и $\alpha_2x \Rightarrow x\beta_2$, соответствует вывод $\alpha_1\alpha_2 \vdash \alpha_2\beta'_1 \vdash \beta'_1\alpha'_2 \vdash \beta_1\alpha_2 \vdash \alpha_2\beta'_1 \vdash \beta'_1\beta'_2 \vdash \beta_1\beta_2$, в котором они разделены словом $\beta_1\alpha_2$. Итак, можно считать, что в выводе $\gamma \vdash \varepsilon_{i_1}$ продукция (6.16) — пусть это будет $\alpha x \Rightarrow x\beta'$ — применена ровно один раз. Но тогда $\gamma = \gamma_1\alpha_2\gamma_2$, применению продукции (6.16) предшествовало несколько (быть может, ни одного) применений (6.17) и место вывода, где применена продукция (6.16), имеет вид $\alpha\gamma_2\gamma'_1 \Rightarrow \gamma_2\gamma'_1\beta'$. Остаток $\gamma_2\gamma'_1\beta'_1 \vdash \varepsilon_{i_1}$ рассматриваемого отрезка не содержит применений (6.16), поэтому в силу свойств продуктов (6.17), (6.18) $\varepsilon_{i_1} = \gamma_1\beta'_2\gamma_2$. Но тогда ε_{i_1} получается из γ подстановкой $\alpha \rightarrow \beta$, т. е. $\gamma \vdash \varepsilon_{i_1}$ в S . Индукцией по числу слов ε_{i_j} доказываем, что $\gamma \vdash \delta$ в S . Отсюда следует эквивалентность S и NS , после чего о справедливости теоремы нетрудно заключить из сопоставления теорем 6.15 и 6.18.

В заключение главы приведем без доказательства один результат об алгоритмической неразрешимости, который доказан Постом с помощью канонических систем и который часто используется для доказательства других неразрешимостей (особенно в теории формальных грамматик).

Пусть дано конечное множество $(\alpha_1, \beta_1), \dots, (\alpha_m, \beta_m)$ пар слов в алфавите A . Поставим следующую проблему: существует ли последовательность i_1, i_2, \dots, i_N индексов, такая, что $\alpha_{i_1}\alpha_{i_2}\dots\alpha_{i_N} = \beta_{i_1}\beta_{i_2}\dots\beta_{i_N}$?

Эта проблема называется комбинаторной проблемой, или *проблемой соответствия Поста*. Имеются два варианта ее формулировки: общая комбинаторная проблема Поста (для произвольного множества пар) и ограниченная комбинаторная проблема (для множества пар с фиксированной мощностью m).

Теорема 6.20. Ограниченная комбинаторная проблема Поста для достаточно больших m алгоритмически неразрешима.

Отсюда следует неразрешимость и общей комбинаторной проблемы.

Формальные системы и алгоритмы. Итак, формальные системы оказываются столь же мощным средством для задания конструктивных объектов, что и алгоритмы. С их помощью можно имитировать поведение машин Тьюринга, т. е. строить формальные системы, в некотором смысле аналогичные алгоритмам. С другой стороны, понятие перечислимого множества в терминах формальных систем (опирающееся на теорему 6.18) выглядит более компактным, чем в терминах, скажем, машины Тьюринга. Поэтому возможны две концепции построения системы основных понятий, формализующих идеи эффективности и конструктивности. Концепция, описанная ранее и являющаяся исторически первой, кладет в основу понятие алгоритма. Вторая концепция, созданная Э. Постом*, опирается на понятия формальной (конкретнее, канонической) системы и перечислимого множества, которое определяется просто как множество теорем формальной системы. Нормальную каноническую систему над алфавитом A можно представить как граф с одной выделенной вершиной — аксиомой, остальные вершины которого помечены словами в A — теоремами, ребра — это применения продукции, а пути из выделенной вершины в данную — возможные выводы данного слова. Множество слов в A , порождаемое системой, — это множество всех вершин графа, помеченных словами в A . Алгоритм — это формальная система особого, детерминированного вида, характеризующаяся тем, что в ней

* Сжатое и блестяще написанное изложение этой концепции имеется в книге Мартин-Лефа [19].

к каждой теореме применима не более чем одна продукция. Соответствующий граф представляет собой цепочку, изображающую вычислительный процесс; аксиома в таком графе — это просто исходные данные алгоритма.

Другой способ детерминизации формальных систем — это фиксация порядка применения правил вывода. Например, нормальный алгоритм Маркова [48] — это упорядоченная система подстановок с двумя дополнительными соглашениями: 1) i -я подстановка может быть применена, только если неприменимы 1, ..., $(i - 1)$ -я подстановки; 2) если подстановка $\alpha \rightarrow \beta$ применима к слову γ , то β подставляется вместо самого левого вхождения α в γ . Основной акцент «алгоритмической» концепции — в ее детерминизме. Поэтому она естественна и удобна при описании вычислительных процессов и устройств. Основной акцент «формально-системной» концепции — в компактности конструктивного описания множеств. Примеры такого описания — формальные теории (§ 6.1 и 6.2). Другая группа примеров, крайне важных в прикладном отношении, — это алгоритмические языки программирования. Методы описания языков и построения компиляторов для них опираются на *теорию формальных грамматик*, представляющих собой еще один вид абстрактных формальных систем. Основные понятия этой теории изложены в следующей главе.

ЯЗЫКИ И ГРАММАТИКИ

До начала XX в., говоря о языках, имели в виду только естественные языки (русский, английский, латинский и т. д.), являющиеся или являвшиеся в прошлом средством общения между людьми в их обычной повседневной жизни. Наука о языках — лингвистика — сводилась в основном к изучению конкретных естественных языков, их классификации, выяснению сходств и различий между ними.

Возникновение и развитие метаматематики (см. гл. 5 и 6), изучающей по существу язык математики, логико-философские исследования языка науки, предпринятые Л. Виттгенштейном, Р. Карнапом и другими в 20–30-е гг. XX в., исследования средств коммуникации у животных, идеи структуралистского подхода к лингвистике привели в 30-х гг. к существенно более широкому представлению о языке, при котором под языком понимается всякое средство общения, состоящее из 1) знаковой системы, т. е. множества допустимых последовательностей знаков; 2) множества смыслов этой системы; 3) соответствия между последовательностями знаков и смыслами, делающего «осмыслившими» допустимые последовательности знаков.

Знаками могут быть буквы алфавита, математические обозначения, звуки, движения брачного танца у животных, ритуальные действия в обрядах различных народов. Наука об осмысливших знаковых системах называется семиотикой. Семиотический подход оказывается весьма плодотворным в различных областях знания —

в биологии, социологии, этнографии, лингвистике; при этом разные ветви семиотики имеют значительную специфику и не везде еще используют точные математические средства. Наиболее продвинутыми являются исследования знаковых систем, в которых знаками являются символы алфавитов, а последовательностями знаков — тексты; к таким знаковым системам относятся естественные языки, языки науки, а также языки программирования. Именно интерес к языкам программирования, совпавший с новыми подходами в структурной лингвистике и необходимостью решать задачу машинного перевода естественных языков, привел в 50-х гг. к возникновению новой науки — математической лингвистики, которая рассматривает языки как произвольные множества осмысленных текстов.

Правила, определяющие множества текстов, образуют синтаксис языка; описание множества смыслов и соответствия между текстами и смыслами — семантику языка. Семантика языка зависит от характера объектов, которые описываются языком, и средства ее изучения для различных типов языков различны. О семантике языка математики — формальных теорий — речь шла выше, в § 6.3; исследование семантики языков программирования стало самостоятельной отраслью теоретического программирования; попытки точного описания семантики естественных языков связаны прежде всего с работами по машинному переводу. Что же касается синтаксиса, то его особенности гораздо меньше зависят от назначения и целей языка; оказывается возможным сформулировать понятия и методы исследования синтаксиса языков, не зависящие от содержания и назначения языков. Кроме того, как уже отмечалось при обсуждении теоремы Райса (см. гл. 5), синтаксические свойства языков проще изучать и распознавать, чем семантические (хотя и при изучении синтаксиса, как будет видно далее, возникают алгоритмически неразрешимые проблемы). Поэтому наибольших успехов математическая лингвистика достигла в изучении синтаксиса, где за последние 30 лет сложился специальный математический аппарат — теория формальных грамматик, очень содержательный и интересный в тео-

ретическом отношении и эффективный в приложениях (языки программирования, искусственный интеллект, машинный перевод). Именно этот аппарат и будет предметом настоящей главы.

С точки зрения синтаксиса язык понимается уже не как средство общения, а как множество формальных (в смысле теории формальных систем — см. начало гл. 6 и § 6.4) объектов — последовательностей символов алфавита. Такие последовательности выше назывались текстами; в теории алгоритмов и формальных систем их называют словами (см. пример 1.5, а также § 5.1, 5.2 и 6.4). В лингвистике естественных языков термины «текст» и «слово» имеют разный смысл; поэтому в математической лингвистике последовательность символов обычно называют нейтральным термином «цепочка» (*string*), а язык, понимаемый как множество формальных цепочек, — *формальным языком*. До конца этой главы, говоря о языках, будем иметь в виду формальные языки, если не оговорено противное.

7.1. ФОРМАЛЬНЫЕ ГРАММАТИКИ И ИХ СВОЙСТВА

Пусть задан алфавит V и тем самым множество V^* всех конечных слов, или цепочек, в алфавите V . Формальный язык L в алфавите V — это произвольное подмножество $L \subseteq V^*$. Конструктивное описание формальных языков осуществляется с помощью формальных систем специального вида, называемых формальными порождающими грамматиками.

Формальная порождающая грамматика G (в дальнейшем — просто грамматика G) — это формальная система, определяемая четверкой объектов $G = \langle V, W, I, P \rangle$, где V — алфавит (или словарь) терминальных (основных) символов; W — алфавит нетерминальных (вспомогательных) символов, $V \cap W = \emptyset$; I — начальный символ (аксиома) грамматики; P — конечное множество правил вида $\xi \rightarrow \eta$, где ξ и η — цепочки в алфавите $V \cup W$. Цепочка β непосредственно выводима из цепочки α в грамматике G (обозначение $\alpha \Rightarrow_G \beta$; индекс G опускается, если понятно, о какой грамматике идет речь), если $\alpha = \gamma\xi\delta$, $\beta = \gamma\eta\delta$ и $\xi \rightarrow \eta$ —

правило G . Цепочка β выводима из α , если существует последовательность $\varepsilon_0 = \alpha, \varepsilon_1, \varepsilon_2, \dots, \varepsilon_n = \beta$, такая, что для всех $i = 0, \dots, n - 1$ $\varepsilon_i \Rightarrow \varepsilon_{i+1}$. Эта последовательность называется *выводом* β из α , а n (число ее элементов, отличных от α) — длиной вывода. Выводимость β из α обозначается $\alpha \Rightarrow^n \beta$ (если нужно указать длину вывода) или $\alpha \Rightarrow^* \beta$ (если длина вывода не указывается). Языком $L(G)$, порождаемым грамматикой G , называется множество всех цепочек в терминальном алфавите V , выводимых из I . Грамматики G и G' *эквивалентны*, если $L(G) = L(G')$.

В теории грамматик сложились свои традиции обозначений, которых мы будем придерживаться. Символы терминального алфавита принято обозначать малыми латинскими буквами, символы нетерминального алфавита — большими латинскими буквами, цепочки в алфавите $V \cup W$ — греческими буквами. Длина цепочки α обозначается $l(\alpha)$ или $|\alpha|$. Множество всех цепочек в алфавите V обозначается, как и прежде, V^* ; множество всех непустых цепочек обозначается V^+ . Иногда принятые здесь обозначения расходятся с традициями теории формальных систем — например, приведенные выше обозначения для выводимости (вместо использованного в § 6.1–6.4 знака \vdash).

Понятие формальной грамматики практически совпадает с введенным в § 6.4 понятием системы подстановок (полусистемы Туэ). Правда, множество, порожденное подсистемой Туэ, — это множество ее заключительных слов, а при отсутствии ограничений на правила вывода в грамматике G цепочки языка $L(G)$ могут не быть заключительными. Однако это обстоятельство не является существенным ввиду следующей леммы.

Лемма 7.1. Для произвольной грамматики G существует эквивалентная ей грамматика G_1 , левые части правил которой не содержат вхождений основных символов.

Пусть $G = \langle V, W, R, I \rangle$. Каждому символу $a \in V$ поставим в соответствие двойник — символ A , не содержащийся в $V \cup W$; множество всех двойников обозначим V' . Определим теперь $G_1 = \langle V_1, W_1, R_1, I_1 \rangle$ следующим образом: $V_1 = V$, $I_1 = I$, $W_1 = W \cup V'$, а $R_1 = R' \cup R''$, где R' — множество всех правил вида $A \rightarrow a$ ($a \in V$, A — двойник a),

а R'' получено из R заменой в каждом правиле всех вхождений терминальных символов вхождениями их двойников. Каждому выводу $I, \varepsilon_1, \dots, \varepsilon_n$ в G соответствует вывод $I, \varepsilon'_1, \dots, \varepsilon'_n, \varepsilon'_{n+1}, \dots, \varepsilon'_{n+m}$ в G' , где ε'_i ($i \leq n$) получено из ε_i заменой всех символов из V их двойниками, а ε'_{n+j} ($j \leq m$) получено из ε_{n+j-1} применением правила из R' , причем к ε'_{n+m} правила из R' уже неприменимы. Ясно, что $\varepsilon'_{n+m} = \varepsilon_n$, поэтому $L(G) \subseteq L(G_1)$. Так как только правила из R' содержат терминальные символы в правых частях, то любой вывод из G_1 цепочки длины m должен содержать m применений правил из R' . Удалив из вывода применения этих правил и приведя в нем обратное переименование двойников в символы V , получим вывод этой же цепочки в G . Отсюда $L(G_1) \subseteq L(G)$, и, следовательно, $L(G) = L(G_1)$. \square

Прием введения двойников, только что продемонстрированный при исследовании грамматик, является весьма распространенным. Сама же лемма позволяет утверждать, что для любого языка L , порожденного некоторой формальной грамматикой, существует порождающая его грамматика, для которой L — множество ее заключительных слов в смысле § 6.4. Используя это утверждение и теорему 6.15, нетрудно доказать следующую теорему.

Теорема 7.1. Для любого перечислимого множества M существует грамматика G , такая, что $M = L(G)$. \square

Отметим, что эта теорема не является непосредственным следствием теоремы 6.15, поскольку не всякая система подстановок является грамматикой; преодоление этой небольшой технической трудности предоставляем читателю.

Итак, формальные грамматики являются частным случаем полусистем Туэ, и при этом они способны порождать любые перечислимые множества. Поэтому, рассматривая грамматики общего вида, мы не выходим за пределы общей теории формальных систем (см. § 6.4). Специфика формально-лингвистического подхода к описанию множеств цепочек начинает проявляться при рассмотрении более узких классов грамматик. Общепринятой классификацией грамматик и порождаемых ими языков является иерархия Хомского, содержащая четыре типа грамматик.

Грамматика типа 0 — это грамматика произвольного вида, без ограничений на правила вывода.

Грамматика типа 1, или контекстная — это грамматика, все правила которой имеют вид $\alpha A\beta \rightarrow \alpha\omega\beta$, где $\omega \in (V \cup W)^+$. Цепочки α и β — это контекст правила. Они не изменяются при его применении.

Грамматика типа 2, или контекстно-свободная (КС-грамматика), все правила которой имеют вид $A \rightarrow \alpha$, где $\alpha \in (V \cup W)^*$.

Грамматика типа 3, или регулярная — грамматика, все правила которой имеют вид либо $A \rightarrow aB$, либо $A \rightarrow a$.

Язык L называется языком типа i ($i = 0, 1, 2, 3$), если существует порождающая его грамматика типа i .

Пример 7.1. а. Множество нечетных чисел в унарном представлении (пример 6.7, а) — это множество цепочек вида $a, aaa, aaaaa \dots$, т. е. язык $\{a^{2n-1}\}$. Он порождается грамматикой $G = \langle V, W, R, I \rangle$, где $V = \{a\}$, $W = \{I\}$, а R содержит правила:

$$I \rightarrow a; I \rightarrow aaI.$$

Если второе правило заменить парой правил $I \rightarrow aB$, $B \rightarrow aI$, то получим грамматику типа 3, эквивалентную исходной.

В последующих примерах грамматик алфавиты V и W не будут указываться в тех случаях, когда принятые нами обозначения терминальных и нетерминальных символов позволяют однозначно извлечь эти алфавиты из правил.

б. Язык $\{a^n b^n\}$ является КС-языком (языком типа 2), поскольку он порождается КС-грамматикой с двумя правилами вывода:

$$I \rightarrow aIb \text{ и } I \rightarrow ab.$$

в. Язык булевых формул с переменными a, b, c порождается КС-грамматикой, где $V = \{a, b, c, \vee, \&, \neg, (\,)\}$; $W = \{I\}$; R содержит правила:

$$\begin{aligned} I &\rightarrow (I \vee I); \\ I &\rightarrow (I \& I); \\ I &\rightarrow \neg I; \\ I &\rightarrow a; \\ I &\rightarrow b; \\ I &\rightarrow c. \end{aligned}$$

Этот язык отличается от языка формул исчисления высказываний (пример 6.7, б) отсутствием импликации.

г. Если в предыдущем языке последние три правила, вводящие операции \vee , $\&$, \neg , заменить четырьмя правилами, вводящими операции $+$, $-$, $*$, $/$, то получим язык арифметических выражений. Этот язык отличается от обычного языка арифметических выражений тем, что не учитывает ассоциативности сложения и умножения, а также приоритетов операций, и поэтому его выражения содержат слишком много скобок (аналогичное замечание относится и к языку из примера «в»). Более близкий к обычному язык арифметических выражений с переменными a , b , c задается болееской КС-грамматикой:

$$\begin{aligned} I &\rightarrow T; \\ I &\rightarrow I + T; \\ I &\rightarrow I - T; \\ T &\rightarrow M; \\ T &\rightarrow T * M; \\ T &\rightarrow T / M; \\ M &\rightarrow (I); \\ M &\rightarrow K; \\ K &\rightarrow a; \\ K &\rightarrow b; \\ K &\rightarrow c. \end{aligned}$$

Для полного совпадения с обычным языком арифметических выражений в эту грамматику надо добавить правила, порождающие числовые константы и произвольные идентификаторы переменных, что мы предоставляем читателю.

д. Язык $a^n b^n c^n$ порождается следующей грамматикой:

$$\begin{aligned} I &\rightarrow aBa; \\ B &\rightarrow aBCa; \\ B &\rightarrow b; \\ bC &\rightarrow BB; \\ aC &\rightarrow Ca. \end{aligned}$$

Эта грамматика не является контекстной (из-за последних двух правил), однако можно убедиться, что ей эквивалентна следующая контекстная грамматика:

$$\begin{aligned}
I &\rightarrow ABA; \\
B &\rightarrow ABCA; \\
B &\rightarrow b; \\
bC &\rightarrow bb; \\
AC &\rightarrow DC; \\
DC &\rightarrow DA; \\
DA &\rightarrow CA; \\
A &\rightarrow a,
\end{aligned}$$

в которой неконтекстное правило $aC \rightarrow Ca$ заменено четырьмя контекстными правилами, а A играет роль двойника a (см. доказательство леммы 7.1). Поэтому язык $\{a^n b^n a^n\}$ имеет тип 1.

Связь языка с порождающей его грамматикой имеет ту же природу, что и связь функции с вычисляющим ее алгоритмом, о которой говорилось в комментарии к теореме Райса (см. гл. 5). Поэтому проблемы распознавания свойств языка по свойствам задающей его грамматики часто оказываются алгоритмически неразрешимыми. В частности, как будет показано ниже, неразрешима проблема распознавания эквивалентности двух грамматик (разрешим лишь ее частный случай, когда обе грамматики имеют тип 3 — о нем будет сказано в гл. 8); неразрешима также проблема: для языка типа i определить, является ли он языком типа j для $j > i$. Как обычно, неразрешимость алгоритмической проблемы в целом не исключает разрешимости ее для конкретных случаев. В частности, для всех $i = 0, 1, 2$ имеются примеры языков типа i , для которых доказано, что они не являются языками типа $i + 1$. Эти доказательства опираются, как правило, на некоторые необходимые (но недостаточные!) условия принадлежности языка к определенному типу. Некоторые из этих условий будут приведены ниже.

Грамматики, в которых все правила обладают тем свойством, что их левые части не короче правых, называются *неукорачивающими*, поскольку в любом выводе такой грамматики длины цепочек могут только возрастать.

Теорема 7.2. Если G — неукорачивающая грамматика, то язык $L(G)$ разрешим.

Пусть $\alpha \in L(G)$ и $|\alpha| = n$. Если вывод α имеет вид $I, \dots, \beta, \gamma, \dots, \beta, \dots, \alpha$, т. е. некоторая цепочка повторяется, то,

удалив из вывода последовательность γ, \dots, β , снова получим последовательность, являющуюся выводом α . Следовательно, для любой цепочки $\alpha \in L(G)$ существует ее бесповторный вывод в G , т. е. вывод, в котором все цепочки различны, причем ввиду того, что G — неукорачивающая грамматика, длины цепочек в выводе не превосходят n . Число $r(n)$ таких цепочек ограничено:

$$r(n) \leq \sum_{i=1}^n |V \cup W|^i,$$

и, следовательно, длина бесповторного вывода цепочки α длины n не превосходит $r(n)!$. Поэтому для любой цепочки ω алгоритм распознавания принадлежности ее $L(G)$ заключается в том, чтобы перебрать все бесповторные последовательности цепочек длины $\leq n$, оканчивающиеся цепочкой ω (число которых ограничено величиной $s_\omega(n)$), и для каждой из них проверить, является ли она выводом в G (сложность проверки также ограничена, так как она заключается в проверке для каждой пары соседних цепочек β_i, β_{i+1} , получается ли β_{i+1} из β_i применением некоторого правила G). \square

В действительности справедливо более сильное утверждение — языки, порождаемые неукорачивающими грамматиками, примитивно рекурсивны.

Очевидно, что все контекстные грамматики являются неукорачивающими. Справедливо и обратное: для любой неукорачивающей грамматики существует эквивалентная ей контекстная грамматика. Мы не будем здесь доказывать это утверждение; отметим лишь, что для его доказательства можно использовать метод двойников — так, как это было сделано в примере 7.1, д. Из этих двух утверждений следует, что класс языков, порождаемых неукорачивающими грамматиками, совпадает с классом языков типа 1. В некоторых вариантах классификации Хомского грамматики типа 1 определяются именно как неукорачивающие. Это, как видим, не изменяет классификации языков.

Таким образом, разрешимость (и даже примитивная рекурсивность) является необходимым условием принадлежности языка типу 1, и, следовательно, любое перечислимое, но не примитивно-рекурсивное множество цепочек является языком типа 0, но не типа 1.

Контекстно-свободные грамматики. Деревья вывода. По определению КС-грамматик они могут быть и укорачивающими (правая часть бесконтекстного правила $A \rightarrow \alpha$ может быть пустой); поэтому, строго говоря, существуют языки типа 2, не являющиеся языками типа 1 (таковы, например, все КС-языки, содержащие пустую цепочку e , поскольку в грамматиках типа 1 пустая цепочка невыводима). Однако в теории грамматик показывается, что для любой укорачивающей КС-грамматики может быть построена неукорачивающая КС-грамматика G' , почти эквивалентная G , т. е. такая, что $L(G') = L(G)$, если $e \notin L(G)$, и $L(G') = L(G) \setminus e$, если $e \in L(G)$. Поэтому достаточно ограничиться изучением свойств неукорачивающих КС-грамматик.

КС-языки являются наиболее хорошо изученным классом языков. Это объясняется тем, что с одной стороны, КС-грамматики оказались весьма подходящим аппаратом для описания строения естественных языков и особенно языков программирования; с другой стороны, благодаря относительной простоте и содержательности структуры КС-языков и наличию удобных средств ее описания исследование КС-языков представляет значительный теоретический интерес.

Появление понятия КС-грамматики (введенного Хомским) практически совпало с появлением метаязыка Бэкуса (или нормальной формы Бэкуса), который впервые был использован при описании языка программирования АЛГОЛ-60 и быстро стал общепринятым средством формального описания языков программирования. Описание языка с помощью нормальных форм Бэкуса представляет собой совокупность «металингвистических формул» — выражений вида $X ::= Y_1 | \dots | Y_n$, где X — некоторый текст, заключенный в угловые скобки и называемый металингвистической переменной, а Y_1, \dots, Y_n — последовательности металингвистических переменных и основных символов языка (букв, цифр, разделителей, неделимых слов типа `begin`, `end`, `else` и т. д.). Знак $::=$ называется металингвистической связкой и читается как «есть» или «— это»; знак $|$ — это металингвистическая связка «или»; металингвистическая переменная представляет собой имя конструкции языка; металингвистиче-

ская формула в целом — это описание различных синтаксических вариантов строения конструкции X , стоящей в левой части, через другие конструкции и основные символы языка, указанные в правой части. Перечисление вариантов производится с помощью связки.

Пример 7.2. а. Множество идентификаторов языка программирования — это множество цепочек из букв и цифр, начинающихся с буквы. Этот язык может быть описан с помощью следующих нормальных форм Бэкуса (металингвистических формул), в которых цепочка из букв и цифр сокращенно называется БЦ-цепочкой:

```
⟨идентификатор⟩ ::= ⟨буква⟩ | ⟨буква⟩ ⟨БЦ-цепочка⟩  
⟨БЦ-цепочка⟩ ::= ⟨буква⟩ | ⟨цифра⟩ | ⟨буква⟩ ⟨БЦ-цепочка⟩ | ⟨цифра⟩ ⟨БЦ-цепочка⟩  
⟨буква⟩ ::= a | b | ... | z  
⟨цифра⟩ ::= 0 | 1 | ... | 9
```

Основные символы языка — это 26 букв латинского алфавита и 10 цифр.

б. Язык арифметических выражений (без констант и с фиксированным множеством переменных a, b, c) в метаязыке Бэкуса описывается следующим образом:

```
⟨арифметическое выражение⟩ ::= ⟨терм⟩ | ⟨арифметическое выражение⟩ + ⟨терм⟩ | ⟨арифметическое выражение⟩ — ⟨терм⟩  
⟨терм⟩ ::= ⟨множитель⟩ | ⟨терм⟩ ⟨множитель⟩ | ⟨терм⟩ / ⟨множитель⟩  
⟨множитель⟩ ::= ((⟨арифметическое выражение⟩)) | ⟨переменная⟩  
⟨переменная⟩ ::= a | b | c
```

Уже из этих примеров ясно, что нормальные формы Бэкуса нетрудно преобразовать в КС-грамматику. Действительно, основные символы соответствуют терминальным символам грамматики, металингвистические переменные — нетерминальным символам, связка $::=$ — знаку \rightarrow , а без связки $|$ можно обойтись, если формулу $X ::= Y_1 | \dots | Y_n$ заменить системой формул $X ::= Y_1, \dots, X ::= Y_n$. Если указанное преобразование провести для последнего примера, заменив $⟨\text{арифметическое выражение}⟩$ на символ I , $⟨\text{терм}⟩$ — на T , $⟨\text{множитель}⟩$ — на M

〈переменная〉 — на L , то получим КС-грамматику из примера 7.1, г. Ясно также, что указанное соответствие является взаимно однозначным, и всякая КС-грамматика легко преобразуется в нормальные формы Бэкуса. В частности, для сокращения записи КС-грамматик используется связка $|$, чем и будем пользоваться в дальнейшем. Такое простое соответствие между двумя видами описаний одного и того же класса языков, которые возникли практически одновременно из разных соображений (грамматики — из теоретических, формы Бэкуса — из прикладных), является свидетельством близости теоретических и практических целей (что бывает не так часто, как хотелось бы) исследования КС-языков и одной из причин необычайно быстрого развития теории КС-языков в течение 1960–1970-х гг. Другая, более глубокая причина широкого использования аппарата КС-грамматик для анализа естественных языков и языков программирования — это возможность описать грамматическую (в смысле традиционной лингвистики) структуру текста языка (т. е. состав и связь языковых конструкций, образующих текст) в терминах вывода этого текста в подходящей грамматике. Изучением связи между структурой текста и его выводом мы сейчас и займемся.

Вывод в формальных системах вообще и в грамматиках в частности определяется как последовательность цепочек, в которой любая пара соседних цепочек α_i, α_{i+1} находится в отношении непосредственной выводимости: $\alpha_i \Rightarrow \alpha_{i+1}$. В КС-грамматиках вывод может быть представлен существенно более компактно — в виде *дерева вывода*. Для его описания введем необходимые определения. Будем рассматривать деревья с корнями, ориентированные в направлении от корня (см. § 4.3). Вершина называется потомком вершины v_i , если существует путь из v_i в v_j , и непосредственным потомком v_i , если существует ребро (v_i, v_j) . Дерево с корнем назовем элементарным, если все его концевые вершины — непосредственные потомки корня. Для деревьев с корнями упорядочение вершин слева направо определяется следующим образом: для любого элементарного поддерева его концевые вершины полностью упорядочены (т. е. расположены на плоскости так, что для любой пары вершин ясно,

какая из них левее другой в обычном смысле слова); если v_i и v_j — непосредственные потомки одной вершины и v_i левее v_j , то любой потомок v_i левее любого потомка v_j . Легко видеть, что любые две вершины лежат либо на одном пути, либо одна левее другой. Если концевые вершины дерева помечены буквами, то цепочка, которая получится, если выписать буквы в соответствии с упорядочением помеченных ими концевых вершин, называется *кроной дерева*.

Каждому правилу $r_i: A \rightarrow s_1s_2\dots s_{l(i)}$, где s_j — терминальный или нетерминальный символ, поставим в соответствие элементарное дерево $t(r_i)$ с $l(i) + 1$ вершиной, корень которого помечен символом A , а $l(i)$ концевых вершин — символами $s_1s_2\dots s_{l(i)}$, причем упорядочение вершин соответствует упорядочению помечающих их символов в правой части r_i . Очевидно, что крона этого дерева совпадает с правой частью r_i .

Для любого вывода $\Delta = I, \alpha_1, \alpha_2, \dots, \alpha_n$ в грамматике G дерево вывода определяется индуктивно. Отрезку вывода $\Delta_1 = I, \alpha_1$ соответствует правило $I \rightarrow \alpha_1$ грамматики G ; этому отрезку ставится в соответствие элементарное дерево, соответствующее этому правилу; заметим, что его кроной является α_1 . Пусть уже определено дерево вывода $t(\Delta_i)$ для отрезка $\Delta_i = I, \alpha_1, \dots, \alpha_i$, и его кроной является α_i . Дерево $t(\Delta_{i+1})$ для отрезка $\Delta_{i+1} = I, \alpha_1, \dots, \alpha_i, \alpha_{i+1}$ определяется так. Если α_{i+1} получается из α_i применением правила $r_j: A \rightarrow \beta$ к некоторому вхождению A в α_i , то этому вхождению A однозначно соответствует концевая вершина в кроне $t(\Delta_i)$. Тогда $t(\Delta_{i+1})$ получается отождествлением вершины элементарного дерева $t(r_j)$ с вершиной v_A , т. е. «приклеиванием» $t(r_j)$ к вершине v_A . Легко видеть, что эта операция над деревьями соответствует применению правила r_j к α_i , и кроной $t(\Delta_{i+1})$ является α_{i+1} . Высотой дерева вывода (или просто *высотой вывода*) называется длина максимального пути от корня дерева к концевой вершине.

