О. П. КУЗНЕЦОВ
Г. М. АДЕЛЬСОН-ВЕЛЬСКИЙ

ДИСКРЕТНАЯ МАТЕМАТИКА ДЛЯ ИНЖЕНЕРА

Кузнецов О. П., Адельсон-Вельский Г. М.
Дискретная математика для инженера. — 2-е изд., перераб. и доп. — М.: Энергоатомиздат, 1988.— 480 с.: ил.

ISBN 5-283-01563-7

Изложены основные понятия теории множеств, общей алгебры, логики, теории графов, теории алгоритмов и формальных систем. По сравнению с изданием 1980 г. существенно переработана и расширена глава по сложности вычислений, добавлен раздел о раскраске графов, включены новые главы по теории формальных языков и линейному программированию.

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

ПРЕДИСЛОВИЕ КО ВТОРОМУ ИЗДАНИЮ

Второе издание книги существенно дополнено. В гл. 4 добавлен раздел, посвященный знаменитой задаче о четырех красках и ее решению, полученному недавно с помощью ЭВМ. Написаны две новые главы: гл. 7 «Языки и грамматики» (О. П. Кузнецов) и гл. 10 «Линейное программирование» (Г. М. Адельсон-Вельский).

В гл. 8 («Автоматы») добавлен раздел о связи между грамматиками и автоматами. Существенно изменена и расширена глава о сложности вычислений (гл. 9 во втором издании). В ней дано изложение и сопоставление двух независимо сложившихся подходов к классификации комбинаторных задач по сложности: хорошо известного в американской литературе NP-подхода, использующего недетерминированные машины Тьюринга и понятие WP-полноты, и подхода, сложившегося в советских работах и основанного на понятии задачи со степенной проверкой. Обсуждению этих подходов предшествует сравнение сложности вычислений одной и той же задачи на абстрактных машинах разных типов. Определения и доказательства в этой главе довольно сложны и громоздки, поскольку они по существу связаны с программированием на абстрактных машинах. Эта глава более трудна для чтения, чем другие.

Подробнее, чем в первом издании, описаны некоторые «хорошие» комбинаторные задачи. В конце гл. 10 приведено полученное недавно доказательство полиномиальности задачи линейного программирования.

Обновлен и дополнен список литературы. Как и в первом издании, 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.

ПАМЯТИ
АРКАДИЯ МАКАРЕВСКОГО

ПРЕДИСЛОВИЕ

к изданию 1980 года

Предлагаемая книга посвящена дискретной математике. И она действительно предназначена для инженеров, хотя из оглавления этого не видно. Дело в том, что излагаемая здесь математика, строго говоря, не является прикладной. В книге нет главного, что ценится в технике — в ней не сказано, как и где это применять, и какая от этого будет польза. Тем не менее она ориентирована на тех, кто занимается техническими науками, и вот почему.

Общеизвестно, что прикладная математика опирается на чистую математику. В книгах по электротехнике не принято объяснять, что такое интеграл по контуру или векторное произведение: само собой разумеется, что нужные сведения читатель получил из курса математического анализа; верно и обратное — курс математического анализа технических вузов, оставаясь чисто математическим, ориентирован на прикладные задачи, которые придется решать его слушателям. Однако в дискретной математике обе эти традиции еще не установились. Прикладные книги обычно начинаются «с азов», объясняя, что такое дизъюнкция и цикл в графе, а курсы чистой дискретной математики, предназначенные для инженеров (будущих или настоящих), хотя и существуют в виде учебных программ и циклов лекций, почти не реализованы в литературе. Одна из главных целей этой книги — попытка создать такой курс. В ней содержатся почти все основные разделы «невероятностной» дискретной математики, за исключением линейного и дискретного программирования.

У книги есть и еще одна цель. Среди практиков очень распространен взгляд на математику как на большой справочник, который нужно уметь открыть на нужной странице. Инженеры любят формулы и методы, но не любят теорем и тем более — их доказательств (да и в соответствующей литературе их обычно печатают более мелким шрифтом). И это понятно. При утилитарном подходе к математике знание доказательства ничего не добавляет к зна-

нию результата: важно знать, «что» и «зачем», но очень редко — «как» и «почему». Такой подход тем не менее может оправдывать себя в областях с давно установившимися моделями объектов и процессов. Однако в задачах управления все чаще главной научной проблемой становится не работа с существующими моделями, а создание новых моделей: вчера это было сетевое планирование и логика цифровых схем, сегодня это — применение формальных грамматик в языках программирования, завтра это будет размытая логика и т. д. В такой ситуации математика нужна уже не как метод расчета, а как метод мышления, как язык, как средство формулирования и организации понятий. Такое владение математикой требует существенно большей культуры: понимания важности точных формулировок и умения обходиться без них там, где они мешают пониманию существа дела, «чувства нетривиальности» — умения понять, что просто, что сложно, а что невозможно, ощущения связи между, казалось бы, далекими идеями и понятиями.

Мы хотели бы надеяться, что наша книга приблизит читателя именно к такой математике. В книге мало методов, но много определений и теорем. Мы старались отбирать их так, чтобы они использовались как можно в большем числе мест, иногда приводятся разные формулировки одних и тех же понятий, чтобы дать почувствовать связи, пронизывающие дискретную математику. Эту же цель преследует и «рваный ритм» книги — наряду с педантичными определениями и выкладками в ней имеются пространные неформальные рассуждения, побуждающие читателя к размышлениям над прочитанным.

В книге можно выделить четыре основных раздела (не вполне соответствующих порядку глав): 1) язык дискретной математики, 2) логические функции и автоматы, 3) теория алгоритмов и формальные системы, 4) графы и дискретные экстремальные задачи.

Первый раздел, посвященный множествам и алгебрам (гл. 1 и 2), довольно традиционен (языку теории множеств сейчас учат уже в школе); по существу он служит развернутым словарем для остальной части книги. Его «языковый» характер проявляется, в частности, в том, что в нем очень много определений и примеров, но почти нет теорем. Примеры подбирались как можно более широко, чтобы продемонстрировать универсальность языка теории множеств. Читатель, знакомый с этим языком, может пропу-

стить первые две главы, возвращаясь к ним в дальнейшем в случае необходимости.

Логика и автоматы (гл. 3 и 7) также уже стали традиционным разделом курсов дискретной математики. Однако в отличие от большинства книг, из которых инженер черпает сведения по этим вопросам, здесь они изложены как главы чистой математики. В главе об автоматах используются понятия теории алгоритмов и формальных систем. Гораздо меньше внимания, чем принято, уделено вопросам синтеза схем и автоматов. На это есть по крайней мере две причины. Во-первых, значительная часть этих вопросов связана с конкретными приложениями к проектированию цифровых устройств и выходит за пределы собственно математики; кроме того, на эту тему имеется обширная литература, с которой нет смысла конкурировать. Во-вторых, теория схем и автоматов, пройдя почти полный «биологический цикл» на глазах одного поколения исследователей, представляет собой весьма поучительный пример того, как сильно подвержены моральному старению отрасли технических наук, слабо связанные с фундаментальными теориями. Еще 10 лет назад все кибернетические журналы были заполнены статьями по минимизации и синтезу. Большинство из них сейчас забыто. А те идеи и результаты теории автоматов, которые в то время считались непрактичными — задачи о распознавании множеств автоматами, асимптотическая теория схем Шеннона—Яблонского—Лупанова и т. д.,— выдержали проверку временем. Именно они находят сейчас применение в различных областях кибернетики — таких, как системное программирование, сложность вычислений, искусственный интеллект.

Теория алгоритмов и формальных систем — центральный раздел книги. Эти вопросы, наоборот, традиция относит к «высокой науке», считая их мало связанными с практикой и трудными для понимания. Первый из этих предрассудков медленно, но верно опровергается развитием теоретической кибернетики. Помимо того, что возникли чисто прикладные ответвления этой теории, связанные с алгоритмическими языками программирования и сложностью вычислений, знание основных неразрешимостей теории алгоритмов и принципов организации формальных исчислений становится существенным элементом математической культуры любого исследователя, имеющего отношение к алгоритмизации процессов управления. По существу оно дает понимание того, что можно и чего нельзя сделать

с помощью вычислительных машин. Сейчас, когда вычислительных машин больше, чем людей, умеющих правильно их использовать, такое понимание особенно важно. Что же касается предрассудка о трудности понимания (который в значительной степени происходит от того, что книги по теории алгоритмов написаны математиками для математиков), то опровергающим примером служат две превосходные книги Б. А. Трахтенброта и М. Минского (см. библиографию к гл. 5 и 6). Вообще, на наш взгляд, инженер-исследователь, работающий с задачами управления и переработки информации, особенно, если он имеет дело с ЭВМ, подготовлен к восприятию основных идей теории алгоритмов ничуть не хуже, чем математик. Более того, при изложении этих идей в прикладной книге появляется благодарная возможность апеллировать к программистской интуиции читателя, которая для понимания здесь не менее важна, чем запас обычных математических знаний.

Раздел, содержащий сведения о графах, состоит из гл. 4 и 8. Глава 4, так же как и первые две главы, — в основном языковая. Большинство понятий, излагаемых здесь, достаточно хорошо известно, поскольку графы благодаря своей наглядности и универсальности за короткое время стали, пожалуй, наиболее распространенным языком задач управления. Напротив, материал гл. 8 не является элементарным и содержится в основном в специальной литературе. Кроме самостоятельного теоретического и прикладного содержания, которое имеют излагаемые здесь экстремальные задачи и методы их решения, представляет особый интерес взгляд на эти задачи с общетеоретической точки зрения, и прежде всего с точки зрения теории алгоритмов. Здесь речь идет о сравнительно новом круге проблем, постепенно становящимся одной из важнейших в дискретной математике в целом — сложности вычислений. Важность результатов в этой области — не только в том, что они помогают оценить расход времени и памяти при решении задач на ЭВМ. Подобно тому, как общая теория алгоритмов впервые показала, что бывают задачи неразрешимые, ее бурно развивающаяся ветвь — теория сложности постепенно приводит к пониманию того, что бывают задачи объективно сложные (пример — так называемые универсальные задачи перебора), причем сложность может оказаться в некотором смысле абсолютной, т. е. практически не устранимой никаким увеличением мощности вычислительных средств. Рассмотрение этих задач поучительно еще

и потому, что почти все они имеют характерный для дискретной математики эффект сочетания простоты формулировки со сложностью решения.

Главы последнего раздела (гл. 4 и 8) написаны Г. М. Адельсоном-Вельским, остальные главы — О. П. Кузнецовым.

К сожалению, объем книги не позволил включить в нее задачи. Некоторой компенсацией за это может служить внимательный разбор многочисленных примеров и настоятельно рекомендуемое читателю восстановление опущенных мест в доказательствах (впрочем, мы не настаиваем на этой рекомендации по отношению к теоремам Черча и Геделя в гл. 6).

Литература сгруппирована в четыре списка, соответствующих указанным разделам; все они помещены в конце книги. Учитывая широкое назначение книги, мы стремились сделать библиографию компактной и общедоступной. Поэтому она состоит в основном из книг на русском языке; журнальные статьи указаны лишь в необходимых случаях.

А. Я. Макаревский, много и плодотворно работавший в теории автоматов, был одним из инициаторов написания книги. Неожиданная смерть в 37 лет помешала ему начать работу над ней. Памяти об этом замечательном человеке и исследователе посвящают авторы эту книгу.

Авторы

ГЛАВА ПЕРВАЯ

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 $a\in\overline{M}$.

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, [35]); 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 «Арарат» (т. е. множество ее футболистов).

M_7 — множество всех футбольных команд высшей лиги в сезоне 1985 г.

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

* Обозначения M_2—M_7 сохраняются до конца параграфа.

тельно содержит неопределяемые 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 бесконечного множества — более сложное понятие. Оно будет рассмотрено после введения понятия соответствия.

Множество мощности 0, т. е. не содержащее элементов, называется пустым и обозначается $\varnothing$. Принято считать, что пустое множество является подмножеством любого множества. Пустое множество введено в математике для удобства и единообразия 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 $N=1, 2, 3...$ — 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 $M_{2^n}=1, 2, 4, 8, 16...$, 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: $\pi = 3{,}1415926536...$ 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, заключающего-

ся 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 $M=\{x\,|\,P(x)\}$, которое читается так: «M — это множество x, обладающих свойством P». Например, $M_{2^n}=\{x\,|\,x=2^k$, где $k\in N_0\}$, $M_4=\{x\,|\,x=\pi/2\pm k\pi$, 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 «Арарат». 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» нуж-

дается 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 устраняли 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:

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

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 $\bigcup\limits_{A \in S} A$, которое читается так: «объединение всех множеств A, принадлежащих совокупности S». Если же все множества совокупности занумерованы индексами, то используются другие варианты обозначений $\bigcup\limits_{i=1}^{k} A_i$ (для случая, когда $S = \{A_1, A_2, \ldots, A_k\}$), $\bigcup\limits_{i=1}^{\infty} A_i$ (для случая, когда S — бесконечная совокупность и ее множества занумерованы подряд натуральными числами), $\bigcup\limits_{i \in I} A_i$ (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: $M_7 = \{\Phi_1, \Phi_2, \ldots \Phi_{16}\}$. Тогда $\bigcup\limits_{i=1}^{16} \Phi_i$ — 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 $\bigcup\limits_{i \in P} N_i$ — 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. $\bigcap\limits_{i=1}^{16}\Phi_i=\varnothing$; более того, для любых i и j $\Phi_i\cap\Phi_j=\varnothing$.

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 $\bigcup\limits_{i=1}^{16}\Phi_i$ может входить только в одну команду. Подробнее о разбиениях см. § 1.3 и 2.2.

г. $\bigcap\limits_{i\in P} N_i=\varnothing$ (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 $A\setminus B=\varnothing$, то $A\subseteq B$.

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

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

в. $M_7\setminus M_6$ — множество всех команд высшей лиги, за исключением «Арарата». Эта запись не вызывает разночтений, 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 $\bigcap\limits_{i=1}^{16}\Phi_i\setminus M_6$ (множество всех футболистов высшей лиги,

не выступающих в «Арарате») 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 M_2 очевидно, что $\bar{M}_2$ — это множество натуральных чисел, бо́льших 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, неопределяемым. 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 $(a_1, \ldots, a_m)$ и $(b_1, \ldots, b_n)$ равны, если $m=n$ и $a_1=b_1$, $a_2=b_2$, ..., $a_m=b_m$.

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, ..., A_n (обозначение $A_1\times ... \times A_n$) называется множество всех векторов $(a_1, ..., a_n)$ длины n, таких, что $a_1\in A_1$, ..., $a_n\in A_n$. $A\times ...\times A$ 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. $A=\{a, b, c, d, e, f, g, h\}$, $B=\{1, 2, ..., 8\}$. Тогда $A\times B=$ $=\{a1, a2, a3, ..., h7, h8\}$ — 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 матриц 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 $A^*=\bigcup\limits_{i\in N} A^i=A^1\bigcup A^2\bigcup A^3\bigcup ...$ При написа-

нии 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 $\{0, 1, ..., 9\}$. Текст, напечатанный на пишущей машинке, является словом в алфавите, определяемом клавиатурой данной машинки (включая знаки препинания и пробел!).

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

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

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, $|A_1\times A_2\times ...\times A_k|=m_1m_2$, ..., m_k. Возьмем любой вектор $(a_1, ..., a_k)$ из $A_1\times A_2\times ...\times A_k$ 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 $A_1\times A_2\times ...\times A_{k+1}$. Таким образом, из всех $m_1...m_k$ векторов приписыванием справа элемента из A_{k+1} можно получить $m_1...m_km_{k+1}$ векторов из $A_1\times A_2\times ...\times A_{k+1}$, причем все они различны, и никаких других векторов в $A_1\times A_2\times ...\times A_{k+1}$ не содержится. Поэтому для $n=k+1$ теорема верна и, следовательно, верна для любых n. □[1]

Следствие, $|A^n|=|A|^n$.

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

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

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: $\mathrm{пр}_i V=\{\mathrm{пр}_i v \mid v\in V\}$. Аналогично определяется проекция множества V на несколько осей: $\mathrm{пр}_{i_1, ..., i_k} V=\{\mathrm{пр}_{i_1, ..., i_k} v \mid v\in V\}$. В частности,

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

если $V=A_1\times A_2\times\ldots\times A_n$, то $\mathrm{пр}_{i_1,\ldots,i_k}V=A_{i_1}\times\ldots\times A_{i_k}$. Отметим, что в общем случае $\mathrm{пр}_iV$ — 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. $V=\{(a,b,d),(c,b,d),(d,b,b)\}$, $\mathrm{пр}_1V=\{a,c,d\}$, $\mathrm{пр}_2V=\{b\}$, $\mathrm{пр}_{2,3}V=\{(b,d),(b,b)\}$.

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 $\mathrm{пр}_1G$ называется *областью определения* соответствия, множество $\mathrm{пр}_2G$ — *областью значений* соответствия. Если $\mathrm{пр}_1G=A$, то соответствие называется *всюду определенным* или полностью определенным (в противном случае соответствие называется частичным); если $\mathrm{пр}_2G=B$, то соответствие называется *сюръективным*.

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 $C\subseteq\mathrm{пр}_1G$, то образом множества C называется объединение образов всех элементов C. Аналогично определяется прообраз множества D для любого $D\subseteq\mathrm{пр}_2G$.

Соответствие G называется *функциональным* (или однозначным), если образом любого элемента из $\mathrm{пр}_1G$ является единственный элемент из $\mathrm{пр}_2G$. Соответствие G между A и B называется *взаимно однозначным* (иногда пишут «1-1-соответствие»), если оно всюду определено, сюръективно, функционально, и, кроме того, прообразом любого элемента из $\mathrm{пр}_2G$ является единственный элемент из $\mathrm{пр}_1G$.

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 $(x-3)^2+(y-2)^2\leqslant 1$, 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 меж-

ду действительными числами на том же рис. 1.1 служит дуга ABC.

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), как это сделано в [6]. 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.

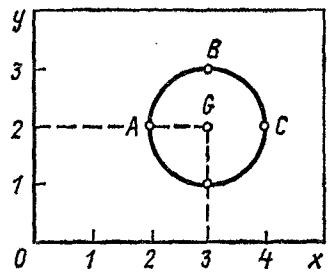

Рис. 1.1

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[1].

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

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

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 г. 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 единственность образа), либо $|A| < |B|$, и тогда, поскольку отображение сюръективно, в B найдутся два элемента, соответствующих одному и тому же $a \in A$ (нарушится единственность прообраза)[1].

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

---

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

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

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 $|B_n|=2^n$. □

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 множество 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 $N_i^2 = \{(a, b) \mid a+b=i+1\}$. 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 $1+2+...+(i-1)+a$. Эта нумерация и доказывает счетность N^2, из которой, в свою очередь, непосредственно следует счетность множества P положительных рациональных чисел, т. е. дробей

вида a/b, где a и b — натуральные числа[1]. 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, ..., 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 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.

Множество всех действительных чисел отрезка [0, 1] не является счетным *(теорема Кантора)*. Действительно, предположим, что оно счетно и существует его нумерация. Расположим все числа, изображенные бесконечными десятичными дробями, в порядке этой нумерации:

$$\begin{array}{l} 0,\ a_{11}\ a_{12}\ a_{13}\ \ldots \\ 0,\ a_{21}\ a_{22}\ a_{23}\ \ldots \\ 0,\ a_{31}\ a_{32}\ a_{33}\ \ldots \\ \ldots\ldots\ldots \end{array}$$

Рассмотрим любую бесконечную десятичную дробь $0,\ b_1,\ b_2,\ b_3\ldots$, 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 отрезка [0, 1] не могут быть пронумерованы, и множество всех действительных чисел отрезка [0, 1] *несчетно*. Его мощность называется *континуум*; множества такой мощности называются *континуальными*. Метод, использованный при доказательстве, называется *диагональным методом* Кантора.

Множество всех подмножеств счетного множества кон-

---

[1] На примере множества P видно, что нумерация числового множества может не иметь ничего общего с упорядочением его элементов по величине. В множестве P 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.

тинуально. 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 отрезка [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 $A \to B$ (обозначение $f: A \to B$). 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: $A \to B$ называется *отображением* A и 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 $A \to A$ 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 $N \to N$, и полностью определена, если ее тип $N \to R$ или $R_+ \to R$ (R_+ — положительное подмножество R).

г. Пусть зафиксирован список $\{a_1, ..., a_n\}$ всех элементов конечного множества A. Тогда любой вектор $v_i = (a_{i_1}, ..., a_{i_n})$ из A^n можно рассматривать как описание функции f_i: $A \to A$ (т. е. преобразования A), определяемой следующим образом: $f_i(a_j) = a_{i_j}$, 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 $N \to N$ 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 неубывания), то получим функцию $q(n)$ типа $N \to \bigcup_{i=1}^{\infty} N^i$, 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 $A_1 \times A_2 \times ... \times A_n \to B$ называется *n-местной функцией*. В этом случае принято считать, что функция имеет n аргументов: $f(a_1, a_2, ..., a_n) = b$, 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 $R^2 \to R$. Таблица выигрышей лотереи задает двухместную не полностью определенную функцию, которая устанавливает соответствие между парами из N^2 (серия, номер) и множеством выигрышей.

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

Пример 1.9. а. Функция $\sin x$ имеет тип $R \to R$. 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 кодирующей функции 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 $f: A \to B$ и $g: B \to C$. Функция h: $A \to C$ 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 $f: A^m \to B$, $g: B^n \to C$ 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 $B \times A^3 \times B^2 \to C$, 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 $A^6 \times B^2 \to C$. Особый интерес представляет случай, когда задано множество функций типа $f_1: A^{m_1} \to A, \ldots, f_n: A^{m_n} \to A$. 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 $f_1, \ldots, f_n$ некоторой подстановкой их друг в друга и переименованием аргументов, на-

зывается *суперпозицией* $f_1, \ldots, f_n$. 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 $R \to R$, 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 $k=0, \pm 1, \pm 2\ldots$ Таким образом, область определения композиции может быть уже областей определения обеих исходных функций и даже казаться пустой.

б. Множество $K=\{k_1, \ldots, k_m\}$ команд ЭВМ отображается в машинные коды этой ЭВМ, т. е. в натуральные числа. Кодирующая функция φ имеет тип $K \to N$. 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_3 в x_2 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: $\alpha=(1\to 3, 2\to 3, 3\to 1)$ и $\beta=(1\to 2, 2\to 1, 3\to 1)$. 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 преоб-

разований — 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 заданий 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 значение 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 $x \notin M_\pi$, 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: $A \to B$ — 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 $R \subseteq M^n$ называется *n-местным отношением* на множестве M. Говорят, что $a_1, \ldots, a_n$ находятся в отно-

шении R, если $(a_1, ..., a_n) \in R$. 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 $\leqslant$ 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. Отношения 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: «жить 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 принадлежа-

щие M_1. Иначе говоря, $R' = R \cap M_1^2$. Строго говоря, R и R' — это разные отношения, с разными областями определения. Однако если не возникает разночтений, этот педантизм не соблюдается; например, вполне можно говорить об отношении «быть делителем», не уточняя, задано оно на N или каком-либо его подмножестве [1].

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 $M = \{a_1, ..., a_m\}$ — это квадратная матрица C порядка m, в которой элемент c_{ij}, стоящий на пересечении i-й строки и j-го столбца, определяется следующим образом:

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

Например, для конечного множества {1, 2, 3, 4, 5, 6} матрицы отношений *1—3* из примера 1.11,а приведены в табл. 1.1, *а—в* 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.

Таблица 1.1

	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

а)

	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

б)

	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

в)

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 $\leqslant$ является объединением отношений $<$ и равенства.

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

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

Свойства отношений. Отношение R называется *рефлексивным*, если для любого $a\in M$ имеет место aRa. Главная диагональ его матрицы содержит только единицы. Отношение R называется антирефлексивным, если ни для какого $a\in M$ не выполняется aRa. Главная диагональ его матрицы содержит только нули. Отношение $\leqslant$ 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 [1]: если первая точка симметрична второй, то и вторая симметрична первой. Пример антисимметричного отношения — отношение $\leqslant$: действительно, если $a\leqslant b$ и $b\leqslant a$, 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», $\leqslant$, «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.

---

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

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 $a=a_1, a_2, ..., a_{n-1}, a_n=b$, в которой между соседними элементами выполнено R: $aRa_2, a_2Ra_3, ..., a_{n-1}Rb$. Если R транзитивно, то $\hat{R}=R$. Действительно, если aRb, то $a\hat{R}b$ (цепочка состоит из двух элементов a, b), поэтому $R\subseteq\hat{R}$. Если же $a\hat{R}b$, то существует цепочка $aRa_2, a_2Ra_3, ..., a_{n-1}Rb$. Но так как R транзитивно [1], то aRb, поэтому $\hat{R}\subseteq R$. Из включения в обе стороны следует $R=\hat{R}$.

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 описы-

[1] 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.

ваются). 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,.., (возможно, бесконечная) такая, что любой элемент из M входит хотя бы в один класс, т. е. $\bigcup_i C_i = M$. 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 $a_1 R a_2$, 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...; 1, 8, 15...; 2, 9, 16...; ...; 6, 13, 20...

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 полностью упорядоченным, если любые два элемента M сравнимы, и частично упорядоченным в противном случае.

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

б. Определим отношения $\leqslant$ и $<$ на R^n следующим образом: $(a_1, ..., a_n) \leqslant (b_1, ..., b_n)$, если $a_1 \leqslant b_1, ..., a_n \leqslant b_n$; $(a_1, ..., a_n) < (b_1, ..., b_n)$, если $(a_1, ..., a_n) \leqslant (b_1, ..., b_n)$ 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 полное 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 $\alpha_1 = a_{11} \ldots a_{1m}$ и $\alpha_2 = a_{21} \ldots a_{2m}$. Тогда $\alpha_1 \prec \alpha_2$, если и только если либо 1) $\alpha_1 = \beta a_i \gamma$, $\alpha_2 = \beta a_j \delta$ и $a_i \prec a_j$ (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 полное упорядочение множества всех конечных слов в алфавите A, которое называется лексико-графическим *упорядочением* слов.

Пример 1.15. а. Наиболее известным примером лексикографического упорядочения является упорядочение слов в словарях. Например, лес $\prec$ лето (случай 1 определения: β = лес, с $\prec$ т, γ пусто, δ = о), поэтому слово «лес» расположено в словаре раньше слова «лето», лес $\prec$ лесть (случай 2 определения: β = ть).

б. Если рассматривать числа в позиционных системах счисления (например, в двоичной или десятичной) как слова в алфавите цифр, то их лексико-графическое 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 выравнять 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 ЭВМ.

в. Лексико-графическое 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 лексико-графически «старше» третьего числа любого года. Чтобы возрастание дат совпадало с лексико-графическим упорядочением, обычное представление надо «перевернуть», т. е. годы поместить слева: 86.08.05. Так обычно делают при представлении дат в памяти ЭВМ.

ГЛАВА ВТОРАЯ

ЭЛЕМЕНТЫ ОБЩЕЙ АЛГЕБРЫ

2.1. ОПЕРАЦИИ НА МНОЖЕСТВАХ И ИХ СВОЙСТВА

Алгебры. Функцию типа φ: $M^n \to M$ будем называть *n-арной операцией* на множестве M; n называется *арностью* операции φ. Множество M вместе с заданной на нем совокупностью операций $\Omega = \{\varphi_1, \ldots, \varphi_m, \ldots\}$, т. е. система $A = (M; \varphi_1, \ldots, \varphi_m, \ldots)$, называется *алгеброй*; M называется *основным,* или несущим, *множеством* (или просто носителем) алгебры A. Вектор арностей операций алгебры называется ее типом, 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 $\varphi_1, \ldots, \varphi_m \ldots$ алгебры A, то система $A' = (M', \varphi_1, \ldots, \varphi_m \ldots)$ называется *подалгеброй* A (при этом $\varphi_1, \ldots, \varphi_m \ldots$ 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 $N_p = \{0, 1, 2, \ldots, p-1\}$. 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 $N_p = \{0, 1, \ldots, 6\}$, $3 \oplus 4 = 0$, $3 \odot 4 = 5$, $4 \oplus 6 = 3$. Часто операции $\oplus$ и $\odot$ обозначают как $a+b \equiv c \pmod p$, $a \cdot b = d \pmod p$. Если p — простое число, то алгебра $\{N_p, \oplus, \odot\}$ называется *конечным полем характеристики p.*

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

г. Множество F одноместных функций на R, т.е. функций $f:R\to R$, вместе с операцией дифференцирования является алгеброй. Элементы основного множества — функции типа $R\to R$, единственной операцией этой алгебры служит дифференцирование — унарная операция типа $F\to F$ (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, $\pi/2$, π, $3\pi/2$, 2π, $5\pi/2$..., однако они задают всего четыре различных отображения множества вершин в себя, сооответствующих первым четырем поворотам. Таким образом, получаем алгебру с основным множеством $\{a_1, a_2, a_3, a_4\}$ и четырьмя унарными операциями α, β, γ, δ. Их можно задать табл. 2.1, в которой на пересечении, например, строки a_3 и столбца γ написано значение функции $\gamma(a_3)$.