Заметим, что одновременно с индуктивным определением дерева вывода мы доказали следующее утверждение: если Δ — вывод α , то крона $t(\Delta)$ совпадает с α . Кроме того, число нетерминальных вершин (т. е. вершин, помеченных нетерминальными символами) дерева вывода

$\Delta = I$, $\alpha_1, \dots, \alpha_n$ терминальной цепочки $\alpha = \alpha_n$ равно n , т. е. числу применений правил в выводе Δ . Действительно, на каждом шаге построения дерева вывода (соответствующем применению одного правила r_i) ровно одна нетерминальная вершина (та, к которой приклеивается дерево $t(r_i)$) перестает быть концевой; все эти вершины для разных шагов различны, а в окончательном дереве $t(\Delta)$ концевых нетерминальных вершин нет. Таким образом, общее число вершин (и, следовательно, символов, использованных при описании) дерева вывода $I, \alpha_1, \dots, \alpha_n$ равно $n + |\alpha_n|$, что гораздо меньше числа

$$\sum_{i=1}^n |\alpha_i|$$

символов в представлении вывода как последовательности цепочек.

Пример 7.3. Арифметическое выражение $a * b + c$ в грамматике из примера 7.1, г имеет вывод $I, I + T, T + T, T * M + T, M * M + T, K * M + T, a * M + T, a * K + T, a * b + K, a * b + c$. Дерево этого вывода приведено на рис. 7.1. Оно содержит 16 вершин, тогда как вывод — 52 символа. Правда, для столь простого выражения и вывод, и его дерево выглядят слишком сложными. Причинами этого являются некоторые особенности данной грамматики, которые будут обсуждены ниже (пример 7.5).

Компактность — не единственное достоинство дерева вывода. Не менее важно то, что дерево вывода сохраняется при некоторых изменениях вывода, что позволяет

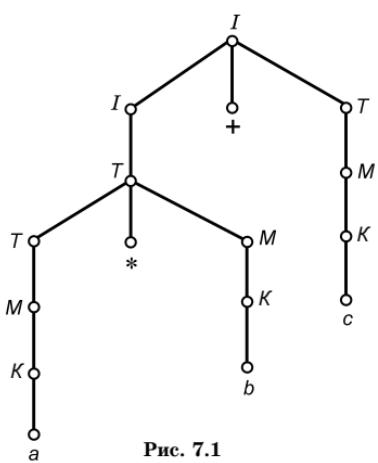


Рис. 7.1

их считать несущественными. Таким несущественным изменением вывода является изменение порядка применения правил к различным вхождениям нетерминальных символов цепочки. Более точно, если в выводе $\Delta = I, \alpha_1, \dots, \alpha_i, \alpha_{i+1}, \alpha_{i+2}, \dots, \alpha_n$ цепочки α_{i+1} и α_{i+2} получены применением правил r_{j_1} и r_{j_2} грамматики к двум различным вхождениям нетерминальных символов A_{j_1} и A_{j_2}

в α_i , то перестановка этих цепочек в выводе допустима в том смысле, что она не изменит результата, т. е. $\Delta' = I$, $\alpha_1, \dots, \alpha_i, \alpha_{i+2}, \alpha_{i+1}, \alpha_{i+3}, \dots, \alpha_n$ — также вывод цепочки α_n . В терминах дерева вывода это очевидно: приклеивание элементарного дерева $t(r_{j_1})$ к вершине A_{j_1} и элементарного дерева $t(r_{j_2})$ к другой вершине A_{j_2} одного и того же дерева дадут одно и то же дерево независимо от порядка этих двух операций.

Множество выводов, получаемых указанными допустимыми перестановками и имеющими одно и то же дерево вывода, образует класс эквивалентности. Среди эквивалентных выводов существует *левосторонний вывод*, в котором на любом шаге правило применяется всегда к самому левому нетерминальному символу. Например, вывод, описанный в примере 7.3, — левосторонний. Между левосторонними выводами и деревьями существует взаимно однозначное соответствие. Очевидно, что если левосторонние выводы равны, то равны и деревья. Пусть $\Delta = I$, $\alpha_1, \alpha_2\dots$ и $\Delta' = I$, $\beta_1, \beta_2\dots$ различные левосторонние выводы, и пусть α_i и β_i — первые несовпадающие цепочки в них. Поскольку $\alpha_{i-1} = \beta_{i-1}$, то α_i и β_i могут быть различны только за счет того, что к самому левому нетерминальному символу α_{i-1} будут применены различные правила; поэтому при построении деревьев $t(\Delta)$ и $t(\Delta')$ на этом шаге к самой левой вершине дерева с короной α_{i-1} будут приклешены различные элементарные деревья, и, следовательно, $t(\Delta)$ и $t(\Delta')$ будут различны. Таким образом, для данного дерева левосторонний вывод — единственный.

Взаимно однозначного соответствия между цепочками данного языка L и деревьями вывода в грамматике G , порождающей $L(G)$, может и не быть. КС-грамматика G называется *неоднозначной*, если существует хотя бы одна цепочка в $L(G)$, имеющая в G более одного дерева вывода (или более одного левостороннего вывода). КС-язык, все порождающие грамматики которого неоднозначны, называется *существенно неоднозначным*.

Пример 7.4. а. Грамматика G из примера 7.1, а, порождающая язык $L(G) = \{a^{2n-1}\}$, однозначна. Действительно, в этой грамматике любой вывод имеет вид $I, aaI, \dots, a^{2n}I, \dots$, т. е. в нем $n - 1$ раз применяется второе

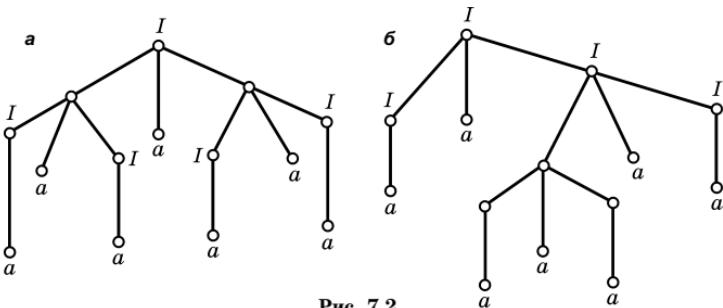


Рис. 7.2

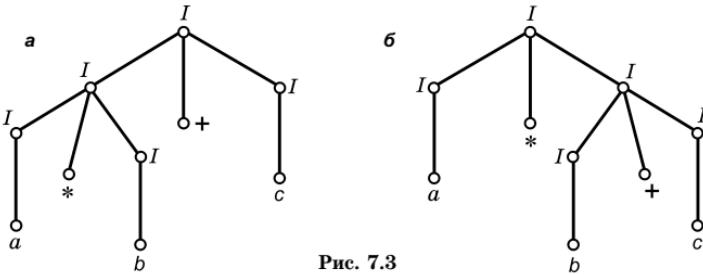


Рис. 7.3

правило; первое правило может быть применено только на последнем шаге, поскольку оно дает терминальную цепочку. Поэтому различные выводы в G обязательно отличаются длиной, а поскольку на каждом шаге добавляется ровно один нетерминальный символ, то выводы разной длины дают различные цепочки языка $L(G)$. Следовательно, в этой грамматике любая цепочка имеет ровно один вывод (в ней нет даже эквивалентных выводов, т. е. различных выводов, имеющих одинаковое дерево). Однако этот же язык можно задать неоднозначной грамматикой $I \rightarrow a$, $I \rightarrow IaI$, в которой, например, цепочка a^7 имеет несколько различных деревьев (например, рис. 7.2, а и б) и соответственно несколько левосторонних выводов.

6. Рассмотрим грамматику, полученную из грамматики примера 7.1, г отождествлением всех нетерминальных символов и удалением всех получившихся тривиальных правил $I \rightarrow I$:

$$I \rightarrow I + I \mid I - I \mid I * I \mid I/I \mid (I) \mid a \mid b \mid c.$$

Эта грамматика эквивалентна исходной; кроме того, выводы и деревья в ней существенно короче. Однако она

неоднозначна. Выражение $a * b + c$ имеет в ней два дерева вывода (рис. 7.3, а и б).

в. Язык $\{a^m b^m c^n \cup a^m b^n c^n\}$ существенно неоднозначен. Доказательство этого имеется, например, в [1].

Неоднозначность языка представляет собой неудобство при его использовании. Дело в том, что дерево вывода является основным средством для интерпретации цепочки; поэтому синтаксическая неоднозначность (т. е. наличие нескольких деревьев вывода) цепочки влечет за собой ее семантическую неоднозначность — наличие разных интерпретаций. Например, для цепочки $a * b + c$ из примера 7.4, б разные деревья вывода интерпретируются как разные способы расстановки скобок (в первом случае $(a * b) + c$, во втором — $a * (b + c)$), что ведет к разной последовательности операций и соответственно к разным результатам вычислений. Отметим, что явное введение скобок после каждой неодноместной операции в языках формул (логических, арифметических, алгебраических и т. д.) является достаточным средством для обеспечения однозначности. Грамматики со скобками (см. пример 7.1, в) хотя и приводят к избыточным скобкам в формулах, однако позволяют уменьшить число нетерминальных символов и тем самым упростить синтаксическую структуру формулы, определяемую ее деревом. Именно поэтому языки формул исчисления выскаживаний и исчисления предикатов (§ 6.1 и 6.2) определены в стиле примера 7.1, в.

Приведение КС-грамматик. Поскольку для одного и того же КС-языка могут существовать различные грамматики (в том числе и разных типов), то возникает проблема выбора грамматики, наиболее подходящей по тем или иным свойствам, или проблема эквивалентного преобразования грамматики к нужному виду. Желаемые свойства могут быть различными — однозначность, минимальное число правил или нетерминальных символов, простота вывода и т. д. Универсальных методов эквивалентных преобразований КС-грамматик не существует из-за неразрешимости алгоритмических проблем распознавания эквивалентности КС-грамматик и существенной неоднозначности КС-языков (см. ниже теоремы 7.6 и 7.7). Однако можно предложить эквивалентные преобразования,

которые с некоторой точки зрения улучшают грамматику. Назовем нетерминальный символ A выводимым, или достижимым, если $I \Rightarrow^* \alpha A \beta$, и производящим, если $A \Rightarrow^* \gamma$, где α, β — произвольные цепочки, а $\gamma \in V^*$ (γ — терминальная цепочка). Нетерминальный символ называется существенным, если он достижимый и производящий; в противном случае он называется несущественным, или бесполезным. Грамматика называется *приведенной*, если она — неукорачивающая и не содержит бесполезных символов.

Покажем, что для любой КС-грамматики можно построить эквивалентную ей приведенную КС-грамматику. Для этого сначала покажем, как выделять достижимые и производящие символы. Алгоритм выделения достижимых символов определим индуктивно. Обозначим через M_1 множество всех нетерминальных символов, содержащихся в правых частях правил вида $I \rightarrow \alpha$. Очевидно, что все символы M_1 достижимы. Пусть уже определено множество нетерминальных символов M_i и показано, что все символы M_i достижимы. Определим $M_{i+1} = M_i \cup M'_i$, где M'_i — множество всех нетерминальных символов из правых частей правил вида $A \rightarrow \alpha$, где $A \in M_i$. Алгоритм останавливается на шаге k , если $M_k = M_{k-1}$; множество достижимых символов совпадает с M_k , причем $k \leq |W|$. Алгоритм выделения производящих символов аналогичен предыдущему, но работает «с конца вывода». Обозначим через Q_1 множество всех нетерминальных символов A , для которых в G имеются правила вида $A \rightarrow \alpha$, где $\alpha \in V^*$. Все символы Q_1 — производящие. Пусть уже определено множество Q_i и показано, что все символы Q_i — производящие. Определим $Q_{i+1} = Q_i \cup Q'_i$, где Q'_i содержит все нетерминальные символы A , для которых в G имеются правила вида $A \rightarrow \alpha$, где $\alpha \in (V \cup Q_i)^*$. Алгоритм останавливается на шаге l , если $Q_l = Q_{l-1}$. Множество производящих символов совпадает с Q_l , причем $l \leq |W|$.

Теперь приведенная грамматика $G' = \langle V', W', R', I \rangle$, эквивалентная предыдущей, определяется так: W' — это множество существенных символов $G' : W' = M_k \cup Q_l$; R' содержит только те правила из R , в которых нет бесполезных символов; V' содержит только такие терминаль-

ные символы, которые встречаются в правых частях правил из R . Равенство $L(G) = L(G')$ доказать нетрудно; это мы предоставляем читателю.

Следующее полезное преобразование грамматики — это устранение правил вида $A \rightarrow B$, которые называются цепными. Алгоритм, который по произвольной неукорачивающей КС-грамматике $G = \langle V, W, R, I \rangle$ строит эквивалентную ей грамматику G' без цепных правил, состоит из двух этапов. На первом этапе для каждого $A \in W$ находится множество W_A всех нетерминальных символов E , таких, что $A \Rightarrow^* E$; ясно, что $E \in W_A$, если имеется последовательность правил $A \rightarrow B, B \rightarrow C, \dots, \rightarrow E$, т. е. отношение $A \Rightarrow^* E$ является транзитивным замыканием (см. § 1.3) отношения $A \rightarrow E$. На втором этапе определяется множество правил R' : если $B \rightarrow \alpha$ содержится в R и не является цепным, то для любого A , такого, что $B \in W_A$, включаем в R' правило $A \rightarrow \alpha$. Полагаем $G' = \langle V, W, R', I \rangle$.

Пример 7.5. Применение описанного алгоритма к грамматике из примера 7.1, г дает грамматику

$$\begin{aligned} I &\rightarrow I + T \mid I - T \mid T * M \mid T/M \mid (I) \mid a \mid b \mid c; \\ T &\rightarrow T * M \mid T/M \mid (I) \mid a \mid b \mid c; \\ M &\rightarrow (I) \mid a \mid b \mid c. \end{aligned}$$

Сравнение этих двух грамматик показывает, что наличие цепных правил имеет как достоинства, так и недостатки. Главным недостатком является громоздкость вывода: построив дерево вывода в последней грамматике для выражения $a * b + c$, можно убедиться, что оно компактнее дерева для того же выражения в грамматике примера 7.1, г (см. рис. 7.1) за счет отсутствия ребер $T \rightarrow M, M \rightarrow K$ и $I \rightarrow T$. С другой стороны, грамматика примера 7.5 содержит 18 правил, тогда как в примере 7.1, г — 11 правил. Цепные правила дают возможность своеобразного «вынесения за скобки» (и, наоборот, описанный выше алгоритм их устраниния «раскрывает скобки», размножая правые части правил), что упрощает описание грамматики в целом. Поэтому такие правила используются при задании языков программирования, хотя они и усложняют вывод.

Докажем теперь, что алгоритм устранения цепных правил в грамматике G дает эквивалентную грамматику G' . Пусть $\Delta_0 = I$, $\alpha_1, \dots, \alpha_m$ — левосторонний вывод в G цепочки α_m , и α_k — самая левая цепочка, к которой применяется цепное правило. Тогда в Δ_0 имеются цепочки вида $\alpha_k = \beta_1 A \beta_2$, $\alpha_l = \beta_1 C \beta_2$, $\alpha_{l+1} = \beta_1 \gamma \beta_2$, причем на отрезке $\alpha_k, \dots, \alpha_l$ применены цепные правила, а α_{l+1} получено из α_l правилом $C \rightarrow \gamma$.

Рассмотрим теперь последовательность $\Delta_1 = I, \alpha_1, \dots, \alpha_k, \alpha_{l+1}, \dots, \alpha_m$. В ней переход от α_k к α_{l+1} осуществляется применением правила $C \rightarrow \gamma$, которое по описанному алгоритму принадлежит R' ; кроме того, Δ_1 содержит по крайней мере на одно цепное правило меньше, чем Δ_0 . Повторяя эту процедуру, в конечном счете получаем последовательность $\Delta_n = I, \dots, \alpha_m$, которая не содержит цепных правил; в ней все переходы к следующей цепочке осуществляются либо нецепными правилами G , либо новыми правилами вида $C \rightarrow \gamma$, которые включены в R' на втором этапе описанного алгоритма. Так как цепочка α_m при этом преобразовании сохранилась, то Δ_n является выводом α_m в G' ; следовательно, $L(G) \subseteq L(G')$. Обратное включение $L(G') \subseteq L(G)$ также доказывается преобразованием вывода.

Одним из явных достоинств грамматик без цепных правил является отсутствие циклов в выводах, т. е. выводимостей вида $A \Rightarrow^* A$. Для неукорачивающих КС-грамматик это равносильно тому, что все выводы в них — бесповторные, ни одна цепочка в выводе не повторяется. В свою очередь, бесповторность вывода позволяет установить фундаментальное для КС-грамматик обстоятельство — линейную зависимость между длиной вывода и длиной выводимой цепочки.

Теорема 7.3. Если $G = \langle V, W, R, I \rangle$ — приведенная КС-грамматика без циклов, то существуют такие константы c_1 и c_2 , зависящие от G , что для любого вывода $\Delta(\alpha)$ цепочки $\alpha \in L(G)$

$$c_1|\alpha| \leq |\Delta(\alpha)| \leq c_2|\alpha|,$$

где $|\Delta(\alpha)|$ — длина вывода $\Delta(\alpha)$.

Пусть $k_0 = |W|$, k_1 — максимальная длина правой части в правилах G . Так как G не содержит циклов и укора-

чивающих правил, то для любого вывода $\beta \Rightarrow^{k_0} \gamma$ $|\gamma| > |\beta|$ и, следовательно, для вывода $I \Rightarrow^{k_0|\alpha|} \gamma$ $|\gamma| > |\alpha|$. Поэтому $|\Delta(\alpha)| \leq k_0|\alpha|$. С другой стороны, $|\alpha| \leq k_1|\Delta(\alpha)|$. Отсюда $|\alpha|/k_1 \leq |\Delta(\alpha)|$. \square

Алгоритмические проблемы для грамматик. Поскольку класс множеств, порождаемых грамматиками, совпадает с классом перечислимых множеств (теорема 7.1), то для языков типа 0 справедлив аналог теоремы Райса: никакое нетривиальное свойство языков типа 0 не является алгоритмически разрешимым; точнее, ни для какого нетривиального свойства языков типа 0 не существует алгоритма, который по произвольной грамматике G выяснял бы, обладает ли этим свойством язык $L(G)$. Для языков типа 1 уже существуют разрешимые проблемы. Например, для любой цепочки α и любого языка L типа 1 разрешима проблема принадлежности α языку L (теорема 7.2). Однако проблемы пустоты и бесконечности языка, порожденного данной грамматикой типа 1, неразрешимы. Для языков типа 2 эти проблемы разрешимы. Разрешимость проблемы пустоты непосредственно следует из наличия алгоритмов приведения КС-грамматик, а именно: язык $L(G)$ непуст, если и только если начальный символ G — производящий.

Разрешимость проблемы бесконечности следует из более сильного утверждения о характере бесконечности КС-языков, называемого иногда «леммой о разрастании».

Теорема 7.4. Для любой КС-грамматики $G = \langle V, W, R, I \rangle$ существуют такие числа p и q , что каждая цепочка $\alpha \in L(G)$ длины, большей p , представима в виде $\alpha = \beta\gamma\delta\epsilon$, где γ непусто, либо δ непусто, $|\gamma\delta| < q$, причем все цепочки вида $\beta\gamma^n\delta^n\epsilon$ также принадлежат $L(G)$.

Построим приведенную грамматику G' , эквивалентную G и не содержащую цепных правил. Пусть k — число нетерминальных символов в G' . Существует лишь конечное число цепочек в $L(G')$, высота вывода которых не превышает k . Обозначим через p максимум длин таких цепочек. Если $|\alpha| > p$, то дерево вывода α имеет высоту $> k$, т. е. содержит путь от корня к концевой вершине длины $> k$. Рассмотрим конец этого пути длиной $k + 1$. Последовательности его вершин соответствует последовательность из $k + 1$ нетерминальных символов

$A_{i_1}, \dots, A_{i_k+1}$, в которой хотя бы один символ должен повториться: $A_{i_r} = A_{i_s} = B$, $r < s \leq k + 1$. Поддереву с вершиной $A_{i_r} = B$ соответствует вывод в G' вида $B \Rightarrow {}^* \rho B \pi \Rightarrow {}^* \gamma \omega \delta$, где $\gamma \omega \delta \in V^*$, причем γ непусто либо δ непусто; это следует из того, что G' — приведенная и не содержит цепных правил. Кроме того, цепочка $\gamma \omega \delta$ является подцепочкой α (поскольку она является кроной поддерева вывода α), и, следовательно, α представима в виде $\alpha = \beta \gamma \omega \delta \epsilon$. При этом, поскольку высота поддерева с вершиной A не превосходит k , то существует такое q (зависящее только от G'), что длина кроны $\gamma \omega \delta$ этого поддерева не превосходит q .

Выводы $B \Rightarrow {}^* \beta B \pi$, $\rho \Rightarrow {}^* \gamma$, $\pi \Rightarrow {}^* \delta$ могут быть повторены в G' любое число раз. Поэтому в G' для любого $n = 1, 2, \dots$ могут быть получены выводы $I \Rightarrow {}^* \beta B \epsilon \Rightarrow {}^* \beta \gamma^n \omega \delta^n \epsilon$ и, следовательно, $\beta \gamma^n \omega \delta^n \epsilon \in L(G') = L(G)$. \square

Из доказательства теоремы 7.4 легко извлечь следующий конструктивный критерий бесконечности: язык $L(G)$ бесконечен, если и только если приведенная грамматика G' , эквивалентная G , содержит такой нетерминальный символ A , что $A \Rightarrow {}^* \rho A \pi$, где либо ρ непусто, либо π непусто. Действительно, если такой символ существует, то в G' возможны выводы $\rho^n A \pi^n$ для любого n и соответственно существуют сколь угодно длинные терминальные цепочки, выводимые в G' . Если же таких символов в G' нет, то на одном пути в дереве вывода нетерминальные символы не могут повторяться. Поэтому в G' возможны только деревья высоты, не превосходящей числа нетерминальных символов G' , и, следовательно, язык $L(G) = L(G')$ конечен.

Однако содержание теоремы 7.4 гораздо глубже названного следствия. Она указывает на периодический характер бесконечности в КС-языках: наряду с любым достаточно длинным словом $\beta \gamma \omega \delta \epsilon$ КС-язык обязательно содержит и все слова вида $\beta \gamma^n \omega \delta^n \epsilon$. Это свойство является необходимым условием бесконтекстности языка; для того чтобы доказать, что язык не является бесконтекстным, достаточно показать, что он не обладает этим свойством. Его можно сформулировать и в терминах длин цепочек: множество длин цепочек КС-языка либо конечно, либо содержит арифметическую прогрессию. Отсюда сразу следует, что, например, язык $\{a^{n^2}\}$ не является КС-языком.

Рассмотрим теперь некоторые неразрешимые проблемы для КС-грамматик. Основным методом доказательства неразрешимостей в теории формальных языков является сведение к комбинаторной проблеме Поста, которая неразрешима (теорема 6.20). Здесь будет удобно пользоваться слегка измененной формулировкой проблемы Поста (не для m пар, как в теореме 6.20, а для двух списков длины m): для списков $X = (\alpha_1, \dots, \alpha_m)$ и $Y = (\beta_1, \dots, \beta_m)$ цепочек в алфавите U решить, существует ли последовательность индексов i_1, \dots, i_N , такая, что $\alpha_{i_1} \dots \alpha_{i_N}$.

Пусть алфавит U и число m зафиксированы. Введем алфавит $U_0 = \{b_1, \dots, b_m\}$, не пересекающийся с U , и обозначим $U_1 = U \cup U_0$. Для списка $X = (\alpha_1, \dots, \alpha_m)$ m цепочек в алфавите U обозначим через $Q(X)$ множество всех цепочек вида $b_{i_N} \dots b_{i_1} \alpha_{i_1} \dots \alpha_{i_N}$ для любых последовательностей индексов i_1, \dots, i_N , $N \geq 1$. Для языков $Q(X)$ справедливы следующие два утверждения.

1. Если $X = (\alpha_1, \dots, \alpha_m)$ и $Y = (\beta_1, \dots, \beta_m)$ — списки цепочек в алфавите U , то комбинаторная проблема для этих списков имеет решение, если и только если множество $Q(X) \cap Q(Y)$ непусто.

Действительно, если $\gamma \in Q(X)$ и $\gamma \in Q(Y)$, то $\gamma = b_{i_N} \dots b_{i_1} \alpha_{i_1} \dots \alpha_{i_N} = b_{i_N} \dots b_{i_1} \beta_{i_1} \dots \beta_{i_N}$ и, следовательно, $\alpha_{i_1} \dots \alpha_{i_N} = \beta_{i_1} \dots \beta_{i_N}$.

2. Для любого списка X цепочек в алфавите U $Q(X)$ является КС-языком в алфавите U_1 .

КС-грамматика, порождающая язык $Q(X)$ для списка $X = (\alpha_1, \dots, \alpha_m)$, задается системой $2m$ правил $I \rightarrow b_i I \alpha_i$, $I \rightarrow b_i \alpha_i$ ($i = 1, \dots, m$). Нетрудно видеть, что эта грамматика однозначна.

Из этих двух утверждений непосредственно следуют две важные теоремы о неразрешимости.

Теорема 7.5. Не существует алгоритма, который по двум произвольным КС-грамматикам G_1 и G_2 определял бы, пусто ли пересечение $L(G_1) \cap L(G_2)$.

Если бы такой алгоритм существовал, то можно было бы определить, пусто ли пересечение КС-языков $Q(X)$ и $Q(Y)$ для любых списков X, Y , состоящих из m цепочек, откуда следовала бы разрешимость комбинаторной проблемы Поста для данного m , что невозможно. \square

Теорема 7.6. Не существует алгоритма, который для произвольной КС-грамматики позволял бы узнать, является ли она однозначной или нет.

Язык $Q(X)$, где $X = (\alpha_1, \dots, \alpha_m)$, порождается однозначной грамматикой с системой правил $R_x: I \rightarrow A; A \rightarrow b_i\alpha_i; A \rightarrow b_iAa_i$. Язык $Q(Y)$, где $Y = (\beta_1, \dots, \beta_m)$, порождается однозначной грамматикой $R_y: I \rightarrow B; B \rightarrow b_i\beta_i; B \rightarrow b_iB\beta_i$. Язык $Q(X) \cup Q(Y)$ порождается грамматикой с системой правил $R_X \cup R_Y$. Эта грамматика неоднозначна в том и только в том случае, когда $Q(X) \cap Q(Y)$ непусто, а это свойство неразрешимо. \square

Из теоремы 7.5 следует, что пересечение КС-языков не обязательно является КС-языком, поскольку для КС-языков проблема пустоты разрешима. Напротив, объединение КС-языков — всегда КС-язык: КС-грамматика для него получается просто объединением правил исходных грамматик. Так как пересечение языков, как и любых множеств, выражается через объединение и дополнение соотношением де Моргана (3.14) в виде

$$L_1 \cap L_2 = \overline{\overline{L}_1 \cup \overline{L}_2}$$

(см. «Булева алгебра и теория множеств» в § 3.2; для языка L в алфавите $V \setminus \overline{L} = V^* \setminus L$), то отсюда следует, что дополнение КС-языка — не всегда КС-язык. Более того, проблемы распознавания типа языка для пересечения и дополнения неразрешимы.

Теорема 7.7. Следующие проблемы для КС-языков неразрешимы:

- 1) является ли КС-языком пересечение КС-языков;
- 2) является ли КС-языком дополнение к КС-языку;
- 3) проблема тривиальности КС-языка ($L = V^*$) или, что то же самое, пустоты дополнения;
- 4) проблема эквивалентности двух КС-грамматик. \square

Предыдущие рассуждения и теорема 7.5 должны были нас подготовить к этим неразрешимостям: поскольку проблема пустоты для пересечения КС-языков неразрешима, а дополнение связано с пересечением и тривиальными множествами V^* и \emptyset соотношениями де Моргана и $\overline{L} = V^* \setminus L$, то кажется естественным, что неразрешимости для пересечения влекут за собой неразрешимости для дополнения (объединение, как мы видели, неприят-

ностей в этом смысле не доставляет). Кроме того, проблема тривиальности — частный случай проблемы эквивалентности, поэтому неразрешимость 4 следует из неразрешимости 3. Однако, например, неразрешимость пустоты дополнения нельзя доказывать «в лоб», опираясь только на указанные связи (т. е. рассуждая примерно так: для L_1 , L_2 и $L_1 \cup L_2$ проблема пустоты разрешима, а для $L_1 \cap L_2$ — нет; но

$$L_1 \cap L_2 = \overline{\overline{L}_1 \cap \overline{L}_2},$$

поэтому причиной неразрешимости является дополнение), поскольку дополнение не сохраняет тип языка и верхнее дополнение в правой части может относиться уже не к КС-языку. Поэтому доказательства утверждений теоремы 7.7 более сложны и опираются на построение специальных КС-языков вида $Q(X)$, позволяющих свести указанные проблемы к проблеме Поста. Подробное доказательство содержится в [46].

7.2. ОПЕРАЦИИ НАД ЯЗЫКАМИ

Операции над языками, связанные с подстановками. Один вид операций над языками — теоретико-множественный (объединение, пересечение, дополнение) — мы фактически уже рассмотрели. Выяснилось, что класс КС-языков не замкнут (в соответствии с гл. 2) относительно пересечения и дополнения. Это говорит о том, что теоретико-множественные операции, за исключением объединения, не имеют естественной синтаксической интерпретации. С этой точки зрения большой интерес представляют другие операции, которые связаны с подстановкой языка в язык.

Пусть L_0 — язык, задаваемый грамматикой $G_0 = \langle V_0, W_0, R_0, I_0 \rangle$, $a \in V_0$; L_1 — язык, задаваемый грамматикой $G_1 = \langle V_1, W_1, R_1, I_1 \rangle$. Подстановка L_0 ($a \rightarrow L_1$) языка L_1 в L_0 вместо символа a — это операция, сопоставляющая языкам L_0 и L_1 язык $L = L_0(a \rightarrow L_1)$, состоящий из всех цепочек L_0 , в которых вместо каждого вхождения a подставлена некоторая цепочка L_1 (вместо различных вхождений a могут подставляться различные

цепочки). Обобщением этой операции является подстановка $L = L_0$ ($a_1 \rightarrow L_1, \dots, a_k \rightarrow L_k$) нескольких языков L_1, \dots, L_k в язык L_0 , при которой вместо каждого вхождения a подставляется некоторая цепочка соответствующего языка L_i ($i = 1, \dots, k$).

Пример 7.6. Пусть L_0 — язык арифметических выражений (примеры 7.1, г и 1.5), L_1 — язык идентификаторов из примера 7.2. Замена в L_0 переменных a, b, c произвольными идентификаторами описывается подстановкой $L = L_0$ ($a \rightarrow L_1, b \rightarrow L_1, c \rightarrow L_1$). Отметим, что a, b, c — не только терминальные символы L_0 , но и однобуквенные цепочки, входящие как в L_0 , так и в L_1 . Поэтому они содержатся в цепочках (т. е. арифметических выражениях) не только языка L_0 , но и L .

Конкатенация L_1L_2 языков L_1 и L_2 — это операция, сопоставляющая языкам L_1 и L_2 язык L , содержащий все цепочки вида $\alpha\beta$, где $\alpha \in L_1, \beta \in L_2$.

Любую цепочку можно рассматривать как конкатенацию ее подцепочек и, в частности, как конкатенацию составляющих ее букв. Как и обычно при мультилипликативной записи операций (см. § 2.2), используются степенные обозначения $LL = L^2, LLL = L^3$ и т. д. В дальнейшем нам также понадобятся степенные обозначения для повторений подстановки вместо одного и того же символа: $[L]_a^1 = L, [L]_a^2 = L (a \rightarrow L), [L]_a^3 = [L_a]^2 (a \rightarrow L)$ и т. д., а также вместо множества символов $Y = \{a_1, \dots, a_k\}$: $[L]_Y^2 = L (a_1 \rightarrow L, \dots, a_k \rightarrow L)$.

Итерация L^* языка L определяется равенством

$$L^* = \{e\} \cup L_1 \cup \dots \cup L^n = \{e\} \cup \bigcup_{i=1}^{\infty} L^i,$$

где e — пустая цепочка.

Важное замечание: не следует смешивать язык $\{e\}$, состоящий из одной пустой цепочки e , с пустым языком \emptyset , не содержащим ни одной цепочки. Например, $L\{e\} = \{e\}L = L; L\emptyset = \emptyset L = \emptyset$ для любого L .

Конкатенация и итерация, а также рассмотренное выше объединение легко выражаются через подстановку в некоторые простые языки. Обозначим $L_{01} = \{ab\}, L_{02} = \{a\}^*, L_{03} = \{a, b\}$. Тогда для любых $L_1, L_2, L_1L_2 = L_{01}(a \rightarrow L_1, b \rightarrow L_2), L_1^* = L_{02} (a \rightarrow L_1), L_1 \cup L_2 = L_{03} (a \rightarrow L_1, b \rightarrow L_2)$. Нетрудно видеть, что из $n + 2$ элементарных языков $\{a_1\}$,

..., $\{a_n\}$, $\{a_1a_2\}$, $\{a_1, a_2\}$ с помощью некоторой последовательности подстановок можно получить любой конечный язык в алфавите $\{a_1, \dots, a_n\}$.

Зацикливанием $[L_0]^*_a$ по символу a (a -рекурсией, или a -итерацией) языка L_0 будем называть операцию, сопоставляющую языку L_0 язык $L = [L_0]^*_a$, содержащий те и только те цепочки языка

$$\bigcup_{i=1}^{\infty} [L_0]^i_a,$$

в которые не входит символ a . Обобщением этой операции является зацикливание $[L_0]^*_Y$ по множеству символов Y , определение которого получается из предыдущего заменой $[L_0]^i_a$ на $[L_0]^i_Y$.

Пример 7.7. Рассмотрим язык сумм $L_S = \{a, (a + a), (a + a + a)\dots\}$ и язык произведений (термов) $L_T = \{a, aa, aaa, \dots\}$. Подстановка L_S ($a \rightarrow L_T$) дает язык L_{ST} сумм произведений, т. е. язык многочленов, который содержит выражения глубины ≤ 2 (о глубине формул см. § 3.1) над переменной a . Язык $[L_{ST}]^n_a$ — это язык выражений глубины $2n$ над переменной a . Однако язык $[L_{ST}]^*_a$ — это не язык выражений произвольной глубины, как могло бы показаться на первый взгляд, а пустой язык, так как для любого n любая цепочка $[L_{ST}]^n_a$ содержит a и поэтому не входит в $[L_{ST}]^*_a$. Если же в исходные языки L_S и L_T ввести вторую переменную b , т. е. рассматривать языки $L_{S'} = \{a, b, (a + a), (a + b)\dots\}$, $L_{T'} = \{a, b, aa, ab\dots\}$, то $[L_{ST'}]^n_a$ — это язык выражений глубины $\leq 2n$ над переменными a, b , а $[L_{ST'}]^*_a$ — язык выражений произвольной глубины над переменной b .

Главное достоинство операции подстановки с точки зрения теории языков заключается в том, что она имеет тот же характер, что и правила КС-грамматик, поскольку применение любого правила КС-грамматики есть некоторая подстановка вместо символа. Точнее, если $A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_k$ — множество всех правил КС-грамматики G с левой частью A , то при порождении $L(G)$ происходит подстановка конечного языка $\{\alpha_1, \dots, \alpha_k\}$ вместо символа A в язык, состоящий из всех цепочек, порождаемых и содержащих A . Наличие связи между подстановкой и КС-правилами позволяет весьма просто представить операцию подстановки над КС-языками L_1 и L_2 как

операцию над порождающими их КС-грамматиками G_1 и G_2 . Пусть $G_1 = \langle V_1, W_1, R_1, I_1 \rangle$, $G_2 = \langle V_2, W_2, R_2, I_2 \rangle$. Тогда язык $L = L_1 (a \rightarrow L_2)$ порождается грамматикой $G = \langle V, W, R, I \rangle$, которая определяется через G_1 и G_2 следующим образом: $V = (V_1 \cup V_2) \setminus \{a\}$; $W = W_1 \cup W_2 \cup \{a\}$; $I = I_1$; $R = R_1 \cup R_2 \cup \{a \rightarrow I_2\}$. Иначе говоря, символ a становится нетерминальным, и грамматика G_2 «подставляется» в грамматику G_1 посредством КС-правила $a \rightarrow I_2$. Правда, такое определение подстановки грамматики в грамматику корректно только при выполнении условий: 1) $a \notin (V_2 \cup W_2)$; 2) $W_1 \cup W_2 = \emptyset$. Действительно, $a \in V_2$ приводит к противоречию: по определению подстановки языков в этом случае $a \in V$, тогда как определению грамматики G , порождающей $L = L_1 (a \rightarrow L_2)$, $a \notin V$; если $W_1 \cap W_2 \neq \emptyset$, то включение в R правила $a \rightarrow I_2$ приводит к подстановке в L_1 не языка L_2 , а языка \tilde{L} , который содержит язык L_2 , но не совпадает с ним; если же $a \in W_2$, то к a можно применить не только специальное правило $a \rightarrow I_2$, но и некоторые правила из R_2 с a в левой части, в результате чего может оказаться, что $L(G) \neq L_1 (a \rightarrow L_2)$. Кроме того, необходимы условия: 3) $V_1 \cap W \neq \emptyset$; 4) $V_2 \cap W_1 = \emptyset$, поскольку при их нарушении $V \cap W \neq 0$. Поэтому, когда хотя бы одно из условий 1–4 не выполнено, перед подстановкой G_2 в G_1 нужно переименовать некоторые символы из V_1 и W_1 так, чтобы эти условия выполнялись. С учетом этих оговорок *подстановка КС-грамматики* $G_2 = \langle V_2, W_2, R_2, I_2 \rangle$ в КС-грамматику $G_1 = \langle V_1, W_1, R_1, I_1 \rangle$ определяется как операция, которая грамматикам G_1 и G_2 сопоставляет грамматику $G = \langle V, W, R, I \rangle$, для которой $V = (V_1 \cup V_2) \setminus \{a\}$; $W = W_1 \cup W_2 \cup \{a\}$; $I = I_1$; $R = R_1 \cup R_2 \cup \{a' \rightarrow I_2\}$, где V_1, W_1, R'_1, a' — результат переименования V_1, W_1, R_1, a соответственно в случае, когда условия 1–4 не выполнены, и $V'_1 = V_1$; $W'_1 = W_1$; $R'_1 = R_1$; $a' = a$, если условия 1–4 выполнены. Независимо от того, производится переименование или нет, $L(G) = L(G_1) (a \rightarrow L(G_2))$.

Таким образом, подстановка КС-языка в КС-язык может быть представлена как подстановка КС-грамматики в КС-грамматику, дающая снова КС-грамматику. Отсюда следует, что класс КС-языков замкнут относительно подстановки, а следовательно, также относитель-

но конкатенации и итерации (выше была показана их представимость через подстановку в КС-языки).

Зацикливание КС-языка также может быть представлено как операция над его грамматикой. Для языка L_1 с грамматикой $G_1 = \langle V_1, W_1, R_1, I_1 \rangle$ грамматика $G = \langle V, W, R, I \rangle$, порождающая язык $[L_1]_a^*$, определяется следующим образом: $V = V_1 \setminus \{a\}$; $W = W_1 \cup \{a\}$; $I = I_1$; $R = R_1 \cup \{a \rightarrow I_1\}$. Доказательство того, что $L(G) = [L_1]_a^*$, предоставляем читателю. Очевидно, что G — КС-грамматика, и, следовательно, класс КС-языков замкнут относительно зацикливания.

Порождающая способность зацикливания сильнее порождающей способности подстановки в следующем смысле: если L_1 и L_2 — конечные языки, то $L_1(a \rightarrow L_2)$ — конечный язык для любого a , тогда как язык $L(G) = [L_1]_a^*$ бесконечен, если в L_1 имеется цепочка длины 2, содержащая a . Действительно, если в $G I \Rightarrow \alpha a \beta$, $|\alpha a \beta| \geq 2$, то по определению $G I \Rightarrow \alpha I \beta$, где либо α , либо β непусто, и, следовательно, для $L(G)$ выполнен критерий бесконечности КС-языка (описанный вслед за доказательством теоремы 7.4).

Пример 7.8. Пусть языки L_S и L_T из примера 7.7 заданы соответственно грамматиками G_S с системой правил $R_S = \{I_1 \rightarrow a \mid (A + A), A \rightarrow a \mid A + A\}$, и G_T с системой правил $R_T = \{I_2 \rightarrow a \mid I_2 a\}$. Подставим G_T в G_S вместо a . Поскольку $a \in V_T$ (т. е. нарушено условие 1), то заменим в G_S a на B , после чего получаем искомую грамматику G_{ST} с начальным символом I_1 и системой правил $\{I_1 \rightarrow B \mid (A + A), A \rightarrow B \mid A + A, B \rightarrow I_2, I \rightarrow a \mid I_2 a\}$, которая порождает язык $L_{ST} = L_S(a \rightarrow L_T)$ из примера 7.7.

Устраним теперь из G_{ST} цепное правило $B \rightarrow I_2$ и введем в язык L_{ST} вторую переменную b (терминальный символ), добавив правила $I_2 \rightarrow b \mid I_2 b$. Получим грамматику G'_{ST} с правилами $\{I_1 \rightarrow I_2 \mid (A + A), A \rightarrow I_2 \mid A + A, I_2 \rightarrow a \mid b \mid I_2 a \mid I_2 b\}$, порождающую язык \tilde{L}_{ST} арифметических многочленов (без вычитания и деления) над двумя переменными. Зацикливание грамматики \tilde{G}_{ST} по символу a дает язык $[\tilde{L}_{ST}]_a^*$ выражений произвольной глубины над переменной b . Его грамматика, полученная зацикливанием \tilde{G}_{ST} , имеет следующие правила (приведенное здесь переименование a в C необязательно

и сделано только для того, чтобы сохранить традиционные обозначения нетерминалов):

$$I_1 \rightarrow I_2 | (A + A); I_2 \rightarrow C | b | I_2 C | I_2 b; A \rightarrow I_2 | A + A; \\ C \rightarrow I_1.$$

Блочные грамматики и сети из языков. Идея использования подстановки языка в правилах грамматики вместо подстановки конечной цепочки, как это делается в правилах обычной КС-грамматики, приводит к понятию *блочной грамматики*, которая определяется как пятерка объектов $G = \langle \mathcal{A}, \mathcal{X}, \mathcal{L}, R, L^0 \rangle$, где $\mathcal{A} = \{A_1, \dots, A_m\}$ — система конечных (возможно, пересекающихся) алфавитов; $\mathcal{X} = \{X_1, \dots, X_m\}$ — система конечных алфавитов, таких, что $X_1 \subseteq A_1, \dots, X_m \subseteq A_m$ и для любых i, j $X_i \cap (A_j \setminus X_j) = \emptyset$; $\mathcal{L} = \{L_1, \dots, L_m\}$ — конечное множество языков в алфавитах A_1, \dots, A_m соответственно; R — конечное множество блочных правил вида $x \rightarrow L_j$, где $x \in X$;

$$X = \bigcup_{k=1}^m X_k;$$

$L_j \in \mathcal{L}; L^0 \in \mathcal{L}$ — начальный язык. Алфавит X_i называется входным алфавитом языка L_i ($i = 1, \dots, m$); X_i — это множество терминальных символов L_i , вместо которых производятся подстановки в L_i , указанные блочными правилами из R . Алфавит $B_i = A_i \setminus X_i$ называется внешним алфавитом G ; алфавит X — внутренним алфавитом G , алфавит

$$B = \bigcup_{i=1}^m B_i$$

— терминальным алфавитом G .

Понятие вывода в блочной грамматике (блочного вывода) аналогично понятию вывода в обычной грамматике с той лишь разницей, что роль начального символа здесь играет начальный язык L^0 (т. е. вывод может начинаться с любого слова из L^0), а применение правила $x \rightarrow L_j$ означает подстановку вместо некоторого вхождения x любого слова из L_j . Таким образом, длина блочного вывода не учитывает сложности вывода «внутри» языков L_1, \dots, L_m . Язык $L(G)$, порождаемый блочной грамматикой G , — это множество всех слов в терминальном алфавите G , выводимых в G из L^0 . Языки $L_i \in \mathcal{L}$ называются блоками языка L .

При блочном подходе к описанию языков с помощью подстановок одних языков в другие понятие нетерминального символа становится относительным: символы из X — нетерминальные для грамматики G , однако они же являются терминальными для языков L_i , подстановки которых друг в друга порождают $L(G)$.

Если все L_i — КС-языки и для каждого L_i зафиксирована порождающая его КС-грамматика $G_i = \langle V_i, W_i, R_i, I_i \rangle$, то обычная КС-грамматика $\tilde{G} = \langle \tilde{V}, \tilde{W}, \tilde{R}, \tilde{I} \rangle$, порождающая $L(G)$, определяется следующим образом: 1) $\tilde{V} = B$; 2) в системе нетерминальных алфавитов W_i производим переименования таким образом, чтобы они не пересекались. Получаем систему алфавитов \tilde{W}_i , после чего полагаем

$$\tilde{W} = \left(\bigcup_{i=1}^m \tilde{W}_i \right) \cup X;$$

3) в правилах R_i символы из W_i заменяют символами из \tilde{W}_i ; полученную систему правил обозначим R'_i . Каждому блочному правилу $x \rightarrow L_j$ из R ставим в соответствие обычное правило $x \rightarrow I_j$; множество таких правил обозначим R' ; полагаем

$$\tilde{R} = \left(\bigcup_{i=1}^m \tilde{R}_i \right) \cup R'_i;$$

4) \tilde{I} совпадает с начальным символом грамматики G^0 начального языка. Доказательство того, что $L(\tilde{G}) = L(G)$, предоставляем читателю. Если же все блоки L_i — конечные языки, то понятие приведенной блочной грамматики определяется так же, как аналогичное понятие для обычной КС-грамматики, с той разницей, что достижимыми и производящими здесь называются не нетерминальные символы, а языки-блоки L_i .

Блочная грамматика может быть представлена сетью из языков — ориентированным графом, в котором вершины соответствуют языкам-блокам, а ребра — блочным правилам. Такие сети удобно интерпретировать как схемы (см. ниже § 8.3), их вершины v_i — как элементы, которым приписаны языки $L(v_i)$, символы входного алфавита языка $L(v_i)$ — как входы элемента v_i , блочное правило $x \rightarrow L_j$ — как соединение в схеме, ведущее от

выхода элемента, которому приписан язык L_j , к элементу v_k , язык $L(v_k)$ которого содержит x в своем входном алфавите. При такой ориентации ребер (от L_j к x) начальный язык L^0 оказывается приписаным выходному элементу сети. Формально *сеть из языков (Л-сеть)* — это пятерка объектов $\lambda = \langle \mathcal{A}, \mathcal{X}, \mathcal{L}, \gamma, v^0 \rangle$, где $\{\mathcal{A} = \{A_1, \dots, A_m\}$ — система конечных алфавитов; $\mathcal{X} = \{X_1, \dots, X_m\}$ — система конечных алфавитов, таких, что $X_1 \subseteq A_1, \dots, X_m \subseteq A_m$, и для любых i, j $X_i \cap (A_i \setminus X_j) = \emptyset$; $\mathcal{L} = \{L_1, \dots, L_m\}$ — конечное множество языков в алфавитах A_1, \dots, A_m соответственно; γ — связный ориентированный граф с множеством вершин $V = \{v_1, \dots, v_n\}$; $v^0 \in V$ — выделенная вершина, называемая выходом сети. Каждой вершине v_i графа γ приписан язык $L(v_i) \in \mathcal{L}$. Каждое ребро, входящее в v_i , помечено символом из $X(v_i)$ — входного алфавита $L(v_i)$, причем кратные ребра имеют различные пометки. Названия алфавитов A, B, X — те же, что и в определении блочной грамматики. Записи вида $A(G), B(\lambda), X(v_i)$ обозначают принадлежность алфавитов объектам, указанным в скобках.

Переход от блочной грамматики $G = \langle \mathcal{A}, \mathcal{X}, \mathcal{L}, R, L^0 \rangle$ к представляющей ее Л-сети $\lambda(G)$ определяется следующим образом. Если $\mathcal{L} = \{L_1, \dots, L_m\}$, то $\lambda(G)$ содержит m вершин v_1, \dots, v_m ; $L(v_i) = L_i$, $i = 1, \dots, m$, а выходом является вершина v^0 , для которой $L(v^0) = L^0$. Для каждого правила $x \rightarrow L_j$ из вершины v_i проводятся ребра с пометкой x ко всем вершинам v_j , таким, что терминальный алфавит $L(v_j)$ содержит x .

Обратный переход — от сети λ к блочной грамматике $G(\lambda) = \langle \mathcal{A}, \mathcal{X}, \mathcal{L}, R, L^0 \rangle$ столь же прост с той разницей, что для непересечения входных алфавитов может понадобиться их переименование: $\mathcal{L}' = \{L'_1, \dots, L'_m\}$ — это множество языков L_1, \dots, L_m сети λ , в которых, возможно, переименованы входные алфавиты так, чтобы они не пересекались. Ребру в λ , ведущему от вершины v_j ко входу x , ставится в соответствие блочное правило $x' \rightarrow L'(v_j)$, где x' — либо x , либо символ, заменивший x при переименовании входных алфавитов; R является объединением всех блочных правил.

Л-сети λ_1 и λ_2 называются слабо эквивалентными, если они представляют один и тот же язык: $L(G(\lambda_1)) = L(G(\lambda_2))$,

и сильно эквивалентными, если блочные грамматики $G(\lambda_1)$ и $G(\lambda_2)$, полученные в результате приведения $G(\lambda_1)$ и $G(\lambda_2)$, совпадают с точностью до переименования внутренних алфавитов.

Сети λ_1 и $\lambda(G(\lambda_1))$ могут и не совпадать, однако они всегда сильно эквивалентны. В частности, при переходе от λ к $G(\lambda)$ из-за переименования может произойти увеличение мощности внутреннего алфавита; при обратном переходе эта мощность не меняется.

Если в блочной грамматике G все языки-блоки конечны: $L_1 = \{\alpha_1, \dots, \alpha_m\}$, то каждое блочное правило $x \rightarrow L_i$ эквивалентно множеству обычных КС-правил $x \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_m$, и обычная КС-грамматика, эквивалентная G , получается объединением всех таких правил. Если же все L_i — одноэлементные языки, то блочные правила G — это обычные КС-правила, а сама G — обычная КС-грамматика. В этом случае $\lambda(G)$ является графическим представлением обычной КС-грамматики.

Пример 7.9. а. КС-грамматика G_S , полученная из примера 7.8 заменой a на c , представляется Л-сетью на рис. 7.4.

б. Л-сетям λ_1 и λ_2 (рис. 7.5) соответствует одна и та же блочная грамматика с начальным языком L_1 и системой правил $R = \{a \rightarrow L_2, a \rightarrow L_3, b \rightarrow L_4, c \rightarrow L_5\}$; следовательно, λ_1 и λ_2 сильно эквивалентны. Поскольку входные символы L_2 и L_3 совпадают, то вход L_3 переименован в c . Если не делать переименования, то грамматика $G(\lambda_1)$ имела бы правило $b \rightarrow L_5$, тогда как соответствующее ребро из L_5 ко входу b языка L_2 в λ_1 отсутствует.

в. В Л-сети λ_3 на рис. 7.6 блоками являются языки L_S и L_T из примера 7.7, а также язык L_I идентификаторов

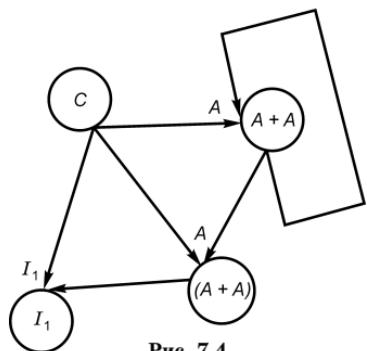


Рис. 7.4

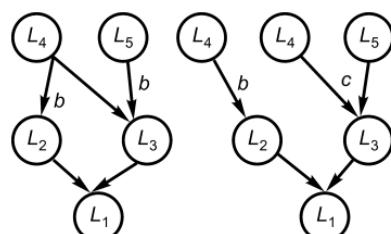


Рис. 7.5

(например, из примера 7.3, а). Язык, представляемый сетью λ_3 , — это язык арифметических выражений (без вычитания и деления) над идентификаторами из L_I . Сеть λ_4 на рис. 7.6 представляет тот же язык, поэтому λ_3 и λ_4 слабо эквивалентны; однако λ_4 имеет ребро из L_I в L_S ,

которое дает еще одно блочное правило в $G(\lambda_4)$, поэтому λ_3 и λ_4 не являются сильно эквивалентными.

Для блочных грамматик и Л-сетей также можно определить операции подстановки и зацикливания. Для Л-сетей λ_1 и λ_2 подстановка λ_2 в λ_1 вместо входа a означает введение ребер от выхода λ_2 ко всем вершинам λ_1 , алфавиты которых содержат a ; зацикливание по a означает введение ребер от выхода λ ко всем вершинам λ , алфавиты которых содержат a .

Блочные грамматики и сети из языков не порождают новых классов языков, однако являются удобным метаязыком для описания операций над языками, для описания структур сложных языков, выделения стандартных языковых компонент в новых языках и т. п.

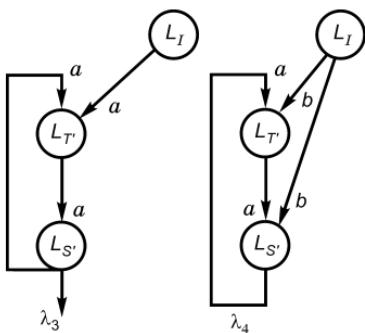


Рис. 7.6

7.3. О СЕМАНТИКЕ ФОРМАЛЬНЫХ ЯЗЫКОВ

До сих пор речь шла о синтаксических свойствах языков. Однако, как указывалось в начале главы, главное назначение любого языка — естественного или искусственно го — быть средством общения, т. е. передавать некоторое содержание, смысл. Множества смыслов, т. е. семантика языков, весьма разнообразны. Для естественных языков их точное определение встречает значительные трудности — и это обстоятельство является главным препятствием на пути машинного перевода. Для искусственных языков множества смыслов определены, как правило, точно: это, например, ходы и позиции для языка шахматной нотации, арифметические функции для языка арифметических формул, программы для языков программирования. Формально любое

множество S можно объявить семантикой, т. е. множеством смыслов (значений) данного языка L , если задать интерпретирующее отображение φ языка L в S ; тогда для цепочки $\alpha \in L$ $\varphi(\alpha)$ будет ее смыслом. Элементы S , являющиеся значениями цепочек L , могут иметь фиксированное символьное воплощение, например, быть цепочками некоторого другого, «понятного» языка (скажем, для человека его родной язык является семантикой любого иностранного языка); но это необязательно: например, говоря об арифметических функциях как о семантике арифметических формул, мы не имеем в виду их конкретное представление.

Итак, чтобы задать семантику языка L , необходимо определить множество смыслов S и интерпретирующее отображение φ языка L в S . Важным достоинством языков, описываемых КС-грамматиками, является возможность задавать отображение φ с помощью деревьев вывода. Краткимзнакомством со средствами такого задания мы завершим изучение формальных языков.

В качестве примера рассмотрим выражения, порождаемые грамматикой из примера 7.5. Их естественная интерпретация — последовательность арифметических действий над переменными a, b, c (точнее, над числами, подставляемыми вместо этих переменных, например, над числами, лежащими в ячейках памяти a, b, c). Вычисления производятся «из глубины формулы», т. е. начинаются в самых глубоких скобках (подформулах); каждая операция может быть выполнена лишь тогда, когда уже вычислены подформулы, являющиеся ее операндами. Структура формулы описывается ее деревом вывода: самым глубоким скобкам соответствуют элементарные поддеревья, все вершины которых концептивные; вершине v , которой синтаксически соответствует правило ρ_i : $A \rightarrow \alpha \circ \beta$ (\circ — знак арифметической операции), семантически соответствует операция, обозначаемая знаком \circ ; число потомков v_j , равное числу нетерминальных символов в ρ_i , соответствует числу operandов \circ ; операция в вершине v может быть выполнена только тогда, когда выполнены все вычисления в поддеревьях, подвешенных к v . Поэтому вычисление формулы можно представить как процесс, идущий по ее дереву вывода снизу вверх — от концевых вершин к корню. Результаты вычислений передаются наверх, в следующую вершину; операция, выполняемая в вершине, однозначно определяется правилом ρ_i , породившим эту вершину. Таким образом, процесс вычисления любого арифметического выражения является последовательностью операций, взятых из фиксированного для данной грамматики множества; само множество операций определяется множеством правил

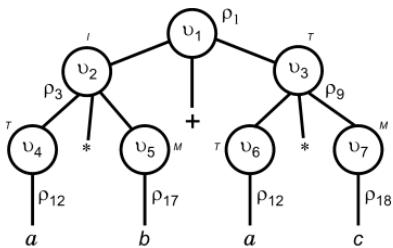


Рис. 7.7

того, каждая вершина помечена соответствующим ей правилом ρ_i грамматики; индекс i — это номер правила в порядке его упоминания в примере 7.5.

Каждому правилу ρ_i поставим в соответствие функцию g_i , число аргументов которой равно числу нетерминальных символов в правой части. Для правил, использованных в дереве рис. 7.7, эти функции таковы:

$$\begin{aligned}\rho_1 : I &\rightarrow I + T; g_1(x_1, x_2) = x_1 + x_2; \\ \rho_3 : I &\rightarrow T * M; g_3(x_1, x_2) = x_1 * x_2; \\ \rho_9 : T &\rightarrow T * M; g_9(x_1, x_2) = x_1 * x_2; \\ \rho_{12} : T &\rightarrow a; g_{12} = a,\end{aligned}$$

кроме того, $g_{17} = b$, $g_{18} = c$.

Следует иметь в виду, что знаки $+$ и $*$ в ρ_i и g_i имеют различное содержание. В синтаксических правилах ρ_i — это абстрактные символы, в функциях g_i — это осмыслиенные знаки знакомых нам арифметических операций, связанные с определенными действиями над числами x_1 и x_2 .

Пусть вершине v_j соответствует правило ρ_i . В нашем примере v_j имеет не более двух потомков; обозначим через v_{j1} левого потомка, а через v_{j2} — правого потомка v_j . Поставим в соответствие вершине v_j функцию-признак $h(v_j)$, определяемый следующим образом: $h(v_j) = g_i$, если ρ_i — терминальное правило ($i = 12, 17, 18$), $h(v_j) = g_i(h(v_{j1}), h(v_{j2}))$, если ρ_i — нетерминальное правило.

Нетрудно убедиться, что формально определенный таким образом вычислительный процесс для нашего примера осуществляет вычисление элементарных арифметических операций, соответствующих вершинам, и передачу результатов от листьев к корню, т. е. работает так, как описано выше. Однако для выражений, содержащих деление, одной функции на правило недостаточно. Дело в том, что деление на 0 недопустимо; поэтому, помимо арифметических операций, необходимо проверять, не равен ли делитель нулю, и если да, то передавать наверх, к корню, сообщение о бессмыслиности всего выражения в целом. Это можно сделать, поставив в соответствие правилу ρ_i , кроме арифметической функции g_i , еще и преди-

граммматики, а их последовательность — структурой конкретного дерева вывода.

Опишем этот процесс более формально на примере дерева вывода для выражения $a * b + a * c$ (рис. 7.7). На этом рисунке нетерминальные вершины выделены кружками и обозначены v_1, \dots, v_7 . Кроме

кат корректности P_i , которой проверяет, определены ли вычисления для потомков вершины v_j (если ρ_i — правило с делением, то P_i , кроме того, проверяет, отличен ли от нуля результат вычисления в v_{j2}), и в случае положительного ответа разрешает вычисление в v_j . В этом случае каждой вершине v_j с правилом ρ_i ставится в соответствие вектор признаков $h(v_j) = (h_1, h_2)$, где h_1 — признак корректности, а h_2 — результат вычисления арифметического выражения, для которого v_j является корнем его дерева вывода (этот результат не определен, если h_1 ложен). Значением всего выражения является результат вычисления в корне дерева. Точную формализацию описанного процесса предоставляем читателю.

В общем случае размерность вектора признаков в вершине может быть любой; кроме того, сами признаки могут вычисляться как снизу вверх (в этом случае они называются синтезируемыми), так и сверху вниз (в этом случае они называются наследуемыми).

Очевидно, что соответствующая грамматика однозначна. Различные варианты формализации вычисления семантики цепочек языка с помощью системы признаков в вершинах дерева, называемые атрибутными грамматиками (или атрибутными переводами, если значением цепочки языка является цепочка другого языка), изложены в [1].

Методы интерпретации текстов, основанные на синтаксически управляемых вычислениях, т. е. на вычислениях, управляемых деревьями вывода, оказались весьма эффективными как в конкретных приложениях при разработке компиляторов для языков программирования, так и для понимания существа проблемы перевода в искусственных и естественных языках. По-видимому, именно этим обстоятельством (наряду с удобством метаязыка Бэкуса и возможностями его расширения на основе операций над КС-языками, описанных выше) объясняется широко распространенная тенденция представлять язык как КС-язык, даже если он и не вполне точно описывается КС-грамматикой. Языки программирования, как правило, содержат конструкции, которые не могут быть точно formalизованы в терминах КС-правил. В этих случаях КС-правила сопровождаются различными ограничениями, соблюдение которых является дополнительным условием принадлежности цепочки языку. Использование деревьев вывода для интерпретации текстов объясняет также тот факт, что проблема синтаксического анализа — одна из важнейших в прикладной лингвистике — заключается не просто в распознавании принадлежности текста языку, а в построении синтаксической структуры текста, соответствующей данной грамматике, что для КС-языков равносильно построению дерева вывода.