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

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

е. Множество $O=\{\alpha, \beta, \gamma, \delta\}$ отображений вершин в себя из предыдущего примера вместе с бинарной операцией композиции отображений (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 $\{a, \gamma\}$, т. е. повороты на углы 0, π, 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

$$a\varphi b = b\varphi a.$$

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

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

и *дистрибутивной справа* относительно ψ, если

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

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 $A=(K; \varphi_1, \ldots, \varphi_p)$ и $B=(M; \psi_1, \ldots, \psi_p)$ одинакового типа. *Гомоморфизмом* алгебры A в алгебру B называется отображение Γ: $K \to M$, удовлетворяющее условию

$$\Gamma\left(\varphi_i\left(k_{j_1}, \ldots, k_{j_{l(i)}}\right)\right) = \psi_i\left(\Gamma\left(k_{j_1}\right), \ldots, \Gamma\left(k_{j_{l(i)}}\right)\right) \quad (2.1)$$

для всех $i=1, \ldots, p$[$l(i)$ — арность операций φ_i и ψ_i, которая у них по условию одинакова] и всех $k_{j_r} \in K$. Смысл усвия (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_i, результат будет одинаков.

Изоморфизмом алгебры A на алгебру B называется взаимно однозначный гомоморфизм. В этом случае существует обратное отображение Γ^{-1}: $M \to K$, 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:

$$\varphi_i\left(\Gamma^{-1}\left(m_{i_1}\right), \ldots, \Gamma^{-1}\left(m_{i_{l(i)}}\right)\right) = \Gamma^{-1}\psi_i\left(m_{i_1}, \ldots, m_{i_{l(i)}}\right)). \quad (2.2)$$

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 $\Gamma_{2n}: n \to 2n$, 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 $\Gamma_{-n}: n \to (-n)$ 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 $\Gamma_7: N \to N_7$ 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 $a \to \log a$. Условие (2.1) имеет вид равенства $\log ab = \log a + \log b$.

г. Рассмотрим алгебры $(K,\ \varphi)$ и $(M,\ \psi)$, 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 следующими таблицами (табл. 2.3, а, б).

Таблица 2.3

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

а

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

б)

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

5. Булевы 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'.

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 существо, 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 совпадет 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 мультипликативной. 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 $e_1 e_2 = e_1$ и $e_1 e_2 = e_2$; 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 $\alpha = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 2 \end{pmatrix}$ и $\beta = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 3 & 2 \end{pmatrix}$. Их произведения имеют вид $\alpha\beta = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 3 & 3 \end{pmatrix}$ и $\beta\alpha = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 2 & 2 \end{pmatrix}$, 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 $\gamma = \beta^2 = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 2 & 3 \end{pmatrix}$, $\delta = \alpha\beta$ и $\zeta = \beta\alpha$, то можно убедиться, что полученное множество $\Gamma = \{\alpha, \beta, \gamma, \delta, \zeta\}$ вместе с операцией композиции образует полугруппу. Таблица Кэли этой полугруппы 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.

Таблица 2.4

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

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

Действительно, пусть задана полугруппа с множеством $M = \{e, a_1, a_2 \ldots\}$. 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 произ-

ведению $a_i a_j$ будет соответствовать преобразование $f_{ij}(x) = xa_i a_j = f_i(x) a_j = f_j(f_i(x))$, 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 $a_i \to f_i(x)$ 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 $\gamma = \beta^2$, $\delta = \alpha\beta$, $\zeta = \beta\alpha$. 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 $\{a_1, \ldots, a_n\}$. Если обозначения операции опустить (как это обычно делается для умножения), то все элементы P можно рассматривать как слова в алфавите $\{a_1, \ldots, a_n\}$. 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 перестановок в списке элементов M, то порядок Σ_M равен $|M|!$ Симметричная 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. □

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, что́ 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 $S = \{s_1, \ldots, s_n\}$, на вход которой может быть подано входное воздействие из множества $\{x_1, \ldots, x_m\}$. 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 $\{x_1, \ldots, x_m\}$. 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 [4]. 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 $\leqslant$. Для элементов a и b из M их верхней гранью называется любой элемент $c \in M$, такой, что $c \geqslant a, c \geqslant b$, а их нижней гранью — любой элемент $d \in M$, такой, что $d \leqslant a$, $d \leqslant b$. В общем случае для некоторых элементов a и b верхняя или нижняя грань может не существовать или быть неединственной, причем различные верхние (или нижние) грани могут быть несравнимы.

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 $\{M; \leqslant; \cap, \cup\}$ 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 полностью 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: $a \leqslant b$, 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: $M_i \leqslant M_j$, 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: $v \leqslant w$, если в векторе w единицы стоят на всех тех местах, на которых они стоят в v (и, быть может, еще на некоторых). Например, $(010) \leqslant (011)$, а (010) и (100) несравнимы. 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 $P = \{\pi_1, \ldots, \pi_m\}$ всех возможных разбиений конечного множества M решетка строится следующим образом. Частичный порядок: $\pi_i \leqslant \pi_j$, 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 од-

ном блоке, если и только если они содержатся в одном блоке в π_i или в π_j. Например, $\pi_1 \bigcup \pi_3 = \{abc, de, f\}$.

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: $a \leqslant b$, если и только

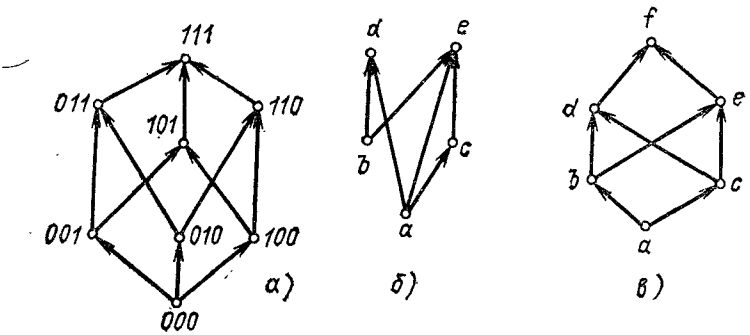

Рис. 2.1

если существует путь из стрелок, ведущий из a и 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 элементы 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, *а* 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 совпадет с рис. 2.1, *в*.

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 структуры — это максимальный элемент решетки, называемый единицей структуры. 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.

ГЛАВА ТРЕТЬЯ

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 (АЛГОЛ и др.) 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 использо-

ваться в дальнейшем. Итак, логическая функция $f(x_1, ..., x_n)$ — 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.

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

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 — лексико-графическом, 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 $f(x_1, ..., x_{i-1}, x_i, x_{i+1}, ..., x_n)$ называется *несущественной* (или *фиктивной*), если $f(x_1, ..., x_{i-1}, 0, x_{i+1}, ..., x_n) = f(x_1, ..., x_{i-1}, 1, x_{i+1}, ..., x_n)$ при любых значениях остальных переменных, т. е. если изменение значения x_i в любом наборе значений $x_1, ..., x_n$ не меняет значения функции. В этом случае функция $f(x_1, ..., x_n)$ по существу зависит от $n-1$ переменной, т. е. представляет

собой функцию $g(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)$ от $n-1$ переменной. Говорят, что функция g получена из 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 (являющегося 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.

Таблица 3.2

x	φ_0	φ_1	φ_2	φ_3
0	0	0	1	1
1	0	1	0	1

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

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: $\varphi_1(x) = x$. Функция $\varphi_2(x)$ называется *отрицанием* x (или функцией НЕ) и обозначается $\bar{x}$, $\neg x$, x', $\sim x$[1]. 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 эти функции отличаются от φ_0 и φ_3 в табл. 3.2; все функции в табл. 3.2 — унарные операции на B, а все функ-

[1] «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 — при вводе формул в ЭВМ, где все символы должны располагаться в строчку. Поэтому все чаще употребляется символ $\neg$, который будет использоваться в гл. 6.

Таблица 3.3

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

ции 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: $x_1 \vee x_2$, $x_1 + x_2$. 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 $x_1 \to x_2$, $x_1 \supset x_2$; 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* $f_1, \ldots, f_m$ 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 $\Sigma = \{f_1, \ldots, f_m \ldots\}$. Символы переменных $x_1, \ldots, x_n, \ldots$ будем считать *формулами глубины* 0. Формула F имеет *глубину* $k+1$, если F имеет вид $f_i(F_1, \ldots, F_{n_i})$, где $f_i \in \Sigma$, n_i — число аргументов f_i, а $F_1, \ldots, F_{n_i}$ — формулы, максимальная из глубин которых равна k. $F_1, \ldots, F_{n_i}$ называются *подформулами* F; f_i называется *внешней* или *главной операцией* формулы F. Все подформулы формул $F_1, \ldots, F_{n_i}$ 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:

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

Все формулы, построенные описанным ранее образом, т. е. содержащие только символы переменных, скобки 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, $x_1(x_2 \vee x_3 x_4)$ и $x_2 x_1(x_2 \vee x_3 x_4)$ 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 $\{\psi_1, \psi_7, \varphi_2\}$, т. е. функции И, ИЛИ, НЕ, то функцию штрих Шеффера можно представить формулами

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

а функцию стрелка Пирса — формулами

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

Формулы, представляющие одну и ту же функцию, называются *эквивалентными* или *равносильными* (см. пример 1.12, б). Эквивалентность формул обозначается знаком равенства; поэтому можно записать $\psi_{14}(x_1, x_2) = \bar{x}_1 \vee \bar{x}_2 = \overline{x_1 x_2}$. Как для двух данных формул выяснить, эквивалентны они или нет? Существует стандартный метод, всегда приводящий к ответу: по каждой формуле восстанавливается таблица[1] функции, а затем полученные две таблицы

[1] 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.

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 функций И, ИЛИ, НЕ.

Разложение функций по переменным. Совершенная дизъюнктивная нормальная форма. Введем обозначение $x^0 = \bar{x}$, $x^1 = x$. 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 $f(x_1, \ldots, x_n)$ может быть представлена в следующем виде:

$$f(x_1, \ldots, x_m, x_{m+1}, \ldots, x_n) = \bigvee_{\alpha_1, \ldots, \alpha_m} x_1^{\alpha_1} \ldots x_m^{\alpha_m} f(\alpha_1, \ldots, \alpha_m, x_{m+1}, \ldots, x_n), \qquad (3.4)$$

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

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

$$f(x_1, x_2, x_3, x_4) = \bar{x}_1 \bar{x}_2 f(0, 0, x_3, x_4) \vee \bar{x}_1 x_2 f \,\&\, (0,$$

$$1, x_3, x_4) \vee x_1 \bar{x}_2 f(1, 0, x_3, x_4) \vee x_1 x_2 f(1, 1, x_3, x_4).$$

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

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

т. е. тождество. □

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

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

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:

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

где дизъюнкция берется по всем наборам $(\sigma_1, \ldots, \sigma_n)$, 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 $(\sigma_1, \ldots, \sigma_n)$ соответствует конъюнкция всех переменных, в которой 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 $f(x_1, \ldots, x_n)$ 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 дизъюнкции и конъюнкции — см. далее (3.8) — формулы, получаемые из (3.6) перестановкой конъюнкций и букв в 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, будем называть *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 конъюнкции, дизъюнкции и отрицания.

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

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 $(P_2; \vee, \&, -)$, 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 бесконечное 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 упро-

щении и т. д., то имеются в виду преобразования формул, реализующих одну и ту же функцию.

Рассмотрим теперь основные свойства булевых операций.

Ассоциативность:

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

Коммутативность:

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

Дистрибутивность конъюнкции относительно дизъюнкции:

$$x_1(x_2 \vee x_3) = x_1x_2 \vee x_1x_3. \tag{3.9}$$

Дистрибутивность дизъюнкции относительно конъюнкции:

$$x_1 \vee (x_2x_3) = (x_1 \vee x_2)(x_1 \vee x_3). \tag{3.10}$$

Идемпотентность:

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

Двойное отрицание:

$$\bar{\bar{x}} = x. \tag{3.12}$$

Свойства констант:

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

Правила де Моргана:

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

Закон противоречия:

$$x\bar{x} = 0. \tag{3.15}$$

Закон «исключенного третьего»:

$$x \vee \bar{x} = 1. \tag{3.16}$$

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 независимы-

ми 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 переменной. 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 и 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 $\overline{x_1 x_3}$ вместо x_1. Получим $\overline{\overline{x_1 x_3} x_2} = \overline{\overline{x_1 x_3} \vee x_2}$, т. е. две формулы, неэквивалентные исходным, но эквивалентные между собой. Если же в правой части нового соотношения $\overline{\overline{x_1 x_3} \vee x_2}$ заменить формулой $\overline{x_1} \vee \overline{x_3}$, эквивалентной ей в силу (3.14), а в полученной подформуле $\overline{\overline{x_1} \vee \overline{x_3}}$ заменить $\overline{x_1}$ на эквивалентную ей в силу (3.12) формулу x_1, то все формулы в построенной цепи преобразований $\overline{x_1 x_3} x_2 \Rightarrow \overline{x_1 x_3} \vee \overline{x_2} \Rightarrow \overline{x_1} \vee \overline{x_3} \vee \overline{x_2} \Rightarrow x_1 \vee \overline{x_3} \vee \overline{x_2}$ 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 $(x_1 x_2) x_3$ пишут $x_1 x_2 x_3$) — это не вызывает неоднозначности ввиду ассоциативности этих операций (3.7); б) если они являются внешними скобками у конъюнкции: например, вместо $(x_1 (x_2 \vee x_3)) \vee (x_4 x_5)$ пишут $x_1 (x_2 \vee x_3) \vee x_4 x_5$. Оба соглашения совершенно аналогичны общепринятому опусканию скобок для умножения в арифметических формулах [1].

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

а) Поглощение:

$$x \vee xy = x; \tag{3.17а}$$

$$x(x \vee y) = x. \tag{3.17б}$$

Докажем подробно первое равенство [для его доказательства используются последовательно соотношения

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

(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:

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

б) Склеивание:

$$xy \vee x\bar{y} = x. \tag{3.18}$$

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

в) Обобщенное склеивание:

$$xz \vee y\bar{z} \vee xy = xz \vee y\bar{z}. \tag{3.19}$$

Доказывается с помощью расщепления, т. е. применения (3.18) в обратную сторону и поглощения (3.17): $xz \vee y\bar{z} \vee xy = xz \vee y\bar{z} \vee xyz \vee xy\bar{z} = xz \vee y\bar{z}$.

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

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

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

$$x_1 \vee f(x_1, x_2 \ldots, x_n) = x_1 \vee f(0, x_2, \ldots, x_n). \tag{3.21}$$

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

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

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 не единственной.

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. $xy \vee \bar{x}(y \vee xz)\,\overline{(x(\bar{y} \vee z) \vee yz)} = xy \vee (\bar{x}y \vee \bar{x}xz)\,\overline{(\bar{x} \vee \overline{(\bar{y} \vee z)})}\,\overline{yz} = xy \vee x\bar{y}\,(\bar{x} \vee yz)\,\overline{(\bar{y} \vee z)} = xy \vee \bar{x}y\,(x\bar{y} \vee y\bar{y}z \vee \bar{x}z \vee yz) = xy \vee \bar{x}yz = y\,(x \vee \bar{x}z) = xy \vee yz.$

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 $F_1 \Rightarrow$ СДНФ $\Rightarrow F_2$. □

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 $F = k_1 \vee \ldots \vee k_m$, где $k_1, \ldots, k_m$ — элементарные конъюнкции. Формулу $\overline{k_1 \vee \ldots \vee k_m}$ приведем к ДНФ $k_1' \vee \ldots \vee k_{m'}'$. Тогда $F = \overline{\overline{k_1 \vee \ldots \vee k_m}} = \overline{k_1' \vee \ldots \vee k_m'} = \overline{k_1'} \overline{k_2'} \ldots \overline{k_m'}$. По правилу де Моргана отрицания элементарных конъюнкций преобразуются в элементарные дизъюнкции, что и даст КНФ.

Пример 3.3. $x\bar{y} \vee \bar{x}y \vee x\bar{z} = \overline{\overline{x\bar{y} \vee \bar{x}y \vee x\bar{z}}} = \overline{(\bar{x} \vee y)\ \&\ (x \vee \bar{y})\ (\bar{x} \vee z)} = \overline{\bar{x}\bar{y} \vee xyz} = (x \vee y)\ (\bar{x} \vee \bar{y} \vee \bar{z})$.

Аналогом СДНФ является СКНФ — совершенная

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

Двойственность. Функция $f_1(x_1, ..., x_n)$ называется *двойственной* к функции $f_2(x_1, ..., x_n)$, если $f_1(x_1, ..., x_n) = \bar{f}_2(\bar{x}_1, ..., \bar{x}_n)$. Беря отрицание над обеими частями равенства и подставляя $\bar{x}_1, ..., \bar{x}_n$ вместо $x_1, ..., x_n$, получаем $\bar{f}_1(\bar{x}_1, ..., \bar{x}_n) = \bar{\bar{f}}_2(\bar{\bar{x}}_1, ..., \bar{\bar{x}}_n) = f_2(x_1, ..., x_n)$, 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 самодвойственно. 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 формуле 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 булевы алгебры множеств. Общий термин «булева алгебра» для алгебр множеств и логических функций не случаен.

Всякая алгебра типа (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 $\mathfrak{B}(U)$), операции & соответствует пересечение, операции $\vee$ — объединение, операции $\bar{}$ 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 $\mathfrak{B}(U)$, где $U=\{a_1, \ldots, a_n\}$, и множеством B_n двоичных векторов длины n: каждому подмножеству $M\subseteq U$ соответствует двоичный вектор $\Gamma(M)=\sigma=(\sigma_1, \ldots, \sigma_n)$, где $\sigma_i=1$, если $a_i\in M$, и $\sigma_i=0$, если $a_i\notin M$. Булева алгебра $(B_n; \vee, \&, \bar{})$ на множестве B_n определяется следующим образом: для любых векторов $\sigma=(\sigma_1, \ldots, \sigma_n)$ и $\tau=(\tau_1, \ldots, \tau_n)$

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

Поскольку компоненты (разряды) σ_i и τ_i 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 любой современной ЭВМ. 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 $(\mathfrak{B}(U); \cap, \cup, {}^-)$ изоморфна булевой алгебре $(B_n, \vee, \&, {}^-)$.

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

$$\Gamma(M_1 \cup M_2) = \sigma \vee \tau;$$
$$\Gamma(M_1 \cap M_2) = \sigma \,\&\, \tau; \qquad (3.23)$$
$$\Gamma(\overline{M}_1) = \bar{\sigma}.$$

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 $a_i \notin M_2$, следовательно, i-й разряд $\sigma \vee \tau$ также равен 0. Аналогично доказываются остальные два равенства. □

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 ЭВМ реализуются очень просто.

Рассмотрим теперь множество $P_2(m)$ всех логических функций m переменных $x_1, \ldots, x_m$. 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 $(P_2(m); \vee, \&, {}^-)$, являющуюся подалгеброй булевой алгебры логических функций.

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

Прежде всего отметим, что эти две алгебры равномощны и содержат 2^{2^m} элементов. Кроме того, поскольку все множества одинаковой мощности порождают изоморфные булевы алгебры множеств (см. пример 2.2, ∂), 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 $(\mathfrak{B}(B_m); \cup, \cap, -)$ и $(P_2(m); \vee, \&, ^-)$.

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 соответствием между $P_2(m)$ и $\mathfrak{B}(B_m)$, 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

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

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

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

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: $\sigma \notin (M_f \cup M_g)$. Тогда $\sigma \notin M_f$ или $\sigma \notin M_g$ 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: $U=\{a_1, \ldots, a_n\}$ — 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 над столбцами значений функций. Пример приведен в табл. 3.4, содержащей две функции f и g и результаты булевых операций над ними.

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

В завершение отметим еще один факт, связывающий логические функции с основными понятиями теории множеств: если $(f \to g) \equiv 1$, то $M_f \subseteq M_g$. Действительно, если $(f \to g) \equiv 1$, 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 — элементариая 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 $f(x_1, \ldots, x_m)$ 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 $F=xz\vee\bar{y}z\vee xy$ соответствует таблица покрытия (табл. 3.5).

Таблица 3.5

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

Из табл. 3.5 видно, что интервал M_{xy} покрывается объединением интервалов M_{xz} и $M_{\bar{y}z}$. 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 поглощения $x\vee xy=x$: 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 содержащийся в F, также представляет f. Поэтому дизъюнкция *всех* простых импликантов f, называемая *сокращенной ДНФ*, также будет представлять 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 $\Sigma_0 = \{\&, \vee, \bar{}\}$ (теорема 3.2). В настоящем параграфе будет показано, как решать этот вопрос для произвольной системы Σ.

Функционально полные системы. Система функции 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 $\Sigma_0 = \{\&, \vee, \bar{}\}$ 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[1] формулами над Σ, представ-

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

ляющими 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 $\Sigma_1=\{\&, \bar{}\}$ и $\Sigma_2=\{\vee, \bar{}\}$ 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:

$$x_1 \vee x_2 = \overline{\bar{x}_1 \& \bar{x}_2}, \; x_1 \& x_2 = \overline{\bar{x}_1 \vee \bar{x}_2}. \qquad (3.24)$$

Булева формула $\overline{x_1x_2}\vee\bar{x}_2$ $(x_3\vee x_4)$ в системе Σ_1 перейдет в формулу $\overline{\overline{x_1x_2}x_2x_3x_4}$, а в системе Σ_2 — в формулу $\overline{\bar{x}_1\vee\bar{x}_2}\vee x_2\vee\overline{x_3\vee x_4}$.

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 $\bar{x}=x|x=x\downarrow x$; $x_1\vee x_2=\overline{x_1\downarrow x_2}=(x_1\downarrow x_2)\downarrow(x_1\downarrow x_2)$; $x_1x_2=\overline{x_1|x_2}=(x_1|x_2)\,|\,(x_1|x_2)$, 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.

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:

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

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

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

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

а также соотношения булевой алгебры, относящиеся к конъ-

юнкции и константам (3.7а), (3.8а), (3.11а), (3.13, а, б). Отрицание и дизъюнкция выражаются так:

$$\bar{x} = x \oplus 1; \tag{3.29}$$

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

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. **а.** $(x_1 \vee x_2)\ (\bar{x}_2 \vee x_1 x_3) = (x_1 x_2 \oplus x_1 \oplus x_2) \& \& (x_1 \bar{x}_2 x_3 \oplus x_1 x_3 \oplus \bar{x}_2) = x_1 (x_2 \oplus 1) x_3 \oplus x_1 x_2 x_3 \oplus x_1 x_3 \oplus x_1 x_2 x_3 \oplus \oplus x_1 (x_2 \oplus 1) = x_1 x_2 x_3 \oplus x_1 x_2 \oplus x_1$;

б. $x_1 \bar{x}_2 \vee \bar{x}_1 x_3 = x_1 \bar{x}_2 \oplus \bar{x}_1 x_3 = x_1 (x_2 \oplus 1) \oplus (x_1 \oplus 1) \& x_3 = = x_1 x_2 \oplus x_1 x_3 \oplus x_1 \oplus x_3$;

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

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 установ-

лено взаимно однозначное соответствие, что и доказывает единственность полинома для каждой функции. □

Функция, у которой полином Жегалкина имеет вид $\Sigma\alpha_i x_i \oplus \gamma$, 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 $x_1 \vee x_2 \vee \ldots \vee x_n$, является замкнутым классом.

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

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

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 $\leqslant$ на множестве векторов одинаковой длины. Напомним, что для двух векторов $\sigma=(\sigma_1, \ldots, \sigma_n)$ и $\tau=(\tau_1, \ldots, \tau_n)$ $\sigma \leqslant \tau$, если и только если $\sigma_i \leqslant \tau_i$ для всех $i=1, \ldots, n$. Здесь воспользуемся этим отношением для двоичных векторов.

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

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, заданные

табл. 3.6. Функция f_1 немонотонная, так как $001<101$, а $f_1(001)>(101)$. Функция f_2 монотонна (проверку представляем читателю).

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

Теорема 3.7. Всякая булева формула, не содержащая отрицаний, представляет монотониую 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 $\sigma=(\sigma_1, ..., \sigma_n)$ $F(\sigma)=1$. Тогда F содержит конъюнкцню (без отрицаний!) $x_{i_1} ... x_{i_k}=1$, равную 1 на этом наборе. Следовательно, $\sigma_{i_1}=...=\sigma_{i_k}=1$. Рассмотрим любой набор τ, больший, чем σ. В нем обязательно $\tau_{i_1}=...=\tau_{i_k}=1$, поэтому $x_{i_1}, ..., x_{i_k}$ 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 $f\not\equiv 0$, так как в невырожденной ДНФ для любой конъюнкции найдется набор, обращающий ее в 1; $f\not\equiv 1$, так как на нулевом наборе $(0, 0, ..., 0)$ $F=0$.

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 условню 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. □

Более подробное изучение импликантов монотонной функцин 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 на табл. 3.6 $M_f^0=\{010, 101\}$, поэтому сокращенная ДНФ f_2 имеет вид $x_2\vee x_1x_3$.

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 покрывается только импликантом k_i, который не может быть удален из сокращенной ДНФ. Следовательно, сокращенная ДНФ — единственная тупиковая, а значит, и минимальная ДНФ монотонной функции. □

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. □

Две теоремы о фукциональной полноте. Теперь можно перейти к основной проблеме этого параграфа: каковы 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 булевы операции.

Лемма 1 (о немонотонных функциях). Если функция $f(x_1, ..., x_n)$ 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 су-

ществуют 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 $\sigma<\omega^1<\omega^2<\ldots<\omega^{k-1}<\tau$, 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^j, ω^{j+1}, таких, что $f(\omega^j)=1$, $f(\omega^{j+1})=0$. Пусть они отличаются в i-й компоненте; тогда $\omega_i^j=0$, $\omega_i^{j+1}=1$, остальные их компоненты одинаковы. Подставим эти значения остальных компонент в f. Получим функцию $f(\omega_1^j, \ldots, \omega_{i-1}^j, x_i, \omega_{i+1}^j, \ldots, \omega_n^j)$ от x_i; обозначим ее $g(x_i)$. Но $g(0)=g(\omega_i^j)=f(\omega^j)=1$; $g(1)=g(\omega_i^{j+1})=f(\omega^{j+1})=0$; следовательно, $g(x_i)=\bar{x}_i$. □

Лемма 2 (о нелинейных функциях). Если функция $f(x_1, \ldots, x_n)$ 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 $K=x_{i_1}x_{i_2}\ldots x_{i_k}$. Положим $x_{i_3}=\ldots=x_{i_k}=1$, 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 $f_i(x, y)$ $(i=0, 1, \ldots, 7)$ — 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). □

Пример 3.9. $f(x_1, x_2, x_3, x_4)=x_1x_3x_4\oplus x_1x_2x_3x_4\oplus x_1\oplus x_4$. Полагаем $x_4=1$, $x_2=0$. Тогда $f(x_1, 0, x_3, 1)=x_1x_3\oplus x_1\oplus 1$,

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	$xy \oplus 1$	$\overline{xy}$	$xy=N(f_1(x, y))$
f_2	0	1	0	$xy \oplus y$	$\bar{x}y$	$xy=f_2(N(x), y)$
f_3	0	1	1	$xy \oplus y \oplus 1$	$\overline{\bar{x}y}=x\vee\bar{y}$	$x\vee y=f_3(x, N(y))$
f_4	1	0	0	$xy \oplus x$	$x\bar{y}$	$xy=f_4(x, N(y))$
f_5	1	0	1	$xy \oplus x \oplus 1$	$\overline{x\bar{y}}=\bar{x}\vee y$	$x\vee y=f_5(N(x), y)$
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	$xy \oplus x \oplus y \oplus 1$	$\overline{x\vee y}$	$x\vee y=N(f_7(x, y))$

что соответствует строке f_5 в табл. 3.7. Отсюда получаем $x_1\vee x_3=f(\bar{x}_1, 0, x_3, 1)$; $x_1x_3=\overline{\bar{x}_1\vee\bar{x}_3}=f(x_1, 0, \bar{x}_3, 1)$.

Замечание. При традиционных обозначениях переменных в выражениях вида $f(x_1, x_2, x_3, x_4)$, 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 набор (серия) типовых логических элементов, а полнота системы означает возможность реализовать с помощью элементов данной схемы любые логические функции. При схемной реали-

зации 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.

Необходимость. 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.

Достаточность. 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. □

Пример 3.10. а. Система $\Sigma_6 = \{\&, \oplus\}$ функционально полна в слабом смысле, так как конъюнкция нелинейна, а сумма по mod2 немонотонна. Константа 0 получается из соотношения (3.27), однако константу 1 с помощью конъюнкции и суммы по mod2 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_5 (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:

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

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

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

Функция $f(x_1, ..., x_n)$ называется *сохраняющей* 0, если $f(0, 0, ..., 0)=0$. Функция $f(x_1, ..., x_n)$ называется *сохраняющей* 1, если $f(1, 1, ..., 1)=1$. Оба класса функций, сохраняющих 0 и сохраняющих 1, являются замкнутыми, что проверяется подстановкой констант в суперпозиции.

Напомним, что функция является самодвойственной, если $f(x_1, ..., x_n)=\bar{f}(\bar{x}_1, ..., \bar{x}_n)$. Класс самодвойственных фуикций замкнут. Его замкнутость доказывается прямой выкладкой. (Чтобы избежать громоздких обозначений, далее она проводится не в самом общем виде; обобщение очевидно.) Пусть $f_1(x_1, ..., x_n)$, $f_2(x_n, x_{n+1}, ..., x_{n+k})$ — самодвойственные функции. Подставим f_2 в f_1 вместо x_n. Получим $g(x_1, ..., x_n, x_{n+1}, ..., x_{n+k})=f_1(x_1, ..., x_{n-1}, f_2(x_n, ..., x_{n+k}))$. Тогда в силу самодвойственности f_1 и f_2 $g(\bar{x}_1, ..., \bar{x}_n, \bar{x}_{n+1}, ..., \bar{x}_{n+k})=\bar{f}_1(\bar{x}_1, ..., \bar{x}_{n-1}, f_2(\bar{x}_n, ..., \bar{x}_{n+k}))=\bar{f}_1(\bar{x}_1, ..., \bar{x}_{n-1}, \bar{f}_2(x_n, ..., x_{n+k}))=f_1(x_1, ..., x_{n-1}, f_2(x_n, ..., x_{n+k}))=g(x_1, ..., x_{n+k})$, т. е. g является самодвойственной.