АВТОМАТЫ

8.1. ОСНОВНЫЕ ПОНЯТИЯ

Определения. Конечным автоматом (в дальнейшем — просто автоматом) называется система $S = \{A, Q, V, \delta, \lambda\}$, в которой $A = \{a_1, \dots, a_m\}$, $Q = \{q_1, \dots, q_n\}$, $V = \{v_1, \dots, v_k\}$ — конечные множества (алфавиты), а $\delta: Q \times A \rightarrow Q$ и $\lambda: Q \times A \rightarrow V$ — функции, определенные на этих множествах. A называется входным алфавитом, V — выходным алфавитом, Q — алфавитом состояний, δ — функцией переходов, λ — функцией выходов. Если, кроме того, в автомате S выделено одно состояние q , называемое начальным (обычно будет считаться, что это q_1), то полученный автомат называется *ициональным* и обозначается (S, q) ; таким образом, по неинициальному автомату с n состояниями можно n различными способами определить инициальный автомат.

Поскольку функции δ и λ определены на конечных множествах, их можно задавать таблицами. Обычно две таблицы сводятся в одну таблицу $\delta \times \lambda: Q \times A \rightarrow Q \times V$, называемую таблицей переходов автомата, или просто автоматной таблицей.

Пример 8.1. Таблица 8.1 задает функции переходов и выходов для автомата с алфавитами $A = \{a_1, a_2, a_3\}$, $Q = \{q_1, q_2, q_3, q_4\}$, $V = \{v_1, v_2\}$.

Еще один распространенный и наглядный способ задания автомата — ориентированный мультиграф, называемый *графом переходов* или *диаграммой переходов*. Вершины графа соответствуют состояниям; если $\delta(q_i, a_j) = q_k$ и $\lambda(q_i, a_j) = v_l$, то из q_i в q_k ведет ребро, на котором написаны a_j и v_l . Граф переходов для табл. 8.1 изобра-

жен на рис. 8.1. Кратные ребра не обязательны; например, два ребра из q_2 в q_1 можно заменить одним, на котором будут написаны обе пары $a_3 | v_1$ и $a_2 | v_1$. Для любого графа переходов в каждой вершине q_i выполнены следующие условия, которые называются условиями корректности: 1) для любой входной буквы a_j имеется ребро, выходящее из q_i , на котором написано a_j (условие полноты); 2) любая буква a_j встречается только на одном ребре, выходящем из q_i (условие непротиворечивости или детерминированности).

Для данного автомата S его функции δ_S и λ_S могут быть определены не только на множестве A всех входных букв, но и на множестве A^* всех входных слов. Для любого входного слова $\alpha = a_{j_1}a_{j_2}\dots a_{j_k}$:

$$\begin{aligned}\delta(q_i, a_{j_1}a_{j_2}\dots a_{j_k}) = \\ = \delta(\delta(\dots\delta(q_i, a_{j_1}), a_{j_2}), \dots, a_{j_{k-1}}), a_{j_k}).\end{aligned}$$

Таблица 8.1

	a_1	a_2	a_3
q_1	q_3, v_1	q_3, v_2	q_2, v_1
q_2	q_4, v_1	q_1, v_1	q_1, v_1
q_3	q_2, v_1	q_3, v_1	q_3, v_2
q_4	q_4, v_1	q_2, v_1	q_1, v_2

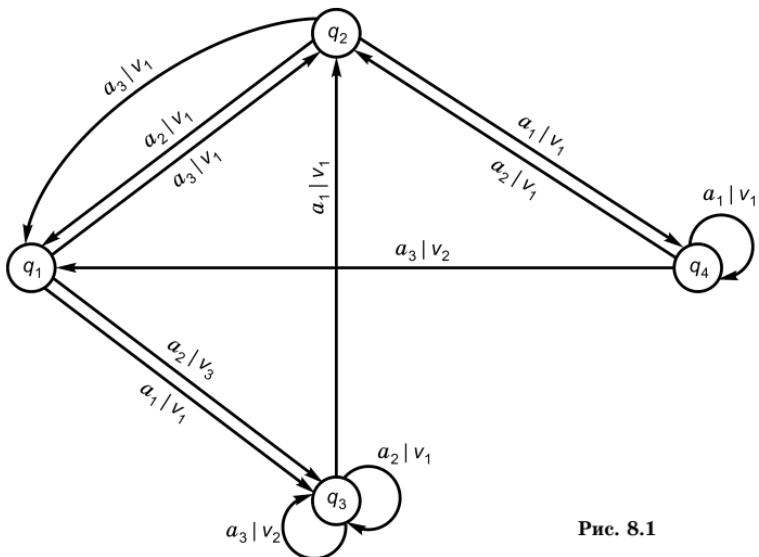


Рис. 8.1

Это традиционное определение «с многоточиями»; намного точнее и лучше читается индуктивное определение, к которому читатель, надеемся, уже привык:

- $\delta(q_i, a_j)$ задается автоматной таблицей S ;
- для любого слова $\alpha \in A^*$ и любой буквы a_j

$$\delta(q_i, \alpha a_j) = \delta(\delta(q_i, \alpha), a_j). \quad (8.1)$$

С помощью расширенной функции δ определяется (также индуктивно) расширенная функция λ :

$$\lambda(q_i, \alpha a_j) = \lambda(\delta(q_i, \alpha), a_j). \quad (8.2)$$

Зафиксируем в автомате S начальное состояние q_1 и каждому входному слову $\alpha = a_{j_1}a_{j_2}\dots a_{j_k}$ поставим в соответствие слово ω в выходном алфавите:

$$\omega = \lambda(q_1, a_{j_1})\lambda(q_1, a_{j_1}a_{j_2})\dots\lambda(q_1, a_{j_1}\dots a_{j_k}). \quad (8.3a)$$

Это соответствие, отображающее входные слова в выходные слова, называется *автоматным отображением*, а также *автоматным (или ограниченно детерминированным) оператором*, реализуемым автоматом (S, q_1) . Иногда будем говорить кратко — оператор (S, q_1) или оператор S (если автомат S — инициальный). Если результатом применения оператора к слову α является выходное слово ω , то это будем обозначать соответственно $S(q_1, \alpha) = \omega$ или $S(\alpha) = \omega$. Число букв в слове α , как обычно, называется *длиной* α и обозначается $|\alpha|$ или $l(\alpha)$. Автоматное отображение также удобно определить индуктивно:

$$\left. \begin{aligned} S(q_i, a_j) &= \lambda(q_i, a_j); \\ S(q_i, \alpha a_j) &= S(q_i, \alpha)\lambda(\delta(q_i, \alpha), a_j). \end{aligned} \right\} \quad (8.3b)$$

Автоматное отображение обладает двумя свойствами, которые следуют непосредственно из (8.3a), (8.3b): 1) слова α и $\omega = S(\alpha)$ имеют одинаковую длину: $|\alpha| = |\omega|$ (свойство сохранения длины); 2) если $\alpha = \alpha_1\alpha_2$ и $S(\alpha_1\alpha_2) = \omega_1\omega_2$, где $|\alpha_1| = |\omega_1|$, то $S(\alpha_1) = \omega_1$; иначе говоря, образ отрезка длины i равен отрезку образа той же длины. Свойство 2 отражает тот факт, что автоматные операторы — это *операторы без предвосхищения* [33], т. е. операторы, которые, перерабатывая слово слева направо, «не заглядывают вперед»: i -я буква выходного слова зависит только от

первых i букв выходного слова. Пример оператора с предвосхищением — оператор, который слову $\alpha = a_{i_1} \dots a_{i_k}$ ставит в соответствие его отражение, т. е. слово $a_{i_k} \dots a_{i_1}$; первая буква выходного слова равна здесь последней букве входного слова.

Указанные два свойства были бы достаточными условиями автоматности отображения, если бы речь шла о бесконечных автоматах, т. е. автоматах с бесконечным Q . Для конечной автоматности этих условий недостаточно: в дальнейшем будут описаны отображения, которые удовлетворяют условиям 1 и 2, но не реализуются в конечном автомате.

Введенные определения (8.1)–(8.3) наглядно интерпретируются на графе переходов. Если зафиксирована вершина q_i , то всякое слово $\alpha = a_{i_1} \dots a_{i_k}$ однозначно определяет путь длины k из этой вершины [обозначим его (q_i, α)], на k ребрах которого написаны соответственно буквы $a_{i_1} \dots a_{i_k}$. Поэтому $\delta(q_i, \alpha)$ — это последняя вершина пути (q_i, α) , $\lambda(q_i, \alpha)$ — выходная буква, написанная на последнем ребре пути (q_i, α) , а отображение $S(q_i, \alpha)$ — слово, образованное k выходными буквами, написанными на k ребрах этого пути.

Пример 8.2. Для автомата S из примера 8.1 $\delta(q_2, a_3a_2) = \delta(q_2, a_3a_1) = q_3$; $\delta(q_2, a_3a_1a_1) = q_2$; $\lambda(q_2, a_3a_2) = v_2$; $\lambda(q_2, a_3a_1) = v_1$; $\lambda(q_2, a_3a_1a_1) = v_1$. Заметим, что $\delta(q_2, a_3a_1) = \delta(q_2, a_3a_2)$, но $\lambda(q_2, a_3a_1) \neq \lambda(q_2, a_3a_2)$. Далее $S(q_2, a_3a_2) = v_1v_2$; $S(q_2, a_3a_2a_1) = v_1v_2v_1$, что иллюстрирует свойство 2 автоматного отображения.

Состояние q_j называется *достижимым* из состояния q_i , если существует входное слово α , такое, что $\delta(q_i, \alpha) = q_j$. Автомат S называется *сильно связным*, если из любого его состояния достижимо любое другое состояние.

Автомат называется *автономным*, если его входной алфавит состоит из одной буквы: $A = \{a\}$. Все входные слова автономного автомата имеют вид $aa\dots a$.

Теорема 8.1. Любое достаточно длинное выходное слово автономного автомата с n состояниями является периодическим (возможно, с предпериодом), причем длины периода и предпериода не превосходят n ; иначе говоря, оно имеет вид $\sigma\omega\dots\omega\omega_1$, где σ — предпериод; ω_1 — начальный отрезок ω ; при этом $0 \leq |\sigma| \leq n$; $1 \leq |\omega| \leq n$.

Действительно, так как в графе автономного автомата из каждой вершины выходит только одно ребро, то его сильно связные подграфы могут быть только простыми циклами, из которых нет выходящих ребер. Поэтому в компоненте связности может быть только один цикл; остальные подграфы компоненты — это деревья, подвешенные к циклу и ориентированные в его сторону. \square

Пример 8.3. Граф автомата, заданного в табл. 8.2, изображен на рис. 8.2 (входные буквы на ребрах опущены; выходной алфавит $V = \{0, 1, 2\}$).

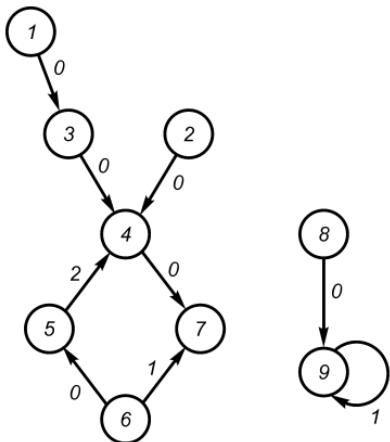


Рис. 8.2

Таблица 8.2

q	a
1	3,0
2	4,0
3	4,0
4	7,0
5	4,2
6	5,0
7	6,1
8	9,0
9	9,1

Здесь и в дальнейшем символы состояний для кратности обозначений будут заменяться их индексами.

Изоморфизм и эквивалентность автоматов. Пусть $S = (A_S, Q_S, V_S, \delta_S, \lambda_S)$ и $T = (A_T, Q_T, V_T, \delta_T, \lambda_T)$ — два автомата. Тройка отображений $f: A_S \rightarrow A_T$, $g: Q_S \rightarrow Q_T$, $h: V_S \rightarrow V_T$ называется *гомоморфизмом* автомата S в автомат T , если для любых $a \in A_S$, $q \in Q_S$, $v \in V_S$ выполнены условия [ср. (2.1)]:

$$\begin{aligned} \delta_T(g(q)), f(a) &= g(\delta_S(q, a)); \\ \lambda_T(g(q), f(a)) &= h(\lambda_S(q, a)). \end{aligned} \quad (8.4)$$

Автомат T называется *гомоморфным* автомату S . Если все три отображения сюръективны, то эта тройка называется *гомоморфизмом* S на T . Если, кроме того, эти три отображения взаимно однозначны, то они называются *изоморфизмом* S на T ; автоматы, для которых существу-

ет изоморфизм, называются изоморфными. Ясно, что мощности соответствующих алфавитов изоморфных автоматов должны быть одинаковыми.

Понятие изоморфизма имеет для автоматов тот же смысл, что и для алгебр, рассматривавшихся в гл. 2: автоматы S и T изоморфны, если входы, выходы и состояния S можно переименовать так, что таблица переходов автомата S превратится в таблицу переходов автомата T . Изоморфизм графов переходов (без учета букв, написанных на ребрах) является необходимым, но недостаточным условием изоморфизма соответствующих автоматов — см. далее пример 8.4, б. При гомоморфизме, помимо переименования, происходит еще и «склеивание» (например, нескольких состояний автомата S в одно состояние автомата T).

Пример 8.4. а. Возьмем в качестве S автономный автомат из примера 8.3, а в качестве T — автономный автомат, график которого приведен на рис. 8.3. Существует гомоморфизм S в T . Соответствующая тройка отображений такова: f тривиально, так как оба входных алфавита

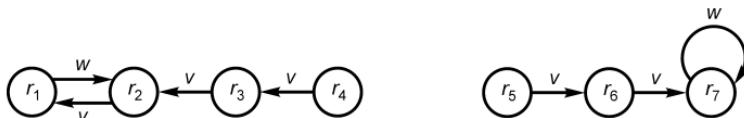


Рис. 8.3

состоят из одной буквы, g и h зададим списками (для краткости здесь и в аналогичных случаях вместо $q_i \rightarrow r_j$ будем писать $i \rightarrow j$); $g = \{1 \rightarrow 4, 2 \rightarrow 3, 3 \rightarrow 3, 4 \rightarrow 2, 5 \rightarrow 1, 6 \rightarrow 2, 7 \rightarrow 1, 8 \rightarrow 6, 9 \rightarrow 7\}$, $h = \{0 \rightarrow v, 1 \rightarrow w, 2 \rightarrow w\}$. Никакое состояние S не отобразилось в r_5 ; заметим, что r_5 недостижимо из других состояний. Это — общее правило: если состояние T не входит в область значений g при гомоморфизме, то оно должно быть недостижимым для любого состояния из этой области, иначе нарушится условие (8.4). Если из автомата T состояние r_5 вместе с инцидентным ему ребром удалить, то получим новый автомат T' ; описанная тройка отображений является гомоморфизмом S в T и S на T' . Как показывает этот пример, число состояний и выходных букв при гомоморфизме может не сохраняться. Для неавтономных автоматов это же относится и ко входному алфавиту.

б. В графе автомата T рис. 8.3 поменяем местами буквы на двух ребрах: на ребре (r_1, r_2) напишем v , а на ребре (r_2, r_1) напишем w . Получим автомат T'' , граф которого изоморден графу T ; однако сам автомат T'' не изоморден T . Действительно, при изоморфизме графов вершина r_4 автомата T изоморфна вершине r_4 автомата T'' ; однако $T(r_4, aaa) = vvv$, $T''(r_4, aaa) = vvw$, и при любом переименовании выходов в выходном слове $T(r_4, aaa)$ все три буквы будут одинаковыми, а в $T''(r_4, aaa)$ — нет.

Пусть S и T — два автомата с одинаковыми входным и выходным алфавитами. Состояние q автомата S и состояние r автомата T называются *неотличимыми*, если для любого входного слова α $S(q, \alpha) = T(r, \alpha)$. В частности, если $T = S$, то речь идет о неотличимых состояниях одного и того же автомата S . Неотличимость состояний q_i и q_j автомата S означает, что инициальные автоматы (S, q_i) и (S, q_j) реализуют одно и то же автоматное отображение.

Автоматы S и T называются *неотличимыми*, если для любого состояния q автомата S найдется неотличимое от него состояние r автомата T и, наоборот, для любого r из T найдется неотличимое от него q из S . Неотличимость автоматов означает, что любое автоматное отображение, реализуемое одним из них, может быть реализовано другим; иначе говоря, их возможности по реализации преобразований входной информации в выходную совпадают. Отношение неотличимости между состояниями и автоматами, как нетрудно показать, рефлексивно, симметрично и транзитивно и, следовательно, является отношением эквивалентности (см. § 1.4). Обычно неотличимость так и называется *эквивалентностью*. Терминологически это не очень корректно и удобно: название свойства отношений используется как имя конкретного отношения. Однако в теории автоматов этот термин стал общепринятым, поэтому будем говорить об эквивалентных состояниях или эквивалентных автоматах, имея в виду отношение неотличимости.

Переход от автомата S к эквивалентному автомatu называется *эквивалентным преобразованием* автомата S . Можно ставить различные задачи о поиске автоматов, эквивалентных данному и обладающих заданными свой-

ствами. Наиболее изученной среди таких задач является задача о минимизации числа состояний автомата, или, короче, о минимизации автомата: среди автоматов, эквивалентных S , найти автомат с наименьшим числом состояний — минимальный автомат.

Теорема 8.2. Для любого автомата S существует минимальный автомат S_0 , единственный с точностью до изоморфизма; если множество состояний S разбивается на l классов эквивалентности ($l \leq n$): $C_1 = \{q_{11}, \dots, q_{l_1}\}$, ..., $C_l = \{q_{l1}, \dots, q_{ll}\}$, то S_0 имеет l состояний.

Если q_{j1} и q_{j2} — состояния из одного класса эквивалентности C_j , то для любой входной буквы a состояния $\delta_S(q_{j1}, a)$ и $\delta_S(q_{j2}, a)$ также находятся в одном классе эквивалентности C_k . Действительно, если $\delta_S(q_{j1}, a)$ и $\delta_S(q_{j2}, a)$ не эквивалентны, то найдется слово α , такое, что $S(\delta_S(q_{j1}, a), \alpha) \neq S(\delta_S(q_{j2}, a), \alpha)$; но тогда в силу (8.1) $S(q_{j1}, a\alpha) \neq S(q_{j2}, a\alpha)$, т. е. q_{j1} и q_{j2} не эквивалентны, что противоречит предположению. Учитывая это обстоятельство, определим автомат $S_0 = (A_S, Q_{S_0}, V_S, \delta_{S_0})$ так: $Q_{S_0} = \{C_1, \dots, C_l\}$; для любого C_i и любой входной буквы a $\delta_{S_0}(C_i, a) = C_j$, где C_j — класс эквивалентности, содержащий состояние $\delta_S(q_{ir}, a)$ (q_{ir} — состояние из C_i ; ввиду отмеченного ранее обстоятельства можно взять любое $q_{ir} \in C_i$); $\lambda_{S_0}(C_i, a) = \lambda_S(q_{ir}, a)$.

Очевидно, что S_0 эквивалентен S ; попутно заметим, что S_0 по построению не имеет эквивалентных состояний. Остается показать, что, во-первых, S_0 минимален, а во-вторых, любой минимальный автомат S'_0 изоморфен S_0 . Предположим, что S_0 не минимален и имеется эквивалентный ему автомат S''_0 с меньшим числом состояний. По определению неотличимости для каждого состояния S_0 найдется эквивалентное ему состояние S''_0 ; поскольку в S''_0 меньше состояний, то какие-то два состояния S_0 эквивалентны одному состоянию S''_0 и в силу транзитивности окажутся эквивалентными между собой, что противоречит отсутствию в S_0 эквивалентных состояний. Поэтому S_0 минимален. Если же S'_0 — другой минимальный автомат, т. е. имеет то же число состояний, то S'_0 эквивалентен S_0 и различные состояния автомата S'_0 эквивалентны различным состояниям S_0 ; легко проверить, что это соответствие состояний S_0 и S'_0 является искомым изоморфизмом. \square

Теорема доказана. Однако, чтобы воспользоваться ею для нахождения минимального автомата, нужно уметь находить классы эквивалентных состояний данного автомата. Само определение неотличимости не содержит метода нахождения, так как оно предполагает перебор по бесконечному множеству входных слов. Наиболее известным алгоритмом нахождения эквивалентных состояний является алгоритм Мили, который будет описан индуктивно.

Пусть дан автомат $S = (A, Q, V, \delta, \lambda)$ с n состояниями. На каждом шаге алгоритма будем строить некоторое разбиение Q на классы, причем разбиение на следующем шаге будет получаться расщеплением некоторых классов предыдущего разбиения. (Отметим, что шаги алгоритма в данном описании вовсе не элементарны. Это, скорее, блоки.)

Шаг 1. Два состояния q и q' относим в один класс C_{1j} , если и только если для любого $a \in A$ $\lambda(q, a) = \lambda(q', a)$.

Шаг $i + 1$. Два состояния q и q' из одного класса C_{ij} относим в один класс $C_{i+1,j}$, если и только если для любого $a \in A$ $\delta(q, a)$ и $\delta(q', a)$ принадлежат одному и тому же классу C_{il} . Если $(i + 1)$ -й шаг не меняет разбиения, т. е. состояния из одного класса C_{ij} принадлежат одному классу $C_{i+1,j}$, то алгоритм заканчивается и получ

ченное разбиение является разбиением на классы эквивалентных состояний; иначе применяем шаг $i + 1$ к полученному разбиению.

Пример 8.5. Для автомата S с девятью состояниями и двумя выходными буквами, заданного табл. 8.3, алгоритм строит следующую последовательность разбиений (классы отделены точкой с запятой):

1	4	6	9;	2	3	8;	5	7
1	4	6;	9;	2	3	8;	5	7
1	4;	6;	9;	2	3	8;	5	7
1	4;	6;	9;	2	8;	3;	5	7

a

C_i	a_1	a_2	a_3
1	4,0	1,1	1,1
2	4,0	3,1	2,1
3	6,0	3,1	6,1
4	1,1	1,0	6,0
5	1,1	2,0	6,0
6	2,1	1,1	5,0

б

Таблица 8.4

q_i	a_1	a_2	a_3
1	2,0	1,1	1,1
2	1,1	1,0	5,0
3	1,1	6,0	5,0
5	6,1	1,1	3,0
6	2,0	9,1	6,1
9	5,0	9,1	5,1

Последнее разбиение является искомым; минимальный для S автомат имеет шесть состояний. Если найденные классы обозначить по порядку C_1, \dots, C_6 , то минимальный автомат описывается табл. 8.4, а, в которой снова обозначения состояний (классов C_i) заменены их индексами.

Обычно, чтобы избежать составления новой таблицы, в исходной таблице оставляют по одному представителю из каждого класса, а строки остальных состояний вычеркивают; в полученной таблице все вхождения этих остальных состояний также заменяют выбранными представителями. В нашем примере вычеркиваются строки 4, 8, 7; в клетках полученной таблицы 4 заменяется на 1, 8 на 2, 7 на 5; в результате получится табл. 8.4, б.

Представляем читателю убедиться в изоморфизме автоматов, заданных двумя табл. 8.4.

Поскольку на каждом шаге число классов увеличивается, а общее их число не превосходит n , то описанный алгоритм заканчивается не позднее чем на $(n - 1)$ -м шаге. Докажем теперь, что алгоритм действительно дает разбиение на классы эквивалентных состояний. Пусть алгоритм остановился на k -м шаге и q и q' принадлежат одному классу C_{ki} из разбиения, полученного на этом шаге. По условию остановки алгоритма для любого a $\delta(q, a)$ и $\delta(q', a)$ принадлежат одному классу C_{kj} ; следовательно, это верно для любого слова α . Но состояния q и q' одного класса заведомо принадлежат одному классу C_{1l} , поэтому $\lambda(q, \alpha) = \lambda(q', \alpha)$, откуда, учитывая (8.3), получаем, что $S(q, \alpha) = S(q', \alpha)$ для любого α , т. е. состояния эквивалентны.

Пусть теперь q и q' принадлежат разным классам C_{ri} и C_{rk} , начиная с некоторого $r \leq k$. Тогда по построению

найдется такое a , что $\delta(q, a)$ и $\delta(q', a)$ принадлежат разным классам $C_{r-1, i}, C_{r-1, j}$; для них, в свою очередь, найдется такое a'' , что $\delta(\delta(q, a), a'')$ и $\delta(\delta(q', a), a'')$ принадлежат разным классам $C_{r-2, i}, C_{r-2, j}$; продолжив по индукции, получим, что для некоторого слова α длины r $\delta(q, \alpha)$ и $\delta(q', \alpha)$ принадлежат разным классам C_{1i}, C_{1j} , а это означает, что $\lambda(q, \alpha) \neq \lambda(q', \alpha)$ и, следовательно, q и q' не эквивалентны.

Частичные автоматы и их минимизация. Автомат S называется *частичным*, или *не полностью определенным автоматом*, если хотя бы одна из его двух функций не полностью определена, т. е. для некоторых пар (состояние — вход) значения функций δ или λ не определены. В автоматной таблице неполная определенность автомата выражается в том, что некоторые ее клетки не заполнены — в них стоят прочерки. В графе частичного автомата в вершинах, где δ не определена, нарушено условие полноты. Для частичных автоматов определения (8.1)–(8.3) нуждаются в изменении. При этом будем пользоваться знаком \cong : запись $A \cong B$ означает, что A и B либо одновременно не определены, либо определены и равны.

Функция $\delta(q_i, \alpha)$:

- $\delta(q_i, a_j)$ задана таблицей автомата S ;
- если $\delta(q_i, \alpha)$ определена, то

$$\delta(q_i, \alpha a_j) \cong \delta(\delta(q_i, \alpha), a_j); \quad (8.5a)$$

в) если $\delta(q_i, \alpha)$ не определена, то $\delta(q_i, \alpha a_j)$ не определена для всех a_j .

Функция $\lambda(q_i, \alpha)$:

$$\lambda(q_i, \alpha a_j) \cong \lambda(\delta(q_i, \alpha), a_j). \quad (8.5b)$$

Автоматное отображение $S(q_i, \alpha)$:

а) $S(q_i, a_j) = \lambda(q_i, a_j)$ (если $\lambda(q_i, a_j)$ не определена, то значение $S(q_i, a_j)$ считается равным прочерку);

- если $\delta(q_i, \alpha)$ определена, то

$$S(q_i, \alpha a_j) = S(q_i, \alpha) \lambda(\delta(q_i, \alpha), a_j) \quad (8.6)$$

(если $\lambda(\delta(q_i, \alpha), a_j)$ не определена, то слово $S(q_i, \alpha a_j)$ получено из $S(q_i, \alpha)$ приписыванием справа прочерка);

е) если $\delta(q_i, \alpha)$ не определена, то и $S(q, \alpha a_j)$ не определено.

Входное слово α , для которого $S(q_i, \alpha)$ определено, называется *допустимым* для q_i .

Из этих определений видно, что функции переходов и выходов неравноправны: если δ не определена на слове α , то она не определена и на всех его продолжениях; для λ это не обязательно. Поэтому, если δ определена на α , а λ не определена на некоторых начальных отрезках α , отображение $S(q_i, \alpha)$ «определенено, но не совсем»: оно представляет собой слово, содержащее прочерки. Эта ситуация естественно интерпретируется на графе: если δ не определена на α , то путь α из состояния q_i не определен, поэтому неясно, как его продолжить. Если же путь α из q_i определен, то, идя по нему, можно прочесть и выходное слово $S(q_i, \alpha)$; ребрам пути α , на которых не написано выходных букв, соответствуют прочерки в слове $S(q_i, \alpha)$.

Понятие неотличимости для частичных автоматов также изменяется. Наиболее простым обобщением обычного понятия неотличимости является следующее. Состояние q_i автомата S и состояние r_j автомата T называются *псевдонеотличимыми* (псевдоэквивалентными), если для любого α $S(q_i, \alpha) \cong T(r_j, \alpha)$, т. е. область определения q_i и r_j одна и та же и в этой области q_i и r_j эквивалентны. Автоматы S и T псевдонеотличимы, если для любого состояния найдется псевдонеотличимое от него состояние T , и наоборот. Достоинство этого определения в том, что для полностью определенных автоматов оно совпадает с обычным; кроме того, отношение псевдонеотличимости является отношением эквивалентности. Нетрудно показать, что если прочерк в функции δ рассматривать как символ нового состояния (переходящего по любому входу только в себя), а в функции λ — как новую выходную букву, то отношение $S(q_i, \alpha) \cong T(r_j, \alpha)$ переходит в обычное отношение равенства $S(q_i, \alpha) = T(r_j, \alpha)$, и, следовательно, применение к частичному автомatu S изложенного ранее алгоритма Мили даст минимальный автомат, псевдонеотличимый от S . Недостаток этого определения в том, что оно требует довольно искусственного условия: совпадения областей определения сравниваемых состояний. Поэтому понятие псевдонеотличимости оказывается слишком слабым и не учитывает всех возможностей минимизации частичных автоматов. Поясним на

Таблица 8.5

	a_1	a_2	a_3
1	2, 0	—	3, —
2	—	1, —	3, 0
3	2, 1	1, —	3, 0

примере, о чём идет речь. Рассмотрим автомат, заданный табл. 8.5.

Псевдонеотличимых состояний здесь просто нет. Рассмотрим состояния 2 и 3 (q_2 и q_3). Область определения для q_2 , т. е. для отображения $S(q_2, \alpha)$, содержитя в области определения для q_3 ; кроме того, на всей области определения q_2 $S(q_2, \alpha) = S(q_3, \alpha)$ для любого α , так как при любой входной букве $\lambda(q_2, \alpha) \cong \lambda(q_3, \alpha)$ и $\delta(q_2, \alpha) \cong \delta(q_3, \alpha)$. Можно сказать, что q_3 «делает больше, чем q_2 » на тех словах, на которых $S(q_3, \alpha)$ определено, а $S(q_2, \alpha)$ нет. Поэтому ясно, что если в S q_2 заменить на q_3 (т. е. вычеркнуть строку 2, а переходы из других состояний в q_2 заменить на переходы в q_3), то получим автомат S' , который «делает больше, чем S ». Это выражение приводит к понятию покрытия для состояний и автоматов. Состояние q_i автомата S покрывает состояние r_j автомата T (S и T , возможно, совпадают), если для любого α из того, что $S(r_j, \alpha)$ определено, следует, что $S(q_i, \alpha)$ определено и $S(q_i, \alpha) = S(r_j, \alpha)$. Автомат S покрывает автомат T , если для любого состояния T найдется покрывающее его состояние S . В частности, состояние, строка которого не содержит прочерков, покрывает все состояния, строки которых получаются из неё заменой некоторых символов прочерками; и обратно, любое состояние q' , полученное из состояния q некоторым доопределением, т. е. заменой прочерков символами, покрывает q . В табл. 8.5 q_3 покрывает q_2 ; автомат S' с двумя состояниями, полученный заменой q_2 на q_3 , описанное ранее, покрывает исходный автомат. Отметим, что отношение покрытия — это отношение нестрогого (ввиду его рефлексивности) порядка; поэтому переход к автомату, покрывающему данный автомат, нельзя называть эквивалентным преобразованием.

Рассмотрим теперь состояния q_1 и q_2 в табл. 8.5. В отличие от пары q_2, q_3 здесь нет оснований считать одно из состояний более сильным, чем другое. Однако эта пара примечательна тем, что можно представить себе состояние, которое покрывает и q_1 , и q_2 . Таким состоянием

является состояние, например, со следующей строкой (в табл. 8.5 она получит номер 4): 2, 0; 1, -; 3, 0, которую можно назвать объединением строк 1 и 2 (можно дать точное определение объединения строк, но надеемся, что оно и так понятно). Это приводит к следующей паре определений. Состояние q_i автомата S и состояние r_j автомата T называются *совместимыми*, если существует состояние p_k (быть может, какого-то третьего автомата W), покрывающее и q_i , и r_j . Автоматы S и T *совместимы*, если существует автомат W , покрывающий S и T . Совместимости можно дать и более прямое определение: q_i и r_j совместимы, если для любого α либо одно из отображений $S(q_i, \alpha)$, $T(r_j, \alpha)$ не определено, либо выходные слова $S(q_i, \alpha)$ и $T(r_j, \alpha)$ (быть может, содержащие прочерки) непротиворечивы, т. е. не содержат на одинаковых местах различных букв (например, пара слов $\omega_1 = v_2v_5 - v_4$ и $\omega_2 = v_2 - v_1v_4$ непротиворечива, а пара слов ω_1 и $\omega_3 = -v_1 - v_4$ противоречива).

Понятия покрытия и совместимости дают общий план минимизации частичных автоматов, аналогичный описанному ранее плану минимизации полностью определенных автоматов: находим совместимые состояния и заменяем их покрывающим состоянием (например, объединением соответствующих строк). Однако в реализации этого плана для частичных автоматов есть свои особенности. Дело в том, что отношение совместимости нетранзитивно (например, в табл. 8.5 пары q_1, q_2 и q_2, q_3 совместимы, а пара q_1, q_3 — нет) и, следовательно, не является отношением эквивалентности, поэтому классы совместимости (т. е. множества попарно совместимых состояний) могут пересекаться.

Назовем систему классов совместимости C_1, \dots, C_l полной, если $C_1 \cup \dots \cup C_l = Q$, и замкнутой, если из того, что состояния q и q' находятся в одном классе C_i , следует, что состояния $\delta(q, \alpha)$ и $\delta(q', \alpha)$ также находятся в одном классе C_j всякий раз, когда $\delta(q, \alpha)$ и $\delta(q', \alpha)$ определены.

Теорема 8.3 (Полл, Ангер). Если для частичного автомата S имеется полная и замкнутая система классов совместимости C_1, \dots, C_l , то существует автомат S' , покрывающий S .

Автомат $S' = (A_S, Q_{S'}, V_S, \delta_{S'}, \lambda_{S'})$ строится так: $Q_{S'} = \{C_1, \dots, C_l\}$; для любого C_i и любой входной буквы a $\delta_{S'}(C_i, a) = C_j$, если для некоторых $q \in C_i$ $\delta_S(q, a) \in C_j$ (классы C_j не могут быть разными для разных q ввиду замкнутости системы классов), и $\delta_{S'}(C_i, a)$ не определена, если для всех $q \in C_i$ $\delta_S(q, a)$ не определена; $\lambda_{S'}(C_i, a) = v$, если для некоторых $q \in C_i$ $\lambda_S(q, a) = v$ (буквы v не могут быть разными для разных q ввиду совместности состояний q из одного класса), и $\lambda_{S'}(C_i, a)$ не определена, если для всех $q \in C_i$ $\lambda_S(q, a)$ не определена. Нетрудно видеть, что состояние C_i автомата S' покрывает все состояния из класса совместности C_i автомата S ; следовательно, ввиду полноты системы классов $\{C_i\}$ автомат S' покрывает S . \square

Эта теорема является аналогом теоремы 8.2 как по содержанию, так и по способу построения искомого автомата; в случае, когда S полностью определен, обе теоремы совпадают. Кроме того, алгоритм Мили можно использовать и для минимизации частичного автомата. Для этого нужно сначала построить различные доопределения исходного автомата (ясно, что все они будут покрывать исходный автомат), а затем минимизировать полученные полные автоматы по алгоритму Мили.

Однако на этом аналогия с полными автоматами кончается и начинаются собственные — и довольно серьезные — трудности минимизации частичных автоматов. Остановимся на них подробнее.

1. Различные доопределения частичного автомата S приводят, вообще говоря, к неэквивалентным между собой полным автоматам S_1, \dots, S_N ; соответствующие минимальные автоматы S_{10}, \dots, S_{N0} могут иметь разное число состояний и также неэквивалентны между собой; следовательно, их нельзя получить друг из друга эквивалентными преобразованиями.

Например, рассмотрим три доопределения клетки $(2, a_1)$ в табл. 8.5: $(2, 0)$, $(2, 1)$ и $(1, 1)$. В первом случае (при очевидном доопределении остальных клеток) получим автомат S_1 , где состояния 1 и 2 эквивалентны; во втором случае получим автомат S_2 , где 1 и 2 не эквивалентны, а эквивалентны 2 и 3; минимальные для них автоматы S_{10} и S_{20} имеют по два состояния, но могут оказаться неизоморфными, если для S_2 доопределить

$\delta(1, a_2) = 3$. Наконец, третье доопределение дает неминимизируемый автомат S_3 . Поэтому, во-первых, результат минимизации может сильно зависеть от выбранного доопределения, а во-вторых, этот результат является тупиковым — его нельзя улучшить эквивалентными преобразованиями и надо просто пробовать другой вариант доопределения. Число же этих вариантов очень велико: если $|Q_S| = n$, $|V_S| = k$, δ_S не определена в p клетках таблицы, а λ_S — в r клетках, то это число равно $n^p k^r$.

2. Даже перебор всех доопределений может не привести к минимальному для S автомата. Дело в том, что алгоритм Мили в любом случае даст систему непересекающихся классов совместимости — а ведь эти классы могут пересекаться! Это иллюстрируется простым, но эффектным примером Полла–Ангера. Автомат S , заданный табл. 8.6, *a*, можно доопределить двумя способами: положив $\lambda(1, a_1) = 0$ либо $\lambda(1, a_1) = 1$. Можно проверить, что при любом из этих доопределений полученный автомат не имеет эквивалентных состояний и, следовательно, не минимизируется. Однако для частичного автомата S это означает всего лишь, что он не имеет нетривиальной замкнутой системы непересекающихся классов совместимости. В то же время для S существует замкнутая система пересекающихся классов: $C_1 = \{1, 2\}$, $C_2 = \{1, 3\}$, которая по теореме 8.3 приводит к автомата S' с двумя состояниями (табл. 8.6, *b*), покрывающему S .

Этот пример говорит о том, что из-за пересечения классов совместимости число различных вариантов минимизации еще больше числа вариантов доопределения частичного автомата.

Таким образом, приходится искать дополнительные методы построения систем классов совместимости. Кратко остановимся на их существе. Всякий класс содержит попарно совместимые состояния, поэтому первая задача

<i>a</i>	a_1	a_2
1	1, –	2, 0
2	3, 0	1, 0
3	2, 1	1, 0

<i>b</i>	<i>Таблица 8.6</i>	
	a_1	a_2
C_1	$C_2, 0$	$C_1, 0$
C_2	$C_1, 1$	$C_1, 0$

заключается в нахождении всех совместимых пар состояний. Решение этой задачи основано на том, что пара состояний q и q' несовместима, если либо $\lambda(q, a_i) \neq \lambda(q', a_i)$, либо пара $\delta(q, a_j)$ и $\delta(q', a_j)$ несовместима для некоторых a_i, a_j . Это дает простой индуктивный процесс (в некотором смысле дополнительный к алгоритму Мили): на 1-м шаге несовместимыми объявляются все пары q, q' , для которых $\lambda(q, a_i) \neq \lambda(q', a_i)$; на $(i + 1)$ -м шаге несовместимыми объявляются все пары q, q' , для которых $\delta(q, a_j)$ и $\delta(q', a_j)$ уже были определены как несовместимые на предыдущих шагах. Процесс останавливается, когда не появляется новых несовместимых пар; все остальные пары являются совместимыми.

Далее из полученных пар совместимых состояний можно образовать максимальные классы совместимости, т. е. классы, в которые нельзя добавить ни одного состояния. Нетрудно понять, что система всех максимальных классов является полной и замкнутой (для любой совместимой пары q, q' $\delta(q, a)$ и $\delta(q', a)$ также совместимы и, следовательно, лежат по крайней мере в одном максимальном классе), поэтому ей соответствует автомат S' , покрывающий исходный автомат S . Однако в общем случае он может иметь даже больше состояний, чем S . (В качестве упражнения предложим читателю построить частичный автомат с пятью состояниями, в котором максимальными классами совместимых состояний будут шесть пар: 12, 23, 34, 45, 14, 25, а остальные четыре пары несовместимы.) Поэтому можно попытаться удалить некоторые классы из этой системы, однако при этом нужно проверять, не нарушаются ли полнота и замкнутость. В общем же случае классы минимальной полной и замкнутой системы $\{C_1, \dots, C_p\}$ не обязаны быть максимальными.

Различные методы минимизации частичных автоматов, эвристические приемы обхода указанных трудностей перебора и обсуждение случаев, когда эти трудности не столь велики, можно найти в книге Р. Миллера [21].

Интерпретация автоматов. Основные проблемы абстрактной теории автоматов. Известно, что конечный автомат представляет собой хотя и абстрактную, но с функциональной точки зрения довольно точную модель

дискретного (цифрового) вычислительного или управляющего устройства. Входная буква — это входной сигнал (точнее, комбинация сигналов на всех входах устройства), входное слово — последовательность входных сигналов, поступающих в автомат в дискретные моменты времени (такты) $t = 1, 2, 3, \dots$; выходное слово — последовательность выходных сигналов, выдаваемых автоматом; состояния автомата — это комбинации состояний запоминающих элементов устройства. Такая интерпретация, безусловно, верна, и именно она довольно долго служила основным стимулом развития и источником задач теории автоматов. Однако обращаем внимание читателя на то, что во всем предшествующем изложении не понадобились ни устройства, ни сигналы, ни даже моменты времени. Все, что действительно существенно в абстрактной (т. е. не исследующей структуру) теории автоматов, — это работа со словами при наличии конечной памяти; именно поэтому мы предпочли не навязывать читателю конкретную интерпретацию с самого начала.

Даже с прикладной точки зрения интерпретация автомата как устройства не является универсальной. Хорошо известно, что всякое вычисление или управление можно реализовать как аппаратурно (в виде устройства), так и программно (в виде компьютерной программы). Это приводит к более общему истолкованию автоматов как алгоритмов с конечной памятью, многие свойства которых можно исследовать безотносительно к способу их реализации. Помня о том, что в этой книге речь идет о математике, будем рассматривать автоматы в основном именно с алгоритмической точки зрения. При подходе к теории автоматов как к части теории алгоритмов центральной проблемой является изучение возможностей автоматов в терминах множеств слов, с которыми работают автоматы. Можно выделить два основных аспекта «работы» автоматов: 1) автоматы распознают входные слова, т. е. отвечают на вопрос, принадлежит ли поданное на вход слово данному множеству (это автоматы-распознаватели); 2) автоматы преобразуют входные слова в выходные, т. е. реализуют автоматные отображения (автоматы-преобразователи). Один аспект можно свести к другому: последовательность ответов распознавателя на входные

слова a_{i_1} , $a_{i_1}a_{i_2}$, $a_{i_1}a_{i_2}a_{i_3} \dots$ образует выходное слово, которое является автоматным отображением; с другой стороны, все выходные буквы преобразователя можно разбить на два класса C_1 и C_2 и считать, что автомат распознает множество тех слов α , для которых $\lambda(q_1, \alpha) \in C_1$ (и, следовательно, дополнение к этому множеству). Тем не менее понятия и проблемы, важные при первом аспекте, оказываются либо несущественными, либо сильно видоизмененными во втором; поэтому указанные два взгляда на автомат имеет смысл рассматривать раздельно.

С проблемой возможностей автоматов связан и другой круг задач, традиционных для теории алгоритмов, — распознавание различных свойств автоматов. В отличие от алгоритмов общего вида, для которых все надежды на успех закрываются теоремой Райса (теорема 5.17), многие свойства автоматов оказываются алгоритмически распознаваемыми (см. далее конец § 8.2).

Наконец, третий круг задач теории автоматов — это задачи описания автоматов и их реализации, т. е. представления автомата как структуры, состоящей из объектов фиксированной сложности (элементов). Помимо важного прикладного значения таких задач для проектирования цифровых схем, их исследование стало, быть может, наиболее существенным вкладом теории автоматов в дискретную математику, поскольку в его ходе впервые было введено и подробно изучено понятие сложности. Это понятие, возникнув как обобщение естественной характеристики цифровой схемы — числа ее элементов, постепенно становится одним из центральных понятий теории алгоритмов вообще; многие количественные характеристики алгоритма — память, быстродействие, объем собственного описания (программы) — являются различными аспектами его сложности. В этом отношении теория автоматов оказалась наиболее продвинутой ветвью теории алгоритмов.

Заканчивая разговор о проблематике и интерпретациях теории автоматов, упомянем еще об одной интерпретации автоматов, которой объем данной книги не позволит коснуться. Фон Нейман рассматривал автоматы как удобный язык для описания основных законов взаимодействия сложных систем, т. е. по существу как

метаязык кибернетики. Этот взгляд на автоматы как на язык, т. е. концептуальное средство (основу некоторой системы понятий), был подробно разработан М. Л. Цетлиным и его учениками при исследовании задач целесообразного поведения взаимодействующих объектов, которые формулировались как задачи коллективного поведения автоматов. Очевидно, что содержательный интерес таких задач вовсе не во взаимодействии цифровых схем, а в поведении любых объектов (быть может, живых существ), возможности которых описаны в терминах конечных автоматов. Подробнее с этими концепциями можно ознакомиться по [41, 50].

8.2. РАСПОЗНАВАНИЕ МНОЖЕСТВ АВТОМАТАМИ

Автоматы Мура. Конечный автомат называется *автоматом Мура*, если его функция выходов зависит только от состояний, т. е. для любых $q, q_i, a_j \lambda(q, a_i) = \lambda(q, a_j)$. Функцию выходов автомата Мура естественно считать одноаргументной функцией; обычно ее обозначают буквой μ и называют функцией отметок (так как она каждому состоянию однозначно ставит в соответствие отметку — выход). В графе автомата Мура выход пишется не на ребрах, а при вершине. Общая модель конечного автомата, которая рассматривалась ранее, называется *автоматом Мили*.

Несмотря на то, что автомат Мура — частный случай автомата Мили, возможности этих двух видов автоматов совпадают.

Теорема 8.4. Для любого автомата Мили существует эквивалентный ему автомат Мура.

Пусть дан автомат Мили $S = (A, Q, V, \delta, \lambda)$, $A = \{a_1, \dots, a_m\}$, $Q = \{q_1, \dots, q_n\}$. Определим автомат Мура S_M следующим образом: $A_M = A$, $V_M = V$, Q_M содержит $mn + n$ состояний: mn состояний q_{ij} ($i = 1, \dots, n$; $j = 1, \dots, m$), соответствующих парам (q_i, a_j) автомата S , и n состояний q_{i0} ($i = 1, \dots, n$). Функции δ_M и μ определяются так: $\delta_M(q_{i0}, a_k) = q_{ik}$; для $i = 1, \dots, n$ $\delta_M(q_{ij}, a_k) = q_{lk}$, где l таково, что $\delta(q_i, a_k) = q_l$; $\mu(q_{i0})$ не определено; для остальных состояний $\mu(q_{ij}) = \lambda(q_i, a_j)$.

Для доказательства теоремы достаточно показать, что для любого q_i и любого α $S(q_i, \alpha) = S_m(q_{i_0}, \alpha)$. Это делается индукцией по длине α ; проведение индукции предоставляется читателю. \square

Рекомендуем в качестве примера построить автомат Мура, эквивалентный автомата Мили из примера 8.1.

Из этой теоремы следует, что при исследовании возможностей автоматов достаточно пользоваться автоматами Мура. Это удобно потому, что автомат Мура можно рассматривать как автомат без выходов, состояния которого различным образом отмечены. Без потери общности можно считать, что этих отметок всего две (например, 0 и 1), и они делят состояния на два класса. Зафиксируем один из этих классов и будем называть его состояния заключительными. Это приводит еще к одному определению автомата — автомата без выходов $S = (A, Q, \delta, q_1, F)$, где $F \subseteq Q$ — множество заключительных состояний. В дальнейшем до конца параграфа будут без специальных оговорок рассматриваться инициальные автоматы без выходов S с начальным состоянием q_1 .

Представление событий в автоматах. Множество слов во входном алфавите называется *событием*. Этот термин стал традиционным в теории автоматов, хотя он и необязателен: можно было обойтись просто «множеством слов». Другой термин для множества слов, пришедший из теории грамматик, — «язык» (см. гл. 7). Событие $E \subseteq A^*$ представимо в автомате $S = (A, Q, \delta, q_1, F)$, если $\delta(q_1, \alpha) \in F$, тогда и только тогда, когда $\alpha \in E$. Всякому автомату (при фиксированных q_1 и F) однозначно соответствует представимое в нем событие; на графе автомата это событие изображается множеством всех путей, ведущих из q_1 в вершины из F . Событие называется представимым (в конечном автомате), если существует конечный автомат, в котором оно представимо. Другие названия этого понятия — множество, определимое, или допускаемое [35], или распознаваемое конечным автоматом. Все эти термины также не обязательны, поскольку *представимое в автомате событие — это конечно-автоматный аналог разрешимого множества*; событие E , представимое в автомате S , можно было бы назвать множеством, разрешимым автоматом S .

Может оказаться, что $q_1 \in F$. В этом случае автомат, еще ничего не получив на входе, уже «что-то представляет». Удобно считать, что это «что-то» — пустое слово (слово нулевой длины); оно содержится в событии, представимом таким автоматом. Пустое слово, как и в гл. 7, будем обозначать e . Для любого слова $\alpha ea = ae = \alpha$; таким образом, в свободной полугруппе (см. гл. 2) слов входного алфавита, где умножением является приписывание слов друг к другу, e играет роль единицы.

Пустое слово не следует путать с пустым событием, т. е. с пустым множеством (см. аналогичное замечание в гл. 7 в связи с операциями над языками). Автомат представляет пустое событие, если ни одно из его заключительных состояний не достижимо из начального состояния.

Пример 8.6. а. Любое конечное множество слов $E = \{\alpha_1, \dots, \alpha_h\}$ представимо в автомате. Идея построения автомата по конечному множеству слов иллюстрируется графом на рис. 8.4, а, где заключительные состояния q_{n-k}, \dots, q_{n-1} изображены двойным кружком. Для конкретных множеств эта идея модифицируется в связи с тем, что слова могут иметь общие начала (тогда начала соответствующих путей нужно объединить, чтобы не нарушить условие автоматности) либо просто содержаться друг в друге (тогда из одного заключительного состояния имеется путь в другое заключительное состояние). Пример автомата для $E = \{ab, ac, aba\}$ с заключительными состояниями $F = \{3, 5, 6\}$ приведен на рис. 8.4, б.

В автомате, представляющем конечное множество слов, путь из начального состояния в любое заключительное состояние не может содержать циклов или содержаться в цикле, так как тогда имелось бы бесконечное

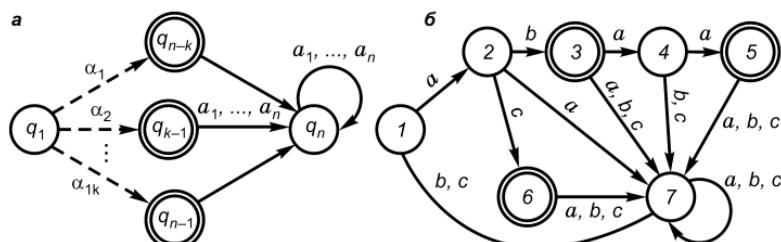


Рис. 8.4

множество путей из начального состояния в F и соответствующее событие было бы бесконечным. Поэтому такой автомат не может быть сильно связным, он является устройством, так сказать, одноразового действия.

б. Автономные автоматы представляют события в однобуквенном алфавите; слова в таких событиях отличаются только длиной. Например, автомат из примера 8.3 с начальным состоянием 1 и $F = \{7\}$ (выходы опускаем) представляет бесконечное событие, состоящее из всех слов,

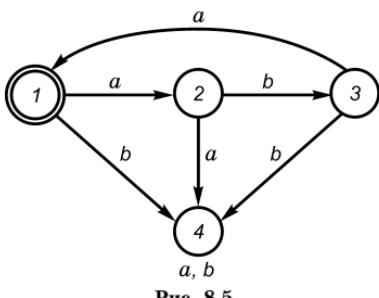


Рис. 8.5

длины которых при делении на 4 дают в остатке 3. Если же положить $F = \{2\}$, то этот автомат представляет пустое событие.

в. Автомат, граф которого приведен на рис. 8.5 ($F = \{1\}$), представляет бесконечное множество $\{e, aba, abaaba, \dots, (aba)^n \dots\}$.

События — это множества конечных слов. Однако можно говорить, что автомат распознает бесконечную последовательность букв $\alpha = a_{i_1}a_{i_2}a_{i_3} \dots$, если он представляет множество $E = \{a_{i_1}, a_{i_1}a_{i_2}, \dots\}$, состоящее из всех начальных отрезков последовательности α .

Теорема 8.5. Существуют события, непредставимые в конечных автоматах; более конкретно: никакая непериодическая бесконечная последовательность не распознается конечным автоматом.

Первая половина теоремы после гл. 5 должна быть очевидной: во-первых, из мощностных соображений (множество событий несчетно, множество автоматов счетно), во-вторых, просто потому, что существуют неразрешимые множества. Поэтому интересно было бы получить пример множества, которое разрешимо (например, машиной Тьюринга), но непредставимо конечным автоматом. Существование такого примера утверждается второй половиной теоремы (пример эффективно заданной непериодической последовательности в алфавите $\{0, 1\}$: 010110111011110...).

Пусть непериодическая последовательность $\alpha = a_{i_1}, a_{i_2} \dots$ распознается автоматом S с n состояниями. Тогда

для любого ее начального отрезка $a_{i_1} \dots a_{i_j}$ $\delta(q_1, a_{i_1} \dots a_{i_j}) = q_{i_j}$, где q_{i_j} — заключительное состояние; поэтому при переработке этой последовательности автомат проходит последовательность заключительных состояний $q_{i_1} \dots q_{i_j} \dots$. Так как Q_S конечно, то некоторое состояние встретится в этой последовательности дважды: $q_{i_j} = q_{i_j+k}$ и, следовательно, $\delta(q_{i_j}, a_{i_j+1}, \dots, a_{i_j+k})$, причем все состояния, проходимые автоматом, заключительные. Поэтому, если на вход автомата в состоянии q_1 подавать бесконечную периодическую последовательность $\alpha_1 = a_{i_1} \dots a_{i_j} (a_{i_j+1} \dots a_{i_j+k})$, где в скобки взят ее период, то автомат будет проходить последовательность заключительных состояний. Следовательно, все начальные отрезки α_1 входят в событие, представимое автоматом, т. е. автомат не отличает α_1 от α и, значит, не распознает α вопреки предположению. \square

Из теоремы 8.5 следует, что класс множеств, представимых конечными автоматами, является лишь частью (собственным подклассом) класса разрешимых множеств. В свою очередь, из этого обстоятельства и теоремы Райса (§ 5.4) вытекает, что *свойство множества «быть представимым в конечном автомате» алгоритмически неразрешимо*. Более точно это утверждение формулируется так. Рассмотрим класс всех вычислимых функций с областью значений в A^* . (Фиксация алфавита A не нарушает общности, так как значения любых других вычислимых функций можно эффективно кодировать словами из A^* .) Каждую функцию из этого класса можно считать функцией, перечисляющей некоторое подмножество A^* , т. е. событие в алфавите A . Тогда в силу теоремы Райса не существует алгоритма, который по данной функции определяет, представимо ли в конечном автомате множество, перечислимое этой функцией, или нет. Поэтому не имеет смысла описывать множества, представимые автоматами, в терминах произвольных разрешимых множеств; следует искать более слабые средства их описания. К изучению таких средств мы и переходим.

Алгебра регулярных событий. Пусть даны события E_1 и E_2 в алфавите A . Напомним три операции над событиями (языками), введенные в гл. 7.

1. *Объединение* $E_1 \cup E_2$ (обычное объединение множеств).

2. Умножение (конкатенация) $E_1E_2 : E = E_1E_2$ — это множество всех слов вида $\alpha_1\alpha_2$, где $\alpha_1 \in E_1$, $\alpha_2 \in E_2$.

3. Итерация $E_1^* = e \cup E_1 \cup E_1E_1 \cup \dots \cup (E_1)^n \cup \dots = \bigcup_{i=0}^{\infty} E_1^i$.

События $\{a_i\}$, где $a_i \in A$, будем называть элементарными и обозначать просто буквами a_i . К элементарным отнесем также событие e .

Событие называется *регулярным*, если оно может быть построено из элементарных событий с помощью конечного числа применений объединения, умножения и итерации; эти операции также назовем регулярными. Иначе говоря, класс R регулярных событий — это наименьший класс подмножеств A^* , содержащий все множества $\{a_i\}$, а также e и замкнутый относительно регулярных операций. Тем самым определена алгебра $(R; \cup, \cdot, ^*)$, основным множеством которой является некоторая система подмножеств A^* , а именно — класс R регулярных событий; элементарные события — образующие этой алгебры. Из определения следует, что каждый элемент этой алгебры (т. е. регулярное событие) может быть описан формулой, содержащей символы образующих (e и буквы A) и знаки регулярных операций над ними. Такие формулы называются *регулярными выражениями*. Регулярные выражения эквивалентны, если они описывают одно и то же регулярное событие.

Приведем некоторые эквивалентные соотношения в алгебре регулярных событий.

Если E, E_1, E_2, E_3 — регулярные события, то

$$E_1(E_2 \cup E_3) = E_1E_2 \cup E_1E_3; \quad (8.7)$$

$$(E_1 \cup E_2)E_3 = E_1E_3 \cup E_2E_3; \quad (8.8)$$

$$(E_1E_2)E_3 = E_1(E_2E_3); \quad (8.9)$$

$$(E^*)^* = E^*; \quad (8.10)$$

$$E^* = e \cup E(E)^*. \quad (8.11)$$

Кроме того, напомним, что объединение ассоциативно и коммутативно.

Пример 8.7. а. Давно употребляемое нами обозначение A^* для множества всех слов в алфавите A может теперь показаться двусмысленным, поскольку $*$ обозначает итерацию. Однако оба смысла этого обозначения

совпадают: регулярное выражение $(a_1 \cup \dots \cup a_n)^*$ действительно задает множество всех слов (включая пустое) в алфавите A .

б. Множество, состоящее из одного слова $a_{i_1} \dots a_{i_k}$, является регулярным событием, поскольку любое выражение вида $a_{i_1} \dots a_{i_k}$ регулярно: оно построено из букв с помощью конкатенации. Любое конечное множество слов $E = \{\alpha_1, \dots, \alpha_k\}$ регулярно и описывается выражением $E = \alpha_1 \cup \dots \cup \alpha_k$, не содержащим итерации.

Обратно, если регулярное выражение E не содержит итерации, то раскрытие скобок [допустимое в силу (8.7) и (8.8)] преобразует его к виду $\alpha_1 \cup \dots \cup \alpha_k$; следовательно, E описывает конечное событие. Если же E содержит итерацию, то оно бесконечно, за исключением случаев, когда итерация применяется только к e ($e^* = e$, т. е. конечно).

в. Регулярное выражение вида A^*E , где E — конечное событие, не содержащее пустого слова, описывает бесконечное событие, содержащее все слова из A^* , оканчивающиеся словами из E . Такие события называются *определенными*, или дефинитными. Например, событие $(a \cup b \cup c)^*(a \cup cb)$ содержит все слова в алфавите $\{a, b, c\}$, кончающиеся на a или cb .

г. Событие $E_1 = (a \cup b)^*c(a \cup b)^*c(a \cup b)^*$ состоит из всех слов в алфавите $\{a, b, c\}$, содержащих c ровно 2 раза. Событие E_1^* состоит из всех слов, содержащих c четное число раз.

д. Регулярное событие E называется *асинхронным событием*, если для любых слов α_1, α_2 и буквы a из того, что $\alpha_1 a^k \alpha_2 \in E$ для некоторого k , следует, что $\alpha_1 a^k \alpha_2 \in E$ для любого $k = 1, 2 \dots$; иначе говоря, если $\alpha \in E$, то в E содержатся все слова, полученные из α повторениями некоторых букв слова α , либо вычеркиванием из α некоторых повторений букв. Например, если E — асинхронное событие и $abbcccd \in E$, то $abcd \in E$, $aabccedd \in E$ и т. д. Регулярные выражения для асинхронных событий могут быть построены из событий aa^* с помощью тех же трех операций; при этом они не должны содержать подформул вида aa^*aa^* . Например, событие $(aa^*bb^* \cup cc^*)^*$ асинхронно, а событие $(aacc^* \cup cc^*)^*$ не является асинхронным, так как оно содержит слово

aac, но не содержит слова *ac*, получающегося из *aac* вычеркиванием повторения *a*.

Регулярные события тесно связаны с автоматами. Изучение этой связи удобно вести, используя описание событий с помощью графов, к которому мы сейчас и переходим.

Источники. Уже говорилось, что на графе автомата событие, представляемое автоматом, изображается множеством путей из начальной вершины в заключительные вершины. Аналогичным образом для описания множеств слов можно использовать произвольные графы (не обязательно автоматные), на ребрах которых написаны буквы. Такие графы называются *источниками*. Более точно, источником над алфавитом A называется ориентированный граф, в котором: 1) выделены начальные и заключительные вершины (вершина может быть одновременно и начальной, и заключительной); 2) на каждом ребре написана либо буква из A , либо пустое слово e (такие ребра назовем пустыми). Каждый источник H однозначно определяет событие E в алфавите A , порожданное множеством всех путей из начальных вершин в заключительные (если путь содержит пустое ребро, то ему соответствует слово $\alpha e \beta = \alpha \beta$). В этом случае говорят, что источник H представляет событие E . Два источника называются эквивалентными, если они представляют одно и то же событие. Граф автомата без выходов — это частный случай источника.

Для любого источника H существует эквивалентный ему двухполюсный источник H_0 (с одной начальной и одной заключительной вершиной, которые, впрочем, могут совпадать), строящийся так. Если в H имеется несколько начальных вершин, то в H_0 вводится новая вершина q_0 , которая объявляется единственной начальной вершиной H_0 и соединяется с прежними начальными вершинами H пустыми ребрами (ребер, заходящих в q_0 , в H_0 нет). Если в H имеется несколько заключительных вершин, то в H_0 вводится новая вершина q_z , которая объявляется единственной заключительной вершиной H_0 ; из прежних заключительных вершин в q_z проводятся пустые ребра; ребер, выходящих из q_z , в H_0 нет. В остальном H_0 совпадает с H .