Теорема 3.10 (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 $f(x_1, ..., x_n)$ несамодвойственна, то подстановкой в нее x и $\bar{x}$ можно получить константу. Действительно, ввиду несамодвойственности f найдется набор $(\sigma_1, ..., \sigma_n)$, такой, что $f(\bar{\sigma}_1, ..., \bar{\sigma}_n)=f(\sigma_1, ..., \sigma_n)$. Но тогда функция $f_c(x)=f(x^{\sigma_1}, ..., x^{\sigma_n})$ является константой, так как $f_c(0)=f(0^{\sigma_1}, ..., 0^{\sigma_n})=f(\bar{\sigma}_1, ..., \bar{\sigma}_n)=f(\sigma_1, ..., \sigma_n)=f(1^{\sigma_1}, ..., 1^{\sigma_n})=f_c(1)$.

Пусть теперь f_0 не сохраняет 0, f_1 не сохраняет 1, f_2 не-

самодвойственна (f_0, f_1, f_2 не обязаны быть различными). Если $f_0(1, ..., 1)=1$, то функция $\varphi(x)=f_0(x, ..., x)$ есть константа 1, так как $\varphi(0)=1$ по определению f_0 и $\varphi(1)=f_0(1, ..., 1)=1$, а функция $\psi(x)=f_1(\varphi(x), ..., \varphi(x))=f_1(1, ..., 1)=0$, т. е. $\psi(x)$ есть константа 0. Если же $f_0(1, ..., 1)=0$, то $\varphi(x)=\bar{x}$, так как $\varphi(0)=1$ по определению f_0 и $\varphi(1)=f_0(1, ..., 1)=0$. 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. □

3.4. ЯЗЫК ЛОГИКИ ПРЕДИКАТОВ

Предикаты. *Предикатом* $P(x_1, ..., x_n)$ называется функция $P: M^n \to B$, где M — произвольное множество, а B — двоичное множество, определенное в начале § 3.1. Иначе говоря, n-местный предикат, определенный на M, — это двузначная функция от n аргументов, принимающих значения в произвольном множестве M. M называется *предметной областью* предиката, а $x_1, ..., x_n$ — *предметными переменными*. В принципе ничто не мешает определить предикат в более общем виде как функцию $P: M \times M_2 \times ... \times M_n \to B$, т. е. разрешить разным аргументам принимать значения из разных множеств. Иногда это оказывается удобным; однако, как правило, в логике предикатов исходят из первого определения.

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

Всякой функции $f: M^n \to M$ можно поставить в соответствие $(n+1)$-местный предикат P, такой, что $P(a_1, ...; a_n, a_{n+1})=1$, если и только если $f(a_1, ..., a_n)=a_{n+1}$. Поскольку функция должна быть однозначной, то это соответствие требует, чтобы для любого $a'_{n+1} \neq a_{n+1}$ $P(a_1, ..., a_n, a'_{n+1})=0$. 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, 40-летний 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 таковые 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. Поэтому логическая интерпретация предикатов является основной, и мы ее будем в дальнейшем придерживаться.

Выражение $P(a_1, ..., a_n)$ (и другие, более сложные выражения логики предикатов), где $a_1, ..., a_n \in M$, будем понимать как высказывание «$P(a_1, ..., a_n) = 1$» или в соответствии с логической интерпретацией как «$P(a_1, ..., a_n)$ истинно», а выражение $P(x_1, ..., x_n)$, где $x_1, ..., x_n$ — переменные, как переменное высказывание, истинность которого определяется подстановкой элементов M вместо $x_1, ..., x_n$. При этом $P(x_1, ..., x_n)$ — это логическая (двоичная) переменная, а $x_1, ..., x_n$ — нелогические переменные. Поскольку предикаты принимают два значения и интерпретируются как высказывания, из них можно образовывать выражения логики высказываний, т. е. формулы вида $P_1(x_1, x_2) \vee (\overrightarrow{P_2(x_3, x_4)} \& \overrightarrow{\& P_1(x_2, x_4)})$. 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 разночтений, 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 $x_1 P x_2$, 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, не доказанная до сих пор, 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 $N(x)\&N(y)\&N(z)\&N(n)\&(n>2)\to\overline{P}_F(x, y, z, n)$ 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 переменные 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 $\overline{(x=y)\vee(i>100)}$».

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 (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 (Ex). Переход от $P(x)$ к $\forall x P(x)$ или к $\exists x P(x)$ называется связыванием переменной x, а также навешиванием квантора на переменную 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 $\sum_{x=1}^{10} f(x)$ 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:

$$\forall x \forall y \forall z \forall n\,(N(x) \,\&\, N(y) \,\&\, N(z) \,\&\, N(n) \,\&\, (n > 2) \rightarrow$$

$$\rightarrow \overline{P}_F(x, y, z, n)).$$

в. Рассмотрим двухместный предикат $x \geqslant y$ на множе-

ствах M с отношением нестрогого порядка и различные квантификации его переменных. $\forall x(x \geqslant y)$ — одноместный предикат от y; если M — множество неотрицательных чисел, то этот предикат истинен в единственной точке: $y=0$. $\forall x \forall y(x \geqslant y)$ — высказывание, истинное на множестве, состоящем из одного элемента, и ложное на любом другом множестве. $\exists x \exists y(x \geqslant y)$ истинно на любом непустом множестве. Высказывание $\exists x \forall y(x \geqslant y)$ («существует x, такой, что для любого y $x \geqslant y$») утверждает, что в M имеется единственный максимальный элемент. Оно истинно на любом конечном множестве целых чисел, но ложно на множестве $\{1/2, 2/3, 3/4, \ldots, n/(n+1)\ldots\}$ или на множестве двоичных векторов, из которого удален вектор, состоящий из одних единиц. Высказывание $\forall y \exists x(x \geqslant y)$ утверждает, что для любого элемента y существует элемент x не меньший, чем y; оно истинно на любом непустом множестве ввиду рефлексивности отношения $\geqslant$. 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: $\exists x P(x, y) \rightarrow \forall x P(x, y)$.

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 *общезначимой*. Например, формула $\exists x P(x, y) \rightarrow \forall x P(x, y)$ тождественно истинна для всех M, состоящих из одного элемента, а 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 $|M| \leqslant 2$ (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 формулы на истинность. Если вспомнить классификацию способов задания множеств (§ 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

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

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

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

Докажем теперь дистрибутивность $\forall x$ относительно

конъюнкции и $\exists x$ относительно дизъюнкции

$$\forall x\,(P_1(x)\,\&\,P_2(x)) \sim \forall x P_1(x)\,\&\,\forall x P_2(x). \qquad (3.33)$$

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, $\forall x P_1(x)\,\&\,\forall x P_2(x)$ истинно. Если же левая часть ложна, то для некоторого $a \in M$ ложно либо $P_1(a)$, либо $P_2(a)$, а следовательно, ложно либо $\forall x P_1(x)$, либо $\forall x P_2(x)$, и правая часть ложна. Аналогично доказывается

$$\exists x\,(P_1(x) \vee P_2(x)) \sim \exists x P_1(x) \vee \exists x P_2(x). \qquad (3.34)$$

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

$$\exists x\,(P_1(x)\,\&\,P_2(x)) \to \exists x P_1(x)\,\&\,\exists x P_2(x); \qquad (3.35)$$

$$(\forall x P_1(x) \vee \forall x P_2(x)) \to \forall x\,(P_1(x) \vee P_2(x)). \qquad (3.36)$$

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:

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

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

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

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

$$\forall x\,(P(x)\,\&\,Y) \sim \forall x P(x)\,\&\,Y; \qquad (3.39)$$

$$\forall x (P(x) \vee Y) \sim \forall x P(x) \vee Y; \tag{3.40}$$

$$\exists x (P(x) \& Y) \sim \exists x P(x) \& Y; \tag{3.41}$$

$$\exists x (P(x) \vee Y) \sim \exists x P(x) \vee Y. \tag{3.42}$$

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

О методах доказательства в логике предикатов. Метод доказательства формул, содержащих переменные, путем непосредственной подстановки в них констант называется методом интерпретаций или методом моделей [1]. Подстановка констант позволяет интерпретировать формулу как осмысленное утверждение об элементах конкретного множества M. Поэтому такой метод, выясняющий истинность формул 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, $M=\{a_1, \ldots, a_n\}$, то кванторы переходят в конечные формулы логики высказываний:

$$\forall x P(x) \sim P(a_1) \& P(a_2) \& \ldots \& P(a_n); \tag{3.43}$$

$$\exists x P(x) \sim P(a_1) \vee P(a_2) \vee \ldots \vee P(a_n). \tag{3.44}$$

Заменив все кванторы в формуле по этим соотношениям, любую формулу логики предикатов можно перевести в формулу, состоящую из предикатов, соединеных 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 вы-

[1] Точные понятия интерпретации и модели вводятся в гл. 6, § 6.3.

вода. 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).

ГЛАВА ЧЕТВЕРТАЯ

ГРАФЫ

4.1. ОСНОВНЫЕ ПОНЯТИЯ И ОПЕРАЦИИ

Графы, их вершины, ребра и дуги. Теорию графов начали разрабатывать для решения некоторых задач о геометрических конфигурациях, состоящих из точек и линий. В этих задачах несущественно, соединены ли точки конфигурации отрезками прямых или криволинейными дугами, какова длина линий и другие геометрические характеристики конфигурации. Важно лишь то, что каждая линия соединяет какие-либо две из заданных точек. Таким образом, можно дать определение графа как совокупности двух множеств V (точек) и E (линий), между элементами которых определено отношение *инцидентности,* причем каждый элемент $e \in E$ инцидентен ровно двум элементам v', $v'' \in V$. Элементы множества V называются *вершинами* графа G, элементы множества E — его *ребрами.* Вершины и ребра графа G называют еще его *элементами* и вместо $v \in V$, $e \in E$ пишут соответственно $v \in G$ и $e \in G$.

В некоторых задачах инцидентные ребру вершины неравноправны, они рассматриваются в определенном порядке. Тогда каждому ребру можно приписать направление от первой из инцидентных вершин ко второй. Направленные ребра часто называют *дугами* (однако в этой книге они будут называться ребрами), а содержащий их граф — *ориентированным* (граф, определенный ранее, называется *неориентированным*). Первая по порядку вершина, инцидентная ребру ориентированного графа, называется его *началом*, вторая — его *концом.* Говорят еще, что ребро ориентированного графа *выходит* из начала и *входит* в конец.

В дальнейшем оказалось, что понятие графа можно применить не только при исследовании геометрических конфигураций. Особенно часто определяют графы при анализе функционирования систем. С отдельными компонентами изучаемой системы удобно связывать вершины графа, а с парами взаимодействующих компонент — его ребра. Построенный таким образом граф называют структурным графом системы.

Изображение графов. На рис. 4.1, *а—з* изображены некоторые неориентированные графы. Множество ребер E мо-

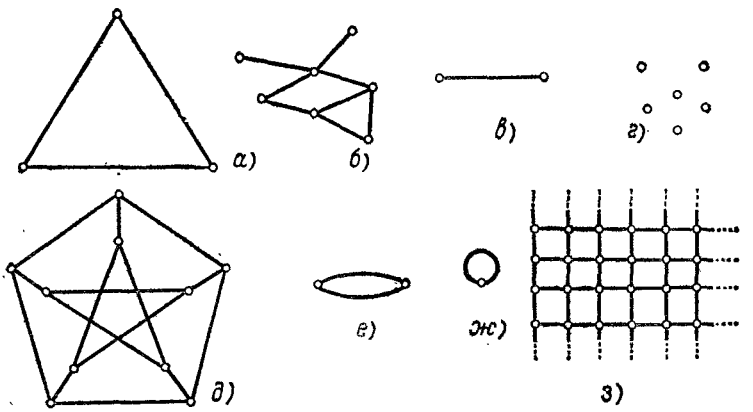

Рис. 4.1

жет быть пустым (рис. 4.1, *г*). Если же множество вершин V пусто, то пусто и E. Такой граф называется пустым. Линии, изображающие ребра графа, могут пересекаться, но точки пересечения не являются вершинами (рис. 4.1, *д*); различные ребра могут быть инцидентны одной и той же паре вершин (рис. 4.1, *е*), в этом случае они называются *кратными*; граф, содержащий кратные ребра, часто называют мультиграфом. Ребро может соединять некоторую вершину саму с собой (рис. 4.1, *ж*), такое ребро называется *петлей*. На рис. 4.1, *з* изображен фрагмент *бесконечного* графа. Его вершины — это точки плоскости с целыми координатами (x, y), а ребра — соединяющие их горизонтальные и вертикальные отрезки длины 1.

Обычно рассматриваемые графы *конечны*, т. е. конечны множества их элементов (вершин и ребер). Поэтому конечность этих графов не будет специально оговариваться. Од-

нако некоторые понятия и результаты, о которых будет идти речь, относятся к произвольным графам.

При изображении ориентированных графов (рис. 4.2, *а—з*) направления ребер отмечаются стрелками, примыкающими к их концам. Ориентированный граф также может иметь кратные ребра (рис. 4.2, *е*), петли (рис. 4.2, *ж*), а также соединяющие одни и те же вершины ребра, идущие в противоположных направлениях (рис. 4.2, *з*).

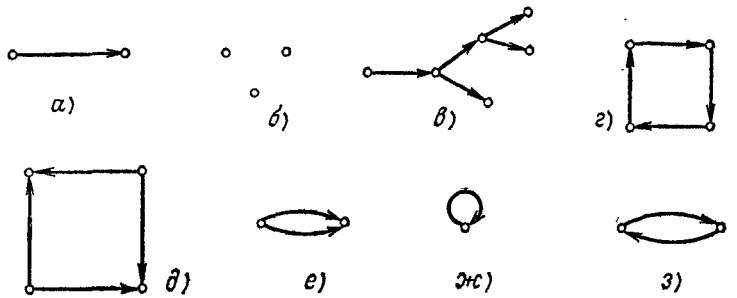

Рис. 4.2.

Каждому неориентированному графу можно поставить в соответствие ориентированный граф с тем же множеством вершин, в котором каждое ребро заменено двумя ориентированными ребрами, инцидентными тем же вершинам и имеющими противоположные направления. Такое соответствие будем называть *каноническим*.

Матрица инцидентности и список ребер. Задать граф — значит описать множества его вершин и ребер, а также отношение инцидентности. Когда граф G — конечный, для описания его вершин и ребер достаточно их занумеровать. Пусть v_1, v_2, ..., v_n — вершины графа G; e_1, e_2, ..., e_m — его ребра. Отношение инцидентности можно определить матрицей $\|\varepsilon_{ij}\|$, имеющей m строк и n столбцов. Столбцы соответствуют вершинам графа, строки — ребрам. Если ребро e_i инцидентно вершине v_j, то $\varepsilon_{ij}=1$, в противном случае $\varepsilon_{ij}=0$. Это так называемая *матрица инцидентности* неориентированного графа G, которая является одним из способов его определения (для графа на рис. 4.3 она дана в табл. 4.1, а).

В матрице инцидентности $\|\varepsilon_{ij}\|$ ориентированного графа G, если вершина v_j — начало ребра a_i, то $\varepsilon_{ij}=-1$; если v_j — конец a_i, то $\varepsilon_{ij}=1$; если a_i — петля, а v_j — инцидент-

Таблица 4.1

	I	II	III	IV	V	VI	VII
1	1	1	0	0	0	0	0
2	1	0	1	0	0	0	0
3	0	1	0	1	0	0	0
4	1	0	0	0	1	0	0
5	0	1	0	0	0	1	0
6	0	0	1	1	0	0	0
7	0	0	1	0	1	0	0
8	0	0	0	1	0	1	0
9	0	0	0	0	1	0	1
10	0	0	0	0	0	1	1

а)

	I	II	III	IV	V	VI	VII
1	—1	1	0	0	0	0	0
2	—1	0	1	0	0	0	0
3	0	—1	0	1	0	0	0
4	0	0	—1	0	1	0	0
5	0	0	—1	0	0	1	0
6	0	0	—1	0	0	0	1
7	0	0	0	0	0	0	2

б)

Ребра	Вершиины
1	I, II
2	I, III
3	II, IV
4	I, V
5	II, VI
6	III, IV
7	III, V
8	IV, VI
9	V, VII
10	VI, VII

в)

Ребра	Вершины
1	I, II
2	I, III
3	II, IV
4	III, V
5	III, VI
6	III, VII
7	VII, VII

г)

ная ей вершина, то $\varepsilon_{ij}=\alpha$, где α — любое число, отличное от 1, 0 и —1, в остальных случаях $\varepsilon_{ij}=0$ (пример — табл. 4.1, б для графа на рис. 4.4).

В каждой строке матрицы инцидентности для неориентированного или ориентированного графа только два элемента отличны от 0 (или один, если ребро является петлей). Поэтому такой способ задания графа оказывается недостаточно экономным. Отношение инцидентности можно еще задать списком ребер графа. Каждая строка этого списка соответствует ребру, в ней записаны номера вершин, инцидентных ему. Для неориентированного графа порядок этих вершин в строке произволен, для ориентированного первым стоит номер или другое наименование начала ребра,

а вторым — его конца. В табл. 4.1, в и 4.1, г приводятся списки ребер для графов, изображенных на рис. 4.3 и 4.4.

По списку ребер графа легко построить его матрицу инцидентности. Действительно, каждая строка этого списка

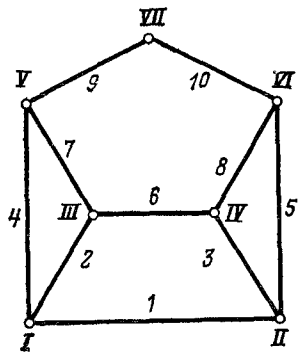

Рис. 4.3.

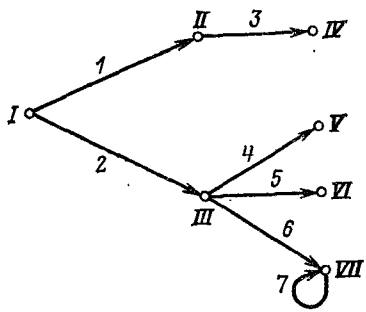

Рис. 4.4

соответствует строке матрицы с тем же номером. Для неориентированного графа в строке списка указаны номера элементов строки матрицы инцидентности, равные 1, и для ориентированного графа в этой строке первым стоит номер элемента строки матрицы, равного — 1, а вторым — номер элемента, равного +1.

Матрица смежности графа. Это квадратная матрица $\|\delta_{ij}\|$, столбцам и строкам которой соответствуют вершины графа. Для неориентированного графа δ_{ij} равно количеству ребер, инцидентных i-й и j-й вершинам, для ориентированного графа этот элемент матрицы смежности равен количеству ребер с началом в i-й вершине и концом в j-й. Таким образом, матрица смежности неориентированного графа симметрична ($\delta_{ij}=\delta_{ji}$), а ориентированного — не обязательно. Если она все же симметрична, то для каждого ребра ориентированного графа имеется ребро, соединяющее те же вершины, но идущее в противоположном направлении. Ориентированный граф с симметричной матрицей смежности канонически соответствует неориентированному графу, имеющему ту же матрицу смежности.

Матрицы смежности рассмотренных ранее графов (рис. 4.3, 4.4) приводятся в табл. 4.2.

Матрица смежности также определяет соответствующий неориентированный или ориентированный граф. Число его

Таблица 4.2

	I	II	III	IV	V	VI	VII
I	0	1	1	0	1	0	0
II	1	0	0	1	0	1	0
III	1	0	0	1	1	0	0
IV	0	1	1	0	0	1	0
V	1	0	1	0	0	0	1
VI	0	1	0	1	0	0	1
VII	0	0	0	0	1	1	0

а)

	I	II	III	IV	V	VI	VII
I	0	1	1	0	0	0	0
II	0	0	0	1	0	0	0
III	0	0	0	0	1	1	1
IV	0	0	0	0	0	0	0
V	0	0	0	0	0	0	0
VI	0	0	0	0	0	0	0
VII	0	0	0	0	0	0	1

б)

вершин равно размерности матрицы n, i-й и j-й вершинам графа инцидентны δ_{ij} ребер. Для неориентированного графа $\delta_{ij}=\delta_{ji}$ и все его ребра определяются верхним правым треугольником матрицы, расположенным над диагональю, включая последнюю. Количество их равно сумме δ_{ij} по этому треугольнику, т. е. $\sum_{i=1}^{n}\sum_{j=i}^{n}\delta_{ij}$. Ребра ориентированного графа определяются всеми элементами δ_{ij} матрицы смежности. В обоих случаях по матрице смежности легко строится, например, список ребер, определяющий граф. Элементу матрицы смежности, расположенному в i-й строке и j-м столбце, соответствуют δ_{ij} строк списка ребер (при $\delta_{ij}=0$ — ни одной строки), в каждой из которых записаны номера i, j. Для неориентированного графа эти строки соответствуют только элементам описанного ранее верхнего правого треугольника матрицы смежности, т. е. элементам δ_{ij} с $j\geqslant i$, а для ориентированного графа нужно рассматривать все элементы δ_{ij}.

Идентификация графов, заданных своими представлениями. Итак, граф может быть представлен различными способами. Он может быть изображен на чертеже, задан матрицей инцидентности, списком ребер или матрицей смежности. Вид чертежа зависит от формы линий и взаимного расположения вершин: Иногда не так легко понять, одинаковы ли графы, изображенные разными чертежами (ср., например, графы на рис. 4.5). Вид матриц и списка ребер зависит от нумерации вершин и ребер графа. Строго говоря, граф считается полностью заданным, если нумерация его вершин зафиксирована; графы, отличающиеся только нумерацией вершин, называются *изоморфными*.

Перенумерация вершин графа задается строкой $\alpha_1, \alpha_2, ..., \alpha_n$ новых номеров вершин, расположенных в исходном порядке. Новая матрица смежности получается из исходной перемещением каждого элемента δ_{ij} в α_i-ю строку и α_j-й столбец, т. е. в результате перестановки $(\alpha_1, \alpha_2, ..., \alpha_n)$ строк и столбцов исходной матрицы. Поэтому, чтобы узнать, представляют ли две матрицы смежности изоморфные графы, можно, например, произвести всевозможные одинаковые перестановки строк и столбцов первой матрицы. Если после одной из этих перестановок возникнет матрица, тождественно совпадающая со второй, графы, изображаемые данными матрицами смежности, изоморфны. Однако чтобы убедиться таким способом в том, что графы неизоморфны, придется выполнить все $n!$ перестановок строк и столбцов, что является достаточно трудоемкой операцией.

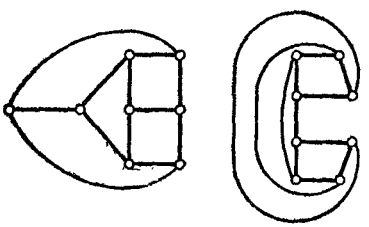

Рис. 4.5

Матрица инцидентности графа и список его ребер зависят от нумерации ребер и вершин. Переход от одной пары нумерации к другой определяется перестановками $(\alpha_1, \alpha_2, ..., \alpha_n)$ вершин и $(\beta_1, \beta_2, ..., \beta_m)$ ребер рассматриваемого графа. Матрица инцидентности получается из первоначальной в результате перестановки строк (i-я — на β_i-е место) и столбцов (j-й — на α_j-е). Строки списка ребер переставляются так же, как и строки матрицы инцидентности, и каждый номер j в строках списка заменяется номером α_j.

Графы без кратных ребер. Пусть даны два множества V_1 и V_2. Их прямое произведение $V_1 \times V_2$ можно *симметризовать*, т. е. рассматривать также пары (v_2, v_1), где $v_2 \in V_2$, а $v_1 \in V_1$, причем пары (v_1, v_2) и (v_2, v_1) считаются одинаковыми. Если $V_1 = V_2 = V$, то несимметризованное произведение $V \times V$ — это множество всех упорядоченных пар (v_1, v_2) элементов множества V, а симметризованное произведение $[V \times V]$ — это множество неупорядоченных пар, когда $(v_1, v_2) = (v_2, v_1)$.

Пусть E — подмножество $[V \times V]$. Тогда V можно считать множеством вершин, а E — множеством ребер неориентированного графа [ребро $(v_1, v_2) \in E$ инцидентно вершинам v_1 и v_2]. Аналогично подмножество $E' \subset V \times V$ определя-

ет ориентированный граф, в котором началом ребра $(v_1, v_2) \in E'$ является вершина v_1, а концом — вершина v_2. Каждая пара (v_1, v_2), принадлежащая подмножеству E или E', фигурирует в качестве ребра только один раз, следовательно, построенные графы не имеют кратных ребер, и наоборот, если граф не имеет кратных ребер, то последние однозначно определяются неупорядоченной или упорядоченной парой (v_1, v_2) инцидентных им вершин. Поэтому множества ребер неориентированного или ориентированного графа считаются подмножеством в первом случае симметризованного произведения, а во втором — несимметризованного.

Степени вершин графа. Пусть G — неориентированный граф. Количество $\rho(v)$ ребер, инцидентных вершине $v \in G$, называется ее *локальной степенью* или просто *степенью*.

Если степени всех вершин конечны, рассматриваемый граф G называется локально-конечным. В частности, любой конечный граф локально-конечен.

Когда заданы матрицы смежности или инцидентности графа, можно определить локальные степени всех его вершин. Действительно, в j-м столбце, соответствующем вершине v_j, единицы стоят на пересечении со строками, которым соответствуют инцидентные этой вершине ребра, а остальные элементы столбца равны 0. Следовательно, $\rho(v_j) = \sum_{i=1}^{m} \varepsilon_{ij}$. Элементы же δ_{ij} матрицы смежности — это количества ребер, инцидентных вершинам v_i и v_j. Отсюда $\rho(v_j) = \sum_{i=1}^{n} \delta_{ij}$.

При подсчете степеней вершин по этим формулам каждая петля дает в степень инцидентной ей вершины вклад 1. Однако при изображении петли на рисунке к этой вершине примыкают два конца петли, т. е. петля дает в эту степень вклад 2. Чтобы таким образом учитывать вклад петель в степень, нужно несколько усложнить формулы для ее вычисления. Через коэффициенты матрицы инцидентности ее можно вычислить, например, по формуле $\rho'(v_j) = \sum_{i=1}^{m} \varepsilon_{ij} \times \left(3 - \sum_{k=1}^{n} \varepsilon_{ik}\right)$. Когда i-е ребро обычное, $\sum_{k=1}^{n} \varepsilon_{ik} = 2$ и соответствующее слагаемое внешней суммы равно ε_{ij}, т. е. 1 для

ребер, инцидентных вершине v_j, и 0 для остальных. Если же оно является петлей, то $\sum_{k=1}^{n} \varepsilon_{ik} = 1$, а слагаемое внешней суммы равно $2\varepsilon_{ij}$, т. е. 2 для петель, инцидентных вершине v_j, и 0 для остальных.

Это же значение степени выражается через коэффициенты δ_{ij} матрицы смежности графа G формулой

$$\rho'(v_j) = \sum_{i=1}^{n} \delta_{ij} + \delta_{jj} = \sum_{k=1}^{n} \delta_{jk} + \delta_{jj}.$$

В зависимости от рассматриваемой задачи может потребоваться тот или иной способ определения степени вершины. Поэтому в каждом случае должно быть указано, является ли петля однажды или дважды инцидентной своей вершине.

Так как каждое ребро имеет два конца, в сумме $\sum_{v \in G} \rho'(v)$ ребра учитываются по 2 раза. Таким образом, эта сумма равна удвоенному числу ребер графа, т. е. четна. Следовательно, четно и количество нечетных слагаемых этой суммы, т. е. число вершин нечетной степени.

Граф называется *однородным* степени k, если степени всех его вершин равны k и тем самым равны между собой. Если однородный граф степени k имеет n вершин и m ребер, то $m = \frac{1}{2} \sum_{v \in G} \rho'(v) = kn/2$, $n = 2m/k$. Граф без кратных ребер называется полным, если каждая пара вершин соединена ребром.

Локальные степени ориентированных графов. Для вершин ориентированного графа определяются две локальные степени: $\rho_1(v)$ — число ребер с началом в вершине v, или, иначе, количество выходящих из v ребер, и $\rho_2(v)$ — количество входящих в v ребер, для которых эта вершина является концом. Петля дает вклад 1 в обе эти степени. Локальные степени вершин ориентированного графа просто выражаются через коэффициенты δ_{ij} его матрицы смежности: $\rho_1(v_i) = \sum_{j=1}^{n} \delta_{ij}$, $\rho_2(v_i) = \sum_{h=1}^{n} \delta_{hi}$.

Выражение их через коэффициенты матрицы инцидентности значительно сложнее.

Так как каждое ребро ориентированного графа G имеет одно начало и один конец, суммы $\sum_{v\in G}\rho_1(v)$ и $\sum_{v\in G}\rho_2(v)$ равны количеству ребер этого графа, а значит, и равны между собой. Отсюда следует, что в однородном ориентированном графе степени k с n вершинами и m ребрами $m=\sum_{v\in G}\rho_1(v)=\sum_{v\in G}\rho_2(v)=kn$, $n=m/k$.

Части, суграфы и подграфы. Граф H называется *частью* графа G, $H\subset G$, если множество его вершин $V(H)$ содержится в множестве $V(G)$, а множество $E(H)$ ребер — в $E(G)$. Если $V(H)=V(G)$, часть графа называется *суграфом*. Например, имеется нулевой суграф, множество ребер которого пусто. Суграф H *покрывает вершины* неориентированного графа G (или является *покрывающим*), если любая вершина последнего инцидентна хотя бы одному ребру из H. Таким образом, если в графе G есть изолированная вершина v, не инцидентная ни одному ребру, покрывающие суграфы этого графа не существуют.

Любое множество B ребер графа G можно считать множеством ребер некоторой части H. Множество вершин этой части состоит из вершин, инцидентных элементам множества B. Если B является множеством ребер другой части H', то $H\subset H'$, причем вершины H', не принадлежащие H, в графе H' изолированы.

Подграфом $G(U)$ графа G с множеством вершин $U\subset V$ называется часть, которой принадлежат все ребра с обоими концами из U. *Звездный граф* для вершины $v\in G$ состоит из всех ребер с началом или концом в вершине v. Множество вершин звездного графа состоит из v и других, инцидентных его ребрам вершин.

Операции с частями графа. *Дополнение* $\bar{H}$ части H определяется множеством всех ребер графа G, не принадлежащих H.

Сумма $H_1\cup H_2$ и *пересечение* $H_1\cap H_2$ частей H_1 и H_2 графа G определяются естественно:

$$V(H_1\cup H_2)=V(H_1)\cup V(H_2);$$
$$E(H_1\cup H_2)=E(H_1)\cup E(H_2);$$
$$V(H_1\cap H_2)=V(H_1)\cap V(H_2);$$
$$E(H_1\cap H_2)=E(H_1)\cap E(H_2).$$

Две части H_1 и H_2 не пересекаются по вершинам, если

они не имеют общих вершин, а значит, и общих ребер. Сумма $H_1 \cup H_2$ не пересекающихся по вершинам частей называется прямой суммой. Аналогично определяется прямая сумма любого числа частей. Части H_1 и H_2 не пересекаются по ребрам, если $E(H_1) \cap E(H_2) = \varnothing$. Например, для любой части H и ее дополнения $\bar{H}$ сумма $G = H \cup \bar{H}$ — прямая по ребрам.

Графы и бинарные отношения. Между ориентированными графами без кратных ребер с множеством вершин $V = \{v_1, \ldots, v_n\}$ и бинарными отношениями на множестве V существует взаимно однозначное соответствие: отношению R соответствует ориентированный граф $G(R)$, в котором ребро (v', v'') существует, если и только если выполнено $v' R v''$. Аналогичное взаимно однозначное соответствие существует между симметрическими бинарными отношениями и неориентированными графами.

Рассмотрим теперь соответствие между операциями над отношениями и операциями над графами. Каждое отношение R имеет отрицание $\neg R$, истинное тогда и только тогда, когда R ложно. Например, для отношения равенства $v' = v''$ отрицанием является отношение неравенства $v' \neq v''$, для отношения ортогональности $v' \perp v''$, определенного для элементов векторного пространства, отрицанием является отношение отличия скалярного произведения от 0: $(v', v'') \neq 0$. Граф $G(\neg R)$ является дополнением графа $G(R)$ по отношению к полному ориентированному графу $K(V)$ с множеством вершин V, на котором задано рассматриваемое бинарное отношение R, и множеством дуг $E(K(V)) = V \times V$. Граф $G(R^{-1})$, где R^{-1} — отношение, обратное R, отличается от графа $G(R)$ тем, что направления всех дуг заменены на обратные.

Отношение R' содержит отношение R, если они определены на одном и том же множестве V и из $v' R v''$ следует $v' R' v''$. В этом случае говорят также, что отношение R' следует из отношения R, и пишут $R' \supset R$. Соответствующие графы $G(R)$ и $G(R')$ имеют одно и то же множество вершин V, а множество $E(R)$ ребер первого является подмножеством множества $E(R)$ ребер второго. Таким образом, $G(R)$ является суграфом графа $G(R')$, т. е. $G(R') \supset G(R)$.

Для любых бинарных отношений R_1 и R_2, заданных на одном и том же множестве V, можно определить сумму (объединение) $R_1 \cup R_2$ и пересечение $R_1 \cap R_2$:

$$v' (R_1 \cup R_2) v'' \equiv v' R_1 v'' \vee v' R_2 v'';$$
$$v' (R_1 \cap R_2) v'' \equiv v' R_1 v'' \,\&\, v' R_2 v''.$$

Соответствующие графы также являются суммой и пересечением

$$G(R_1 \cup R_2) = G(R_1) \cup G(R_2);$$
$$G(R_1 \cap R_2) = G(R_1) \cap G(R_2).$$

Некоторые типы графов хорошо описываются на языке бинарных отношений. Например, нуль-граф $\varnothing(V)$, не имеющий ребер, соответствует нулевому отношению $v'0v''$, не содержащему ни одной пары $(v', v'') \in V \times V$; полному ориентированному графу $K(V)$ соответствует универсальное отношение $v'Uv''$, всегда истинное.

Если R рефлексивно, то $G(R)$ имеет петли во всех вершинах; если R антирефлексивно, то $G(R)$ не имеет петель. Если R транзитивно, то в графе $G(R)$ для каждой пары ребер (v', v'') и (v'', v''') имеется замыкающее ребро (v', v''').

4.2. МАРШРУТЫ, ЦЕПИ И ЦИКЛЫ

Определения. Пусть G — неориентированный граф. *Маршрутом* в G называется такая конечная или бесконечная последовательность ребер $(...e_0, e_1, ..., e_n...)$, что каждые два соседние ребра $e_{i=1}$ и e_i имеют общую инцидентную вершину. Одно и то же ребро может встречаться в маршруте несколько раз. В дальнейшем будут рассматриваться в основном конечные маршруты, т. е. конечные последовательности ребер $(e_1, e_2, ..., e_n)$. В таких маршрутах имеется первое ребро e_1 и последнее ребро e_n. Вершина v_0, инцидентная ребру e_1 и не инцидентная e_2, называется началом маршрута. Если же ребра e_1 и e_2 — кратные, необходимо специальное указание, какую из двух инцидентных им вершин считать началом маршрута. Аналогично определяется конец маршрута.

Вершины, инцидентные ребрам маршрута, кроме начальной и конечной, называются *внутренними* или *промежуточными*. Так как различные ребра маршрута могут быть инцидентными одной и той же вершине, начало и конец маршрута может одновременно оказаться и внутренней вершиной [см. маршрут $(e_1, e_2, e_3, e_4, e_5)$ на рис. 4.6].

Пусть маршрут M $(e_1, e_2, ..., e_n)$ имеет начало v_0 и конец v_n. Тогда его называют соединяющим вершины v_0 и v_n. Число ребер маршрута называется его длиной. Если $v_0 = v_n$, маршрут называют *циклическим*. Отрезок $(e_i, e_{i+1}, ..., e_j)$ конечного или бесконечного маршрута M сам является маршрутом. Он называется участком маршрута M.

Маршрут M называется *цепью*, если каждое ребро встречается в нем не более одного раза, и *простой цепью*, если любая вершина графа G инцидентна не более чем двум его ребрам. Циклический маршрут называют *циклом*, если он является цепью, и *простым циклом*, когда это простая цепь. Однако фактическим циклом (соответственно простым цик-

лом) считают циклически упорядоченное множество ребер, в котором два соседних ребра имеют общую инцидентную вершину. Таким образом, последовательности (e_1, e_2, e_3, e_4), (e_2, e_3, e_4, e_1), (e_3, e_4, e_1, e_2), (e_4, e_1, e_2, e_3) представляют один и тот же цикл. Часто считается, что можно менять по-

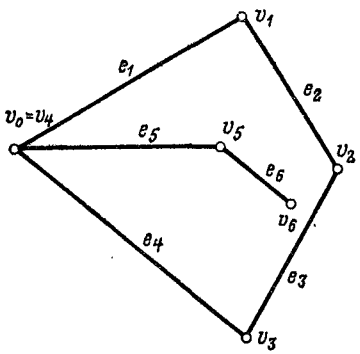

Рис. 4.6

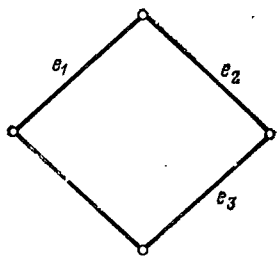

Рис. 4.7

рядок ребер цикла на противоположный, т. е. например, последовательность (e_4, e_3, e_2, e_1) представляет тот же цикл (рис. 4.7).

Участок цепи или цикла является цепью, а участок простой цепи или простого цикла — простой цепью.

Связные компоненты графа. Вершины $v', v'' \in G$ называются связанными, если существует маршрут M с началом v' и концом v''. Легко видеть, что в этом случае существует также маршрут с началом v'' и концом v'. Для этого ребра маршрута M должны идти в противоположном порядке. Пусть вершина $v \in G$ инцидентна более чем двум ребрам маршрута M, связывающего вершины v' и v'', e_i — первое из этих ребер, e_j — последнее $(j > i+1)$. Тогда из маршрута M можно выбросить участок от $(i+1)$-го ребра до $(i-1)$-го. Получится маршрут $M'(e_1, ..., e_i, e_j, ..., e_n)$. Если M' — не простая цепь, то можно продолжать этот процесс выбрасывания его внутреннего участка. В конце концов получится простая цепь M^*, связывающая v' и v''. Таким образом, связанные маршрутом вершины связаны также и простой цепью.

Если вершина $v \in G$ связана с какой-либо другой вершиной v', она связана и сама с собой. Пусть v и v' связывает маршрут M $(e_1, e_2, ..., e_n)$, тогда v и v связывает маршрут M' $(e_1, e_2, ..., e_n, e_{n-1}, ..., e_2, e_1)$, в котором сначала идут ребра

маршрута M, а затем они же, но в обратном порядке. Однако обычно считают, что изолированная вершина также связана сама с собой и отношение связанности двух вершин, заданное на множестве вершин графа G, рефлексивно. Как было указано ранее, оно симметрично. Наконец, оно транзитивно. Если вершина v' связана с вершиной v'' маршрутом M' $(e'_1, \ldots, e'_n)$, а v'' с v''' — маршрутом M'' $(e''_1, \ldots, e''_p)$, то v' связана с v''' маршрутом M $(e'_1, e'_2, \ldots, e'_n, e''_1, \ldots, e''_p)$, в котором сначала идут ребра маршрута M', а затем ребра маршрута M''. Последовательность ребер M — это маршрут. Действительно, кроме пары ребер e'_n, e''_1, остальные пары соседних ребер являются соседними в одном из маршрутов M' или M'' и имеют общую инцидентную вершину. Ребра же e'_n и e''_1 инцидентны вершине v''.

Итак, отношение связанности вершин обладает свойствами отношения эквивалентности и определяет разбиение множества вершин графа на непересекающиеся подмножества V_i. Вершины одного и того же множества V_i связаны друг с другом, а вершины различных множеств V_i и V_j не связаны между собой. Поэтому в графе G нет ребер с концами в разных множествах V_i и V_j, и он может быть разложен в прямую сумму подграфов: $G = \bigcup G(V_i)$.

Граф G называется *связным*, если все его вершины связаны между собой. Поэтому все подграфы $G(V_i)$ связны и называются связными компонентами рассматриваемого графа.

Расстояния. Пусть G — связный неориентированный граф, v' и v'' — любые две его вершины. Тогда существует связывающая их простая цепь M $(e_1, e_2, \ldots, e_q)$. Если количество q ребер этой цепи — не минимальное из возможных, то существует цепь M' $(e'_1, e'_2, \ldots, e'_{q'})$, связывающая v' и v'' и имеющая меньшее число ребер. Если и M' не минимально, можно найти связывающую v' и v'' цепь с еще меньшим количеством ребер и т. д. Однако этот процесс уменьшения числа ребер можно повторить не более q раз, так как это число каждый раз уменьшается не меньше чем на единицу. Поэтому существует связывающая v' и v'' цепь $\tilde{M}$ $(\tilde{e}_1, \tilde{e}_2, \ldots, \tilde{e}_p)$ с минимальным количеством ребер p. Минимальная длина простой цепи с началом v' и концом v' называется *расстоянием* d (v', v'') между этими вершинами (в § 4.4 будут рассмотрены более общие понятия длины цепи и расстояния между вершинами).

Считая каждую вершину v неориентированного графа связанной с самой собой, мы по существу ввели нулевые маршруты, не содержащие ребер, с началом и концом в любой вершине $v \in G$. В соответствии с этим расстояние $d(v, v)$ между вершиной v и ею самой равно 0. Для любой пары $v', v'' \in G$ различных вершин $d(v', v'') > 0$, так как связывающая эти вершины цепь состоит хотя бы из одного ребра.

Расстояние $d(v', v'')$ удовлетворяет *аксиомам метрики:*

1) $d(v', v'') \geqslant 0$, причем $d(v', v'') = 0$ тогда и только тогда, когда $v' = v''$;

2) $d(v', v'') = d(v'', v')$;

3) справедливо *неравенство треугольника*: $d(v', v'') + d(v'', v''') \geqslant d(v', v''')$.

В особом доказательстве нуждается только последнее свойство. Пусть $M'(v', v''')$ и $M''(v'', v''')$ — минимальные простые цепи, связывающие v' с v'' и v'' с v'''. Тогда последовательность ребер $M(e'_1, e'_2, \ldots, e'_{p'}, e''_1, e''_2, \ldots, e''_{p''})$, в которой сначала идут ребра цепи M', а затем ребра цепи M'', — это маршрут, связывающий v' и v''' и имеющий длину $d(v', v'') + d(v'', v''')$. Как показано ранее, если этот маршрут не является простой цепью, то можно найти более короткую цепь $\tilde{M}$, связывающую v' и v''. Поэтому во всяком случае минимальная связывающая эти вершины цепь имеет длину, не превосходящую суммы $d(v', v'') + d(v'', v''')$.

Диаметр, радиус и центр графа. Пусть G — конечный неориентированный связный граф. Тогда можно определить его *диаметр* $d(G) = \max\limits_{v', v'' \in G} d(v', v'')$. Кратчайшие простые цепи, связывающие вершины $v', v'' \in G$ с максимальным расстоянием между ними, называются *диаметральными простыми цепями.*

Пусть v — произвольная вершина рассматриваемого графа G. *Максимальным удалением* в графе G от вершины v называется величина $r(v) = \max\limits_{v' \in G} d(v, v')$. Вершина v называется *центром* графа G, если максимальное удаление от нее принимает минимальное значение $r(G) = \min\limits_{v' \in G} r(v')$. Максимальное удаление $r(G)$ от центра называется *радиусом* графа, а любая кратчайшая цепь от центра v до максимально удаленной от него вершины v' — *радиальной цепью.*

Центр не обязательно должен быть единственным. Например, в *полном неориентированном графе* $K(V)$, в кото-

ром любые две различные вершины v', $v''\in V$ соединены ребром, радиус равен единице и любая вершина $v\in V$ является центром.

Протяженности. Пусть G — конечный, связный неориентированный граф, m — число его ребер. Количество последовательностей ребер этого графа без повторений равно $m!$, т. е. конечно. Следовательно, конечно и число простых цепей, в которых ребра не повторяются. Протяженностью $g(v', v'')$ между вершинами $v', v''\in G$ называется максимальная из длин связывающих эти вершины простых цепей.

Если исключить из этих простых цепей циклы, то для любой вершины $v\in V$ $g(v, v)=0$. В этом случае протяженность также удовлетворяет аксиомам метрики. Как и для расстояния, в доказательстве нуждается лишь неравенство треугольника. Пусть $L(v', v''')$ — самая длинная простая цепь, соединяющая вершины v' и v'''; $L^*(v'', v')$ — какая-то простая цепь с началом v'' и концом v'; $\widetilde{L}(v'', v^{IV})$ — участок последней цепи до первого пересечения с цепью L (рис. 4.8). Тогда

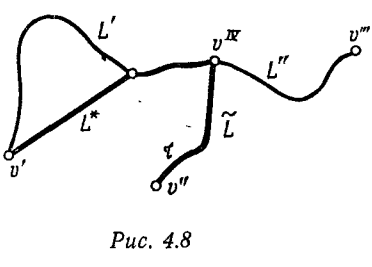

Рис. 4.8

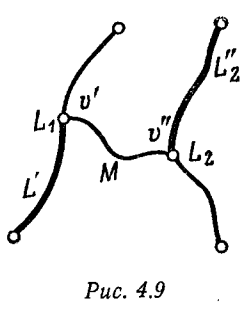

Рис. 4.9

v^{IV} делит цепь на два участка L' и L'' (один из этих участков или же участок $\widetilde{L}$ может быть пустым). Участки L' и $\widetilde{L}$ составляют простую цепь с началом v' и концом v'', а L'' и $\widetilde{L}$ — простую цепь, соединяющую v'' и v''', причем сумма их длин не меньше длины цепи L. Значит, и сумма максимальных длин путей с теми же началами и концами, равная $g(v', v'')+g(v'', v''')$, не меньше длины цепи L, которая равна $g(v', v''')$.

В графе G существуют *диаметральные по протяженности* или *длиннейшие простые цепи*. Их длина l_0 называется диаметром протяженности. Для каждой вершины v существуют самые длинные простые цепи с концом в этой вершине. Их длина $g(v)=\max\limits_{v'\in V} g(v, v')$ называется *числом протяженности* для вершины v. Центрами протяженности называются вершины s_0 с минимальным числом протяженности $g_0=g(s_0)=\min\limits_{v\in V} g(v)$. Самые длинные простые цепи с началом в центре

протяженности называются радиальными по протяженности, а их длина g_0 — радиусом протяженности.

Любые диаметральные по протяженности цепи L_1 и L_2 пересекаются. Если бы это было не так, нашлась бы простая связывающая цепь $M(v', v'')$, такая, что v' лежит на L_1, v'' — на L_2, а остальные вершины M не принадлежат L_1 и L_2 (рис. 4.9). Пусть L_1' и L_2'' — более длинные из участков, на которые разбиваются L_1 и L_2 вершинами v' и v''. Тогда длина простой цепи $L_1' \cup M(v', v'') \cup L_2''$ больше l_0.

Матрица и цепи. Произведение графов. Пусть G и H — два графа, оба ориентированные или неориентированные, без кратных ребер с одним и тем же множеством вершин V. В *произведении* этих графов $F=GH$ ребро (v', v'') существует в том и только том случае, когда для некоторой вершины $v \in V$ существуют ребра $(v', v) \in G$, $(v, v'') \in H$, т. е. существует маршрут из двух ребер с началом v' и концом v'', причем первый элемент маршрута принадлежит графу G, а второй — графу H.

Матрица смежности произведения графов равна произведению матриц смежности сомножителей, только вместо обычных произведений и сумм нужно рассматривать логические:

$$\delta_{ij}^{(F)} = \bigvee_{k=1}^{n} \left(\delta_{ik}^{(G)} \,\&\, \delta_{kj}^{(H)}\right).$$

Для графов с кратными ребрами в произведении GH кратностью ребра (v', v'') считается число маршрутов длины 2 в графе $G \cup H$ таких, что первое ребро содержит v' и принадлежит G, а второе ребро содержит v'' и принадлежит H. При таком определении

$$\delta_{ij}^{(F)} = \sum_{k=1}^{n} \delta_{ik}^{(G)} \delta_{kj}^{(H)}.$$

Таким образом, произведение графов без кратных ребер может оказаться имеющим кратные ребра.

В дальнейшем (см. § 4.4) в связи с проблемой изоморфизма графов будет дано другое определение произведения графов, точнее, их квадратов.

Прямые произведения графов. Пусть даны графы G_1 и G_2 с множествами вершин V_1 и V_2. У прямого произведения этих графов $F=G_1 \times G_2$ множеством вершин является прямое произведение $V=V_1 \times V_2$. Ребра произведения могут быть определены различными способами.

При одном из определений в произведении F графов G_1 и G_2 ребро (v_1', v_2'), (v_1'', v_2'') имеется тогда, когда в графе G_1 есть ребро (v_1, v_1'') или в графе G_2 — (v_2', v_2''). В другом определении для этого требуется существование обоих этих ребер. В первом случае элементы матрицы смежности определяются формулой $\delta_{(i_1,i_2)(j_1,j_2)} = \delta_{i_1j_1} \vee \delta_{i_2j_2}$, во втором $\delta_{(i_1,i_2)(j_1,j_2)} = \delta_{i_1j_1} \cdot \delta_{i_2j_2}$. Таким образом, в обоих случаях матрица смежности произведения является кронеккеровским произведением матриц смежности сомножителей, но в первом случае произведение элементов матрицы понимается как логическая сумма $\vee$ (или), а во втором — как логическое (и обычное) произведение (и).

Чаще всего используется третье определение, согласно которому ребро (v_1', v_2'), (v_1'', v_2'') существует в тех и только тех случаях, когда в графах G_1 и G_2 есть соответственно ребра (v_1', v_1'') и (v_2', v_2'') или $v_1' = v_2'$, а в графе G_2 есть ребро (v_2', v_2'') или, наконец, в графе G_1 есть ребро (v_1', v_2'), а $v_1'' = v_2''$. Пусть $\Delta' = E \vee \Delta$, где E — единичная матрица, а Δ — матрица смежности рассматриваемого графа:

$$\delta'_{ij} = \begin{cases} \delta_{ij}, & i \neq j; \\ 1, & i = j. \end{cases}$$

Тогда матрица смежности прямого произведения графов G_1 и G_2 при таком определении равна $\Delta^{(F)} = \Delta'^{(G_1)} \times \times \Delta'^{(G_2)} - E^{(G_1)} \times E^{(G_2)}$, т. е.

$$\delta_{(i_1,i_2)(j_1,j_2)} = \begin{cases} \delta'_{i_1j_1}\delta'_{i_2,j_2}, & i_1 \neq j_1 \text{ или } i_2 \neq j_2; \\ 0, & i_1 = j_1 \text{ и } i_2 = j_2. \end{cases}$$

Аналогично определяются прямые произведения любого множества графов.

Задача о кенигсбергских мостах. Постановка и решение этой задачи Эйлером знаменует начало разработки теории графов. Расположение мостов в г. Кенигсберге в его время приведено на рис. 4.10, *а*. Требуется пройти каждый мост по одному разу и вернуться в исходную часть города. Можно построить граф задачи, в котором каждой части города соответствует вершина, а каждому мосту — ребро, инцидентное вершинам, относящимся к соединяемым им частям (этот граф изображен на рис. 4.10, *б*).

Попробуем каким-либо способом обойти мосты. После того как мы пройдем какой-нибудь из них, нужно будет ид-

ти по мосту, связывающему часть города, в которую мы попали, с некоторой другой частью. Следовательно, обходу мостов соответствует последовательность ребер графа задачи, в которой два соседних ребра имеют общую вершину, т. е. маршрут. Так как в конце обхода нужно вернуться в исходную часть города и на каждом мосту нужно побывать по одному разу, этот маршрут является простым циклом, содержащим все ребра графа. В дальнейшем такие циклы

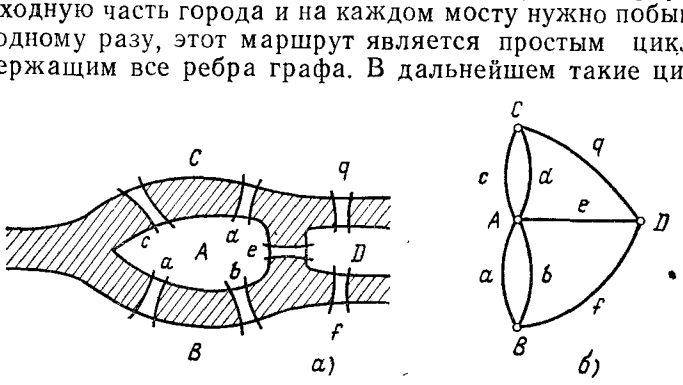

Рис. 4.10

стали называть *эйлеровыми,* а графы, имеющие эйлеровы циклы, — *эйлеровыми графами.*

Эйлеров цикл можно считать следом пера, вычерчивающего этот граф, не отрываясь от бумаги. Таким образом, эйлеровы графы — это такие графы, которые можно изобразить одним росчерком пера, причем процесс такого изображения начинается и кончается в одной и той же точке.

Условия, при которых граф эйлеров.

Теорема 4.1 (Эйлера). Конечный неориентированный граф G эйлеров тогда и только тогда, когда он связен и степени всех его вершин четны.

Необходимость этих условий доказать легко. В несвязном графе каждый цикл принадлежит какой-либо его связной части, т. е. не проходит через все его ребра (кроме случая, когда все компоненты связности графа, кроме одной, — изолированные вершины). Кроме того, каждый раз, когда эйлеров цикл приходит в какую-нибудь вершину, он должен выйти из нее по другому ребру, т. е. инцидентные вершинам ребра можно разбить на пары соседних в эйлеровом цикле. Исключением являются ребра, инцидентные началу эйлерова цикла, совпадающему с его концом. Сначала цикл выходит из этой вершины по какому-либо ребру. Затем он, возможно, несколько раз возвращается в эту

вершину, каждый раз входя по новому ребру и выходя также по новому, но другому ребру. В конце концов он возвращается в исходную вершину по не затронутому ранее ребру. Таким образом, и в этом случае инцидентные этой вершине ребра также распадаются на пары соседних в эйлеровом цикле, но одна из этих пар состоит из ребра, проходимого в начале процесса, и другого, замыкающего этот процесс.

Теперь нужно доказать достаточность этих условий. Пусть G — конечный связный неориентированный граф с четными степенями всех вершин. Начнем построение эйлерова цикла с произвольной вершины v графа G. Каждый раз, когда мы добавляем к маршруту новое ребро и приходим в новую вершину, число свободных ребер в этой вершине изменяется на единицу. Если до этого оно было четным, то теперь оно становится нечетным и не может оказаться нулем. После ухода из этой вершины число свободных инцидентных ей ребер уменьшается еще на единицу и вновь становится четным. Исключением является исходная вершина, у которой после начала процесса число свободных ребер нечетно и остается нечетным после каждого возвращения в эту вершину и ухода из нее.

Описанный ранее процесс построения цикла может закончиться лишь в той вершине, откуда он начинался, но при этом не обязательно, чтобы он проходил через все ребра графа. Принадлежащие ему ребра порождают связную часть P графа G, в которой степени всех вершин четны. Значит, они четны и для разности $G \setminus P$. Так как граф G связен, в дополнении $G \setminus P$ существует хотя бы одна вершина v', принадлежащая также части P. Начиная с этой вершины, можно провести, как и ранее, построение цикла P' в $G \setminus P$, кончающегося также в вершине v'.

Эта вершина, кроме того, разбивает цикл P на два участка: P_1 с началом v и концом v' и P_2 с началом v' и концом v. Тогда $P_1 \cup P' \cup P_2$ — также цикл, начинающийся и кончающийся в вершине v, но имеющий большее число ребер. Если и этот цикл не проходит через все ребра, этот процесс расширения цикла можно повторить. Каждый раз число ребер в цикле увеличивается не менее чем на два, значит, в конце концов эйлеров цикл будет построен. □

В графе задачи о Кенигсбергских мостах (рис. 4.10) все вершины имеют нечетную степень. Следовательно, ее решение невозможно.

Эйлеровы цепи. Так называется цепь, включающая все ребра данного конечного неориентированного графа G, но

имеющая различные начало v' и конец v''. Чтобы в графе существовала эйлерова цепь, необходимы его связность и четность степеней всех вершин, кроме начальной v' и конечной v''. Последние две вершины должны иметь нечетные степени: из v' мы лишний раз выходим, а в v'' лишний раз входим. Эти условия достаточны для существования эйлеровой цепи. Доказательство — то же самое, как и для условий, достаточных для существования эйлерова цикла. Можно искать наименьшее число не пересекающихся по ребрам цепей, покрывающих конечный связный граф G. Пусть G не является эйлеровым графом, k — число его вершин нечетной степени. Ранее было доказано, что k — четно. Каждая вершина нечетной степени должна быть концом хотя бы одной из покрывающих цепей. Следовательно, число последних не меньше, чем $k/2$. Но ограничиться $k/2$ цепями не так трудно. Соединим вершины нечетной степени попарно $k/2$ ребрами произвольным образом. Тогда степень каждой из этих вершин увеличится на единицу и станет четной. Получится эйлеров граф, в котором существует эйлеров цикл. Выбросим теперь обратно присоединенные ребра. При выбрасывании первого ребра эйлеров цикл превратится в эйлерову цепь, а при выбрасывании каждой следующей цепи одна из возникших к этому моменту цепей разобьется на две части. Таким образом, общее число этих цепей станет равно $k/2$. (Заметим, что при построении никогда не выбрасываются соседние по эйлерову циклу ребра.)

Случай конечного ориентированного графа. Чтобы в конечном ориентированном графе G существовал эйлеров цикл, необходимо и достаточно равенство степеней вершин этого графа по входящим и выходящим ребрам: $\forall v \in G(\rho_1(v) = \rho_2(v))$. Доказательство буквально то же, что и для неориентированного графа, только вместо определения четности числа оставшихся свободных ребер показывается, что в процессе построения эйлерова цикла для всех вершин, кроме исходной для построения очередного подцикла, сохраняется баланс между количествами свободных (еще не пройденных) входящих и выходящих ребер. Так как любому неориентированному графу канонически соответствует ориентированный, в котором каждое ребро заменено двумя ребрами, инцидентными тем же вершинам и идущими в противоположном направлении (причем у этого графа для каждой вершины степени по входящим и выходящим вершинам равны степени этой вершины

в исходном неориентированном графе, а значит, и между собой), то отсюда следует справедливость утверждения: *в конечном связном графе всегда можно построить ориентированный цикл, проходящий через каждое ребро по одному разу в каждом из двух направлений.*

Такой цикл иногда называют *способом обхода всех ребер графа.* Он используется во многих прикладных задачах, связанных с графами.