Теорема 8.6. Для любого регулярного события E существует двухполюсный источник, представляющий E .

Теорема доказывается индукцией по глубине регулярного события E . Элементарное событие представляется источником, состоящим из двух вершин — начальной и заключительной и ребра, идущего из начальной вершины в заключительную, на котором написано данное событие (a_i или e).

Пусть теперь построены двухполюсные источники: H_1 , представляющий регулярное событие E_1 и H_2 , представляющий регулярное событие E_2 ; их начальные вершины — q_{10} и q_{20} , заключительные вершины — q_{1z} и q_{2z} соответственно. Тогда источник H с начальной вершиной q_0 и заключительной вершиной q_z , который представляет событие E — результат регулярной операции над E_1 и E_2 , строится так.

1. $E = E_1 \cup E_2$. H строится «параллельным соединением» H_1 и H_2 (рис. 8.6, а). Он состоит из источников H_1 и H_2 и новых вершин q_0 и q_z ; из q_0 проводятся пустые ребра в q_{10} и q_{20} , из q_{1z} и q_{2z} проводятся пустые ребра в q_z .

2. $E = E_1 E_2$. H строится «последовательным соединением» H_1 и H_2 (рис. 8.6, б): из a_{1z} проводится пустое ребро в q_{20} , начальной вершиной H объявляется q_{10} , заключительной вершиной H объявляется q_{2z} .

3. $E = E^*$. H строится зацикливанием H_1 (рис. 8.6, в): из q_{1z} проводится пустое ребро в q_{10} , q_{10} объявляется начальной и заключительной вершиной H .

Доказательство того, что построенные таким образом источники действительно представляют соответствующие события, несложно. Пусть, например, $E = E_1 E_2$ и $\alpha \in E$. Тогда $\alpha = \alpha_1 \alpha_2$, где $\alpha_1 \in E_1$, $\alpha_2 \in E_2$. По предположению, H_1 представляет E_1 , H_2 представляет E_2 ; поэтому существует путь α_1 из q_{10} в q_{1z} и путь α_2 из q_{20} в q_{2z} . Но тогда

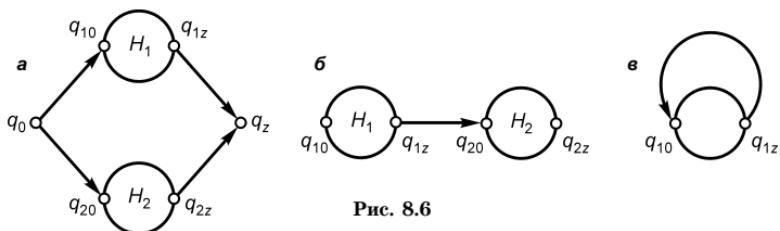


Рис. 8.6

по построению существует путь $\alpha_1\alpha_2$ из q_{10} (начальной вершины H) в q_{2z} (заключительную вершину H). И наоборот, всякий путь α из q_{10} в q_{2z} обязательно проходит через q_{1z} и q_{20} , поэтому он имеет вид $\alpha = \alpha_1\alpha_2$, где α_1 — путь из q_{10} в q_{1z} и α_2 — путь из q_{20} в q_{2z} , откуда следует, что $\alpha_1 \in E_1$ и $\alpha_2 \in E_2$. Доказательства для объединения и итерации аналогичны. \square

Введение пустых ребер (вместо простого склеивания вершин) объясняется необходимостью избежать «ложных путей». Некоторые из введенных пустых ребер в дальнейшем можно удалять, не меняя представляемого события.

Детерминизация источников и синтез автоматов. Если источник имеет одну начальную вершину, не содержит пустых ребер и удовлетворяет условиям автоматности, то он является графом автомата без выходов. Такой источник часто называют *детерминированным источником*. Этот термин связан с интерпретацией произвольного источника как *недетерминированного автомата*, т. е. автомата, для которого $\delta(q, a)$ определена неоднозначно; точнее, значением $\delta(q, a)$ является подмножество Q (быть может, пустое). Иначе говоря, при фиксированной вершине q символу a соответствует множество ребер, а слову α — множество путей, ведущих из q ; на каждом шаге недетерминированный автомат как бы совершает выбор из возможных ребер (q, a) . Понятие представимости в недетерминированном автомате совпадает с понятием представимости в источнике.

Теорема 8.7 (теорема детерминизации). Для любого источника H с n вершинами существует эквивалентный ему детерминированный источник H' , имеющий не более чем 2^n вершин.

Назовем множество \tilde{q} вершин источника замкнутым, если из того, что $q_i \in \tilde{q}$, следует, что \tilde{q} принадлежит любая вершина, в которую из q_i ведет пустое ребро. Для источника без пустых ребер все множества вершин замкнуты. Обозначим через \tilde{q}_1 наименьшее замкнутое множество вершин H , содержащее все начальные вершины H .

Источник H' строится так. Образуем все замкнутые подмножества вершин H (их не более чем 2^n) и каждому такому подмножеству поставим в соответствие вершину

H' (в дальнейшем эти подмножества и соответствующие им вершины будем отождествлять и обозначать \tilde{q}_i). Начальной вершиной H' объявим \tilde{q}_0 , заключительными вершинами — все подмножества \tilde{q}_i , содержащие хотя бы одну заключительную вершину H . Если в источнике H из множества вершин \tilde{q}_i пути a (они могут содержать пустые ребра) ведут в множество \tilde{q}_j (т. е. \tilde{q}_j — это множество концов всех ребер a , начала которых принадлежат множеству \tilde{q}_i), то в источнике H' из вершины \tilde{q}_i проводится ребро a в вершину \tilde{q}_j . Если же в H никакая из вершин множества \tilde{q}_i не имеет выходящего из нее пути a , то в H' из вершины \tilde{q}_i проводится ребро a в вершину \emptyset , соответствующую пустому подмножеству вершин H . Таким образом, каждой вершине \tilde{q}_i источника H' и каждому входному символу a соответствует ровно одно ребро a , выходящее из вершины \tilde{q}_i , и, следовательно, источник H' — детерминированный. Другими словами, H' — это граф автомата с начальным состоянием \tilde{q}_1 ; описанное ранее построение ребер H' определяет функцию переходов автомата $\delta_{G'}(\tilde{q}_i, a) = \tilde{q}_j$.

Источник H' обладает следующим свойством: в H' непустой путь α из \tilde{q}_1 в \tilde{q}_j существует тогда и только тогда, когда в H для любой вершины $q \in \tilde{q}_j$ существует путь α из некоторой начальной вершины $q_1 \in \tilde{q}_1$ в q (если $\alpha = e$, то $\tilde{q}_j = \tilde{q}_1$ по условию замкнутости; пустых ребер в H' по построению нет). Доказательство проведем индукцией по длине слова α . Если $\alpha = a$, то это свойство выполняется по построению ребер a в H' . Предположим, что оно выполняется для всех слов α длины $\leq k$, и докажем, что оно выполняется для αa , где a — произвольный входной символ.

Пусть в H' имеется непустой путь αa из \tilde{q}_1 в \tilde{q}_j : $\delta_G(\tilde{q}_1, \alpha a) = \tilde{q}_j$. Если $\delta_G(\tilde{q}_1, \alpha 0) = \tilde{q}_l$, то из \tilde{q}_l в \tilde{q}_j ведет ребро a . По предположению, в H для любой вершины $q^* \in \tilde{q}_l$ существует путь α из некоторой начальной вершины $q_1 \in \tilde{q}_1$ в q^* . По построению H' из того, что в H' есть ребро a из \tilde{q}_l в \tilde{q}_j , следует, что в H для любой вершины $q \in \tilde{q}_j$ найдется вершина \tilde{q}_l , из которой ведет путь a в q ; поэтому в H имеется путь a из q^* в q и, следовательно, путь αa из q_1 в q .

Аналогично доказывается обратное утверждение: в предположении, что в H для любой вершины $q \in \tilde{q}_j$ есть

путь αa из некоторой начальной вершины $q_1 \in \tilde{q}_1$ в q , рассматривается множество всех путей αa из начальных вершин в вершины из \tilde{q}_j и множество \tilde{q}_l всех вершин, в которые ведут отрезки α этих путей. С использованием индуктивного предположения и построения ребер H' показывается, что в H' $\delta(\tilde{q}_1, \alpha a) = \tilde{q}_j$.

Из доказанного свойства H' и определения заключительных вершин H' следует, что в H' путь α из \tilde{q}_1 в заключительную вершину существует тогда и только тогда, когда в H имеется путь α из начальной вершины в заключительную. Поэтому $\alpha \in E'$, если и только если $\alpha \in E$. \square

Из теорем 8.6 и 8.7 непосредственно следует одна из важнейших теорем теории автоматов, доказанная впервые Клини.

Теорема 8.8 (теорема синтеза). Для любого регулярного события E существует конечный автомат, представляющий это событие. \square

Метод построения (синтеза) автомата, представляющего E , заключается в том, что сначала строится источник, представляющий E , а затем этот источник детерминизируется путем процедуры, описанной при доказательстве теоремы 8.7. На практике процедура детерминизации несколько модифицируется с целью ее упрощения. Дело в том, что некоторые подмножества вершин H (т. е. состояния H') не достижимы из начальной вершины; их удаление не изменит события, представляемого источником. Поэтому в таблицу переходов H' включаются только те подмножества, которые порождаются процедурой детерминизации, начатой с подмножества q_1 (см. пример 8.8).

При такой модификации построенный автомат может иметь меньше чем 2^n состояний (n — число вершин исходного источника). Однако в общем случае эту оценку понизить нельзя. Например, в трехбуквенном алфавите для любого n существует источник с n состояниями, такой, что любой эквивалентный ему автомат имеет не менее чем 2^n состояний. Пример такого источника для $n = 5$ приведен на рис. 8.7; доказательство того, что он обладает указанным свойством, можно найти в [33].

Пример 8.8. Построим автомат, представляющий событие $(ab \cup c^*) (a \cup bc)^* a$. В соответствии с ранее изложенным синтез автомата проводится в два этапа. Снача-

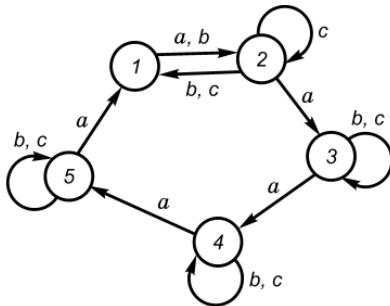


Рис. 8.7

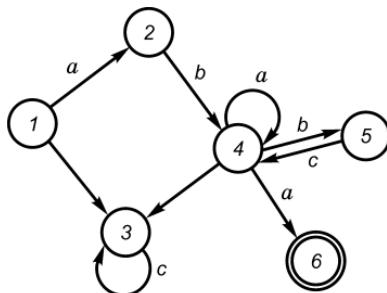


Рис. 8.8

ла по событию строится представляющий его источник H . Он изображен на рис. 8.8. В этом источнике ребра, которым не приписано букв, — пустые; некоторые «лишние» пустые ребра, возникающие, если строго следовать методу теоремы 8.6, удалены. Вершина 1 является начальной, вершина 6 — заключительной.

Затем методом теоремы 8.7 построенный источник детерминизируется, при этом строятся только подмножества, достижимые из начального подмножества {1}. Функция переходов детерминированного источника H' , вершинами которого служат подмножества вершин H , приведена на табл. 8.7, а (знаком \emptyset , как обычно, обозначено пустое множество); после перенумерации этих подмножеств она приобретает обычный вид таблицы переходов (табл. 8.7, б).

В табл. 8.7, б заключительные состояния 2 и 5 выделены; они соответствуют подмножествам из табл. 8.7, а, содержащим заключительное состояние 6 источника H .

	<i>a</i>	<i>b</i>	<i>c</i>
1	2, 4, 6	5	3, 4
2, 4, 6	4, 6	4, 5	\emptyset
5	\emptyset	\emptyset	4
3, 4	4, 6	5	3, 4
4, 6	4, 6	5	\emptyset
4, 5	4, 6	5	4
\emptyset	\emptyset	\emptyset	\emptyset
4	4, 6	5	\emptyset

	<i>a</i>	<i>b</i>	<i>c</i>
1	2	3	4
2	5	6	7
3	7	7	8
4	5	3	4
5	5	3	7
6	5	3	8
7	7	7	7
8	5	3	7

Анализ автоматов. Если процесс построения автомата по заданному событию (и вообще по некоторому описанию множества слов во входном алфавите) называется синтезом, то обратный процесс — получение описания множества входных слов, представимого заданным автоматом, называется анализом автомата. Из теоремы 8.9 видно, что результат анализа также может быть описан с помощью регулярных событий (этот факт также доказан Клини).

Теорема 8.9 (теорема анализа). Всякое событие, представимое конечным автоматом, регулярно.

Определим индуктивно событие E_{ij}^k : 1) E_{ij}^0 — множество всех входных символов a , таких, что $\delta(q_i, a) = q_j$; 2) $E_{ij}^k = E_{ij}^{k-1} \cup E_{ik}^{k-1}(E_{kk}^{k-1})^*E_{kj}^{k-1}$. Обозначим через M_{ij}^k множество всех входных слов, ведущих из q_i в q_j и не проходящих при этом через состояния с номерами, большими, чем k .

Лемма. $E_{ij}^k = M_{ij}^k$.

Докажем эту лемму по индукции. Для $k = 0$ она очевидна. Пусть она верна для $k = r - 1$. Тогда $M_{ij}^r = E_{ij}^{r-1} \cup \tilde{M}_{ij}^{r-1}$, где \tilde{M}_{ij}^r — множество всех слов из M_{ij}^r , проходящих через q_r . Если $\alpha \in \tilde{M}_{ij}^r$ и проходит через q_r s раз, то α представимо в виде $\alpha = \alpha_0\alpha_1\dots\alpha_{s-1}\alpha_s$, где $\alpha_0 \in E_{ir}^{r-1}$, $\alpha_s \in E_{rj}^{r-1}$, $\alpha_1\dots\alpha_{s-1} \in E_{rr}^{r-1}$ (т. е. содержит $s - 1$ цикл, проходящий через q_r). Поэтому

$$\alpha \in E_{ir}^{r-1}(E_{rr}^{r-1})^*E_{rj}^{r-1}.$$

И наоборот, если это включение выполнено, то из регулярной формулы его правой части и индуктивного предположения следует, что $\alpha \in \tilde{M}_{ij}^r$. Таким образом, $M_{ij}^r = E_{ir}^{r-1}(E_{rr}^{r-1})^*E_{rj}^{r-1}$ и, следовательно, $M_{ij}^r = E_{ij}^{r-1} \cup \cup E_{ir}^{r-1}(E_{rr}^{r-1})^*E_{rj}^{r-1} = E_{ij}^r$, что и доказывает лемму.

Очевидно, что для автомата с n состояниями, начальным состоянием q_1 и заключительными состояниями q_{z_1}, \dots, q_{z_p} представляемое им событие имеет вид

$$M_{1z_1}^n \cup \dots \cup M_{1z_p}^n.$$

Из леммы следует, что это событие регулярно. \square

Доказательство теоремы по существу представляет собой алгоритм анализа. Изложенный здесь алгоритм (алгоритм Макнотона–Ямады) очень компактен и удобен для доказательств, однако приводит, как, впрочем, и

другие алгоритмы анализа, к довольно громоздким регулярным выражениям (пример анализа в [21], т. II). Заметим при этом, что в алгебре регулярных событий нет нетривиальных методов отыскания минимального регулярного выражения, эквивалентного данному. В то же время из теоремы синтеза следует, что проблема распознавания эквивалентности регулярных выражений разрешима, так как по выражениям E_1 , E_2 можно построить представляющие их автоматы S_1 , S_2 и проверить их на эквивалентность (сравнив автоматы, минимальные для S_1 , S_2); следовательно, метод минимизации регулярных выражений существует, хотя он заведомо нереалистичен из-за своей громоздкости.

Итак, теоремы анализа и синтеза устанавливают взаимно однозначное соответствие между автоматами (с точностью до эквивалентности) и регулярными событиями. Поэтому регулярные события — это один из основных языков для описания поведения автоматов, используемых в теоретических исследованиях. Важным свойством регулярных событий является их замкнутость относительно булевых операций: пересечение регулярных событий и дополнение к регулярному событию также регулярны (объединение — регулярная операция по определению). Действительно, если событие E описывается источником H , то \bar{E} описывается источником \bar{H} , который отличается от H тем, что заключительными вершинами \bar{H} служат те и только те вершины, которые не являются заключительными в H . Так как для любых множеств E и E_2

$$E_1 \cap E_2 = \overline{\bar{E}_1 \cup \bar{E}_2},$$

то из источников для E_1 и E_2 можно построить источник для $E_1 \cap E_2$; в силу теорем синтеза и анализа всякое событие, заданное источником, регулярно и искомая замкнутость доказана. Этот факт позволяет расширить язык регулярных событий, дополнив его операциями дополнения и пересечения.

Пример 8.9. а. Автономные автоматы представляют регулярные события в однобуквенном алфавите. Если для автомата, указанного в примерах 8.3 и 8.6, б (с начальным состоянием 1, множество заключительных состояний

$F = \{7\}$, то он представляет событие $E = aaa(aaaa)^*$, а если $F = \{3, 4, 6\}$, то $E = a \cup aa(aaaa)^* \cup aaaa(aaaa)^* = a \cup aa(e \cup aa)(aaaa)^*$.

б. Автомат на рис. 8.5 представляет событие $(aba)^*$.

в. Автоматы, представляющие определенные события (см. пример 8.7, в), называются определенными автоматами или автоматами с конечной глубиной памяти. Смысл последнего термина — в том, что такие автоматы одинаково реагируют на слова, у которых их «хвосты» (заранее фиксированной для данного автомата длины) совпадают. Свойства этих автоматов описаны в [42].

г. Автомат называется *асинхронным*, если в нем для любых q_i и a $\delta(q_i, aa) = \delta(q_i, a)$. Состояние $q_i = \delta(q, a)$ с таким свойством называется устойчивым по a ; поэтому можно сказать, что в асинхронном автомате любое состояние устойчиво по любому входу, ведущему в это состояние. Событие является асинхронным (см. пример 8.7, д), если и только если оно представимо в асинхронном автомате (докажите!).

д. Пусть $E = (a \cup b \cup c)^*(ab \cup c)$. Тогда $\bar{E} = e \cup b \cup (a \cup b \cup c)^*(bb \cup cb \cup a)$.

Автоматы и теория алгоритмов. Полученные результаты можно проинтерпретировать в терминах теории алгоритмов. Выше уже отмечалось, что событие, представимое в автомате, — это множество, разрешимое автоматом, или, как говорят, автоматно-разрешимое множество. Для инициального автомата с выходами и заключительными состояниями вводится понятие автоматно-перечислимого множества: множество, перечислимое автоматом, — это множество всех выходных слов, образованных путями из начального состояния в заключительные состояния, иначе говоря, множество всех слов $S(\alpha)$, таких, что $\delta(q_1, \alpha)$ — заключительное состояние. В § 5.4 рассматривалась связь между перечислимыми и разрешимыми множествами и было установлено, что разрешимые множества всегда перечислимы, обратное же верно не всегда. Как обстоит дело в автоматном случае?

Пусть даны автоматно-разрешимое множество M в алфавите A и представляющий его автомат S . Автомат S' с выходами, перечисляющий множество M , строится так. Входной и выходной алфавиты S' совпадают и равны A . Граф S' получается из графа S заменой на каждом ребре символа a_i парой a_i, a_i ; начальное и заключительные состояния автомата S' — те же, что и в S . Автомат S' просто переписывает на выход

все, что он получает на входе; однако всякий раз, когда на вход поступило слово из M , он, переписав его на выход, оказывается в заключительном состоянии; следовательно, S' перечисляет M . Итак, если множество M автоматно-разрешимо, то оно и автоматно-перечислимое, причем существует автомат, перечисляющий M , который по сложности не превосходит автомат, разрешающий M (он имеет тот же граф и ту же мощность входного алфавита).

Пусть теперь даны автоматно-перечислимое множество M в алфавите A и перечисляющий его автомат S с n состояниями, для которого A является выходным алфавитом. В графе S удалим с ребер входные символы; получим источник (вообще говоря, недетерминированный), описывающий событие M . Из теоремы 8.7 о детерминизации следует, что существует автомат S' с 2^n состояниями, представляющий M , причем, как показывает пример автомата на рис. 8.7, в некоторых случаях эту оценку числа состояний понизить нельзя. Таким образом, если множество автоматно-перечислимое, то оно автоматно-разрешимо, однако число состояний разрешающего автомата может экспоненциально возрасти. Аналогия с общим случаем эффективно задаваемых множеств будем полной, если учесть, что автоматные множества задаются устройством с конечным числом состояний, поэтому экспоненциальный переход от перечислимости к разрешимости снова дает конечное число состояний. Машина Тьюринга — это устройство со счетным множеством состояний (если считать и ленту); поэтому экспоненциальный (от множества к системе его подмножеств) переход от перечислимости к разрешимости может вывести за пределы счетных множеств, т. е. за пределы эффективно вычисляющих устройств.

Для автоматов алгоритмически разрешимы следующие проблемы, которые в силу теоремы Райса неразрешимы для произвольных алгоритмов: 1) проблема эквивалентности автоматов; 2) проблема непустоты множества, представляемого автоматом; 3) проблема бесконечности множества, представляемого автоматом. Разрешимость первой проблемы уже отмечалась; она следует из результатов § 8.1. Проблема непустоты равносильна проблеме достижимости заключительного состояния q_z из начального состояния q_1 : если путь из q_1 в q_z существует, то существует и простой (без циклов) путь из q_1 в q_z длины, меньшей n (n — число состояний); поэтому проблема решается вычислением $\delta(q_1, \alpha)$ для всех α длины, меньшей n . Что же касается третьей проблемы, то множество, представимое автоматом, бесконечно, если и только если оно содержит слово α , такое, что $n \leq |\alpha| < 2n$. Действительно,

в этом случае существует путь из q_1 в q_z , проходящий дважды через некоторое состояние, т. е. содержащий цикл; повторяя этот цикл нужное число раз, можно получить сколь угодно длинные слова, представимые в автомате.

Автоматные множества довольно просто описываются в терминах формальных систем: множество слов автоматно (регулярно), если и только если оно порождается нормальной системой Поста с продукцией вида $\alpha x \rightarrow \beta x$ (более подробно о связи автоматов с формальными системами см. в [44]).

Автоматы и языки. На естественный вопрос о связи конечных автоматов с языками или, точнее, о том, как конечно-автоматные множества характеризуются в терминах теории языков, имеется простой ответ.

Теорема 8.10. Множество слов регулярно, если и только если оно является языком типа 3 в классификации Хомского (см. гл. 7).

Пусть множество E регулярно и S — конечный автомат, представляющий E . Построим по S грамматику G_S типа 3 следующим образом. Терминальный алфавит G_S совпадает с входным алфавитом S , нетерминальный алфавит G_S взаимно однозначно соответствует алфавиту состояний S (для сохранения обозначений, принятых в грамматиках, обозначим через B_i нетерминальный символ, соответствующий состоянию q_i), начальный символ G_S соответствует начальному состоянию q_1 автомата S . Каждой паре (q_i, a) , такой, что $\delta(q_i, a) = q_j$, поставим в соответствие одно правило $B_i \rightarrow aB_j$, если q_j — незаключительное состояние, и два правила $B_i \rightarrow aB_j$, $B_i \rightarrow a$, если q_j — заключительное состояние. Нетрудно убедиться, что $\alpha \in L(G_S)$, если и только если $q_j = \delta(q_1, \alpha)$ — заключительное состояние S .

Пусть теперь G — грамматика типа 3. Построим по G источник H_G следующим образом. Каждому нетерминальному символу B_i грамматики G ставится в соответствие вершина q_i ; вершина, соответствующая начальному символу G , объявляется начальной в H_G . Кроме того, в H_G вводится дополнительная вершина q_z , которая объявляется заключительной. Каждому правилу вида $B_i \rightarrow aB_j$ соответствует ребро с символом a из q_i в q_j ; каждому правилу вида $B_i \rightarrow a$ соответствует ребро с символом a из q_i в q_z . Легко видеть, что путь α из начальной вершины

в заключительную существует в H_G тогда и только тогда, когда $\alpha \in L(G)$. \square

Из этой теоремы непосредственно следует разрешимость проблем непустоты и бесконечности, о которых говорилось выше, поскольку они разрешимы уже для языков типа 2. Напротив, разрешимость проблемы эквивалентности — это специфическая особенность языков типа 3, поскольку для произвольных КС-языков эта проблема неразрешима (теорема 7.7).

Эквивалентность автоматных множеств и языков типа 3 говорит о том, что алгебра регулярных событий (алгебра Клини), рассмотренная в данном параграфе, является адекватным алгебраическим описанием языков типа 3. Алгебра Клини содержит два типа операций над языками: конечные операции — объединение и конкатенацию, которые из конечных языков порождают конечные языки, и итерацию, которая из конечного языка порождает бесконечный язык. КС-языки имеют аналогичное алгебраическое описание (хотя и не столь изящное), а именно: все КС-языки, и только они, порождаются из элементарных языков с помощью конечных операций (объединения и конкатенации либо подстановки, которая им эквивалентна, — см. гл. 7) и операций типа зацикливания. В отличие от итерации L^* , результат которой однозначно определяется языком L , зацикливание $[L]_a^*$ — это множество операций, зависящих от параметра a . Это, с одной стороны, объясняет большую порождающую способность этих операций по сравнению с итерацией, но, с другой стороны, делает алгебраическое описание КС-языков более громоздким, чем описание языков типа 3.

В конце § 6.4 говорилось о двух подходах к формализации понятия конструктивности — алгоритмическом и формально-системном. В теории формальных языков — в разных ее аспектах — оказываются полезными оба подхода. Описание языков дается в терминах порождающих процедур, которые типичны для формально-системного подхода; эти процедуры реализуются в виде порождающих грамматик, которые рассматривались в гл. 7. Задачи же синтаксического анализа, заключающиеся в распознавании принадлежности текста

языку и построении его дерева, ставятся как задачи разработки детерминированных распознающих процедур, которые типичны для алгоритмического подхода. Эти процедуры реализуются в виде различных моделей автоматов, адекватных различным классам языков. Для языков типа 3, как следует из теоремы 8.10, адекватной автоматной моделью является конечный автомат. Для более сложных языков адекватными являются другие автоматные модели, отличающиеся от конечных автоматов бесконечностью памяти; однако на эту бесконечность в зависимости от вида модели и связанного с ней языка накладываются различные ограничения — память может быть магазинной (доступной только с одного конца), линейно ограниченной (линейно зависящей от длины распознаваемого слова) и т. д., что делает эти модели более ограниченными в своих возможностях, чем машины Тьюринга, которые можно считать автоматной моделью языков типа 0. Рассмотрение этих моделей выходит за рамки настоящей книги.

8.3. СЕТИ ИЗ АВТОМАТОВ, ИХ АНАЛИЗ И СИНТЕЗ

Комбинационные и логические автоматы. Автомат^{*} называется *комбинационным*, если для любого входного символа a и любых состояний q_i и q_j $\lambda(q_i, a) = \lambda(q_j, a)$, иначе говоря, если выходной символ не зависит от состояния и определяется текущим входным символом. В таком автомате все состояния эквивалентны, и, следовательно, минимальный комбинационный автомат имеет одно состояние. Функция переходов в нем вырождена; его поведение однозначно задается функцией выходов с одним аргументом: $\lambda(a_i) = v_j$.

Автомат называется *логическим*, если его входной алфавит состоит из 2^m двоичных наборов длины m , а выходной — из 2^n двоичных наборов длины n . Функция выходов логического комбинационного автомата — это просто система n логических функций от m переменных

* Здесь и далее, если не оговорено противное, рассматриваются автоматы общего вида — автоматы Мили.

(i -я функция определяет значения i -й компоненты в выходном векторе автомата).

Последовательные автоматные вычисления. В этом параграфе речь будет идти о различных способах комбинирования автоматов друг с другом. Если рассматривать автоматы просто как частный случай алгоритмов, то естественно прежде всего изучить автоматные блок-схемы, т. е. блок-схемы (введенные в § 5.1 для произвольных алгоритмов), все блоки которых являются конечно-автоматными алгоритмами. Поскольку общие свойства схем оказываются очень простыми, ограничимся рассмотрением этих свойств на примере.

На рис. 8.9 дана блок-схема, на которой S_1 и S_2 — автоматные операторы, а S_3 — автоматный предикат. Как и обычно, сама блок-схема не содержит указаний о том, с какой информацией работают автоматы; она лишь указывает передачи управления: сначала работает S_1 , затем S_2 , затем автомат S_3 проверяет условие; при положительном ответе работает S_1 , при отрицательном — S_2 . Понятно, что при таком взаимодействии автоматы должны «уметь останавливаться», как алгоритмы в гл. 5. Формально будем считать, что каждый из автоматов в заключительном состоянии не определен, т. е. не воспринимает последующих входных сигналов. Автомат, вычисляющий предикат, будет иметь два заключительных состояния — для «да» и для «нет». Зафиксируем (пока без входного алфавита) конкретные таблицы переходов для S_1 , S_2 , S_3 (табл. 8.8, a – e) и рассмотрим два крайних случая, которые возможны для входной информации автомата. (Выходы для данных рассмотрений

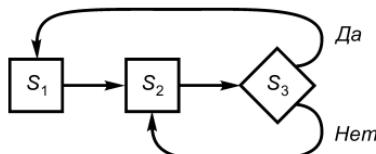


Рис. 8.9

а

q_{11}	q_{11}	q_{1z}	q_{12}
q_{12}	q_{1z}	q_{11}	q_{12}
q_{1z}	—	—	—

б

q_{21}	q_{23}	q_{22}
q_{22}	q_{2z}	q_{22}
q_{23}	q_{21}	q_{23}
q_{2z}	—	—

в

<i>Таблица 8.8</i>	q_{31}	q_{31}	q_{3z0}	q_{32}
q_{32}	q_{3z1}	q_{3z1}	q_{31}	
q_{3z0}	—	—	—	
q_{3z1}	—	—	—	

несущественны: для определенности будем считать, что они во всех клетках различны.)