Гамильтоновым циклом называется простой цикл, проходящий через все вершины рассматриваемого графа. Такой цикл существует далеко не во всяком графе. Более того, через любые две вершины рассматриваемого графа может проходить простой цикл (в этом случае граф G называется *циклически связным),* а гамильтонов цикл при этом может отсутствовать.

Рис. 4.11

На рис. 4.11 изображены граф с гамильтоновым циклом (рис. 4.11, *а*) и циклически связный граф, в котором нет гамильтонова цикла (рис. 4.11, *б*). В последнем, чтобы пройти через вершины A, B, C, гамильтонов цикл должен содержать все лежащие на этих сторонах ребра. Но тогда он не проходит через расположенную в центре треугольника вершину О.

Иногда можно построить проходящую через все вершины графа простую цепь с началом и концом в различных заданных вершинах v', $v''\in G$. Такая цепь тоже называется *гамильтоновой.*

4.3. НЕКОТОРЫЕ КЛАССЫ ГРАФОВ И ИХ ЧАСТЕЙ

Дерево и лес. Неориентированное дерево — это связный неориентированный граф без циклов, а значит, без петель и кратных ребер. Несвязный (неориентированный) граф без циклов называется *лесом;* связные компоненты леса являются деревьями. Любая часть леса или дерева также не имеет циклов, т. е. является лесом или деревом. Любая цепь в таком графе является простой. Иначе она содержала бы цикл.

Теорема 4.2. Любые две вершины дерева v' и v'' связаны одной и только одной цепью.

Действительно, пусть L_1 и L_2 — две различные цепи с началом v' и концом v''. Тогда в цепи L_2 есть хотя бы одно ребро e, не принадлежащее цепи L_1. Если какая-либо из инцидентных ребру e вершин не лежит на L_1, то соседнее ребро цепи L_2, имеющее концом эту вершину, также не принадлежит цепи L_1. Таким образом, можно, если это понадобится, расширить участок цепи L_2, состоящий из не принадлежащих цепи L_1 ребер, пока в обоих концах этого участка не окажутся вершины, лежащие на цепи L_1 (встречи с этой цепью обязательно произойдут, ведь в обоих концах цепи L_2 расположены вершины v' и v'', лежащие также и на цепи L_1). Между этими вершинами, отличающимися друг от друга, так как в дереве G нет циклов, лежит участок цепи L_1, не имеющий с построенным ранее участком цепи L_2 общих вершин, кроме концов (участок цепи L_2 вообще не имеет с цепью L_1 общих вершин, кроме концов). Тогда из обоих этих участков составляется цикл, которого не может быть в дереве G. □

Верно и обратное: если любые две вершины графа G связаны единственной цепью, этот граф является деревом. Действительно, из простого цикла всегда можно составить две непересекающиеся простые цепи с одними и теми же началом и концом.

Концевые вершины и ребра. Вершина v графа G называется *концевой*, или висячей, если ее степень $\rho(v)$ равна единице. Инцидентное концевой вершине ребро также называется концевым. *Если конечное дерево состоит более чем из одной вершины, оно имеет хотя бы две концевые вершины и хотя бы одно концевое ребро.* Действительно, пусть v — вершина дерева G. Так как она связана с другими вершинами, из нее выходит хотя бы одно ребро. Если другой конец v' этого ребра не является концевой вершиной, из него выходит еще одно ребро. Из другого его конца v'' выходит еще одно ребро и т. д. Таким образом, строится цепь, проходящая все время через новые вершины. Иначе часть этой цепи оказалась бы циклом. Так как наше дерево конечно, процесс построения этой цепи должен закончиться, причем последнее ее ребро и одна из инцидентных ей вершин являются концевыми.

Если v — концевая вершина, она является и второй концевой вершиной дерева. Если же вершина v — не концевая, из нее выходит еще хотя бы одно ребро. Начиная с него, можно построить цепь, идущую из v в другую сторону, и в конце ее найти другую концевую вершину.

Дерево с корнем; ветви. Пусть в дереве G отмечена вершина v_0. Эту вершину называют *корнем* дерева G, а само дерево — *деревом с корнем*. В таком дереве можно естественным образом ориентировать ребра. Вершину v' ребра (v', v'') можно соединить единственной цепью L с корнем v_0. Если эта цепь не содержит ребра (v', v''), в последнее вводится ориентация от v' к v''; в противном случае — от v'' к v'. Такая ориентация согласована с ориентацией того же ребра, определенной через вершину v''. Действительно, если цепь L не содержит ребра (v', v''), то, добавив это ребро, мы получим единственную цепь L', связывающую v_0 и v'' и проходящую через рассматриваемое ребро. Если же цепь L содержит ребро (v', v''), то, отбросив его, мы получим цепь L', связывающую v_0 и v'' и не содержащую этого ребра.

Ориентированное таким образом дерево с корнем называется *ориентированным деревом*. В нем все ребра имеют направление от *корня* (рис. 4.12). Если изменить направления всех ребер ориентированного дерева на противоположные (*к корню*), получится ориентированный граф, который тоже называют ориентированным деревом, а иногда *сетью сборки*.

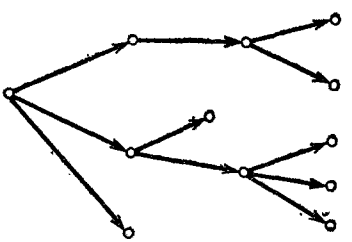

Рис. 4.12

В каждую вершину ориентированного дерева (за исключением корня) входит только одно ребро, т. е. эта вершина является концом одного и только одного ребра. Действительно, такое ребро является последним в цепи, связывающей рассматриваемую вершину с корнем, а эта цепь единственна. В корень не входит ни одно ребро, все инцидентные корню ребра связывают его со своими вторыми концами, значит, корень является их началом. Любое дерево можно ориентировать, выбрав в качестве корня любую его вершину. Это доказывает, что в конечном дереве число вершин на единицу больше числа ребер.

Пусть v — вершина дерева G с корнем v_0, $B(v)$ — множество всех вершин, связанных с корнем цепями, проходящими через вершину v (т. е. имеющими ребра, инцидентные v). Это множество порождает подграф $G(v)$, который называется *ветвью* вершины v в дереве с корнем v_0. Как

подграф дерева, т. е. графа без циклов, $G(v)$ тоже не имеет циклов; любые вершины $v', v'' \in B(v)$ связаны в этой ветви маршрутом, составленным из участков цепей $L(v')$ и $L(v'')$ от корня v_0 до этих вершин, начинающихся в вершине v. Нужно только доказать, что эти участки целиком погружены в $G(v)$, но это справедливо, так как в каждую вершину такого участка ведет цепь из корня v_0, являющаяся начальным участком соответствующей цепи $L(v')$ или $L(v'')$ и проходящая через вершину v. Таким образом, ветвь $G(v)$ связна и сама является деревом.

Рассмотрим $G(v)$ как дерево с корнем v. Для любой вершины $v' \in G(v)$ связывающая ее с v цепь $L(v, v')$ — это конец цепи $L(v_0, v')$, связывающей в дереве G эту вершину с корнем v_0. Следовательно, последнее ребро этой цепи, входящее в вершину v' в исходном дереве G и в его ветви $G(v)$, ориентируется в обоих этих деревьях с корнем одинаково, так что вершина v' является его концом.

Типы вершин и центры деревьев. Пусть дано конечное дерево G. Назовем его концевые вершины *вершинами типа 1*. Отметим, что если дерево имеет более двух вершин, то среди них есть неконцевые вершины. Действительно, пусть v_1 и v_2 — концевые вершины. Они должны быть связаны цепью. Если эта цепь состоит из одного ребра, то v_1 и v_2 не связаны ни с какими другими вершинами, что противоречит связности дерева; если же эта цепь состоит из двух или более ребер, то она проходит через вершины степени большей или равной двум, т. е. неконцевые вершины

Удалим из дерева G все вершины типа *1* и инцидентные им концевые ребра. Останется связный граф G' без циклов, т. е. дерево. Действительно, для любых вершин $v', v'' \in G'$ связывающая их в дереве G цепь $L(v', v'')$ не может проходить через концевые ребра и целиком лежит в G'. Дерево G' также имеет концевые вершины, которые будем называть *вершинами типа 2* в дереве G. Аналогично определяются вершины типов *3*, *4* и т. д.

Легко видеть, что в конечном дереве G имеются вершины лишь конечного числа типов, причем число вершин максимального типа равно единице или двум, так как в соответствующем дереве каждая вершина является концевой. Каждая вершина дерева G, кроме концевых, имеет соседей на единицу меньшего типа. Действительно, в дереве $G^{(k-1)}$, определяющей предыдущий тип, она не была концевой, а в $G^{(k)}$ стала. Значит, при построении этого дерева были

удалены некоторые инцидентные этой вершине концевые ребра дерева $G^{(k-1)}$ и другие их концы — соседние с рассматриваемой вершиной вершины типа k—1. Остальные инцидентные рассматриваемой вершине ребра, кроме, может быть, одного, были удалены раньше, и соответствующие им соседние вершины имеют меньший тип. Одно же инцидентное ребро остается, если только рассматриваемая вершина —не единственная, имеющая максимальный тип. Другой конец этого ребра — вершина старшего типа, если тип рассматриваемой вершины немаксимальный, и вершина того же типа — в противном случае (рис. 4.13, цифры у вершин означают их типы).

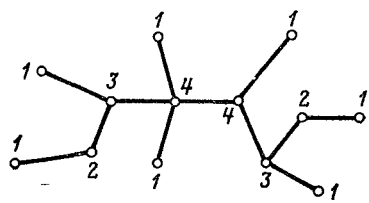

Рис. 4.13

Теорема 4.3. Центрами дерева являются вершины максимального типа k и только они.

Любая цепь L с началом в вершине v_0 максимального типа идет последовательно по вершинам уменьшающихся типов. Ее первое ребро (v_0, v_1) ведет либо в вершину v_1 ранга k, либо в вершину меньшего типа. Следующие ребра (v_1, v_2), (v_2, v_3) и т. д. уже обязательно ведут в вершины меньшего типа, так как единственная соседняя вершина большего или равного типа — это вершина, инцидентная уже использованному ребру. Таким образом, для длины l цепи L с началом в вершине максимального типа k имеем: $l(L) \leqslant k$, если есть еще одна вершина типа k; $l(L) \leqslant k-1$, если такая вершина единственна. Так как у каждой вершины дерева есть хотя бы одна соседняя, тип которой на 1 меньше, то из вершины максимального типа k можно построить путь длины k—1. Следовательно, максимальное удаление от вершины v_0 максимального типа в дереве равно ее типу k или k—1. Остается показать, что если вершина v имеет немаксимальный тип, она является началом более длинной цепи. Построим сначала цепь $L(v, v_0)$, связывающую эту вершину с вершиной максимального типа k. Если v_0 — единственная вершина типа k, она связана по крайней мере с двумя вершинами типа k—1, так как в графе $G^{(k-1)}$ не является концевой. Следовательно, можно построить цепь $L(v_0, v')$ длины k—1, не проходящую через последнее ребро цепи $L(v, v_0)$ и не имеющую с этой цепью общих

вершин (иначе в дереве G был бы цикл). Цепь $L=L(v, v_0) \cup L(v_0, v')$ имеет длину большую, чем k—1.

Если в дереве G есть две вершины v_0 и v_1 максимального типа k, то тоже можно построить связывающую v с v_0 цепь $L(v, v_0)$. В случае, когда эта цепь проходит через другую вершину v_1 максимального типа k, ее длина не меньше двух. Тогда вершина v_0 является началом цепи $L(v, v')$ длиной k—1 с первым ребром (v_0, v''), ведущим в вершину v'' типа k—1. Эта цепь не имеет с первоначально построенной цепью $L(v, v_0)$ общих вершин, иначе в дереве G был бы цикл. Следовательно, $L=L(v, v_0) \cup L(v_0, v')$ — цепь длиной, не меньшей $k+1$, с началом в рассматриваемой вершине v.

Наконец, если цепь $L(v, v_0)$ не проходит через другую вершину v_1 максимального типа, длину $k+1$ или больше имеет начинающаяся в вершине v цепь $L=L(v, v_0) \cup \cup (v_0, v_1) \cup L(v_1, v'')$, где цепь $L(v_1, v')$ длиной k—1 строится так же, как в предыдущем случае цепь $L(v_0, v')$.

Итак, для любой вершины немаксимального типа ее максимальное удаление больше k. □

Из теоремы непосредственно следует, что дерево имеет либо один, либо два центра.

Диаметральные цепи в деревьях проходят через его центр v_0 или через оба центра v_0 и v_1, если их два. В первом случае длина диаметральной цепи равна $2k$—2, а во втором $2k$—1, где k — максимальный тип вершин дерева G (подразумевается, что оно конечное). Такая цепь строится из двух радиальных цепей $L(v_0, v')$ и $L(v_0, v'')$ или $L(v_0, v')$ и $L(v_1, v'')$ длиной k—1, не имеющих общих вершин и существующих, как это показано ранее. Когда в дереве G два центра — v_0 и v_1, эти цепи соединяются ребром (v_0, v_1) и получается цепь длиной $2k$—1.

Докажем, что более длинных цепей в дереве G нет. Пусть L — какая-либо цепь, $v_0, v_1, \ldots, v_h$ — последовательность ее вершин, $k_0, k_1, \ldots, k_h$ — их типы. Вначале типы вершин могут расти. Пусть $k_0<k_1<\ldots<k_g$ $(g \leqslant h)$. Число вершин в этом участке последовательности вершин не превосходит максимального типа k, а число соединяющих их ребер пути L не больше k—1. Соседних вершин одинакового ранга не больше двух, это центры дерева G, если их два. В последнем случае к участку возрастания типа вершин вдоль цепи L может добавиться одно ребро неубывания типа.

Если цепь L длиннее, возрастание типа вершин вдоль

нее должно смениться убыванием. Пусть $v_{g'}$ — первая вершина цепи L, тип которой меньше типа предыдущей: $k_{g'-1} > k_{g'}$. Тогда тип следующей вершины еще меньше, и далее вдоль пути типы вершин убывают: $k_{g'-1} > k_{g'} > \ldots > k_h$. Действительно, у вершины $v_{g'}$ только одна соседняя вершина имеет больший тип — это вершина $v_{g'-1}$, а все остальные соседние вершины, в том числе и вершина $v_{g'+1}$, имеют тип меньший, чем $k_{g'}$. Точно так же у вершины $v_{g'+1}$ больший тип имеет только вершина $v_{g'}$, а типы остальных соседей меньше $k_{g'+1}$ и т. д.

Таким образом, последовательность $v_0, v_1, \ldots, v_h$ можно разбить на участки возрастания типа $v_0, v_1, \ldots, v_g$, его убывания $v_{g'}, v_{g'+1}, \ldots, v_h$ ($g' = g$ или $g+1$) и, когда цепь проходит через две вершины v_g и $v_{g'}$ максимального типа, участок постоянства типа. Длина участка убывания типа также не превосходит $k-1$, а общая длина цепи L не больше $2k-2$ или $2k-1$ в случае, когда эта цепь проходит через два центра дерева G.

Остается доказать, что цепь L, не содержащая центра, имеет меньшую длину и не является диаметральной. Пусть $k' < k$ — максимальный ранг вершины v', через которую проходит эта цепь. Тогда до этой вершины типы растут, а после нее падают. Следовательно, длина участков этой цепи до и после вершины v' не превышает $k'-1$, а общая длина цепи L не больше $2k'-2 < 2k-2$. Если в дереве G два центра — v_0 и v_1, а цепь L проходит только через один из них, ее длина не больше $2k-2$ и она также не является диаметральной.

Цикломатическое число графа. Пусть G — конечный неориентированный граф. Его *цикломатическим числом* называется

$$\gamma(G) = \nu_c + \nu_e - \nu_v,$$

где ν_c — число связных компонент графа; ν_e — число его ребер, а ν_v — число вершин. Цикломатическое число дерева равно нулю, цикломатическое число леса — сумме цикломатических чисел своих связных компонент-деревьев, т. е. также равно нулю. Цикломатические числа остальных конечных графов положительны. Так как цикломатическое число несвязного графа равно сумме цикломатических чисел связных компонент, достаточно рассмотреть связный граф G.

Если этот граф — дерево, его цикломатическое число равно нулю. В противном случае в графе G есть цикл $Z(e_1, e_2, \ldots, e_h)$. Можно выбросить любое ребро этого

цикла, и граф останется связным. Действительно, пусть $M(e_1', e_2', \ldots, e_l')$ — маршрут, связывающий вершины v' и $v'' \in G$ и содержащий выброшенное ребро e_q цикла Z (если маршрут не содержит такого ребра, он связывает вершины v' и v'' и в новом графе G'). Это ребро можно заменить маршрутом $M'(e_q, e_{q+1}, \ldots, e_n, e_1, \ldots, e_{q-1})$, состоящим из остальных ребер цикла Z и имеющим те же концы, что и само ребро.

Если в G' еще имеются циклы, можно выбросить следующее ребро, не нарушая связности, и т. д. В конце концов получится неориентированный связной граф G^ν без циклов, т. е. дерево (так как число вершин всех конструируемых графов $G', G'', \ldots, G^\nu$ не меняется, если оно больше единицы, то все ребра выбросить нельзя). Число оставшихся ребер на единицу меньше числа вершин графа G^ν, а значит, и исходного графа G, но в последнем на γ ребер больше. Следовательно, $\gamma(G) = \nu_c(G^\nu) + \nu_e(G^\nu) - \nu_v(G^\nu) + \gamma = \gamma > 0$.

Максимальные графы исключения. Пусть дано семейство $\{F_i\}_1^s$ частей (называемых запрещенными) некоторого графа G^1. Часть H этого графа называется графом исключения, если она не содержит ни одной части из этого семейства, и максимальным графом исключения, если она не вложена в другой граф исключения. Обычно считается, что пустой граф $\varnothing$ не принадлежит семейству запрещенных частей $\{F_i\}_1^s$, иначе графов исключения нет. Если графы исключения существуют, то существуют и максимальные графы исключения. Когда рассматриваемый граф конечный, цепочка графов исключения, в которой каждый следующий содержит все предыдущие, должна оборваться, так как переход к следующему графу исключения в этой цепочке связан с добавлением новых элементов графа G, а их количество конечно. Следовательно, цепочка должна оборваться, и последний входящий в нее граф — это максимальный граф исключения.

Отсюда следует, например, что любой граф G содержит максимальные части без циклов (семейство $\{Z_i\}$ состоит из всех циклов графа G). Можно также ограничить степени $\rho_H(v)$ вершин $v \in G$ в рассматриваемых графах исключения заданными значениями $k(v)$. Для этого семейство $\{F_i\}$ запрещенных частей должно включать все звездные графы $s(v)$ графа G с $(k(v)+1)$ ребрами.

Максимальные паросочетания и устойчивые множества

вершин в графе. Пусть для всех вершин v графа G $k(v)=1$. Тогда графы исключения $H \subset G$ состоят из изолированных вершин и пар вершин v', $v'' \in G$, связанных одним ребром $(v', v'') \in G$. Части H, содержащие только такие пары, называются *паросочетаниями* в графе G. Однако в отличие от приведенной ранее терминологии *максимальным паросочетанием* в конечном графе G называется не просто паросочетание, которое не вложено ни в какое другое, а содержащее максимально возможно число ребер.

Пусть теперь запрещенным множеством частей графа G является множество пар вершин v', $v'' \in G$, инцидентных какому-либо ребру этого графа. Тогда любой граф исключения H состоит из изолированных вершин, не являющихся соседними в графе G. Такие множества вершин графа G называются *устойчивыми*. Как и паросочетание, устойчивое множество называется *максимальным*, если оно содержит максимально возможное число вершин.

Две теоремы о максимальных графах исключения.

Теорема 4.4. Если никакая конечная запрещенная часть F_i не имеет концевых ребер, то максимальный граф H покрывает все вершины графа G.

Действительно, если некоторый граф исключения H' не содержит ребер, инцидентных некоторой вершине $v \in G$, эту вершину вместе с любым инцидентным этой вершине ребром можно добавить в часть H'. В полученной части H'' вершина y является концевой, а любая часть H'' либо не содержит v и тогда является частью H', т. е. не принадлежит семейству $\{F_i\}$, либо содержит вершину v, являющуюся в этой части концевой (изолированные вершины тоже считаются концевыми), и не принадлежит семейству $\{F_i\}$, состоящему из частей без концевых вершин. Таким образом, часть H' не максимальна. □

Теорема 4.5. Пусть G — связный граф и $\{F_i\}_1^s$ — семейство его частей, каждая из которых по ребрам двусвязна. Тогда максимальные графы исключения связны и покрывают все вершины графа G.

Граф G называется двусвязным по ребрам, если после удаления любого его ребра он остается связным. В двусвязном графе нет концевых вершин. Действительно, после удаления концевого ребра, единственного инцидентного такой вершине, она становится изолированной, а соответствующая часть G' графа G — несвязной. Следовательно, максимальный граф исключения H для семейства двусвяз-

ных конечных частей $\{F_i\}_1^s$ покрывает все вершины графа G.

Пусть H — несвязный граф исключения, H_1 — его связная компонента. Последняя не покрывает всех вершин графа G. Значит, в любом маршруте $M(v', v'')$, соединяющем вершины $v' \in H_1$ и $v'' \in G \setminus H_1$, найдется ребро (v_1, v_2) с концами $v_1 \in H_1$ и $v_2 \notin H_1$. Это ребро соединяет компоненту связности $H_1 \subset H$ с некоторой другой компонентой связности $H_2 \subset H$ (так как H покрывает все вершины графа G, $v_2 \in H$). Добавим ребро (v_1, v_2) к части H и получим объемлющую часть H'. Любая часть $H'' \subset H$ либо не содержит ребра (v_1, v_2), т. е. принадлежит графу исключения H и не является частью из семейства $\{F_i\}_1^s$, либо содержит это ребро и не является графом, двусвязным по ребрам. Действительно, после удаления этого ребра H'' становится несвязным графом, так как вершины v_1 и v_2 связаны только им. Значит, $H'' \notin \{F_i\}_1^s$, т. е. H' — тоже граф исключения и H — не максимален. □

Максимальные деревья. Пусть G — связный граф, $\{Z_i\}$ — семейство всех его циклов. Тогда максимальный граф исключения T не содержит циклов. Он покрывает все вершины графа G, так как циклы Z_i не имеют концевых вершин, и связен, так как любой цикл двусвязен по ребрам. Следовательно, T — дерево. Будем называть его *максимальным деревом* в графе G.

Максимальное дерево в конечном связном графе G уже было раньше построено при определении цикломатического числа $\gamma(G)$ графа G. При этом было удалено $\gamma(G)$ ребер. Легко видеть, что все максимальные деревья T графа G имеют одно и то же число вершин, равное числу ν_v вершин графа G, и число ребер, равное $\nu_v - 1$. Следовательно, при построении этих деревьев нужно удалить одинаковое число ребер, равное цикломатическому числу $\gamma(G)$ графа G.

Пусть в графе G удалено $\gamma(G)$ ребер. Если остался связный граф H, его цикломатическое число уменьшилось на $\gamma(G)$ и стало равно нулю. Таким образом, при удалении произвольных $\gamma(G)$ ребер из конечного связного графа G получаем либо максимальное дерево, либо несвязный граф.

Задача о минимальном соединении. Пусть каждому ребру e конечного неориентированного связного графа G приписано некоторое действительное число $\mu(e)$ — *вес* этого ребра. *Минимальным соединением* называется максималь-

ное дерево $T \subset G$ с минимальным *общим весом* $\mu(T) = \sum_{e \in T} \mu(e)$.

Задача о минимальном соединении внешне похожа на значительно более сложную *задачу о коммивояжере,* в которой требуется найти в полном неориентированном конечном графе K_n гамильтонов цикл Z минимального общего веса $\mu(Z) = \sum_{e \in Z} \mu(e)$.

В отличие от этой задачи минимальное соединение легко найти. Его построение начинается с выбора ребра e_1 минимального веса $\mu(e_1)$. На каждом следующем шаге к уже построенной части T_{i-1} добавляется ребро e_i, такое, что часть $T_i = T_{i-1} \cup e_i$ не имеет циклов и из всех ребер, обладающих этими свойствами, e_i имеет минимальный вес $\mu(e_i)$. Если имеется несколько таких ребер, то можно выбрать любое из них.

Пусть лес T_{i-1} не покрывает всех вершин графа G. Тогда существует ребро e, инцидентное вершине $v \notin T_{i-1}$. Его добавление к T_{i-1} не приводит к появлению цикла (e — концевое ребро), и часть T_{i-1} можно расширить. Если T_{i-1} покрывает все вершины G, но несвязен, его тоже можно расширить. Действительно, в маршруте $M(v', v'')$, соединяющем в графе G вершины v' и v'' из различных компонент T_{i-1}, найдется хотя бы одно ребро (v_1, v_2), концы которого принадлежат разным компонентам леса T_{i-1}. В графе $T_i = T_{i-1} \cup (v_1, v_2)$ это ребро является *разделяющим* (т. е. после удаления граф T_i становится несвязным), и через него не проходит никакой цикл. Нет в том графе и циклов, не проходящих через ребро (v_1, v_2), они принадлежали бы лесу T_{i-1}. Следовательно, пока не возникает дерево T_{n-1}, покрывающее все вершины графа G (n — их количество), процесс расширения части T_i продолжается.

Отметим, что все ребра дерева T_{n-1} можно занумеровать в порядке их включения в эту часть графа G. Пусть T — дерево минимального общего веса, покрывающее вершины графа G. Оно существует, так как в конечном графе количество покрывающих деревьев конечно. Рассмотрим ребро $e_i \in T_{n-1} \setminus T$ минимальным номером i. Граф $T' = T \cup e_i$ имеет цикломатическое число 1, т. е. не является деревом. В нем имеется некоторый цикл Z и другого цикла Z' быть не может. Действительно, в последнем имелось бы ребро e', не принадлежащее Z, и, выбросив его из Z', мы полу-

чили бы связный граф T'' с цикломатическим числом 0 и циклом Z. Наконец, цикл Z проходит через ребро e_i: в дереве $T=T'\setminus e$ циклов нет.

В дереве T_{n-1} нет циклов, значит, Z содержит ребро $e'\notin T_{n-1}$. Тогда граф $T_1=T'\setminus e'=T\cup e_i\setminus e'$ покрывают все вершины G и не имеет циклов, т. е. является максимальным деревом в графе G. Так как дерево T имеет минимальный общий вес,

$$\mu(T_1)-\mu(T)=\sum_{e\in T'}\mu(e)-\sum_{e\in T}\mu(e)=\mu(e_i)-\mu(e')\geqslant 0.$$

Однако неравенство весов $\mu(e_i)$ и $\mu(e')$ невозможно, иначе при построении T_i вместо e_i было бы выбрано ребро e'. Действительно, все ребра T_{i-1} принадлежат дереву T, принадлежит ему и ребро e'. Значит, $T'=T_{i-1}\cup e'\subset T$ и в этой части графа G нет циклов.

Итак, $\mu(e_i)=\mu(e')$ и $\mu(T_1)=\mu(T)$, т. е. T_1 — дерево минимального общего веса, максимальное в графе G. В нем больше общих ребер с деревом T_{n-1}, чем в дереве T. Повторив при необходимости такую замену ребер, мы придем в конце концов к дереву T_{n-1} и докажем, что $\mu(T_{n-1})=\mu(T)$, т. е. мы построили максимальное в графе G дерево T_{n-1} минимального общего веса.

Двудольные графы. Граф G называется двудольным, если множество его вершин V распадается на два непересекающихся подмножества V' и V'', таких, что каждое ребро графа G имеет один конец из V', а другой — из V''. Двудольные графы рассматриваются во многих задачах дискретной математики. Одна из них — *задача о различных представителях подмножеств.* Пусть дано множество W и семейство его подмножеств $\{U_i\}$, требуется в каждом множестве этого семейства выбрать некоторый элемент u_i — представитель этого множества — так, чтобы разным множествам U_i и U_j соответствовали бы различные представители u_i и u_j.

Множество V' вершин двудольного графа G, соответствующего этой задаче, — это множество всех элементов W; множество V'' состоит из элементов v_i'', соответствующих подмножествам U_i рассматриваемого семейства. Вершины $w\in W$ и $v_i''\in V''$ являются концами ребра (w, v_i'') в том и только том случае, когда $w\in U_i$. Требуется найти *максимальное паросочетание* (нерасширяемое множество ребер (w, v_i''), в котором все вершины w и v_i'' различны) и прове-

рить, каждая ли вершина $v_i \in V''$ является концом некоторого ребра из этого паросочетания.

Другой пример. Пусть заданы два произвольных множества V' и V'' элементов разной природы (тем самым эти множества не имеют общих элементов) и отношение $v'Rv''$, являющееся истинным или ложным для всех пар $(v', v'') \in V' \times V''$. Тогда это отношение определяет двудольный граф G: ребро (v', v'') принадлежит G, если отношение $v'Rv''$ истинно.

Теорема 4.6. Граф G является двудольным тогда и только тогда, когда все его циклы имеют четную длину.

Для цикла Z двудольного графа это условие выполняется: вершины, через которые этот цикл проходит, поочередно принадлежат множествам V' и V''. Пусть теперь все циклы графа G имеют четную длину. Выберем вершину v_0 в некоторой связной компоненте G_0 графа G. Множество вершин G_0 распадается на множество V вершин с четным расстоянием от v_0 и множество V' вершин с нечетным расстоянием от v_0. Если $v_1 \in V$, $v_2 \in V$, то кратчайшие простые цепи $L_1(v_0, v_1)$ и $L_2(v_0, v_2)$ имеют четную длину. Предположим, что ребро (v_1, v_2) существует. Пусть v' — последняя общая вершина для $L_1(v_0, v_1)$ и $L_2(v_0, v_2)$, т. е. $L_1(v_0, v_1) = L'(v_0, v') \cup L''(v', v_1)$, $L_2(v_0, v_2) = L'(v_0, v') \cup L'''(v', v_2)$ и начальные ребра L'' и L''' различны. Тогда цепь $L''(v', v_1) \cup (v_1, v_2) \cup L'''(v_2, v')$, где $L'''(v_2, v')$ обозначает цепь $L'''(v', v_2)$, пройденную в обратном порядке, является циклом нечетной длины, что противоречит предположению. Таким образом, никакие вершины из V не соединены ребром; аналогичное утверждение доказывается и для V'. Поэтому всякое ребро из G_0 имеет один конец в V, а другой — в V' и G_0 — двудольный граф. Поскольку это верно для любой связной компоненты графа G, то и сам граф G — двудольный. □.

Максимальные двудольные части графа. Пусть запрещенное семейство частей неориентированного графа G состоит из простых циклов Z нечетной длины. Тогда графы исключения можно рассматривать как двудольные, а максимальные графы исключения — это максимальные двудольные части графа G.

Циклы нечетной длины не имеют концевых вершин и являются двусвязными по ребрам. В силу соответствующих общих теорем из этого следует, что максимальные двудоль-

ные части графа G покрывают все его вершины и в каждой связной компоненте этого графа лежит одна связная компонента максимальной двудольной части.

Для построения максимальной двудольной части графа G выберем в каждой его связной компоненте G_0 вершину v_0. Для каждой вершины $v \in G_0$ можно определить ее ранг $r(v, v_0)$ относительно вершины v_0, равный ее расстоянию от вершины v_0. Удалим из G_0 все ребра (v', v''), связывающие вершины v' и v'' рангов одинаковой четности. Так как все ребра кратчайшего пути $L(v_0, ..., v)$, соединяющего вершину v_0 с вершиной $v \in G_0$, соединяют вершины соседних рангов i и $i+1$, имеющих разную четность, они не будут удалены. Значит, часть G_0' графа G_0, состоящая из вершин $v \in G_0$ и неудаленных ребер, — связный граф. Легко видеть, что он двудольный: множество V' состоит из всех вершин четных рангов, множество V'' — из вершин нечетных рангов, и каждое оставшееся ребро соединяет вершину из множества V' с вершиной из множества V''. Значит, в нем нет циклов нечетной длины. Докажем, что, добавив любое удаленное ребро (v_1, v_2), мы получим некоторый цикл нечетной длины. Пусть $r(v_1, v_0) = i$, $r(v_2, v_0) = j$. Числа i и j имеют одинаковую четность. Рассмотрим кратчайшие цепи $L_1(v_0, ..., v_1)$ и $L_2(v_0, ..., v_2)$, соединяющие вершину v_0 с вершинами v_1 и v_2. Будем идти от вершины v_2 к v_0 по цепи L_2, пока первый раз не попадем в вершину $v_h \in L_1$, в крайнем случае $v_h = v_0$. Тогда цикл, состоящий из отрезков $L_1'(v_h, ..., v_i)$, $L_2'(v_h, ..., v_j)$ путей L_1 и L_2 и ребра (v_1, v_2), является простым циклом нечетной длины. Действительно, все вершины этого цикла различны (а значит, различны и ребра), так как цепи L_1 и L_2 — кратчайшие, длина цепи L_1' равна $i-h$, а длина цепи L_2' равна $j-h$. Следовательно, длина цикла Z равна $(i-h)+(j-h)+1$, т. е. нечетна, и G_0' — максимальная двудольная часть графа G_0. Из таких частей, построенных для каждой компоненты, максимальные двудольные части графа G можно составить 2^l способами, где l — число связных компонент графа G.

О максимальном паросочетании в двудольном графе. Пусть G — двудольный граф, $\{(v_i', v_i'')\}(i=1, 2...)$ — паросочетание в нем, т. е. семейство ребер (v_i', v_i''), в котором все вершины $v_i' \in V'$ и $v_i'' \in V''$ различны между собой. Если, кроме вершин v_i'', в множестве V'' имеется еще какая-

либо вершина v'', можно построить часть $S(v'')$ графа G следующим образом.

Каждая вершина множества $S(v'')$ принадлежит *одному из множеств* $V^{(i)}$ ($i=0, 1 \ldots$), определяемых условиями $V^{(0)}=\{v''\}$; для четного $i=2j$:

$$V^{(i+1)} = V^{(2j+1)} = \{v' \mid (v', v'') \in G, v'' \in V^{(2j)}\} \setminus$$
$$\setminus V^{(1)} \cup V^{(3)} \cup \ldots \cup V^{(2j-1)};$$

для нечетного $i=2j+1$:

$$V^{(i+1)} = V^{(2j)} = \{v'' \mid (v', v'') \in \Pi, v' \in V^{(2j-1)}\} \setminus$$
$$\setminus V^{(0)} \cup V^{(2)} \cup \ldots \cup V^{(2j-2)},$$

где Π — рассматриваемое паросочетание. Будем говорить, что вершина $v \in S(v'')$ имеет индекс i, если $v \in V^{(i)}$. Множество ребер части $S(v'')$ состоит из всех ребер $(v', v'') \in G$, инцидентных вершинам v'' четных индексов. Легко видеть, что другой конец такого ребра принадлежит части $S(v'')$. Построение части $S(v'')$ изображено на рис. 4.14.

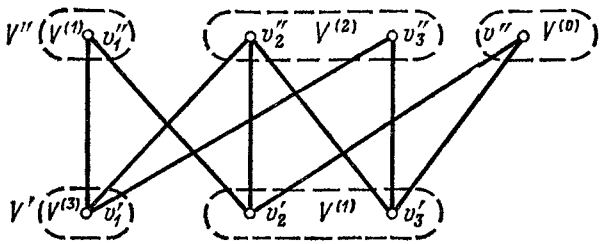

Рис. 4.14

Таким образом, каждое ребро $(v_1, v_2) \in S(v'') \cap \Pi$ соединяет вершину v_1 нечетного индекса с вершиной v_2 следующего, а ребро (v_1, v_2), соединяющее вершину v_2 четного индекса с вершиной v_1 следующего, не принадлежит паросочетанию Π. Это можно доказать по индукции.

Пусть $v' \in S(v'')$ — вершина нечетного индекса, не являющаяся концом никакого ребра из паросочетания Π. В эту вершину ведет *чередующаяся цепь* $L(e_1, e_2, \ldots, e_{2j+1})$ с началом v'', четные ребра которой принадлежат паросочетанию Π, а нечетные — не принадлежат. Тогда паросочетание Π можно расширить: выбросить из него j ребер $e_2, e_4, \ldots, e_{2j}$ и добавить $j+1$ ребер $e_1, e_3, \ldots, e_{2j+1}$.

Если же паросочетание Π максимально, то все вершины

$S(v'')$ нечетного индекса — это концы $v_i' \in V'$ некоторых ребер $(v_i', v_i'') \in \Pi$, а множество вершин четного индекса состоит из других концов v_i' этих ребер и вершины v''. Так как все ребра, инцидентные вершинам v четного индекса из $S(v'')$, принадлежат этой части графа G, то в этом случае число вершин $v \in S(v'') \cap V'$, имеющих в $S(v'')$ нечетный индекс и соседних в графе G вершинам $v \in S(v'') \cap V''$, на единицу меньше числа последних. Таким образом доказана *теорема Холла.*

Теорема 4.7. Если максимальное паросочетание Π в конечном двудольном графе G не инцидентно всем вершинам из множества V'', то существует подмножество последнего, для которого множество соседних вершин (лежащих в множестве V') имеет на единицу меньшую мощность. □

В задаче о различных представителях множеств в каждой вершине $v'' \in V''$ соответствует подмножество $U(v'')$ элементов множества V', связанных в двудольном графе G ребрами с вершиной v''. Для любого подмножества $\tilde{V}'' \subset V''$ множество соседних вершин — это объединение всех подмножеств $U(v'')$ для $v'' \in \tilde{V}''$. Если число элементов $v' \in \bigcup_{v'' \in \tilde{V}''} U(v'')$ меньше числа различных множеств $U(v'')$, входящих в это объединение, задачу о различных представителях, очевидно, нельзя решить. Теорема Холла показывает, что в противном случае, когда для любого подмножества $\tilde{V}'' \subset V''$ выполнено условие $|V| \leqslant |\bigcup_{v'' \in \tilde{V}''} U(v'')|$, задача о максимальных представителях решается.

4.4. ОРИЕНТИРОВАННЫЕ ГРАФЫ

Пути и циклы в ориентированном графе. Пусть V — множество вершин ориентированного графа G, E — множество его ребер. Каждое ребро $e \in E$ имеет начало $v' \in V$ и конец $v'' \in V$; говорят также, что ребро e выходит из вершины v' и входит (или заходит) в вершину v''. Таким образом, заданы два отображения φ_1 и φ_2, где $\varphi_1(e)$ — начало ребра e, $\varphi_2(e)$ — конец ребра e.

Можно дать несколько определений пути в ориентированном графе G. *Путь из вершин и ребер* — это последовательность $L(v_0, e_1, v_1, ..., e_n, v_n)$, где $v_{i-1} = \varphi_1(e_i)$, $v_i = \varphi_2(e_i)$. Вершина v_0 называется началом пути L, вершина v_n —

концом L, число n ребер — его длиной. Путь, состоящий из одной вершины, имеет нулевую длину.

Каждому пути L ненулевой длины взаимно однозначно соответствует последовательность $\tilde{L}(e_1, ..., e_n)$ ребер пути L. Она называется *путем из ребер*. Это понятие пути — аналог понятия маршрута в неориентированном графе. Наконец, для графов, не содержащих кратных ребер (т. е. ребер с одинаковым началом и одинаковым концом), каждому пути L из вершин и ребер однозначно соответствует последовательность $\tilde{L}(v_0, v_1, ..., v_n)$ вершин пути L. Она называется *путем из вершин*. Ввиду взаимной однозначности объектов, порождаемых этими тремя определениями пути, всякий раз можно пользоваться тем из них, которое окажется удобнее.

Путь $Z(v_0, v_1, v_1, ..., a_n, v_n)$ называется *ориентированным циклом* (или просто циклом, когда ясно, что рассматриваются только ориентированные циклы), если он состоит более чем из одного элемента и его начало совпадает с его концом. Начало цикла обычно не фиксируется; иначе говоря, все пути вида $v_i, e_{i+1}, v_{i+1}, ..., v_n = v_0, e_1, ..., e_i, v_i$, получающиеся друг из друга циклическим сдвигом, — это один и тот же цикл.

Как и для неориентированного графа, цикл называется *простым*, если каждая принадлежащая ему вершина инцидентна ровно двум его ребрам. Если граф содержит циклы, то он содержит и простые циклы: отрезок цикла между повторениями одной и той же вершины также является циклом, поэтому любой цикл можно «укоротить» до простого. Граф, не содержащий циклов, называется *ациклическим*.

Пусть $L'(v'_0, v'_1, ..., v'_k)$ и $L''(v''_0, v''_1, ..., v''_l)$ — пути, причем конец L' совпадает с началом L'': $v'_k = v''_0$. Тогда последовательность $L(v'_0, ..., v'_{k-1}, v''_0, ..., v''_l)$ — также путь, называемый *композицией путей* L' и L''. Он обозначается $L = L_1 * L_2$ или $L = L_1 L_2$. Длина композиции равна сумме длин ее компонент.

Начальные и конечные вершины графа. Ранги вершин. Вершина графа называется начальной, если в нее не входит ни одно ребро, и конечной — если из нее не выходит ни одно ребро. Во всяком конечном ациклическом графе G есть хотя бы одна начальная и хотя бы одна конечная вершина. Действительно, все пути G конечны и имеют длину, не превосходящую числа его вершин, так как в путях ациклического графа вершины не могут повторяться. Поэтому

существует максимальный путь (быть может, не единственный), который нельзя удлинить ни в начале, ни в конце. Его начало будет начальной вершиной G, а конец — конечной вершиной. *Максимальным рангом* $R(v)$ вершины v ориентированного графа G называется максимальная из длин путей этого графа с концом в v. $R(v)=0$, если и только если v — начальная вершина G. Если через v проходит цикл, то v может быть концом сколь угодно длинного пути (повторяющегося цикла); в этом случае $R(v)=+\infty$. Как видно из предыдущего, в ациклическом графе все ранги конечны.

Покажем, что в ациклическом графе для любой неначальной вершины v $R(v)=\max\limits_{(v',v)\in G} R(v')+1$.

Пусть v — неначальная вершина, $L(v_0, v_1, ..., v_{k-1}, v)$ — путь максимальной длины с концом в ней. Тогда L — это композиция пути $L_1(v_0, ..., v_{k-1})$ и ребра (v_{k-1}, v). Так как длина L' не превышает максимального ранга v_{k-1}, то $R(v)\leqslant \max\limits_{(v',v)\in G} R(v')+1$.

Пусть теперь (v', v) — ребро и $L'(v_0, ..., v')$ — путь максимальной длины с концом v'. Композиция L' и ребра (v', v) — это путь L с концом v. Его длина не превосходит $R(v)$ и на единицу больше длины L', которая равна $R(v')$. Поэтому $R(v)\geqslant R(v')+1$ и, следовательно (так как (v', v) — произвольное ребро с концом в v), $R(v)\geqslant \max\limits_{(v',v)\in G} R(v')+1$.

Из этих двух неравенств следует требуемое равенство.

Минимальным рангом $r(v)$ вершины v ориентированного графа G называется минимум длин путей $L(v_0, ..., v)$ с началом в какой-либо начальной вершине v_0 графа G и с концом в рассматриваемой вершине v. Если ни одного такого пути не существует, $r(v)=+\infty$. В конечном ориентированном ациклическом графе G минимальные ранги всех неначальных вершин удовлетворяют условию $r(v)=\min\limits_{(v',v)\in G} r(v')+1$. Доказательство аналогично предыдущему.

Отношение достижимости. Базисный граф. Вершина v'' ориентированного графа G называется *достижимой* из вершины $v'\in G$, если существует путь $L=(v', ..., v'')$ с началом v' и концом v''. Легко видеть, что отношение достижимости рефлексивно и транзитивно. С его помощью определяется разбиение множества вершин G на классы эквивалентности; вершины $v', v''\in G$ принадлежат одному классу, ес-

ли v'' достижима из v', а v' — из v''. Пусть $L_1(v', ..., v'')$ и $L_2(v'', ..., v')$ — соответствующие пути, связывающие эти вершины. Тогда композиция этих путей $L_1 * L_2$ — это цикл, проходящий через вершины v' и v''. Таким образом, любые вершины одного и того же класса эквивалентности принадлежат некоторому циклу, однако простого цикла, проходящего через эти вершины, может и не быть.

На множестве V_i классов эквивалентности вершин графа отношение достижимости индуцирует отношение частичной упорядоченности: $V_1' \leqslant V_2'$ в том и только том случае, когда некоторая вершина $v_2 \in v_2'$ достижима из некоторой вершины $v_1 \in V_1'$. Если граф — ациклический, то каждый класс эквивалентности состоит из одной вершины и само отношение достижимости является отношением частичной упорядоченности.

Ориентированный граф G' называется транзитивным замыканием G, если отношение, соответствующее G', является транзитивным замыканием отношения, соответствующего G. Граф G' получается из G добавлением ребра (v', v'') (если оно отсутствует в G) всякий раз, когда в G есть путь $(v', ..., v'')$ и $v' \neq v''$. Если $G' = G$ (т. е. все такие ребра есть уже в G), то граф G изображает транзитивное отношение и сам называется транзитивным. Граф G' определяет то же отношение транзитивности, что и исходный граф.

Минимальный граф G_B, индуцирующий на множестве вершин V то же отношение достижимости, что и данный ориентированный граф G, т. е. граф с неуменьшаемым далее множеством ребер, называется *базисным графом* для графа G. Если G — конечный граф, то базисный граф для него можно построить причем для ациклического графа единственным образом. В каждом классе эквивалентности вершин, порожденном отношением достижимости, вершины нумеруются произвольным образом и определяются ребра (v_i, v_{i+1}) $(i = 1, ..., k,$ где k — число элементов класса) и (v_k, v_1). Таким образом, каждая вершина рассматриваемого класса достижима из остальных вершин этого класса, чего с меньшим числом ребер добиться нельзя.

Пусть V_1' и V_2' — различные классы эквивалентности вершин графа G и в последнем есть ребро (v_{i1}, v_{j2}), где $v_{i1} \in V_1'$, $v_{j2} \in V_2'$. Это ребро можно заменить ребром (v_{11}, v_{12}), после чего параллельные ребра склеить. Действительно, в любом пути $L(...v_{i1}, v_{j2}...)$ его можно заменить участ-

ком $(v_{i1}, v_{i+1,1}, ..., v_{k1}, v_{11}, v_{12}, v_{22}, ..., v_{j2})$. Теперь в ациклическом подграфе G, порожденном множеством первых вершин каждого класса эквивалентности, выбрасываются все ребра, замыкающие какой-либо путь. При этом для любой пары вершин (v', v''), удовлетворяющей отношению достижимости, в определенном выше графе $\widetilde{G}$ сохраняется путь $L(v', ..., v'')$ максимальной длины, который в конечном ациклическом графе $\widetilde{G}$ существовал. Действительно, если бы какое-либо ребро (v_i, v_{i+1}) пути L было выкинуто, в графе $\widetilde{G}$ имелся бы более длинный путь $L'(v', ..., v_i, ..., v_{i+1}, ..., v'')$, в котором это ребро было бы заменено стягиваемым им путем.

В графе $G_B = \widetilde{G} \cup \bigcup Z_j$ (j — класс эквивалентности для отношения достижимости) нельзя выкинуть ни одного ребра; при выбрасывании ребра $(v_{ji}, v_{j,i+1})$ или (v_{jk}, v_{j1}) из цикла Z_j или $(v_{ji}, v_{j''1}) \in G$ конец его перестает быть достижимым из начала.

Цепи, неориентированные циклы. Любому ориентированному графу G соответствует неориентированный граф G', в котором ребро (v', v'') имеется тогда и только тогда, когда в исходном графе G есть ребро (v', v'') или (v'', v'). Цепи и циклы графа G' называются также *цепями и неориентированными циклами* графа G. Среди ориентированных графов можно выделить подмножество графов, для которых их неориентированные образы не имеют циклов, т. е. являются деревьями или лесами. Таковы определенные в § 4.3 ориентированные деревья, но не только они. На рис. 4.15 изображен ориентированный граф без неориентированных циклов, не являющийся ориентированным деревом, хотя его неориентированный образ — дерево.

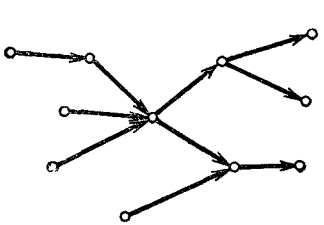

Рис. 4.15

Неориентированный цикл $\zeta(v_0, v_1, ..., v_k, v_0)$ ориентированного графа G является либо ориентированным циклом G, либо ориентированным циклом графа $\widetilde{G}$ с обратными направлениями ребер, либо, наконец, в нем существует такая вершина v_i, что $(v_{i-1}, v_i) \in G$, $(v_{i+1}, v_i) \in G$. Здесь индексы рассматриваются по модулю $k+1$. Таким образом, когда $i=k$, то $i+1=0$, а когда $i=0$, то $i-1=k$.

Назовем вершину $v_i \in \zeta$ положительной, если $(v_i, v_{i+1}) \in G$, и отрицательной, когда $(v_{i+1}, v_i) \in G$. Если все вершины неориентированного цикла ζ положительны, то это — ориентированный цикл, если все они отрицательны, то это — ориентированный цикл в графе $\widetilde{G}$ с обратными направлениями ребер. В остальных случаях существует такая отрицательная в ζ вершина v_i, что предыдущая вершина положительна. Но тогда наши условия для вершины v_i выполняются.

Правильная нумерация. Пусть вершины конечного ориентированного графа занумерованы от 1 до n. Нумерация называется *правильной на ребре* $(v_i, v_j) \in G$, если $i<j$, и *правильной на графе* G, если она правильна на всех его ребрах. Правильная нумерация вершин графа G может существовать лишь в том случае, когда G — ациклический граф. Правильную нумерацию можно рассматривать как расширение отношения частичной упорядоченности, заданной на множестве V вершин конечного ориентированного ациклического графа, до отношения полной упорядоченности. Строится это расширение следующим образом.

Пусть G — конечный ориентированный ациклический граф, $v' < v''$ — соответствующее отношение строгой частичной упорядоченности. Если вершины v', v'' не удовлетворяют ни отношению $v' < v''$, ни обратному отношению $v'' < v'$, то к множеству истинных формул можно добавить одно из этих соотношений, например $v' < v''$, и замкнуть это расширение отношения $<' : v_1 <' v_2 \equiv v_1 < v_2 \vee (v_1 < v' \& v'' < v_2)$.

Когда выполняются отношения $v_1 <' v_2$ и $v_2 <' v_3$, возможны три случая:

1) $v_1 < v_2 \& v_2 < v_3$, тогда $v_1 < v_3$, т. е. $v_1 <' v_3$;

2) $v_1 < v_2 \& v_2 < v' \& v'' < v_3$, тогда $v_1 < v' \& v'' < v_3$, т. е. $v_1 <' v_3$;

3) $v_1 < v' \& v'' < v_2 \& v_2 < v_3$, тогда $v_1 < v' \& v'' < v_3$, т. е. $v_1 <' v_3$.

Четвертый случай — $v_1 < v' \& v'' < v_2 \& v_2 < v' \& v'' < v_3$ невозможен. По транзитивности старого отношения $<$ в этом случае $v'' < v'$, что не выполняется по условию. Таким образом, новое отношение транзитивно.

Оно антирефлексивно, так как выполнение отношений $v_1 <' v_2$ и $v_2 <' v_1$ означает, что:

1) $v_1 < v_2 \& v_2 < v_1$ — сочетание условий, невозможное из-за антирефлексивности старого отношения $<$;

2) $v_1 < v_2$, $v_2 < v'$, $v'' < v_1$, тогда по транзитивности старого отношения $v'' < v'$, а это по условию не так;

3) $v_1 < v'$, $v'' < v_2$, $v_2 < v_1$ — так же, как и для второго случая, отсюда следует $v'' < v'$;

4) $v_1 < v'$, $v'' < v_2$, $v_2 < v'$ и $v'' < v_1$, но и в этом случае по транзитивности отношения $< v'' < v'$.

Продолжая, если нужно, этот процесс расширения отношения строгой частичной упорядоченности, вследствие конечности числа вершин графа G через конечное число шагов получим отношение строгой полной упорядоченности $\lesssim$. Для любых вершин v_1, v_2, различных между собой, будет выполняться одно и только одно из условий $v_1 < v_2$ или $v_2 > v_1$. Для этого отношения есть единственная правильная нумерация. Номер 1 получает вершина v', для которой выполняются все условия $v' < v (v \neq v')$. Существование этой вершины легко доказать — это начальная вершина ациклического графа $\widetilde{G}$, соответствующего нашему порядку. Номер 2 имеет вершина, для которой выполняются условия $v'' < v [v \in G \setminus (v' \cup v'')]$, начальная в подграфе $\widetilde{G} \setminus v'$ и т. д. Такая нумерация правильна и для исходного отношения $<$.

Длины путей, протяженности и расстояния между вершинами графа уже были определены ранее. Часто рассматриваются более общие определения этих понятий. Пусть каждому ребру $(v', v'') \in G$ поставлено в соответствие действительное число $l(v', v'')$ — его длина. Тогда длина любого пути $L(v_{i_0}, v_{i_1}, ..., v_{i_p})$ определяется как сумма длин входящих в него ребер:

$$l(L) = \sum_{k=1}^{p} l\left(v_{i_{k-1}}, v_{i_k}\right).$$

Расстоянием $d(v_i, v_j)$ между двумя вершинами v_i и v_j графа называется нижняя грань длин путей $L(v_i, ..., v_j)$ с началом в первой и концом во второй из этих вершин; протяженностью $g(v_i, v_j)$ — верхняя грань этих длин.

Считается, что длина пути $L_0(v_i)$, состоящего из одной вершины, равна 0; если вершины v_i и v_j в графе G не связаны, то $d(v_i, v_j) = +\infty$, $g(v_i, v_j) = -\infty$. Приведенные ранее определения длины пути, расстояния и протяженности являются частными случаями этого определения, когда длины всех ребер равны единице. Так как при изменении знаков длин $l(v', v'')$ всех ребер (v', v'') на противоположные расстояния и протяженности меняются местами с заменой знаков на противоположные, достаточно исследовать свойства расстояний.

Если существует путь $L(v_i, ..., v_h, ..., v_j)$, в котором некоторая вершина v_h принадлежит циклу $Z(v_h, ..., v_h)$ отрицательной длины $l'<0$, то $d(v_i, v_j)=-\infty$. Действительно, в этом случае путь $L_q(v_i, ..., v_h, ..., v_h, ..., v_j)$, состоящий из отрезка $(v_i, ..., v_h)$ пути L, q раз повторенного цикла Z и отрезка $(v_h, ..., v_j)$ пути L имеет длину $l(L)+ql'$, т. е. может оказаться меньше любого отрицательного числа. Если же в любом связывающем рассматриваемые вершины пути нет вершин, принадлежащих циклам отрицательной длины, то, выбросив из таких путей циклы, получим простые пути не большей длины. В конечном графе G количество таких путей конечно и нижняя грань их длин достигается.

Теорема 4.8. Расстояние $d(v_i, v_j)$ между различными вершинами графа G удовлетворяет условию $d(v_i, v_j)=\inf\limits_{(v_h, v_j)\in G}(d(v_i, v_h)+l(v_h, v_j))$.

При доказательстве будем считать, что нижние грани достигаются (для конечных графов это всегда верно). Пусть $(v_h, v_j)\in G$ и $L(v_i, ..., v_h)$ — путь минимальной длины, соединяющий v_i с v_h. Добавив к нему ребро (v_h, v_j), получим путь $L'(v_i, ..., v_h, v_j)$, соединяющий v_i с v_j. Следовательно, $d(v_i, v_j)\leqslant l(L')=l(L)+l(v_h, v_j)=d(v_i, v_h)+l(v_h, v_j)$, откуда $d(v_i, v_j)\leqslant \inf\limits_{(v_h, v_j)\in G}(d(v_i, v_h)+l(v_h, v_j))$. Если же $L(v_i, ..., v_g, v_i)$ — путь минимальной длины, соединяющий v_i с v_g (он не состоит из одной вершины, так как $v_i\neq v_j$), то $L'(v_i, ..., v_g)$ — путь, соединяющий v_i и v_g, и $d(v_i, v_j)=l(L)=l(\widetilde{L'})+l(v_h, v_j)\geqslant d(v_j, v_g)+l(v_g, v_j)\geqslant \inf\limits_{(v_h, v_j)\in G}(d(v_i, v_h)+l(v_h, v_j))$. □

Если вершина v_i принадлежит какому-либо циклу отрицательной длины, то $d(v_i, v_i)=-\infty$. В противном случае $L_0(v_i)$—кратчайший путь с началом и концом v_i и $d(v_i, v_i)=0$. □

4.5. ГРАФЫ С ПОМЕЧЕННЫМИ ВЕРШИНАМИ И РЕБРАМИ

Разбиение вершин графа на классы. Нередко приходится иметь дело с различиями между вершинами графа. Тогда последние разбивают на классы. Каждый класс состоит из вершин, имеющих некоторое общее свойство. Свойства, определяющие разбиение вершин графов на классы, могут быть «графовыми»: иметь данную степень, данное расстояние от фиксированной вершины — корня, данный ранг (в ориентированном графе), в двудольном графе вершины каждой доли составляют класс и т. д. В других случаях разбиение определяется свойствами объектов, описываемых при помощи графов. Например, структурная формула химического соединения — это граф, в котором вершины соответствуют атомам молекулы соединения, ребра — валент-

ным связям, а классы состоят из вершин, соответствующих атомам одного и того же элемента (рис. 4.16).

Помеченные вершины. Пусть дано разбиение вершин графа на классы. Каждой вершине можно соотнести *метку*, указывающую, какому классу она принадлежит. Метки являются элементами заданного множества. Иногда они явно указывают на свойства, определяющие классы: степени, ранги вершин и их расстояния от корня можно метить соответствующими числами, вершины структурной формулы — обозначениями химических элементов и т. п. Однако часто отвлекаются от конкретного характера различий между вершинами, и тогда метки указывают только на факты сходства вершин и их различия. Соотнесение таких меток вершинам интерпретируют как *раскраску* последних в разные цвета. Аналогичным образом говорят о раскраске ребер графа и вообще о раскраске элементов произвольного множества.

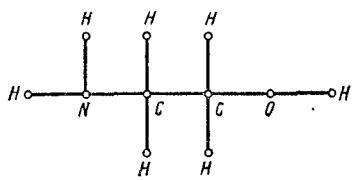

Рис. 4.16

Задачи о раскраске вершин графов. Часто рассматривают не произвольные раскраски вершин графов, а только удовлетворяющие некоторым заданным условиям. Так, во многих исследованиях запрещается красить соседние вершины в один цвет. В этих работах количество цветов задается или требуется найти его минимум.