Случай 1: входные алфавиты исходных автоматов не пересекаются. Пусть $A_1 = \{a, b, c\}$, $A_2 = \{d, f\}$, $A_3 = \{g, h, i\}$. Тогда схеме на рис. 8.9 соответствует табл. 8.9, а. В ней заключительные состояния отождествлены с начальными состояниями последующих автоматов (аналогичные преобразования производились при композиции машин Тьюринга в § 5.2). Полученную таблицу переходов можно минимизировать по правилам минимизации частичных автоматов; результат приведен в табл. 8.9, б. Разумеется, автомат, описываемый в табл. 8.9, б, будет работать в соответствии с блок-схемой рис. 8.9 только в том случае, если в нужные моменты времени будут происходить «переключения внешней среды», т. е. переходы от алфавита A_1 к алфавиту A_2 и т. д. Поскольку эти моменты соответствуют переходам от одного подавтомата к другому и, следовательно, автоматом с табл. 8.9, б распознаются, то переключения можно осуществлять с помощью выходных сигналов. В нашем примере переключающими сигналами будут $\lambda(1, b)$, $\lambda(2, a)$, $\lambda(2, d)$, $\lambda(1, h)$, $\lambda(2, g)$, $\lambda(2, h)$. Итак, в этом случае блок-схема из автоматов — это также автомат, алфавит которого является

а

Таблица 8.9

	a	b	c	d	f	g	h	i
q_{11}	q_{11}	q_{21}	q_{12}					
q_{12}	q_{21}	q_{11}	q_{12}					
q_{21}				q_{23}	q_{22}			
q_{22}				q_{31}	q_{22}			
q_{23}				q_{21}	q_{23}			
q_{31}						q_{31}	q_{21}	q_{32}
q_{32}						q_{11}	q_{11}	q_{31}

б

	a	b	c	d	f	g	h	i
1	1	1	2	3	2	1	1	2
2	1	1	2	1	2	1	1	1
3	—	—	—	1	3	—	—	—

объединением алфавитов исходных автоматов, а число состояний не превосходит $\max |Q_i|$, где максимум берется по мощностям множеств состояний исходных автоматов. Можно сказать, что такой автомат (табл. 8.9, б) реализует блок-схему рис. 8.9 на общей памяти с переключением входов.

Случай 2: входные алфавиты исходных автоматов совпадают или включаются друг в друга. Если в рассматриваемом примере положить $A_1 = A_3 = \{a, b, c\}$, $A_2 = \{a, b\}$, то для блок-схемы получим табл. 8.10.

В этом случае объединения состояний, вообще говоря, не происходит, если только состояния из разных подавтоматов не оказываются эквивалентными. Общий входной алфавит является объединением исходных алфавитов; его мощность равна мощности максимального из исходных алфавитов. Число состояний полученного автомата не превосходит суммы чисел состояний исходных автоматов.

Очевидно, что остальные возможные случаи соотношения алфавитов — это их частичное пересечение; получающиеся таблицы имеют вид, «промежуточный» между табл. 8.9 и 8.10; возможности их минимизации будут зависеть от конкретного вида исходных автоматов. Наконец, заметим, что для графов автоматов указанные преобразования крайне просты и сводятся к отождествлению заключительных вершин с соответствующими начальными вершинами. Правда, возможности минимизации видны на графе не столь наглядно.

Подведем итог. Блок-схема из конечных автоматов S_1, \dots, S_k , работающих последовательно (т. е. неодновременно), — это конечный автомат S , следовательно, множество автоматов замкнуто относительно условного и безусловного перехода. Мощности входного алфавита и множества состояний S не превосходят суммы мощностей соответственно входных алфавитов и множеств состояний S_1, \dots, S_k . Автомат S называют суммой S_1, \dots, S_k ;

Таблица 8.10

	<i>a</i>	<i>b</i>	<i>c</i>
q_{11}	q_{11}	q_{21}	q_{12}
q_{12}	q_{21}	q_{11}	q_{12}
q_{21}	q_{23}	q_{22}	—
q_{22}	q_{31}	q_{22}	—
q_{23}	q_{21}	q_{23}	—
q_{31}	q_{31}	q_{21}	q_{32}
q_{32}	q_{11}	q_{11}	q_{31}

часто, впрочем, термин «сумма» относят лишь к операции безусловного перехода (т. е. последовательного соединения с неодновременной работой).

Синхронные сети из автоматов. Если автоматы рассматривать как устройства с входами и выходами, то присоединение выходов одних автоматов ко входам других дает *схему*, или *сеть из автоматов*, все автоматы которой работают одновременно. Под состоянием сети из m одновременно работающих автоматов S_1, \dots, S_m (компонент сети) понимается вектор $(q_{i_1}, \dots, q_{i_m})$, где q_{i_j} — состояние автомата S_j . Поэтому в общем случае число возможных состояний сети равно произведению чисел состояний составляющих ее компонент. Возникает вопрос, является ли сеть из автоматов автоматом; если да, то как получить ее автоматное описание из описаний ее подавтоматов.

Прежде всего заметим, что вектор-состояние сети указывает, в каких состояниях находятся компоненты сети в один и тот же момент времени. Таким образом, при описании автоматных сетей — в отличие от абстрактных автоматов — необходимо явно вводить понятие времени. Существуют два основных способа введения времени — *синхронный* и *асинхронный*. Синхронный способ заключается в следующем. Вводится шкала времени, которая делится на отрезки одинаковой длины (такты): границы тактов называются моментами автоматного времени и нумеруются натуральными числами, начиная с нуля. Длина такта принимается за единицу времени. Входное слово (последовательность букв) рассматривается как времененная последовательность сигналов или импульсов (каждый сигнал соответствует букве); интервал между соседними импульсами равен в точности длине такта*. Следовательно, слово длины k занимает во времени ровно k тактов; его буквы можно считать функциями от времени: $a(t)$ — буква, появившаяся на входе в момент t . Автоматные функции δ и λ реализуются с задержкой. Время задержки функции δ равно единице:

* При такой интерпретации импульс является «точечным». Можно считать, что сигнал длится на протяжении всего такта; но тогда в случае, если сигналы в соседних тактах одинаковы, нужны внешние часы, указывающие границы такта.

$\delta(q(t), a(t)) = q(t + 1)$; состояние $q(0)$ определено заранее. Время задержки функции λ обычно считается равным нулю: $\lambda(q(t), a(t)) = v(t)$, но иногда равным единице: $\lambda(q(t), a(t)) = v(t + 1)$; во втором случае должно быть определено $v(0)$. Таким образом, под действием последовательности входных сигналов одинаковой длины каждый из автоматов порождает последовательность промежуточных или внутренних сигналов (реализующих состояния) и последовательность выходных сигналов, причем длины тактов этих последовательностей совпадают с длиной входного такта. Таким образом на практике достигается такая всеобщая синхронизация — за счет внешних синхронизирующих часов либо за счет идеальных временных характеристик автоматов и среды, в которой они функционируют, — на данном уровне рассмотрения несущественно (но, разумеется, становится существенным при физической реализации таких сетей).

Прежде чем говорить о сетях из автоматов общего вида, рассмотрим некоторые основные виды соединения автоматов.

1. Параллельное соединение (рис. 8.10): *a* — с раздельными входами и алфавитами A_1 и A_2 ; *б* — с общим входом и алфавитом A .

Могут сказать, что соединение *a* — это вообще не соединение; тем не менее пару автомата $S_1 = (A_1, Q_1, V_1, \delta_1, \lambda_1)$; $S_2 = (A_2, Q_2, V_2, \delta_2, \lambda_2)$ можно рассматривать как один автомат $S = (A, Q, V, \delta, \lambda)$, который определяется так: $A = A_1 \times A_2$, $Q = Q_1 \times Q_2$, $V = V_1 \times V_2$; следовательно, входной символ автомата S — это пара символов: $a = (a^1, a^2)$, состояние автомата S — это пара состояний: $q = (q^1, q^2)$, где $q^1 \in Q_1$, $q^2 \in Q_2$. Далее $\delta(q, a) = \delta((q^1, q^2), (a^1, a^2)) = (\delta_1(q^1, a^1), \delta_2(q^2, a^2))$, т. е. смена состояний происходит независимо и одновременно; аналогично

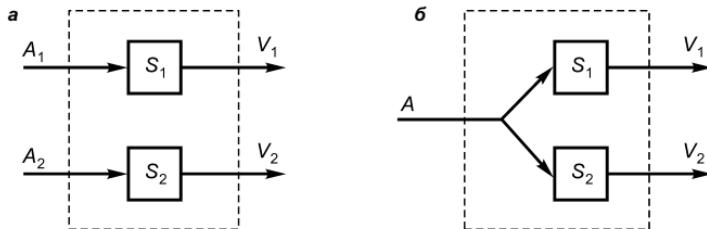


Рис. 8.10

$\lambda(q, a) = (\lambda_1(q^1, a^1), \lambda(q^2, a^2)) = (v^1, v^2)$, где $v_1 \in V_1; v^2 \in V_2$. Автомат S называется *прямым произведением автомата* S_1 и S_2 .

До сих пор речь шла об автоматах с одним входом и одним выходом. Сеть на рис. 8.10, *a* дает пример автомата, о котором удобно говорить, что он имеет несколько входов (в данном случае два) и несколько выходов. В общем случае про автомат, входные символы которого — векторы длины k , говорят, что он имеет k входов; это же касается и выходов. В частности, логический автомат с входными наборами длины m и выходными наборами длины n имеет m двоичных входов и n двоичных выходов.

Автомат на рис. 8.10, *b* определяется аналогично, с той разницей, что входные алфавиты S_1, S_2 и S совпадают; поэтому, например, $\delta(q, a) = (\delta_1(q^1, a), \delta_2(q^2, a))$.

2. Последовательное соединение (рис. 8.11). Эту сеть также можно описать как автомат $S = (A, Q, V, \delta, \lambda)$, причем $A = A_1, V = V_2, V_1 = A_2$; как и прежде, $Q = Q_1 \times Q_2$. Для определения δ и λ существенна задержка функции λ_1 . Если задержка λ_1 равна нулю: $\lambda_1(q^1(t), a(t)) = v^1(t)$, то

$$\begin{aligned} q(t+1) &= (q^1(t+1), q^2(t+1)) = \\ &= (\delta_1(q^1(t), a(t)), \delta_2(q^2(t), \lambda_1(q^1(t), a(t)))). \end{aligned} \quad (8.12)$$

Выражение в правой части зависит только от $q(t) = (q^1(t), q^2(t))$ и $a(t)$ и, следовательно, определяет $q(t+1)$ как функцию от этих переменных. Эта функция и есть функция δ для S : $q(t+1) = \delta(q(t), a(t))$; ее конкретная таблица зависит от таблиц δ_1, δ_2 и λ .

Если же время задержки λ_1 равно единице: $\lambda_1(q^1(t), a(t)) = v^1(t+1)$, то $q(t+1) = \delta_1(q^1(t), a(t)), \delta_2(q^2(t), \lambda(q^1(t-1), a(t-1)))$ и зависимости только от предыдущего такта не получается. Общий метод получения автоматного описания для этого случая будет изложен позже, а пока рассмотрим очень простой пример. Пусть автоматы S_1 и S_2 имеют по одному состоянию, двоичные вход и выход, а на выходе переписывают то, что подано на вход, но с задержкой на такт: $\lambda(q(t), a(t)) = a(t+1)$. В этом случае S_1 и S_2 на рис. 8.11 — это просто устройства задержки на такт или, как говорят, элементы задержки. Для этих элементов необходимо определить начальное значение выхода, положив его равным нулю. Казалось бы, такая

Таблица 8.11

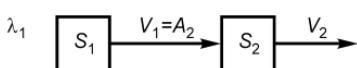


Рис. 8.11

	0	1
q_0	$q_0, 0$	$q_1, 0$
q_1	$q_0, 1$	$q_1, 1$

сеть тривиальна и по свойствам прямого произведения $Q = Q \times Q_2$ должна иметь одно состояние. Однако она осуществляет (при введенной ранее синхронной интерпретации тактов) отображение $S(\alpha) = 00\alpha_{-2}$ (α_{-2} обозначает слово α , у которого отброшены две последние буквы), которое нельзя реализовать автоматом с одним состоянием. Выход заключается в том, чтобы интерпретировать элемент задержки с двоичным входом и выходом как автомат с двумя состояниями, значения которых совпадают со значениями входа. Его таблица переходов приведена в табл. 8.11. В этой таблице предполагается, что λ не имеет задержки: $\lambda(q(t), a(t)) = v(t) = q(t)$. Таким образом, автомат с одним состоянием и единичной задержкой на выходе оказывается эквивалентным автомату с двумя состояниями без задержки на выходе. Но тогда сеть на рис. 8.11 оказывается автоматом с четырьмя состояниями; ее таблица переходов строится из табл. 8.11 по формуле (8.12) и приведена в табл. 8.12, где пары (q_i, q_j) обозначены ij , а значение выхода совпадает с состоянием второго элемента задержки.

Кажется парадоксальным, что цепь из двух элементов задержки удается описать автоматом, в котором функция λ не имеет задержки. Это происходит потому, что задержки «загоняются» в состояния. Аналогично можно описать цепь из любого числа элементов задержки.

3. Обратная связь (рис. 8.12): это соединение заключается в том, что выход автомата S_1 присоединяется к одному из его входов.

Таблица 8.12

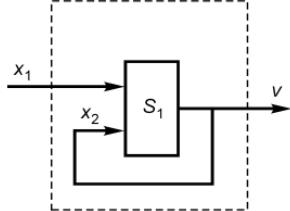


Рис. 8.12

ij	0	1	Выход
00	00	10	0
01	00	10	1
10	01	11	0
11	01	11	1

Рассмотрим случай, когда S_1 — комбинационный автомат без задержки, и предположим, что S_1 реализует логическую функцию $x_1 \bar{x}_2$; вход x_1 оставлен внешним, на вход x_2 подана замкнутая обратная связь. Пусть в момент $t v = 0$; тогда $x_2(t) = 0$ и $v(t) = x_1(t)x_2(t) = 1$, т. е. равен и нулю, и единице в один и тот же момент. Если же $v(t) = 1$, то $x_2(t) = 1$ и при $x_1(t) = 1$ $z(t) = x_1(t)x_2(t) = 0$, т. е. опять получаем противоречие. Предоставляем читателю убедиться, что введение в обратную связь задержки устраняет противоречия.

Этот пример показывает, что сеть из автоматов, содержащая контур обратной связи без задержки, может не иметь конкретной автоматной интерпретации. Правда, это бывает не всегда, но тем не менее сети с контурами обратной связи без задержек в теории автоматов, как правило, исключаются из рассмотрения.

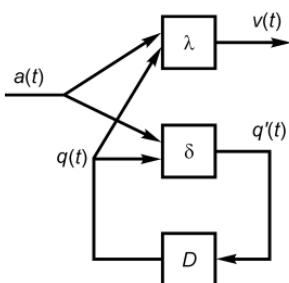


Рис. 8.13

С другой стороны, обратная связь с задержкой оказывается мощным средством построения автоматов. Всякий автомат при синхронной интерпретации может быть реализован как сеть, состоящая из комбинационных автоматов и элементов задержки. Действительно, такая сеть для автомата с функциями $q(t + 1) = \delta(q(t), a(t))$, $v(t) = \lambda(q(t), a(t))$ приведена

на рис. 8.13. На этом рисунке блок λ — комбинационный автомат с входным алфавитом $A \times Q$ и выходным алфавитом V , блок δ — комбинационный автомат с тем же входным алфавитом и выходным алфавитом Q . Блок D — это блок, задерживающий поступающие в него сигналы на один такт (блок задержки). Он представляет собой автомат Мура, входной и выходной алфавиты которого совпадают и равны $Q = \{q_1, \dots, q_n\}$, алфавит состояний $R = \{r_1, \dots, r_n\}$ имеет ту же мощность, что и Q , $\lambda_D(r_i) = q_i$, $\delta_D(r_i, q_j) = r_j$ для всех $i, j = 1, \dots, n$. Частный случай D — двоичный элемент задержки. Сигнал $q'(t)$ на выходе блока δ — это вычисленное значение $(\delta(q(t), a(t)))$, которое, будучи задержано блоком D на такт, появится в момент $t + 1$ на входах блоков δ и λ , т. е. $q'(t) = \delta(q(t), a(t)) = q(t + 1)$.

В важном частном случае, когда алфавиты A , V , Q состоят из двоичных наборов, блоки λ и δ — это логические комбинационные автоматы, двоичные выходы которых в момент t являются логическими функциями от двоичных переменных, образующих наборы $a(t)$ и $q(t)$; блок D — это параллельное соединение двоичных элементов задержки. Число этих элементов равно n — длине вектора Q , а число состояний блока D , как и в общем случае, равно мощности его входного алфавита, т. е. $|Q| = 2^n$.

Поскольку любые конечные алфавиты A , Q , V можно закодировать двоичными кодами подходящей длины (т. е. установить взаимно однозначное соответствие между элементами любого из этих алфавитов и двоичными наборами), получаем один из первых и фундаментальных результатов теории автоматов.

Теорема 8.11. Любой конечный автомат при любом двоичном кодировании его алфавитов A , Q , V может быть реализован синхронной сетью из логических комбинационных автоматов и двоичных задержек, причем число задержек не может быть меньше $\log_2|Q|$. \square

В дальнейшем логические комбинационные автоматы для краткости будем называть логическими блоками (их входы и выходы двоичные), а блок задержки на один такт с одним двоичным входом и выходом — элементом задержки. Вход элемента задержки будем обозначать Y , а выход — y .

Сеть из логических блоков и элементов задержки (и те и другие будем называть блоками) называется правильно построенной логической сетью (ППЛС), если: 1) к каждому входу блока сети присоединен не более чем один выход блока сети (однако допускается присоединение выхода более чем к одному входу, т. е. разветвление выходов); 2) в каждом контуре обратной связи, т. е. в каждом цикле, образованном блоками и соединениями между ними, имеется по крайней мере один элемент задержки. Входами такой сети называются те входы блоков, к которым не присоединены никакие выходы; выходами сети называются те выходы блоков, которые не присоединены ни к каким входам.

Сеть на рис. 8.13 превращается в ППЛС, если алфавиты A , Q , V закодировать двоичными наборами, число

входов и выходов блоков δ , λ , D согласовать с длинами соответствующих наборов, а блок D реализовать, как указано ранее, параллельным соединением двоичных задержек. Поэтому в формулировке теоремы 8.11 фактически подразумевается представление автомата синхронной ППЛС (СЛС) и следующую теорему можно считать обратной теоремой 8.11.

Теорема 8.12. Всякая СЛС со входами x_1, \dots, x_m , выходами z_1, \dots, z_n и k элементами задержки является конечным автоматом, входной алфавит которого состоит из 2^m двоичных наборов длины m , выходной алфавит — из 2^n наборов длины n , множество состояний — из 2^k наборов длины k .

Начнем с рассмотрения СЛС без задержек. Такая сеть по определению не может иметь циклов. Рассмотрим любой ее выход z .

Блок, которому он принадлежит, реализует на этом выходе логическую функцию $z = f^1(p_1^1, \dots, p_r^1)$, где p_1^1, \dots, p_r^1 — входы блока. Каждый из них либо является входом сети, т. е. переменной x , либо присоединен к выходу другого блока: $p_i = f_i^2(p_{i1}^2, \dots, p_{im_i}^2)$ и, следовательно, $z = f^1(f_1^2(p_{11}^2, \dots, p_{1m_1}^2), \dots, f_r^2(p_{r1}^2, \dots, p_{rm_r}^2))$, где вместо некоторых f_j^2 , возможно, стоят переменные x . Еще одно повторение этой процедуры даст суперпозицию глубины три с функциями f_j^3 и переменными p_i^3 и т. д., причем верхний индекс переменной p_i^l равен числу блоков между узлом p_i^l и выходом z . Поскольку сеть ациклическая, этот процесс закончится и все переменные p_i^l будут заменены переменными x . Отсюда (с учетом отсутствия задержек в сети) видно, что $z(t)$ является логической функцией от некоторых $x_1(t), \dots, x_m(t)$ и, следовательно, ППЛС без задержек всегда является логическим комбинационным автоматом.

Пусть теперь дана произвольная СЛС G . Если из нее удалить элементы задержки, то получим ациклическую сеть G_0 без задержек, которая, как только что было показано, является логическим комбинационным автоматом. Входами G_0 служат, во-первых, входы G , а во-вторых, выходы y_1, \dots, y_k элементов задержки G ; выходы G_0 — это выходы G и входы Y_1, \dots, Y_k элементов задержки G (рис. 8.14). Поэтому входной набор G_0 име-

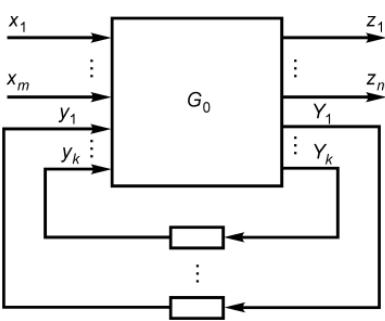


Рис. 8.14

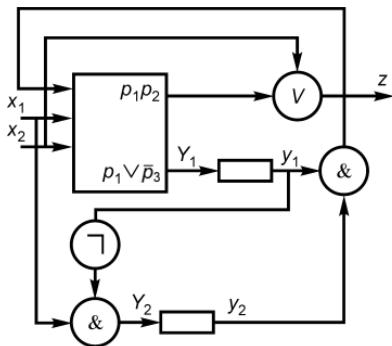


Рис. 8.15

ет вид $(x_1, \dots, x_m, y_1, \dots, y_k)$, выходной набор — $(z_1, \dots, z_n, Y_1, \dots, Y_k)$. Если теперь набор $(x_1(t), \dots, x_m(t))$ считать входным сигналом $a(t)$ сети G , набор $(z_1(t), \dots, z_n(t))$ — выходным сигналом $v(t)$ сети G , а набор $(y_1(t), \dots, y_k(t))$ — состоянием $q(t)$ сети G и учесть, что $(Y_1(t), \dots, Y_k(t)) = (y_1(t+1), \dots, y_k(t+1)) = q(t+1)$, то получим, что сеть G_0 вычисляет две системы логических функций от набора $(x_1, \dots, x_m, y_1, \dots, y_k) = a(t) \times q(t)$ — систему $(Y_1(t), \dots, Y_k(t)) = q(t+1)$, т. е. функцию δ , и систему $z_1(t), \dots, z_n(t)$, т. е. функцию λ . Эти две системы называются *каноническими уравнениями* сети G .

Сеть G в целом принимает вид рис. 8.14, где блоки δ и λ образуют логический блок G_0 (но при этом δ и λ не обязательно отдельные блоки и могут иметь общие части), а блок задержки D состоит из k двоичных задержек. Это и дает автоматное описание СЛС G . \square

Для ациклических СЛС с задержками справедливы следующие два утверждения: 1) любая ациклическая ППЛС реализует определенный автомат (см. пример 8.9, в); 2) любой определенный автомат можно реализовать ациклической СЛС с задержками. Доказательство этих утверждений не очень сложно и предоставляем читателю.

Пример 8.10. Для иллюстрации метода теоремы 8.12 приведем сеть на рис. 8.15, которая содержит три логических блока (один из них — с тремя входами и двумя выходами, реализующими функции p_1p_2 и $p_1 \vee \bar{p}_3$ от входов p_1, p_2, p_3 блока) и две задержки. Канонические уравнения сети: $Y_1 = \bar{x}_2 \vee y_1 y_2$, $Y_2 = x_1 \bar{y}_1$, $z = x_1 y_2 y_2 \vee x_2$.

Переходы автомата, реализуемого этой сетью, приведены в табл. 8.13. Заметим, что первые два состояния этого автомата эквивалентны.

СЛС с единичными задержками и конечные автоматы — адекватные друг другу описания в том смысле, что параметры m , n , k СЛС определяют соответствующие параметры $|A|$, $|V|$, $|Q|$ автомата, и наоборот. Если же попытаться обобщить понятие задержки и считать, что задержки могут быть не только единичными, то связь между k и $|Q|$ становится менее очевидной. Рассмотрим, например, сеть (см. рис. 8.10, а), где S_1 и S_2 — двоичные элементы задержки с величинами τ_1 и τ_2 (τ_1 и τ_2 — произвольные натуральные числа). Со структурной точки зрения сложность этой сети всегда одна и та же: она определяется числом элементов и соединений между ними и не зависит от значений τ_1 и τ_2 . Сложность же соответствующего автоматного описания существенно

Таблица 8.13

y_1y_2	x_1x_2			
	00	01	10	11
00	10,0	00,1	11,0	01,1
01	10,0	00,1	11,0	01,1
10	10,0	00,1	10,0	00,1
11	10,0	10,1	10,1	10,1

зависит от τ_1 и τ_2 . Дело в том, что значения выходов y_1 , y_2 сети в момент t определяются не только значениями входов Y_1 , Y_2 в момент t и выходов в момент $t - 1$; если $y(t) = 0$, а $Y(t) = 1$, то для вычисления $y(t + 1)$ нужно знать, сколько тактов назад Y изменился на единицу. Автоматное описание можно получить, разбив каждую задержку на единичные. Полученный автомат будет иметь $\tau_1 + \tau_2$ единичных задержек и $2^{\tau_1 + \tau_2}$ состояний, откуда видно, что сложность автоматного описания простых сетей с различными целочисленными задержками может быть сколь угодно большой. Поэтому для описания синхронных сетей с нетривиальными временными зависимостями применяются методы, не использующие понятие абстрактного автомата.

Описание поведения асинхронных логических сетей, в которых значениями задержек могут быть произвольные действительные числа, дано в первом издании настоящей книги. Здесь отметим лишь, что в общем случае это поведение не может быть описано никаким конечным автоматом.

Синтез автоматных сетей. Рассмотренные ранее методы получения единого автоматного описания для сетей из автоматов называются методами *композиции* автоматов. Методы композиции решают задачу анализа сетей, поскольку исследование поведения сетей и, в частности, реализуемого сетью отображения удобно проводить при наличии автоматного описания: таблицы переходов, графа переходов или системы канонических уравнений. Обратная задача — построение сети по заданному автомата — называется задачей *декомпозиции* или разложения автоматов. Возможны различные задачи декомпозиции в зависимости от того, какие условия накладываются на искомую автоматную сеть. Например, существуют методы разложения автомата на сеть из автоматов, соединенных заданным образом — последовательно, параллельно и т. д. Некоторые из этих методов и литература по ним приведены в [21, т. II, § 8.4].

Наиболее практически важной и хорошо изученной задачей декомпозиции является задача реализации автомата в заданном автоматном базисе, т. е. задача построения сети, реализующей данный автомат, из заданного набора автоматов (автоматного базиса), называемых в этом случае элементарными автоматами, или просто элементами. Эта задача обычно называется задачей *структурного синтеза* или структурной реализации автоматов, а получаемые сети — *схемами из функциональных элементов*. Из проблем, которые здесь возникают, кратко коснемся двух основных: 1) любой ли автомат S можно реализовать схемой из множества элементов Σ (если да, то Σ называется автоматно-полным набором элементов); 2) среди всех схем в базисе Σ , реализующих S , найти минимальную схему. Начнем со структурного синтеза логических комбинационных автоматов, т. е. со схемной реализации систем логических функций. Если функция f задана формулой F над множеством функций Σ (см. гл. 3), то формуле F можно поставить в соответствие схему G из комбинационных автоматов, реализующих функции Σ . Построение схемы G определяется индукцией по глубине формулы: 1) если $F = \phi(x_{i_1}, \dots, x_{i_k})$, где $\phi \in \Sigma$, x_{i_1}, \dots, x_{i_k} — исходные переменные, то схема G состоит из одного элемента ϕ , входы которого отождествлены

с переменными x_{i_1}, \dots, x_{i_k} , а выход — с переменной z ;
 2) если $F = \phi(F_1, \dots, F_k)$, где F_i — переменная x_{j_i} или функция, уже реализованная схемой G_i , то схема G для F строится так: к i -му входу элемента ϕ присоединяется выход схемы G_i (если F_i — функция) или переменная (если $F_i = x_{j_i}$), выходом z схемы G объявляется выход элемента ϕ . Например, формуле $x_1x_2x_3 \vee x_2x_3x_4$ соответствует схема на рис. 8.16, а.

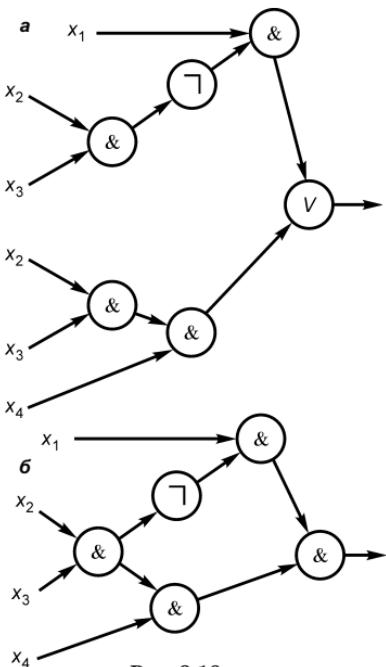


Рис. 8.16

Схема, полученная описанным методом, всегда имеет вид ориентированного дерева; ее входы соответствуют концевым вершинам, а выход — корню дерева. Между множеством формул над Σ и множеством древовидных схем из элементов, реализующих функции Σ , существует взаимно однозначное соответствие: по любой формуле F над Σ описанный метод однозначно строит схему G ; с другой стороны, анализ схемы G , проведенный методом теоремы 8.12, даст исходную формулу F (второй факт и дает основание утверждать, что G действительно

реализует F). Число знаков операций в F равно числу элементов в G . Это обстоятельство сводит задачу преобразования (и, в частности, минимизации) древовидных схем к задаче преобразования логических формул, рассматривавшейся в гл. 3.

Итак, по формуле над Σ всегда можно построить древовидную схему из элементов Σ ; обратное утверждение в силу теоремы 8.11 верно для произвольных (а не только древовидных) схем над Σ . Отсюда получаем следующий важный, хотя и очевидный, факт: *для того чтобы произвольная функция могла быть реализована схемой над Σ , необходимо и достаточно, чтобы множество функций Σ было функционально полным в соответствии с из-*

ложенным в § 3.3. Ясно, что для реализации системы функций ответ в точности тот же.

Вторая задача — задача минимизации — оказывается гораздо более сложной. Проблема минимизации формул сама по себе довольно трудна, однако она не исчерпывает всех возможностей минимизации схем. Например, схема на рис. 8.16, *b* реализует ту же формулу, что и рис. 8.16, *a*, однако схема *b* не древовидная и имеет меньше элементов, чем любая формула в булевом базисе.

До настоящего времени известно очень небольшое количество классов функций, для которых найдены минимальные схемные реализации. Исследования в этой области дают — пока косвенные — свидетельства того, что в общем случае поиск минимальных схемных решений невозможен без большого перебора и практически не реализуем; достаточно точно оценить по данной функции хотя бы число элементов в минимальной схеме — не проводя синтеза — также не удается. Поэтому теоретические исследования задачи синтеза пошли по пути глобальных характеристик сложности схем. Основной результат здесь формулируется следующим образом.

Пусть $L_\Sigma(f)$ — число элементов минимальной схемы в базисе Σ , реализующей функцию f . Обозначим

$$L_\Sigma(n) = \max_{f \in P_2(n)} L_\Sigma(f),$$

где максимум берется по всем функциям от n переменных. Функция $L_\Sigma(n)$ называется *функцией Шеннона* для базиса Σ ; она равна минимальному числу элементов из Σ , достаточному для реализации любой функции от n переменных.