Начнем с довольно простой задачи. Сколько цветов понадобится для раскраски вершин графа, если степени всех его вершин не превосходят некоторого числа k? Можно показать, что $k+1$ цветов достаточно. Будем окрашивать вершины по очереди в произвольном порядке. Пусть уже окрашенные вершины не имеют соседей, окрашенных в те же цвета, и мы вибираем цвет очередной вершины. Ее соседи — это вершины, имеющие с ней по общему инцидентному ребру. Значит, их не больше k. Некоторые из них могут быть уже окрашены, в крайнем случае все. Все равно на их окраску ушло не более k цветов, и имеется свободный цвет, который можно выбрать для данной вершины. Так будут по очереди окрашены все вершины графа.

Пусть G — полный граф, т. е. каждая пара его вершин соединена ребром. Тогда все вершины должны быть окрашены в разные цвета. В полном графе с $k+1$ вершиной сте-

пени всех вершин равны k. Значит, существуют графы со степенями вершин, не большими k, для раскраски которых $k+1$ цвет необходим. Однако далеко не все графы такие. Например, вершины двудольного графа можно раскрасить в два цвета, каковы бы ни были степени его вершин: в нем соседние вершины обязательно принадлежат разным долям, и достаточно окрасить вершины каждой доли в свой цвет.

Проблема четырех красок. Как уже говорилось, раскрашивать можно не только вершины графа. Рассмотрим так называемые карты, т. е. разбиения плоскости на связные области (рис. 4.17). Достаточно ли четырех цветов для раскраски любой карты? Эта проблема имеет большую историю. По некоторым сведениям, еще в 1840 г. о ней знал известный немецкий математик Мебиус. Первое из ошибочных «доказательств» было дано Кемпе в 1879 г., но ошибка была обнаружена не сразу. Ее нашел Хейвуд в 1890 г. и тогда же доказал, что области любой карты можно раскрасить требуемым образом в пять цветов. С тех пор проблему четырех красок, как ее назвали, долго не удавалось решить.

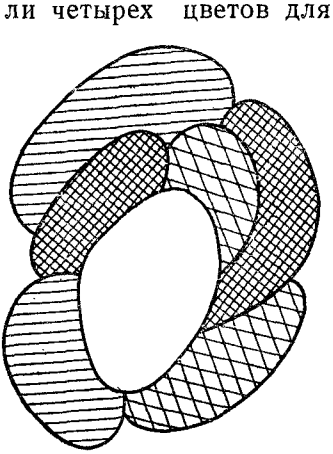

Рис. 4.17

Недавно американские математики Аппель и Хакен доказали гипотезу о четырех красках, существенно использовав машинные вычисления. Это первый случай, когда столь знаменитая математическая задача была решена при помощи машины. Доказательство задачи требует грозмоздкого описания машинных расчетов, поэтому здесь не приводится. Однако его идеи поучительны, и мы их рассмотрим.

Сначала несколько замечаний. Некоторые области могут граничить между собой только в изолированной точке, в которой могут сходиться сколько угодно областей (рис. 4.18), но мы не будем считать их соседними. Такие точки можно «раздуть», т. е. окружить их областями, не отгораживающими друг от друга областей с протяженными границами. В каждой точке новой карты будут сходиться не более чем по три области. Если ее можно раскрасить в четыре цвета, то, сохранив цвета областей старой карты, получим

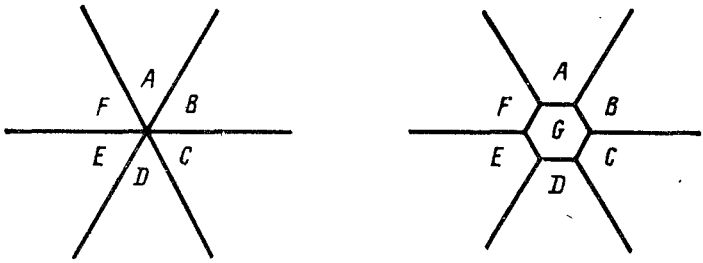

Рис. 4.18

допустимую раскраску последней. В дальнейшем будем рассматривать карты, в точках которых сходятся не более чем по три области.

Двойственный граф. Границы областей карты — это плоский граф. Его вершины — точки, где сходятся по три области, ребра — соединяющие их линии — общие границы двух областей. В каждой области выберем точку — ее центр, и центры соседних областей соединим линиями. Получится плоский граф, который называется двойственным определенному выше (оба графа изображены на рис. 4.19). Области двойственного графа окружают вершины графа границ карты. Так как нас интересуют карты, в вершинах графа границ которых сходятся по три области, в двойственном графе каждая область — треугольник (криволинейный, причем внешняя область тоже ограничена треугольником). Таким образом, двойственный граф определяет так называемую триангуляцию плоскости. Заметим еще, что он всегда связен. Раскрашивать области карты все равно, что вершины двойственного графа. Поэтому проблему четырех красок формулируют еще и так: можно ли раскрасить вершины любого связного плоского графа в четыре цвета так, чтобы соседние вершины были окрашены в разные цвета?

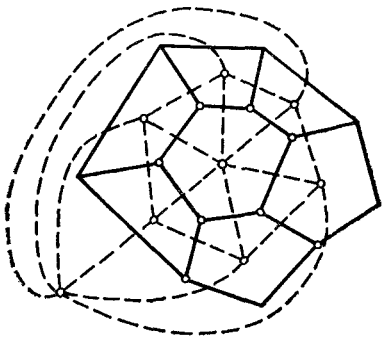

Рис. 4.19

Редуцируемые конфигурации. Конфигурация — это связный подграф плоского графа, порожденный некоторым подмножеством его вершин (обычно небольшим). Остальные вершины и ребра графа составляют внешнюю часть конфигурации. Конфигурацию можно «стянуть», т. е. исключить из нее некоторые вершины, не имеющие инцидентных ребер из внешней части, отождествить некоторые другие и соединить оставшиеся вершины ребрами, может быть, поиному. Внешняя часть присоединяется к соответствующим вершинам новой конфигурации, причем требуется, чтобы полученный граф оказался плоским. Если исходный граф был триангуляцией, то новый граф тоже должен быть триангуляцией. Конфигурация называется редуцируемой, если из правильной раскраски вершин нового графа можно получить правильную раскраску исходного графа. Это определение зависит от мощности средств редукции. Приведем примеры редуцируемых конфигураций.

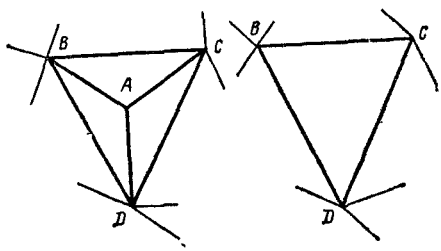

Рис. 4.20

Вершина степени 3 (рис. 4.20). Исключив внутреннюю вершину, снова получим триангуляцию (тот же рисунок). Если редуцированный граф можно раскрасить в четыре цвета, то оставшиеся вершины конфигурации окрашены в три из них. Поэтому остается свободный цвет для окраски исключительной вершины.

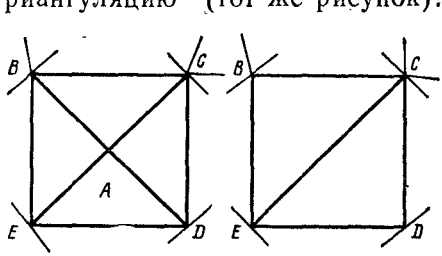

Рис. 4.21

Вершина степени 4 (рис. 4.21). Исключив внутреннюю вершину и соединив ребром какую-либо пару противоположных вершин, получим триангуляцию (тот же рисунок). Пусть редуцированный граф раскрашен в четыре цвета. Если оставшиеся вершины конфигурации окрашены только в три цвета, то есть свободный цвет для исключенной верши-

ны. Пусть они окрашены в четыре цвета. Перекрасим вершину C в цвет 1, соседние с ней вершины цвета 1 — в цвет 3, соседние с ними вершины цвета 3 — в цвет 1 и т. д. Этот процесс мы называем цепной перекраской цветов 1 и 3, начинающейся с вершины C. Когда он окончится, редуцированный граф будет правильно раскрашен. Если при этом цвет вершины A не изменится, то цвет 3 окажется свободным для окраски вершины E, исключенной в редуцированном графе. В противном случае в редуцированном графе есть цепь из вершин цветов 1 и 3, соединяющая вершины A и C. Однако тогда в нем нет цепи вершин цветов 2 и 4, соединяющей вершины B и D, кроме ребра BD, но последнее не принадлежит внешней части конфигурации и потому при перекрасках цветов не определяет соседства. Применим цепную перекраску цветов 2 и 4, начинающуюся с вершины B. Новая окраска будет иметь только один дефект: вершины B и D окрашены одинаково. Однако в исходном графе они не соседние. Вместе с тем при перекраске освободился цвет 4 для окраски вершины E.

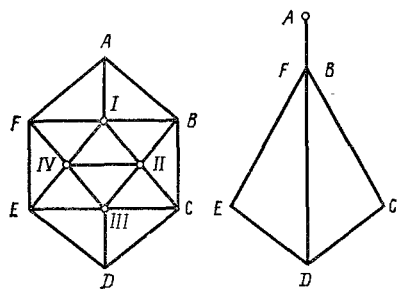

Рис. 4.22

Четыре вершины степени 5, расположенные, как показано на рис. 4.22. Исключим эти вершины, отождествим вершины B и F, а вершины B и D соединим ребром. С точностью до перенумерации цветов и симметрии относительно конфигурации возможны раскраски оставшихся вершин последней, приведенные в строках табл. 4.3 и соответствующие шести графам рис. 4.23. Все они, кроме первой, продолжаются внутрь конфигурации (рис. 4.23). При первой раскраске либо вершины B и D не связаны цепью вершин цветов 2 и 3, либо вершины A и C не связаны цепью вершин цветов 1 и 4. Применив соответствующие цепные перекраски, перейдем к последним двум раскраскам рис. 4.23, которые тоже можно продолжить внутрь конфигурации.

Минимальный нераскрашиваемый граф. Если гипотеза о четырех красках неверна, то существует граф, вершины которого нельзя раскрасить в четыре цвета, имеющий минимальное число вершин среди всех таких графов. В этом гра-

Таблица 4.3

A	B	C	D	E	F	I	II	III	IV
1	2	1	3	1	2	—	—	—	—
1	2	1	3	4	2	3	4	2	1
1	2	3	1	3	2	3	1	2	4
1	2	3	1	4	2	3	4	2	1
1	2	3	4	3	2	3	1	2	4
1	3	1	3	1	2	4	2	4	3

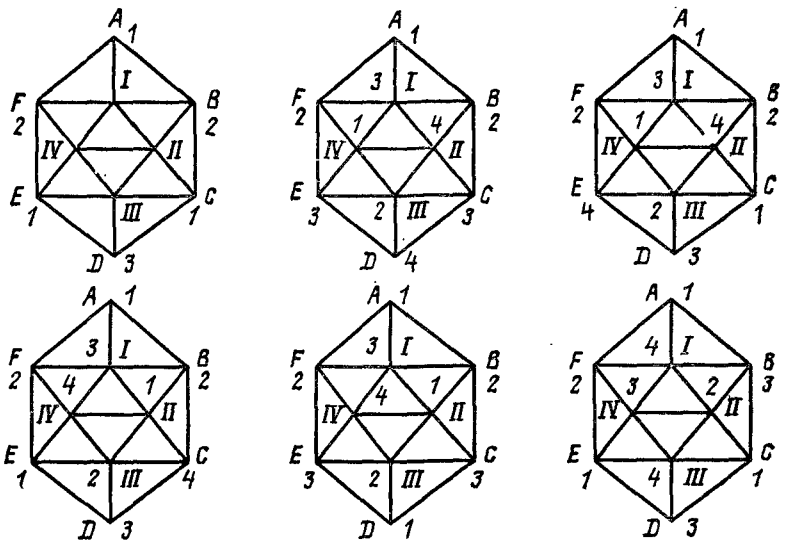

Рис. 4.23

фе нет никакой редуцируемой конфигурации, иначе редуцированный граф тоже нельзя было бы раскрасить в четыре цвета, а ведь у него меньше вершин. Таким образом, если в любом связном плоском графе с достаточно большим количеством вершин есть редуцируемая конфигурация, то гипотеза о четырех красках справедлива (доказано, что все связные плоские графы с небольшим числом вершин раскрасить можно). Плоские графы вообще заслуживают изучения.

Порождение плоских графов. Будем рассматривать пло-

ские графы вместе с их отображениями на плоскость. Таким образом, в дальнейшем вершины плоского графа — это точки на плоскости, а ребра — соединяющие их линии. Концами ребер являются инцидентные им вершины, другие вершины им не принадлежат, и они не имеют других общих точек, кроме общих инцидентных вершин. Можно считать, что ребра являются ломаными линиями, состоящими из конечного числа отрезков прямых. Остальные точки плоскости разбиты графом на некоторое количество связных областей.

Минимальный связный граф состоит из единственной вершины и не имеет ребер. Любые точки плоскости, не совпадающие с вершиной этого графа, можно соединить ломаной линией, не проходящей через нее. Значит, у минимального графа на плоскости одна область (речь идет о связных областях). Любой другой связный плоский граф может быть порожден из графа с меньшим количеством вершин или ребер одной из следующих операций.

1. Добавление вершины степени 1. Вне связного плоского графа G, т. е. в некоторой области r, ставится вершина p и соединяется ребром q с некоторой вершиной графа G, расположенной на границе области r (рис. 4.24). Ясно, что в других областях ничего не меняется, но то, что область r остается связной, требует доказательства (хотя и очевидно). Мы его не приводим, так как оно требует довольно глубокого знания топологии плоскости. У нового связного плоского графа G' количества вершин и ребер на 1 больше, чем у графа G, а количество областей — такое же.

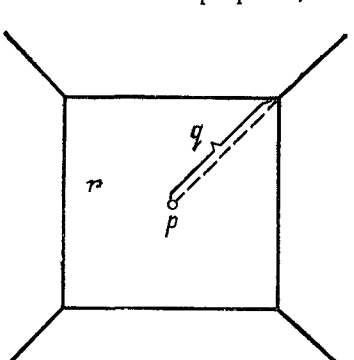

Рис. 4.24

2. Добавление вершины степени 2. Внутри некоторого ребра q связного плоского графа G ставится вершина p. Таким образом, это ребро разбивается на два — q' и q'' (рис. 4.25), т. е. в новом связном плоском графе G' на одну вершину и на одно ребро больше. Вне графа ничего не меняется, значит, число областей остается тем же.

3. Разбиение области. Новое ребро q соединяет вершины p' и p'', расположенные на границе области r,

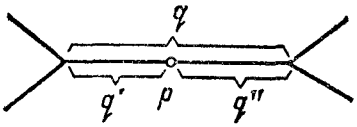

Рис. 4.25

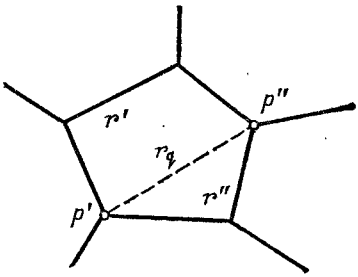

Рис. 4.26

и лежит в этой области (рис. 4.26). Из известной теоремы Жордана следует, что область разбивается на две — r' и r''. Значит, в графе G' на одно ребро и на одну область больше, чем в графе G, а количество вершин то же.

Для доказательства возможности породить при помощи указанных операций любой связный плоский граф используем обратные операции: удаление вершин степени 1 и 2, а также разрыв цикла. Пусть дан не минимальный связный плоский граф G' с v' вершинами и e' ребрами. Если в нем есть вершина p' степени 1 или 2, то можно произвести операцию ее удаления, причем получится связный плоский граф G. Обратно граф G' может быть получен из графа G операцией добавления вершины степени 1 или 2. Пусть теперь степени всех вершин графа G' не меньше, чем 3. Сложим все эти степени $n_{p'}$ $(p' \in G')$. Сумма равна удвоенному числу ребер $2e'$, так как каждое ребро увеличивает на 1 степени двух вершин. С другой стороны, она не меньше утроенного числа вершин, так как степень каждой вершины не меньше, чем 3. Таким образом, $2e' = \sum\limits_{p' \in G} \geqslant 3v'$.

Цикломатическое число графа G' равно $e' - v' + 1 \geqslant \geqslant 3/2v' - v' + 1 > 0$, значит, в нем есть некоторый цикл Z. Если удалить из G' какое-нибудь ребро q' цикла Z, то граф останется связным. Обратно граф G' может быть получен из связного плоского графа G операцией соединения вершин p' и p'' ребром q'. Так как при этом возникает новый цикл Z, область r, по которой проходит ребро q', разбивается на две.

Теорема Эйлера (о многогранниках). Пусть l^0 — выпуклый многогранник. Эйлер доказал, что количества его вершин v, ребер e и граней g связаны соотношением $v - e + g = 2$.

Выпуклый многогранник можно спроектировать из внут-

ренней точки на поверхность сферы, а с последней — на плоскость. При этом его вершины перейдут в вершины некоторой карты, ребра — в ее ребра, грани — в области (одна область будет внешней). Оказывается, соотношение Эйлера справедливо для количеств вершин, ребер и областей любого связного плоского графа.

Теорему Эйлера легко доказать по индукции. В минимальном связном плоском графе G_0 одна вершина, одна область и нет ребер. Соотношение Эйлера для него выполняется. Пусть оно выполняется для всех связных плоских графов с e ребрами, и G' — произвольный связный плоский граф с v' вершинами, $e'=e+1$ ребрами и r' областями. Уже было показано, что он может быть получен из некоторого связного плоского графа G с e ребрами при помощи одной из указанных выше операций. Пусть v — количество вершин графа G, r — количество его ребер. Если граф G' порожден из него при помощи первой или второй операции, то $v'=v+1$, $g'=g$; если при помощи третьей, то $v'=v$, $g'+1$. В обоих случаях $v'-e'+g'=v-e+g=2$. □

Неизбегаемые наборы конфигураций. Можно показать, что в любой триангуляции плоскости G есть вершина, степень которой не больше, чем 5. Пусть v — число вершин триангуляции, e — число ребер, g — число областей. Последние имеют $3g$ граничных ребер, но каждое ребро является границей двух областей. Следовательно, $2e=3g$, т. е. $g=2/3e$. С другой стороны, как уже отмечалось, $2e=\sum\limits_{p\in G} n_p$, где n_p — степени вершин триангуляции. Кроме того, $v=\sum\limits_{p\in G} 1$. Подставим эти значения v, e и g в соотношение Эйлера

$$2 = v - e + g = v - e + 2/3g = v - 1/3e = \sum_{p\in G} 1 - 1/6 \sum_{p\in G} n_p = 1/6 \sum_{p\in G} (6 - n_p).$$

Отсюда $\sum\limits_{p\in G} (6-n_p) = 12$.

В левой части положительны только члены суммы с 2—5 степенями (можно показать, что вершины степени 1 в триангуляции нет), и, так как вся сумма положительна, такие члены существуют, что и требовалось доказать.

Таким образом, набор конфигураций, состоящий из вершин степеней 2—5 (рис. 4.27), обладает тем свойством, что

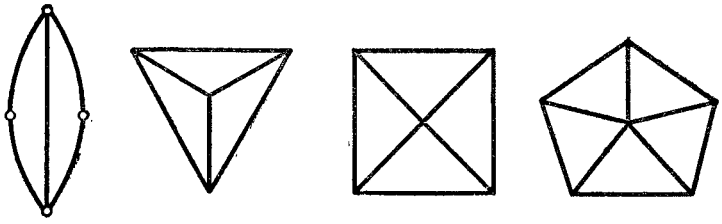

Рис. 4.27

в любой триангуляции плоскости есть хотя бы одна конфигурация из этого набора (на самом деле их больше). Такие наборы конфигураций называются неизбегаемыми. Приведенный выше набор — не единственный. Ниже будут приведены еще два примера. В оба входят вершины степеней 2, 3 и 4.

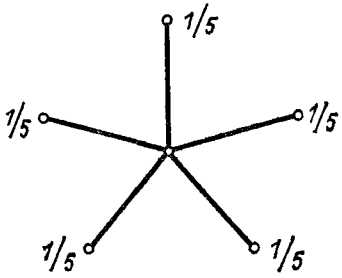

Рис. 4.28

Каждая вершина p триангуляции G дает в приведенную выше сумму, полученную из соотношения Эйлера, вклад, равный $(6-n_p)$, где n_p — ее степень. Эти вклады можно различным образом перераспределить между соседними вершинами. Например, оставим при каждой вершине степеней 6, 7... их вклады, а вклады вершин степени 5 перенесем в соседние вершины, распределив их поровну (рис. 4.28). Пусть ν_p — количество вершин степени 5, соседних с вершиной p триангуляции G. Новый вклад вершины p равен $\nu_p/5$, если ее степень равна 5, и $6+\nu_p/5-n_p$, если она больше. Легко видеть, что

$$12 = \sum_{p\in G}(6-n_p) = \sum_{p\in G, n_p=5}\nu_p/5 + \sum_{p\in G, n_p>5}(6+\nu_p/5-n_p) =$$

$$= \sum_{p\in G, n_p<6}\nu_p/5 + \sum_{p\in G, n_p>6}(6+\nu_p/5-n_p).$$

Значит, в последней сумме есть положительные члены. Все члены первой суммы положительны, но они присутствуют только в случаях, когда в триангуляции G есть расположенные рядом две вершины степени 5 или степеней 5 и 6 (рис. 4.29).

Для исследования слагаемых первой суммы заметим, что $\nu_p \leqslant n_p$. Значит, $6+\nu_p/5-n_p \leqslant 6-4n_p/5$. При $n_p \geqslant 8$ $6-4n_p/5<0$. Остается исследовать случай $n_p=7$, $6+\nu_p/5-n_p>0$, т. е. $\nu_p/5>1$, откуда следует $\nu_p>5$. Однако, если среди соседей вершины степени 7 есть шесть вершин степени 5, то среди последних есть соседи между собой. Достаточно даже, чтобы были четыре соседние вершины степени 5 (вообще, когда у вершины степени k больше, чем $k/2$ соседних вершин имеют степень 5, среди них есть соседние между собой). Итак, в триангуляции G есть либо вершины степеней 2, 3 или 4, либо конфигурации, изображенные на рис. 4.29.

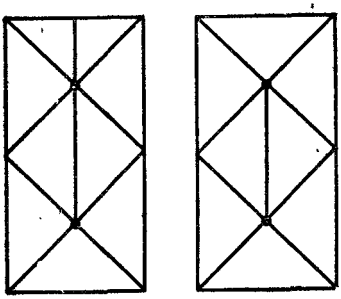

Рис. 4.29

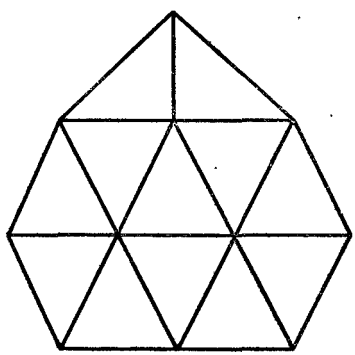

Рис. 4.30

Можно указать несколько иной набор конфигураций: вместо двух соседних вершин степеней 5 и 6 он содержит конфигурацию из трех взаимно соседних вершин степеней 5, 6 и 6 (рис. 4.30). Если в триангуляции нет вершин степеней 2, 3 и 4, двух соседних вершин степени 5 и указанной выше конфигурации, то у каждой вершины степени 5 не менее трех соседей имеют степени, большие, чем 6. Будем переносить положительные вклады вершин степени 5 только в соседние вершины со степенями 7, 8, ..., причем поровну. Значит, в каждую из этих вершин перенесем не больше, чем $\nu_p/3$, т. е. новые вклады вершин будут не больше, чем $6-n_p+\nu_p/3$. Среди них должны быть положительные, но условие $6-n_p+\nu_p/3>0$ при $n_p=7$ дает $\nu_p>3$, при $n_p=8$ $\nu_p>6$, а при $n_p \geqslant 9$ оно невозможно. Таким образом, в триангуляции G есть либо вершина степени 7, у которой четыре соседа имеют степень 5, либо

вершина степени 8, у которой есть семь таких соседей. Однако в обоих случаях две вершины степени 5 должны быть соседними.

Основная идея. Если бы существовал неизбегаемый набор, состоящий только из редуцируемых конфигураций, то минимальным нераскрашиваемым графом, о котором уже говорилось, мог бы быть только граф G_0 с единственной вершиной. Однако его можно раскрасить даже в один цвет. Значит, минимального нераскрашиваемого графа не существовало бы. Но тогда все связные плоские графы можно было бы раскрасить в четыре цвета. Таким образом, чтобы доказать гипотезу о четырех красках, достаточно найти неизбегаемый набор редуцируемых конфигураций. Приведенные выше примеры не подходят: вершины степеней 2, 3 и 4 редуцируемы, но редуцировать остальные конфигурации не удается. Однако нужно искать неизбегаемые наборы не для всех триангуляций, а только для тех, в которых степени всех вершин не меньше 5.

Аппель и Хакен строили такие наборы, состоящие из большого количества конфигураций, при помощи разнообразных перераспределений вкладов вершин в сумму $\sum_{p \in G}(6-n_p)$. Но как проверить, что все входящие в набор конфигурации редуцируемы? Методы редукции аналогичны тем, которые использовались при исследовании конфигурации, изображенной на рис. 4.22. Из-за многочисленности вариантов применения этих методов рассмотреть их все удалось только с помощью машины. Была создана программа, производившая многочисленные попытки редуцировать предъявленные ей конфигурации. Все же она прекращала эти попытки, если ей не удавалось произвести редукцию за заранее заданное время. Тогда Аппель и Хакен строили новый набор, который такую конфигурацию не содержал (хотя, возможно, исключаемая конфигурация была редуцируемой).

Так весьма быстродействующие машины проработали около 1500 ч, пока не был построен набор из 1932 конфигураций, и машина показала, что 1931 из них редуцируема. Оставшуюся конфигурацию исследовали вручную, применив более тонкие методы редукции. Она оказалась тоже редуцируемой. Таким образом, проблема четырех красок была решена.

Графы — язык дискретной математики. При формулировке задач дискретной математики или описании методов

их решений часто употребляется язык теории графов. Ряд примеров этого приведен ниже. Здесь же укажем лишь один достаточно общий способ постановки задач дискретной математики, при котором естественным образом возникает граф.

Пусть рассматривается множество V объектов v, каждый из которых может находиться в состояниях x_v из заданного для этого элемента v множества состояний X_v. Состояния различных объектов взаимозависимы так, что для каждого объекта v заданы множество Q_v непосредственно влияющих объектов и функция влияния $F_v(v_1, ..., v_s)$, определяющая состояние x_v объекта v по состояниям x_{v_i} объектов $v_i \in Q_v : x_v = F_v(v_1, ..., v_s)$, $\{v_1, ..., v_s\} = Q_v$ (в частности, для некоторых v множества Q_v пусты, тогда $x_v = F_v$, где F_v — заданное состояние).

В данном случае объекты $v \in V$ можно рассматривать как вершины структурного графа определенной ранее системы соотношений между состояниями этих объектов. Это ориентированный граф с ребрами (v', v''), $v' \in Q_v$. Иногда рассматриваются неоднозначные функции $F_v(v_1, ..., v_s)$ и ищутся системы допустимых состояний объектов, удовлетворяющие каким-либо дополнительным условиям, чаще всего некоторым условиям оптимальности.

ГЛАВА ПЯТАЯ

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 типичного 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 неэвклидовых 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. [36]). Все это потребовало точного изучения принципов математических рассуждений (до сих пор казавшихся интуитивно ясными) математическими же средствами. Возникла особая отрасль математики — основания математики, или метаматематики.

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). Существо 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 слишко 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 изобре-

тения» 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 техническую кибернетику, 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 графе, то дело даже не в том, что в рисунке

больше несущественных деталей и два человека один и тот же граф изобразят по-разному (в конце концов, разные 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 АЛГОЛе; 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 поразному.

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 ЭВМ. 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 ЭВМ третьего поколения 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 алгоритма (например, ЭВМ), 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.

Большинство читателей сочтет 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 $\sqrt[7]{3}, \sqrt[5]{2}, (1,2)^{\pi}$? 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 лю-

бое 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: 90811557001,15 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 основные 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 неэлементарен). Кроме того, становится ясным, что выбор механизма реализации (скажем, человека или ЭВМ) 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 закаичивается, 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 граф изображен на рис. 5.1.

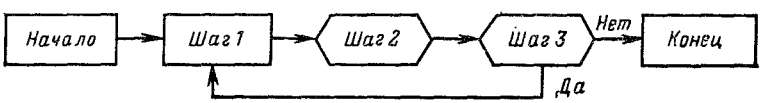

Рис. 5.1

Такой граф, 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 выходят два ребра (их называют логическими условиями, или предикатами). Кроме того, имеется единственный оператор конца, 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 «разблочивать» 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

Рис. 5.2

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 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) $x<5$, $5\leqslant x<20$, $x=20$, $x<20$.

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 поня-

тий 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 ЭВМ и, следовательно, к инженерной интуиции. Основной теоретической моделью этого типа (созданной в 30-х годах — раньше ЭВМ!) 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 и нормальные алгоритмы Маркова.