Теорема 8.13 (Шеннон, Лупанов). Для любого базиса Σ

$$L_\Sigma(n) \sim c_\Sigma \frac{2^n}{n},$$

где c_Σ — константа, зависящая от базиса, причем для любого $\varepsilon > 0$ доля функций f , для которых

$$L_\Sigma(f) \leq (1 - \varepsilon)c_\Sigma \frac{2^n}{n},$$

стремится к нулю с ростом n . \square

Здесь знак \sim обозначает асимптотическое равенство

$$(a(n) \sim b(n), \text{ если } \lim_{n \rightarrow \infty} \frac{a(n)}{b(n)} = 1).$$

Смысл второго утверждения теоремы в том, что с ростом n почти все функции реализуются со сложностью, близкой к верхней границе, т. е. $L(n)$.

Доказательство этой теоремы содержится, например, в [47].

Для автоматов общего вида уже задача существования схемы в заданном наборе Σ оказывается довольно трудной. Правда, из обсуждения схемной реализации логических функций и теоремы 8.11 следует, что набор K элементов, состоящий из элемента единичной задержки и любого функционального полного набора логических элементов, является автоматно полным. Однако в общем случае проблема автоматной полноты для произвольного набора автомата оказывается алгоритмически неразрешимой (в отличие от проблемы функциональной полноты для логических функций, решение которой дается теоремой 3.9). Сравнительно простое доказательство этого результата имеется в [49], где данная проблема сводится к проблеме эквивалентности слов в ассоциативных исчислениях.

Практические методы синтеза автоматов разработаны в основном для набора K , описанного ранее, либо для наборов, полученных из набора K заменой элемента задержки на какой-либо другой элементарный автомат. Долгое время основной считалась каноническая схема синтеза Хаффмена–Глушкова, которая разбивает процесс структурной реализации автоматов на следующие этапы: 1) построение автомата по какому-либо описанию автоматного отображения (например, по регулярному событию); 2) минимизация числа состояний автомата; 3) двоичное кодирование состояний (а также входных и выходных алфавитов, если исходный автомат абстрактный, а не логический) и получение канонических уравнений будущей сети, описывающих блоки δ и λ как системы логических функций; 4) реализация системы канонических уравнений схемой в функционально полном базисе. Число элементов задержки определяется на этапе 3, число логических элементов — на этапе 4.

Эта схема сыграла большую роль в развитии методов синтеза автоматов. Однако, как показал 30-летний опыт, надежды на ее широкое практическое использование ока-

зались сильно преувеличенными. Это объясняется в основном следующими причинами. Во-первых, как было установлено ранее, схемы с k элементами задержки имеют 2^k состояний, поэтому описание сколько-нибудь больших схем в терминах состояний и таблиц переходов оказывается довольно громоздким. Во-вторых, на разных этапах решаются разные задачи минимизации, которые плохо связаны между собой. Известны примеры, когда уменьшение числа состояний приводит к усложнению логики схемы. Кроме того, различные варианты кодирования (для k задержек существует $(2^k)!$ вариантов) приводят к разным системам канонических уравнений; предвидеть сложность их реализации заранее, на этапе кодирования, невозможно. Поэтому в практике получают все большее распространение методы, обходящие каноническую схему синтеза.

8.4. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ЛОГИЧЕСКИХ ФУНКЦИЙ И АВТОМАТОВ

Представление автомата схемой из элементов — это исторически первый и наиболее исследованный вид структурной реализации автомата. Другой ее вид — реализация автомата программой. Здесь мы ограничимся вопросами программной реализации комбинационных логических автоматов, т. е. систем логических функций.

Под программой будем понимать пронумерованную последовательность команд k_1, \dots, k_s , взятых из некоторого фиксированного набора (системы команд). Программа работает над конечным множеством пронумерованных (или проименованных) двоичных ячеек. Номер (или имя) ячейки называется ее адресом; именем ячейки часто будет служить имя логической переменной, значения которой хранятся в этой ячейке.

Система команд содержит команды-операторы вида $b := f(a_1, \dots, a_p)$ (выполнить операцию f над содержимым ячеек a_1, \dots, a_p и результат положить в ячейку b) и двухадресные условные переходы двух видов: 1) «если a , то i , иначе j » (если $a = 1$, то перейти к выполнению команды k_i , иначе перейти к k_j); 2) «если \bar{a} ($a = 0$), то i , иначе j ».

Операция $f(a_1, \dots, a_p)$ — это логическая функция p переменных; в частности, она может быть константой 0 или 1. Если j — номер следующей команды, то переход можно считать одноадресным: «если a , то i (иначе перейти к следующей команде)», а если $i = j$, то безусловным: «перейти к i ». Любая из команд указанных типов может быть заключительной, что указывается словом «конец».

Процессом вычисления программы k_1, \dots, k_s называется последовательность шагов $k(1), k(2), \dots, k(t)$, на каждом из которых выполняется одна команда программы. Эта последовательность определяется так: 1) $k(1) = k_1$; 2) если $k(i) = k_r$ — оператор, то $k(i + 1) = k_{r+1}$; 3) если $k(i)$ — условный переход, то номер команды $k(i + 1)$ указывается этим переходом; 4) если $k(i)$ — заключительная команда, то процесс вычисления останавливается после ее выполнения. Число t называется временем вычисления.

Программа Π вычисляет (или реализует) логическую функцию $f(x_1, \dots, x_n) = y$, если для любого двоичного набора $\sigma = (\sigma_1, \dots, \sigma_n)$ при начальном состоянии памяти $x_1 = \sigma_1, x_2 = \sigma_2, \dots, x_n = \sigma_n$ (состояние остальных ячеек несущественно) программа через конечное число шагов останавливается и при этом в ячейке y лежит величина $f(\sigma_1, \dots, \sigma_n)$.

Если под сложностью схемы, реализующей автомат, обычно понимается число элементов в схеме (см. § 8.3), то сложность программы можно понимать в различных смыслах: 1) число команд в тексте программы; 2) объем промежуточной памяти, т. е. число ячеек для хранения промежуточных результатов вычислений; 3) время вычисления, которое будет характеризоваться двумя величинами: средним временем

$$t_{\text{ср}}(\Pi) = \frac{1}{2^n} \sum_{\sigma} \tau_P(\sigma)$$

и максимальным временем

$$t_{\text{макс}}(\Pi) = \max_{\sigma} \tau_P(\sigma),$$

где сумма и максимум берутся по всем 2^n двоичным наборам σ , а τ_P — время работы программы Π на наборе σ .

Любой схеме, реализующей функцию f и содержащей N элементов, нетрудно поставить в соответствие программу, реализующую f и состоящую из N команд, сле-

дующим образом. Занумеруем элементы схемы числами $1, 2, \dots, n$ так, чтобы на любом пути от входа к выходу номера элементов возрастили; при этом номер 1 получит один из входных элементов, а номер N — выходной элемент. Пусть элемент e_i схемы реализует функцию φ_i и к его входам присоединены выходы элементов e_{j_1}, \dots, e_{j_p} (некоторые из них, возможно, являются входами схемы). Поставим элементу e_i в соответствие либо ячейку a_i и команду $a_i = \varphi_i(a_{j_1} \dots a_{j_p})$, если $i \neq N$; либо ячейку y и команду « $y := \varphi_N(a_{j_1} \dots a_{j_p})$ конец», если $i = N$. Получим программу, не содержащую условных переходов (такие программы будем называть *операторными*), в которой порядок команд в точности соответствует нумерации элементов в схеме, а система команд — базису схемы.

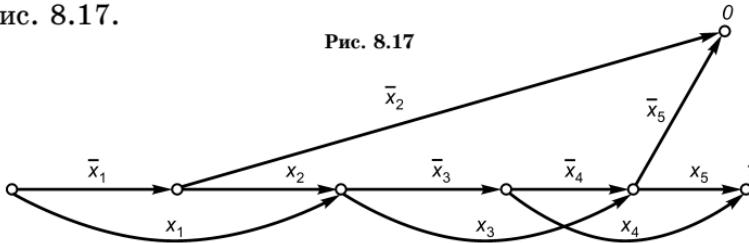
Проблемы синтеза операторных программ в основном сводятся к проблемам синтеза схем: в частности, вопросы функциональной полноты системы команд и минимизации собственного текста операторной программы совпадают соответственно с задачами о функциональной полноте системы функций и о минимизации схем. Поскольку операторная программа не содержит условных переходов, время ее вычисления на любом наборе одно и то же; отсюда $t_{\max} = t_{\text{ср}} = N$, т. е. оба вида временной сложности совпадают со сложностью текста программы и, следовательно, в силу теоремы 8.13 при достаточно больших n почти для всех функций близки к $2^n/n$. Набором, проблема минимизации памяти (за счет многократного использования одной ячейки для нескольких промежуточных результатов) для операторных программ является нетривиальной комбинаторной задачей.

Другой «крайний» вид программ, вычисляющих логические функции, — это программы, состоящие из команд вида $y := \sigma$ ($\sigma = 0$ или $\sigma = 1$) и условных переходов. Такие программы называются *бинарными программами*. Всякую булеву формулу F , содержащую N букв, можно реализовать бинарной программой, вычисляющей F за время $t_{\max} = N$ и содержащей N команд условного перехода (а также две команды $y := 0$ и $y := 1$, которые в оценках не учитываются). Для описания метода реализации используем представление программы в виде графа (в котором вершины соответствуют командам, а ребра —

переходам). Пусть G_1 — граф программы для функций f_1 с начальной вершиной v_{10} и двумя заключительными вершинами v_{1z}^0 (с командой « $y = 0$ ») и v_{1z}^1 (с командой « $y = 1$ »); G_2 — граф программы для f_2 с началом v_{20} и концами v_{2z}^0 и v_{2z}^1 . Тогда: 1) программа, граф G которой получен присоединением G_2 к «нулю» G_1 (т. е. отождествлением вершин v_{1z}^0 и v_{20} ; команда « $y: = 0$ » при этом отбрасывается), вычисляет функцию $f = f_1 \vee f_2$; 2) программа, граф G' которой получен присоединением G_2 к «единице» G_1 (отождествлением v_{1z}^1 и v_{20}), вычисляет $f = f_1 \& f_2$; 3) программа, граф которой получен из G_1 заменой команд в v_{1z}^0 и v_{1z}^1 на инверсные (« $y: = 0$ » на « $y: = 1$ » и наоборот), вычисляет \bar{f}_1 . Заметим, что в графе G получаются две единичные, а в графе G' — две нулевые заключительные вершины. В обоих случаях их надо отождествить.

Пример 8.11. Формула $(x_1 \vee x_2)(\bar{x}_3x_4 \vee x_5)$ реализуется бинарной программой, граф которой приведен на рис. 8.17.

Рис. 8.17



Описанный метод не гарантирует минимальность получаемых программ по времени и числу команд. Существуют другие методы, которые, в частности, для любой функции n переменных дают бинарную программу с $t_{\max} \leq n$ независимо от длины исходной формулы. В общем же случае сложность бинарных программ характеризуется следующими асимптотическими оценками. Пусть $L_B(n)$ — функция Шеннона для числа команд бинарных программ (см. аналогичное определение для $L_\Sigma(n)$ в конце § 8.3).

Теорема 8.14. 1) $L_B(n) \sim 2^n/n$, причем существует метод синтеза бинарных программ, удовлетворяющих этой оценке, для которых $t_{\max} \sim n$; 2) для любого $\varepsilon > 0$ доля функций, для которых: а) $L_B(f) \leq (1 - \varepsilon)2^n/n$; б) $t_{\text{cp}}(f) \leq (1 - \varepsilon)n$, стремится к 0 при $n \rightarrow \infty$. \square

Таким образом, сложность бинарных программ, вычисляющих логические функции, по числу команд асимптотически равна сложности операторных программ. Однако бинарные программы обладают двумя важными достоинствами, которых нет у операторных программ. Первое из них — это отсутствие промежуточной памяти в процессе работы программы. Оно связано с тем, что команды условного перехода обращаются только к значениям входных переменных, которые не меняются в процессе вычисления; новых же значений переменных эти команды не создают. Это позволяет реализовать бинарные программы на постоянной памяти. Второе достоинство бинарных программ — более высокое быстродействие, т. е. меньшие величины t_{\max} и t_{cp} . Напомним, что для операторных программ $t_{\max} = t_{\text{cp}} = N$. Для времени вычисления бинарных программ справедливы следующие утверждения.

1. Для любой функции f , существенно зависящей от всех n переменных, $\lceil \log_2 n + 1 \rceil \leq t_{\max}(f) \leq n$, причем обе границы достигаются.

2. Существуют последовательности функций $f_1(x_1)$, $f_2(x_1, x_2)$, ..., $f_n(x_1, \dots, x_n)$..., для которых $\lim_{n \rightarrow \infty} t_{\text{cp}}(f_n) = 2$; иначе говоря, с ростом n $t_{\text{cp}}(f_n)$ практически не увеличивается. Примером такой последовательности является любая последовательность f_1, \dots, f_n ..., где $f_1 = x_1, \dots, f_{n+1} = x_{n+1} \circ f_n$ (\circ — дизъюнкция или конъюнкция).

3. Для любой системы m логических функций n переменных существует бинарная программа, вычисляющая ее за время, не превосходящее $n + m$. (В отличие от оценок времени для вычисления одной функции здесь число команд вида $y := \sigma$ не меньше m и входит в оценку.)

Как уже отмечалось, число команд в бинарных и операторных программах асимптотически равно $2^n/n$. Если же в программе использовать и операторы, и условные переходы, то эта оценка понижается вдвое.

Более подробные сведения об оценках сложности и методах программной реализации логических функций и автоматов можно получить в [45]. Здесь отметим лишь, что *логический автомат всегда может быть реализован бинарной программой, не требующей промежуточной памяти*.

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

ОСНОВНАЯ ЛИТЕРАТУРА

1. Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. Т. 1, 2. М.: Мир, 1978.
2. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов. М.: Мир, 1979.
3. Братчиков И. Л. Синтаксис языков программирования. М.: Наука, 1972.
4. Бунос Дж., Джейффи Р. Вычислимость и логика. М.: Мир, 1994.
5. Гиндикин С. Г. Алгебра логики в задачах. М.: Наука, 1972.
6. Гладкий А. В. Математическая логика. М.: РГГУ, 1998.
7. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982.
8. Дискретная математика и математические вопросы кибернетики. Т. 1 / Под общей ред. С. В. Яблонского и О. Б. Лупанова. М.: Наука, 1974.
9. Зыков А. А. Основы теории графов. М.: Наука, 1987.
10. Катленд Н. Вычислимость: Введение в теорию рекурсивных функций. М.: Мир, 1983.
11. Клини С. К. Математическая логика. М.: Мир, 1973.
12. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. М.: МЦНМО, 2002.
13. Лавров И. А., Максимова Л. Л. Задачи по теории множеств, математической логике и теории алгоритмов. М.: Физматлит, 2001.
14. Линдон Р. Заметки по логике. М.: Мир, 1968.
15. Майника Э. Алгоритмы оптимизации на сетях и графах. М.: Мир, 1981.
16. Мальцев А. И. Алгебраические системы. М.: Наука, 1970.
17. Мальцев А. И. Алгоритмы и рекурсивные функции. М.: Наука, 1986.
18. Манин Ю. И. Вычислимое и невычислимое. М.: Сов. Радио, 1980.
19. Мартин-Леф П. Очерки по конструктивной математике. М.: Мир, 1975.
20. Мендельсон Э. Введение в математическую логику. М.: Наука, 1984.
21. Миллер Р. Теория переключательных схем. Т. 1, 2. М.: Наука, 1970, 1971.
22. Минский М. Вычисления и автоматы. М.: Мир, 1975.
23. Новиков П. С. Элементы математической логики. М.: Физматгиз, 1959.
24. Оре О. Теория графов. М.: Наука, 1980.
25. Рейнгольд Э., Нивергельт Ю., Део Н. Комбинаторные алгоритмы. Теория и практика. М.: Мир, 1980.
26. Робертс Ф. С. Дискретные математические модели с приложениями к социальным, биологическим и экологическим задачам. М.: Наука, 1986.

27. Роджерс Х. Теория рекурсивных функций и эффективная вычислимость. М.: Мир, 1972.
28. Сачков В. Н. Введение в комбинаторные методы дискретной математики. М.: Наука, 1982.
29. Свами М., Тхуласираман К. Графы, сети, алгоритмы. М.: Мир, 1984.
30. Смальян Р. Теория формальных систем. М.: Наука, 1981.
31. Таран Т. А. Основы дискретной математики. Киев: Просвіта, 2003.
32. Трахтенброт Б. А. Алгоритмы и вычислительные автоматы. М.: Сов. Радио, 1974.
33. Трахтенброт Б. А., Барздинь Я. М. Конечные автоматы. Поведение и синтез. М.: Наука, 1970.
34. Харари Ф. Теория графов. М.: Мир, 1973.
35. Холкрофт Дж., Мотвани Р., Ульман Дж. Введение в теорию автоматов, языков и вычислений. М.: Издательский дом «Вильямс», 2002.
36. Чень Ч., Ли Р. Математическая логика и автоматическое доказательство теорем. М.: Наука, 1983.
37. Шалыто Л. А. Логическое управление. Методы аппаратной и программной реализации алгоритмов. СПб.: Наука, 2000.
38. Шиханович Ю. А. Введение в современную математику. М.: Наука, 1965.
39. Шоломов Л. А. Основы теории дискретных логических и вычислительных устройств. М.: Наука, 1980.
40. Шрейдер Ю. А. Равенство, сходство, порядок. М.: Наука, 1971.

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

41. Варшавский В. И., Поспелов Д. А. Оркестр играет без дирижера. М.: Наука, 1984.
42. Гилл А. Введение в теорию конечных автоматов. М.: Наука, 1966.
43. Китаев А., Шень А., Вяльй М. Классические и квантовые вычисления. М.: МЦНМО, ЧеРо, 1999.
44. Кратко М. И. Формальные исчисления Поста и конечные автоматы // Проблемы кибернетики. М.: Физматгиз, 1966. Вып. 17. С. 41–65.
45. Кузнецов О. П. О программной реализации логических функций и автоматов // Автоматика и телемеханика. 1977. №7. С. 163–174. № 9. С. 137–149.
46. Ландебер П. Проблемы разрешения в грамматиках непосредственных составляющих // Кибернетический сборник. Новая серия. М.: Мир, 1973.
47. Лупанов О. Б. О синтезе некоторых классов управляющих систем // Проблемы кибернетики. М.: Физматгиз, 1963. Вып. 10. С. 63–97.
48. Марков А. А., Нагорный Н. М. Теория алгорифмов. М.: Наука, 1984.
49. Орлов В. Л. Простое доказательство неразрешимости некоторых задач о полноте автоматных базисов // Кибернетика. 1973. № 4. С. 109–113.
50. Цетлин М. Л. Исследования по теории автоматов и моделированию биологических систем. М.: Наука, 1969.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

А

- Автомат 328
 - автономный 331
 - асинхронный 362
 - инициальный 328
 - комбинационный 366
 - логический 366
 - Мили 347
 - Мура 347
 - недетерминированный 356
 - сильно связный 331
 - частичный (неполностью определенный) 338
- Аксиома 243
- Аксиоматизируемость 271
- Алгебра 35
 - булева логических функций 58, 65
 - множеств 36, 65
 - Жегалкина 73
 - логики 50, 212
 - регулярных событий 351
- Алгоритм 151, 212
 - нормальный Маркова 163, 290
- Алфавит 13, 155, 243
- Арифметика формальная 263
- Ассоциативность 37

Б

- Базис линейного пространства 122
 - Блок-схема алгоритма 159
 - машины Тьюринга 175
 - Булеан 36
-
- ## В
- Вектор 12, 124
 - линейного пространства 121, 124
 - циклический 120

- Векторное пространство
 - (см. линейное пространство)
- Вершина 91
 - концевая 94
- Вывод 244
 - в грамматике 294
- Вычислительная сложность 227, 230

Г

- Глубина формулы 54
- Гомоморфизм 38
 - автоматов 332
- Грамматика 293
 - блочная 320
 - контекстная 296
 - контекстно-свободная 296
 - неоднозначная 305
 - неукорачивающая 298
 - приведенная 308
 - регулярная 296
 - типа 0 296
- Граф 91
 - ациклический 109
 - вершинно-порожденный 94
 - двудольный 125
 - знаковый 126
 - неориентированный 91
 - однородный 94
 - ориентированный 96
 - переходов 323
 - планарный 128
 - полный 92
 - простой 91
 - пустой 92
 - реберно-порожденный 95
 - связный 101
 - сильно связный 107
 - эйлеров 103
- Группа 44

- Д**
- Данные 154, 164
 - Двойственность 64
 - Декомпозиция
 - автоматов 379
 - Дерево 114
 - вывода 302
 - ориентированное 115
 - покрывающее 118, 137
 - Детерминизация 356
 - Диагональный метод 20
 - Диаметр графа 102
 - Дизъюнкция 53
 - Дистрибутивность 38
 - Длина пути 100
 - Доказательство 89, 244
 - Дополнение графа 95
 - множества 12
 - отношения 29
 - Достижимость 100, 107
- Е**
- Единица полугруппы 41
 - решетки 49
- З**
- Задача комбинаторная 233
 - NP-полная 238
 - о выполнности КНФ 239
 - — коммивояжере 106, 229
 - — кратчайшем пути 138
 - — максимальном паросочетании 126, 150
 - — потоке в сети 146
 - — сетевом планировании 142
 - Замкнутость 35, 74
 - Звезда 95
- И**
- Изоморфизм 39
 - автоматов 332
 - графов 98, 239
 - Импликант 69
 - Импликация 53
 - Интервал 70
 - Интерпретация 89, 266
 - Инцидентность 93
 - Источник (вершина графа) 110
 - (недетерминированный автомат) 354, 356
 - Исчисление 243
 - ассоциативное 279
 - высказываний 245
 - предикатов 254
 - — с равенством 262
 - Итерация 316, 352
- К**
- Квантор общности 85, 254
 - существования 85, 254
 - Класс NP 235
- P 235
 - эквивалентности
- Л**
- Лес 114
 - Линейная комбинация 122
 - независимость 122
 - Линейное пространство 121
 - Логика высказываний 84
 - Логика предикатов 83
- М**
- Матрица
 - инцидентности 93
 - отношения 28
 - смежности 93, 111
 - Машина Тьюринга 163
 - многоленточная 187
 - — недетерминированная 236
 - — универсальная 177
 - Метатеорема 244
 - Множество 4
 - бесконечное 5
 - конечное 5
 - континуальное 20
 - перечислимое 219
 - разрешимое 218
 - счетное 18
 - Модель (алгебраическая система) 46
 - Модель теории 267
 - Монoid 41
 - Мощность множества 6, 20
 - Мультиграф 92
- Н**
- NP-полнота 238
 - Неотличимость 334
 - Непротиворечивость 268
 - Неразрешимость
 - алгоритмическая 186, 215
 - Нормальная
 - форма Бэкуса 300
 - дизъюнктивная (ДНФ) 63
 - — совершенная (СДНФ) 57
 - — конъюнктивная (КНФ) 64
 - Нумерация алгоритмов 213

O

- Область значений 15, 21
 - истинности 82
 - определения 15, 21
- Образ 15
- Образующая 43
- Общезначимость 87, 268
- Объединение графов 95
 - множеств 9
 - отношений 29
 - элементов решетки 46
- Оператор (операция над функциями) 189
 - автоматный 330
 - наименьшего числа (μ -оператор) 196
 - примитивно-рекурсивный (ПР-оператор) 194, 203
- Операция 35
- Отношение 27
 - бинарное 27
 - обратное 29
 - порядка 33
 - рефлексивное 29
 - симметричное 30
 - транзитивное 31
 - эквивалентности 31
- Отображение 21
 - автоматное 330
- Отицание 52

П

- Паросочетание 126
- Переменная
 - несущественная 51
 - свободная 85
 - связанная 85
- Пересечение графов 95
 - множеств 10
 - элементов решетки 46
- Петля 92
- Подалгебра 35
- Подграф 94
- Подмножество 5
- Подформула 54
- Подстановка 23, 60, 278
- Покрытие 70
 - вершинное 135
- Поле 35, 36
- Полином Жегалкина 73
- Полиномиальная сводимость 237
 - сложность 231
- Полнота формальной теории 271
 - функциональная 71, 79
- Полугруппа 41
 - свободная 43
- Полупуть 107
- Поток 146
- Правило вывода 243
 - MP 247

— подстановки 247

- Предикат 82
- примитивно-рекурсивный 193
- Предметная область 82
- Проблема остановки 185
 - самоприменимости 215
 - соответствия (комбинаторная проблема Поста) 289
 - эквивалентности алгоритмов 225
 - — слов в полугруппе 282
- Программа бинарная 385
- Продукция 283
- Проекция 15
- Прообраз 15
- Пространство циклов 122
- Противоречивость 87, 268
- Прямое произведение автоматов 372
 - — множеств 12
- Путь 98, 107

P

- Радиус графа 103
- Разбиение 10, 48
- Размерность вектора 124
 - векторного пространства 122
- Разность множеств 11
 - симметрическая графов 96
 - множеств 11
- Разрешимость формальной теории 271
- Расстояние в графе 102
- Ребро 91
 - кратное 92
- Рекурсия двойная 202
 - кратная 203
 - одновременная 197
 - примитивная 191
- Решетка 46
 - разбиений 48

C

- Самодвойственность 64
- Самоприменимость 215
- Связность 100, 107
- Семантика 226, 268
 - формальных языков 325
- Сетевой график 142
- Сеть из автоматов 370
 - — языков 322
- Синтаксис 226, 268
- Система команд 164
 - подстановок (полусистема Туз) 278
 - Поста каноническая 282
 - — нормальная 287
 - Туз 279
 - формальная 240, 277
- Слово 14, 282, 348
- Сложение по модулю 35, 53

Смежность 92
Событие 348
— асинхронное 353
— определенное 353
— регулярное 352
Соответствие 15
— взаимно однозначное 16
— инъективное 15
— сюръективное 15
— функциональное 16
Сортировка топологическая 110
Степень вершины 94
Сток 110
Стрелка Пирса 53
Суперпозиция 23, 54
Сходимость алгоритма 159

Т

Тезис Тьюринга 184
— Черча 205
Теорема Геделя о неполноте
вторая 272
— — — первая 272
— — — полноте исчисления
предикатов 271
— Кантора 20
— Кэли 45
— Маркова–Поста 281
— — функциональной
полноте 79, 81
— Полла–Ангера 341
— Понтрягина–Куратовского
132
— Райса 224
— формальной теории 244
— Черча 272
— Шеннона–Лупанова 381
— Эйлера 104, 129
Теория формальная 243
Терм 254
Транзитивное замыкание 30,
114

У

Упорядочение
лесикографическое 34
Условный переход 174, 383
Устойчивость множества вершин
внешняя 135
— — — внутренняя 134
— состояния автомата 362

Ф

Формула 24
— булева 58
— выполнимая 87
— исчисления 243
— общезначимая 87, 268
— противоречивая 87, 268
Функция 21
— Аккермана 201
— выходов 328
— вычислимая 212
— — по Тьюрингу 167
— логическая (алгебры логики) 50
— — линейная 74
— — монотонная 75
— обратная 22
— общерекурсивная 205
— переходов 328
— перечисляющая 219
— примитивно-рекурсивная 191
— рекурсивная 162, 189
— характеристическая 193
— частично-рекурсивная 204
— Шеннона 381

Ц

Центр графа 103
— дерева 116
Цепочка 293
Цепь 100
— простая 100
Цикл 100
— гамильтонов 106
— простой 100
— эйлеров 103
Цикломатическое число 118

Ч

Часть графа 94

Ш

Штрих Шеффера 53

Э

Эквивалентность 53
— автоматов 334
— алгоритмов 225
— грамматик 294
— формул 55, 87

Я

Язык формальный 235, 294

ОГЛАВЛЕНИЕ

Предисловие к третьему изданию	3
<i>Г л а в а п е р в а я.</i>	
Множества, функции, отношения	4
1.1. Множества и операции над ними	4
1.2. Соответствия и функции	15
1.3. Отношения	27
<i>Г л а в а в т о р а я.</i>	
Элементы общей алгебры	35
2.1. Операции на множествах и их свойства	35
2.2. Полугруппы, группы, решетки	41
<i>Г л а в а т р е т ъ я.</i>	
Введение в логику	50
3.1. Логические функции (функции алгебры логики)	50
3.2. Булева алгебра	56
3.3. Полнота и замкнутость	71
3.4. Язык логики предикатов	82
<i>Г л а в а ч е т в е р т а я.</i>	
Графы	91
4.1. Основные понятия	91
4.2. Пути и связность в неориентированных графах	99
4.3. Пути и связность в ориентированных графах	107
4.4. Деревья	114
4.5. Пространство циклов	118
4.6. Двудольные и планарные графы	125
4.7. Раскраски, устойчивость, покрытия	132
4.8. Оптимизационные задачи на графах	137
<i>Г л а в а п ят а я.</i>	
Теория алгоритмов	151
5.1. Предварительное обсуждение	151
5.2. Машины Тьюринга	163
5.3. Рекурсивные функции	188
5.4. Вычислимость и разрешимость	211
5.5. Вычислительная сложность и NP-трудные задачи	227

<i>Г л а в а ш е с т а я.</i>	
Формальные системы	240
6.1. Формальные теории (логические исчисления). Исчисление высказываний	242
6.2. Исчисление предикатов и теории первого порядка	254
6.3. Метатеория логических исчислений	265
6.4. Абстрактные формальные системы	274
<i>Г л а в а с е д ь м а я.</i>	
Языки и грамматики	291
7.1. Формальные грамматики и их свойства	293
7.2. Операции над языками	315
7.3. О семантике формальных языков	324
<i>Г л а в а в о с ь м а я.</i>	
Автоматы	328
8.1. Основные понятия	328
8.2. Распознавание множеств автоматами	347
8.3. Сети из автоматов, их анализ и синтез	366
8.4. Программная реализация логических функций и автоматов	383
Рекомендуемая литература	388
Основная литература	388
Дополнительная литература	389
Предметный указатель	390

Олег Петрович КУЗНЕЦОВ

**ДИСКРЕТНАЯ МАТЕМАТИКА
ДЛЯ ИНЖЕНЕРА**

Издание шестое,
стереотипное

Художественный редактор *С. Л. Шапиро*
Редактор *И. Л. Яновская*
Корректоры *И. А. Короткова, А. К. Райхчин*
Подготовка иллюстраций *В. В. Воскресенская*
Верстальщик *С. Ю. Малахов*
Выпускающие *Н. К. Белякова, О. В. Шилкова*

ЛР № 065466 от 21.10.97
Гигиенический сертификат 78.01.07.953.П.004173.04.07
от 26.04.2007 г., выдан ЦГСЭН в СПб

Издательство «ЛАНЬ»
lan@lpbl.spb.ru; www.lanbook.com
192029, Санкт-Петербург, Общественный пер., 5.
Тел./факс: (812)412-29-35, 412-05-97, 412-92-72.
Бесплатный звонок по России: 8-800-700-40-71

Подписано в печать 28.04.09.
Бумага офсетная. Гарнитура Школьная. Формат 84×108 1/32.
Печать офсетная. Усл. п. л. 21. Тираж 2000 экз.

Заказ № .

Отпечатано в полном соответствии
с качеством предоставленных диапозитивов
в ОАО «Издательско-полиграфическое предприятие «Правда Севера».
163002, г. Архангельск, пр. Новгородский, д. 32.
Тел./факс (8182) 64-14-54; www.ippps.ru