5.2. МАШИНЫ ТЬЮРИНГА

Основные определения. Машина Тьюринга состоит из: 1) управляющего устройства, которое может находиться в одном из состояний, образующих конечное множество $Q=\{q_1, \ldots, q_n\}$; 2) ленты, разбитой на ячейки, в каждой из которых может быть записан один из символов конеч-

ного алфавита $A=\{a_1, \ldots, a_m\}$; 3) устройства обращения к ленте, т. е. считывающей 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_j 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

$$q_i a_j \to q_i' a_j' d_k, \tag{5.1}$$

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 $a_j \to a'_j d_k$. 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 $abcq_ide$. Стандартной 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 $q_z\alpha$. 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 $K \underset{T}{\to} K'$; если из контекста ясно, о какой машине T идет речь, индекс T будем опускать. Если для K_1 и K_n существует последовательность конфигураций $K_1, K_2, \ldots, K_n$, такая, что $K_1 \underset{T}{\to} K_2 \underset{T}{\to} \ldots \underset{T}{\to} K_n$, обозначим это $K_1 \underset{T}{\Rightarrow} K_n$. Например, если в системе команд машины T имеются команды $q_2a_5 \to q_3a_4R$

и $q_3a_1L\to q_4a_2$, то $q_2a_5a_1a_2\to a_4q_3a_1a_2\to q_4a_4a_2a_2$ и, следовательно, $q_2a_5a_1a_2\underset{T}{\Rightarrow}q_4a_4a_2a_2$. Последовательность конфигураций $K_1\underset{T}{\to}K_2\underset{T}{\to}K_3\underset{T}{\to}\ldots$ 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 $q_11\to q_11R$, $q_1\lambda\to q_11R$ из любой начальной конфигурации будет работать бесконечно, заполняя единицами всю ленту вправо от начальной точки.

б. Для любой машины T, если $K_1\underset{T}{\Rightarrow}K_i\underset{T}{\Rightarrow}K_j$ и $K_i=K_j$, последовательность $K_1\Rightarrow K_i\Rightarrow K_j\Rightarrow\ldots$ является бесконечной: ее отрезок $K_i\Rightarrow K_j$ будет повторяться (зациклится).

Если $\alpha_1q_1\alpha_2\underset{T}{\Rightarrow}\beta_1q_2\beta_2$, 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 $(\alpha_1, \ldots, \alpha_k)$ назовем правильной, если она имеет вид $\alpha_1\lambda\alpha_2\lambda\ldots\alpha_{k-1}\lambda\alpha_k$ (при условии, что $\lambda\notin A_{\text{исх}}$) либо $\alpha_1*\alpha_2*\ldots*\alpha_{k-1}*\alpha_k$, где $*$ — специальный символ-разделитель, 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 совокупность слов (разделенных пробелами) 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 $f(V)=W$, $q_1V^* \underset{T}{\Rightarrow} q_zW^*$, 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_1V^*, 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 $A_{\text{исх}}$ будет 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 $q_ia_j \to q'_ta'_jE$ и $q''_ta''_j \to q''_ta''_jd_k$, то, заменив эти две команды командой $q_ia_j \to q''_ta''_j\,d_k$, получим машину T', эквивалентную T. Путем таких преобразований можно в машине T убрать все команды, содержащие E, для случая, когда q'_t — незаключительное состояние; при этом может сократиться число состояний (некоторые q'_t не войдут в правые части новых команд и станут недостижимыми из q_1). Если q'_t — заключительное состояние, то, введя новое заключительное состояние q_{n+1} и заменив команду $q_ia_j \to q'_ta'_jE$ на $m+1$ команду $q_ia_j \to q'_ta'_jR$, $q'_ta_1 \to q_{n+1}a_1L$, ..., $q'_ta_m \to q_{n+1}a_mL$ (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 $1\ldots1=1^x$, состоящим из x единиц. Таким образом, числовая функция $f(x_1, \ldots, x_n)$ правильно вычислима по Тьюрингу, если существует машина T, такая, что $q_1 1^{x_1} * 1^{x_2} * \ldots * 1^{x_n} \underset{T}{\Rightarrow} q_z 1^y$, когда $f(x_1, \ldots, x_n)=y$, и T работает бесконечно, начиная с $q_1 1^{x_1} * 1^{x_2} * \ldots * 1^{x_n}$, когда $f(x_1, \ldots, x_n)$ 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 осуществляет машина T_+ с четырьмя состояниями и следующей системой команд (первая команда введена для случая, когда $a=0$ и исходное слово имеет вид $*1^b$):

$$q_1 * \to q_z \lambda R;$$
$$q_1 1 \to q_2 \lambda R;$$
$$q_2 1 \to q_2 1R;$$
$$q_2 * \to q_3 1L;$$
$$q_3 1 \to q_3 1L;$$
$$q_3 \lambda \to q_z \lambda R.$$

В этой системе команд перечислены не все сочетания состояний машины и символов ленты: опущены те из них, которые при стандартной начальной конфигурации никогда не встретятся. 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.

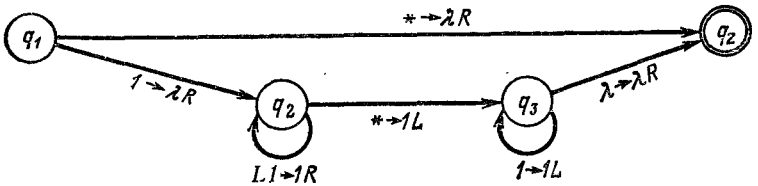

Рис. 5.3

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.

Машина $T_{\text{коп}}$ 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 со-

стояние 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

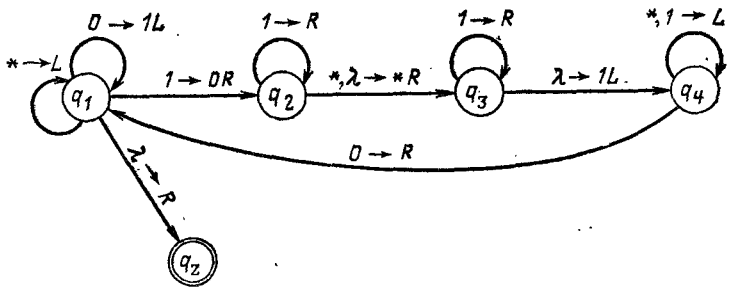

Рис. 5.4

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 $g(x)=f_2(f_1(x))$, 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 вычислима по Тьюрингу.

Пусть T_1 — машина, вычисляющая f_1, а T_2 — машина, вычисляющая f_2, и множества их состояний соответственно $Q_1=\{q_{11}, \ldots, q_{1n_1}\}$ и $Q_2=\{q_{21}, \ldots, q_{2n_2}\}$. 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. Начальным 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 $T_1(1^x)=1^{f_1(x)}$ и $q_{11}1^x \underset{T_1}{\Rightarrow} q_{1z}1^{f_1(x)}$. 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 $q_{21}1^{f_1(x)} \underset{T_2}{\Rightarrow} q_{2z}1^{f_2(f_1(x))}$. Но так как все команды T_2 содержатся в T, то $q_{11}1^x \underset{T}{\Rightarrow} q_{21}1^{f_1(x)} \underset{T}{\Rightarrow} q_{2z}1^{f_2(f_1(x))}$ и, следовательно, $T(1^x)=1^{f_2(f_1(x))}$. 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 $f_2(f_1(x))$. □

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.

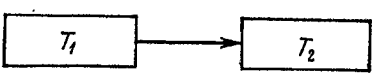

Рис. 5.5

Поскольку машина Тьюринга вектор над $A_{\text{исх}}$ воспринимает как слово в алфавите $A_{\text{исх}} \cup \{*\}$, 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 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;

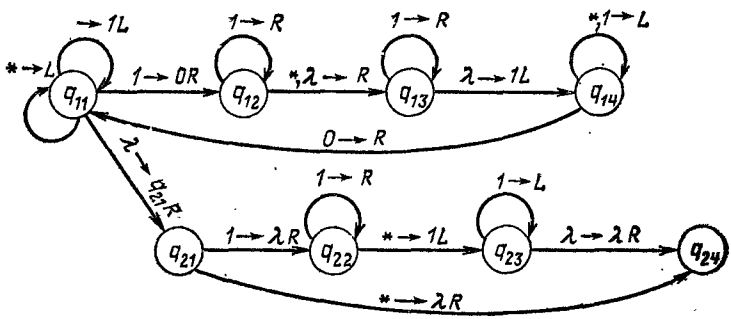

Рис. 5.6

иначе говоря, для любой машины Тьюринга T существует эквивалентная ей машина T_R с правой полулентой. Аналогичная теорема формулируется для левой полуленты.

Можно предположить 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 сдвинет 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 [47]; 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 $\omega=$И, когда $P(\alpha)$ истинно, и $\omega=$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 восста-

новлением, если $T(\alpha) = \omega\alpha$. Если существует машина T, вычисляющая $P(\alpha)$, то существует и $\widetilde{T}$, вычисляющая $P(\alpha)$ с восстановлением. Действительно, вычисление с восстановлением можно представить следующей последовательностью конфигураций: $q_1\alpha \Rightarrow q_{n_1}\alpha * \alpha \Rightarrow \alpha * q_{n_2}\alpha \Rightarrow \alpha * q_{n_3}\omega \Rightarrow q_{n_4}\omega\alpha$. 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

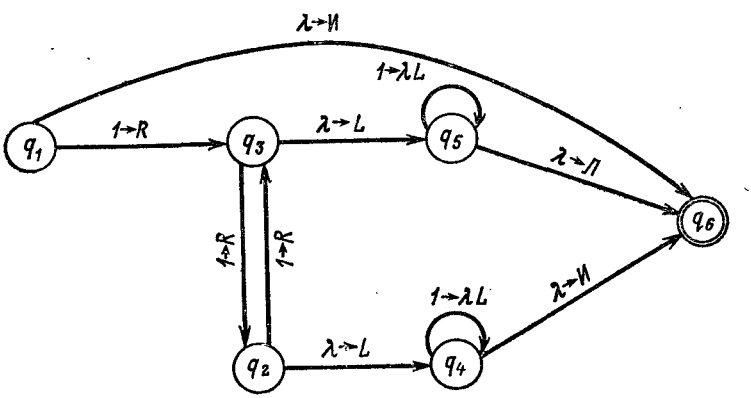

Рис. 5.7

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 $1 \to \lambda L$ на команды $1 \to L$.

Так как машина Тьюринга с алфавитом $A_{\text{исх}} = \{И, Л\}$ и командами $q_1И \to q_z ЛЕ$ и $q_1Л \to q_z ИЕ$ вычисляет отрицание логической переменной, то из вычислимости всюду определенного $P(\alpha)$ следует вычислимость $\overline{P(\alpha)}$.

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 $q_{11}, q_{12}, \ldots, q_{1n_1}$ и системой команд Σ_1, вычисляющая g_1; T_2 — машина с состояниями $q_{21}, q_{22}, \ldots, q_{2n_2}$ 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:

$$\Sigma_3 = \Sigma_1 \cup \Sigma_2 \cup \{q_{31}\, И \to \lambda q_{11} R,\ q_{31}\, Л \to \lambda q_{21} R,\ q_{1z} \to q_{2z} E\}.$$

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. □

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 ($n=1, 2\ldots$). Цикл из состояний q_1, q_2, q_3 — это «зацикленная» машина T_+, в которой заключительное состояние совмеще-

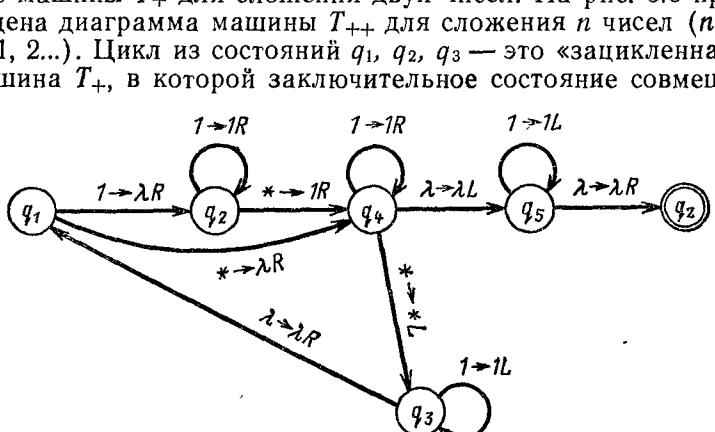

Рис. 5.8

но 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 чем говорит наличие маркера *), то происходит переход к следующему циклу; если нет (о чем 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 (например, в АЛГОЛе) 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, 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 $T_\times$, осуществляющей умножение двух чисел: $T_\times(1^a * 1^b)=1^{ab}$. 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 $\sim$;

$T_{\text{коп}}$ — см. пример 5.4, б, в котором команду $q_1\lambda\to q_zR$ надо заменить на команду $q_1\sim\to q_zR$;

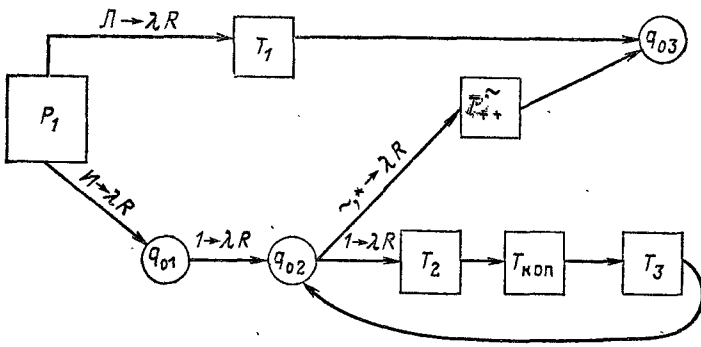

Рис. 5.9

$T_{\text{коп}}(a-1)$ раз копирует 1^b; после i-го цикла она останавливается в конфигурации $1^{a-i-1}(\sim 1^b)^{i-1}\sim q1^b\ *\ 1^b$;

T_3 — вернуть головку к крайнему слева непустому символу. После $(a-1)$-го цикла этим символом окажется $\sim$ (или $*$, если $a=1$), и происходит выход из цикла и переход к $T\tilde{+}+$.

$T\tilde{+}+$ работает как T_{++} (см. пример 5.7, а) с той разницей, что числа, которые она суммирует, разделены двумя видами маркеров: $\sim$ и $*$; для этой цели в нее к командам с маркером $*$ в левой части добавлены такие же команды для маркера $\sim$.

Аналогично тому, как из машины T_+ была построена T_{++}, можно из $T_\times$ построить машину $T_{\times\times}$, которая перемножает несколько чисел, и по схеме, аналогичной приведенной на рис. 5.9, с использованием $T\tilde{\times}\times$ вместо $T\tilde{+}+$ построить машину, осуществляющую возведение в степень. Другой важный пример, который читатель может либо построить, либо найти в [44, 47], — 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. со-

вокупность 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 наделают 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 $q'_t a'_j R$, то заменить в текущей конфигурации $q_i a_j$ на $a'_j q'_t$ (получится конфигурация $\alpha_1 a_k a'_j q'_t \alpha_2$); если же правая часть имеет вид $q'_t a'_j L$, то заменить $a_k q_i a_j$ на $q'_t a_k a'_j$ (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 кото-

рой 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[1]. 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 полуленту), а в левой полуленте записан код системы команд $S(\Sigma_T)$. В частности, начальная конфигурация U имеет вид $u_1S(\Sigma_T)ZS(q_1)S(\alpha)$ (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 $S(K)$. Благода-

[1] 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 $T_{пр}$, реализующую этот алгоритм, и рассматривать универсальную машину $U(T_{пр}(\Sigma_T),\alpha)$.

ря 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 $q_i a_j \rightarrow$

$\rightarrow q_i' a_j' L$). 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 $S(\Sigma_T) Z\lambda ... \lambda u_z \beta$, 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

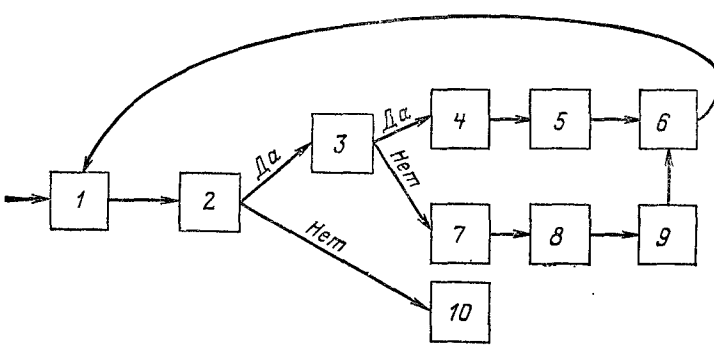

Рис. 5.10

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 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 $A_{\text{исх}} = \{a_1, \ldots, a_m, «, », \|\}$ и m вспомогательными символами $b_1, \ldots, b_m$ приведена на рис. 5.11.

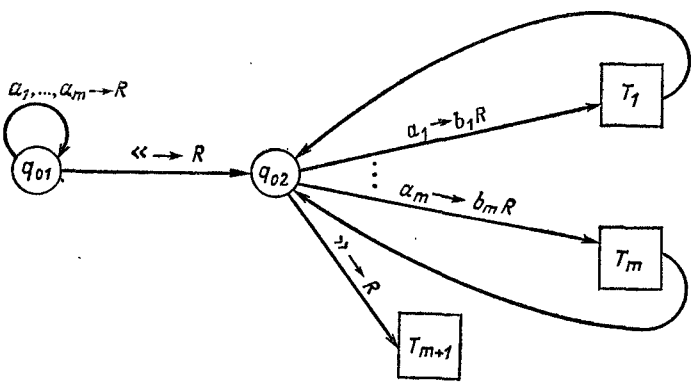

Рис. 5.11

Все машины $T_1, \ldots, T_m$ 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 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

Таблица 5.2

	$a_1, \ldots, a_m$	$\gg$	$\|$	$b_1, \ldots, b_m$
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}	$q_{i4}L$	—	—	$q_{i2}R$

отметки, т. е. заменяет b_i на a_i, и останавливается. Система команд машины T_i $(i=1, \ldots, m)$ 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 « и ». 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 → 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 → (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. Шеннон установил менее очевидный факт — он построил универсальную машину с двумя состояниями. В то же время показано (Боброу и Минский), что универсальная машина с двумя состояниями и двумя символами невозможна. Вообще в определенных пределах уменьшение числа сим-

волов U ведет к увеличению числа состояний, и наоборот. Подробная сводка результатов о минимальных универсальных машинах приведена в [44].

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 универсальной управляющей ЭВМ. 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 годах, до создания современных ЭВМ.

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? Утвердительный ответ на этот вопрос содержится в тезисе Тьюринга, который формулируется так: «Всякий алгоритм может быть реализован машиной Тьюринга».

Доказать тезис Тьюринга нельзя, поскольку само поня-

тие 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 результативности. 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) = $И, если $A(\alpha)$ дает результат, и $B(A, \alpha) = $Л, 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) = $И, если машина $T(\alpha)$ останавливается, и $T_0(\Sigma_T, \alpha) = $Л, 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) = $ = И, если машина $T(\Sigma_T)$ останавливается, и $T_1(\Sigma_T) = $ 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) $q_{1n}\text{Л} \to q_{1,n+1}E$, $q_{1n}a_j \to q_{1n}R$ 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' остановится, если она не останавливается, и не остановится, 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. □

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[1]. Например, она решается для всех машин, приведенных в примерах 5.2—5.8. Поэтому неразрешимость общей проблемы остановки вовсе не снимает необходимости 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 устранить ее. При этом решающую роль играют опыт и искусство программиста.

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 годах нашего века теории рекурсивных функций. В этой теории, как и вообще в теории алгоритмов, принят конструктивный, финитный подход (см. § 5.1), основной чертой кото-

[1] Однако существуют конкретные машины Тьюринга (например, любая универсальная машина) с неразрешимой проблемой остановки.

рого 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):

$$I_m^n(x_1, \ldots, x_n) = x_m \ (m \leqslant n).$$

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 $h(x_1, \ldots, x_m)$, $g_1(x_1, \ldots, x_n)$, $\ldots$, $g_m(x_1, \ldots, x_n)$

$$S_m^n(h, g_1, \ldots, g_m) = h(g_1(x_1 \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)) =$$
$$= f(x_1, \ldots, x_n).$$

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:

$$f(x_2, x_1, x_3, \ldots, x_n) = f\left(I_2^2(x_1, x_2), I_1^2(x_1, x_2), x_3, \ldots, x_n\right);$$

$$f(x_1, x_1, x_3, \ldots, x_n) = f\left(I_1^2(x_1, x_2), I_1^2(x_1, x_2), x_3, \ldots, x_n\right).$$

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:

$$\left.\begin{aligned} &f(x_1, \ldots, x_n, 0) = g(x_1, \ldots, x_n); \\ &f(x_1, \ldots, x_n, y+1) = h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y)). \end{aligned}\right\} \quad (5.2)$$

Пара равенств (5.2) называется *схемой примитивной рекурсии*. Тот факт, что функция f определена схемой (5.2), выражается равенством $f(x_1, \ldots, x_n, y) = R_n(g, h)$. 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 $f(x_1, \ldots, x_n, k)$ понадобится $k+1$ вычислений по схеме (5.2) — для $y=0, 1, \ldots, k$.

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 $x_1, \ldots, x_n$ 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 $m \leqslant n$, являются примитивно-рекурсивными.

2. Если $g_1(x_1, ..., x_n), ..., g_m(x_1, ..., x_n)$, $h(x_1, ..., x_m)$ — примитивно-рекурсивные функции, то $S_m^n(h, g_1, ..., g_m)$ — примитивно-рекурсивные функции для любых натуральных n, m.

3. Если $g(x_1, ..., x_n)$ и $h(x_1, ..., x_n, y, z)$ — 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:

$$f_+(x, 0) = x = I_1^1(x);$$

$$f_+(x, y + 1) = f_+(x, y) + 1 = (f_+(x, y))'.$$

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:

$$f_\times(x, 0) = 0;$$

$$f_\times(x, y + 1) = f_\times(x, y) + x = f_+(x, f_\times(x, y)).$$

3. Возведение в степень $f_{\exp}(x, y) = x^y$ примитивно-рекурсивно:

$$f_{\exp}(x, 0) = 1;$$

$$f_{\exp}(x, y + 1) = x^y x = f_\times(x, f_{\exp}(x, y)).$$

Определим функцию $x \dot{-} y$ (арифметическое или урезанное вычитание) следующим образом:

$$x \dot{-} y = \begin{cases} x - y, & \text{если } x > y; \\ 0 & \text{в противном случае.} \end{cases}$$

Пример 5.10 (вычитание). Примитивно-рекурсивными являются следующие функции:

1) $f(x) = x \dot{-} 1$, определяемая схемой:

$$f(0) = 0 \dot{-} 1 = 0;$$

$$f(y + 1) = y;$$

2) $f(x, y) = x \dot{-} y$, определяемая схемой:

$$f(x, 0) = x;$$

$$f(x, y + 1) = x \dot{-} (y + 1) = (x \dot{-} y) \dot{-} 1 = f(x, y) \dot{-} 1$$

(для определения функции из п. 2 использована функция из п. 1);

$$3)\ f(x, y) = |x - y| = (x \dot{-} y) + (y \dot{-} x);$$

$$4)\ \operatorname{sg}(x) = \begin{cases} 0, & \text{если } x = 0; \\ 1, & \text{если } x \neq 0; \end{cases}$$

ее схема имеет вид:

$$\operatorname{sg}(0) = 0;$$

$$\operatorname{sg}(x + 1) = 1;$$

$$5)\ \min(x, y) = x \dot{-} (x \dot{-} y);$$

$$6)\ \max(x, y) = y + (x \dot{-} y).$$

С помощью функции sg (сигнум) из примера 5.10,г и ее отрицания $\overline{\operatorname{sg}}(x) = 1 \dot{-} \operatorname{sg} x$ построим примитивно-рекурсивное описание функций, связанных с делением.

Пример 5.11 (деление). **а.** Функция $r(x, y)$ — остаток от деления y на x:

$$r(x, 0) = 0;$$

$$r(x, y + 1) = (r(x, y) + 1) \operatorname{sg}(|x - (r(x, y) + 1)|).$$

Смысл второй строки определения в следующем: если $y+1$ не делится на x, то $\operatorname{sg}(|x \dot{-} (r(x, y) + 1)|) = 1$ и $r(x, y+1) = r(x, y) + 1$; если же $y+1$ делится на x, то $r(x, y+1) = \operatorname{sg}(|x - (r(x, y) + 1)|) = 0$.

б. Функция $q(x, y) = [y/x]$ — частное от деления y на x, т. е. целая часть дроби y/x:

$$q(x, 0) = 0;$$

$$q(x, y + 1) = q(x, y) + \overline{\operatorname{sg}}(|x - (r(x, y) + 1)|).$$

Второе слагаемое, как и в случае $r(x, y)$, зависит от делимости $y+1$ на x. Если $y+1$ делится на x, то $q(x, y+1)$ на единицу больше, чем $q(x, y)$; если нет, то $q(x, y+1) = q(x, y)$.

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 при-

митивная рекурсивность «арифметизованных» логических функций, т. е. числовых функций, которые на множестве {0, 1} ведут себя как логические функции. Действительно, если $x, y \in \{0, 1\}$, то

$$\begin{aligned} \bar{x} &= 1 \dot{-} x; \\ x \vee y &= \max(x, y); \\ x \& y &= \min(x, y). \end{aligned} \tag{5.4}$$

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 $R(x_1, ..., x_n)$ называется примитивно-рекурсивным, если примитивно-рекурсивна его характеристическая функция χ_R:

$$\chi_R(x_1, \ldots, x_n) = \begin{cases} 1, & \text{если } R(x_1, \ldots, x_n) \text{ выполняется;} \\ 0 & \text{в противном случае.} \end{cases}$$

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 типа АЛГОЛ 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 $P_1, .., P_k$ 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:

$$\chi_{Pd_n}(x) = \overline{\operatorname{sg}}(r(n, x)).$$

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:

$$\chi_{>}(x_1, x_2) = \operatorname{sg}(x_1 \dot{-} x_2).$$

г. Если $f(x)$ и $g(x)$ примитивно-рекурсивны, то предикат «$f(x) = g(x)$» примитивно-рекурсивен, так как его функция χ имеет вид:

$$\chi(x) = \overline{\operatorname{sg}}(|f(x) - g(x)|).$$

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 $g_1(x_1, \ldots, x_n)$, $g_2(x_1, \ldots, x_n)$ и предикату $P(x_1, \ldots, x_n)$ строит функцию $f(x_1, \ldots, x_n) = B(g_1, g_2, P)$:

$$f(x_1, \ldots, x_n) = \begin{cases} g_1(x_1, \ldots, x_n), \text{ если} \\ \quad P(x_1, \ldots, x_n) \text{ истинно;} \\ g_2(x_1, \ldots, x_n), \text{ если} \\ \quad P(x_1, \ldots, x_n) \text{ ложно.} \end{cases} \tag{5.5}$$

Теорема 5.3 утверждает, что B вычислим по Тьюрингу. Примитивная рекурсивность B видна из следующего соотношения, эквивалентного (5.5):

$$f(x_1, \ldots, x_n) = g_1(x_1, \ldots, x_n)\,\chi_P(x_1, \ldots, x_n) + \\ + g_2(x_1, \ldots, x_n)\,\chi_{\overline{P}}(x_1, \ldots, x_n).$$

Обобщение оператора B на случай многозначного перехода по предикатам $P_1, \ldots, P_k$, из которых истинен всегда один и только один предикат: $f(x_1, \ldots, x_n) = B(g_1, \ldots, g_k, P_1, \ldots, P_k)$, также примитивно-рекурсивно, поскольку

$$f(x_1, \ldots, x_n) = g_1\,\chi_{P_1} + \ldots + g_k\,\chi_{P_k}$$

(отметим, что это соотношение определяет B и в том случае, когда ни один из $P_1, \ldots, P_k$ не истинен: B при этом равно нулю).

С помощью оператора B удобно задавать функции, оп-

ределенные 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 $f(x_1, ..., x_n, y)$ — функция от $(n+1)$-й переменной. Хорошо известные операции суммирования $\sum\limits_{y<z}$ и перемножения $\prod\limits_{y<z}$ по переменной y с пределом z — это операторы, которые из функции $f(x_1, ..., x_n, y)$ порождают новые функции $g(x_1, ..., x_n, z)=\sum\limits_{y<z} f(x_1, ..., x_n, y)$ и $h(x_1, ..., x_n, z)=\prod\limits_{y<z} f(x_1, ..., x_n, y)$. Покажем их примитивную рекурсивность:

$$g(x_1, \ldots, x_n, 0)=0 \text{ (по определению);}$$

$$g(x_1, \ldots, x_n, z+1)=g(x_1, \ldots, x_n, z)+f(x_1, \ldots, x_n, z);$$

$$h(x_1, \ldots, x_n, 0)=1;$$

$$h(x_1, \ldots, x_n, z+1)=h(x_1, \ldots, x_n, z)\, f(x_1, \ldots, x_n, z).$$

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 $\sum\limits_{y<z}$ и $\prod\limits_{y<z}$ нетрудно показать, что операторы $\sum\limits_{y=k}^{z}$ и $\prod\limits_{y=k}^{z}$ (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 огра-*

ниченным оператором минимизации, который применяется к предикатам и определяется так:

$$\mu y_{y\leqslant z} P(x_1, \ldots, x_n, y) = \begin{cases} \text{наименьшему } y \leqslant z, \text{ такому, что } P(x_1, \ldots, x_n, y) \text{ истинно, если такой } y \text{ существует;} \\ z \text{ в противном случае.} \end{cases}$$

Из предиката $P(x_1, \ldots, x_n, y)$ с помощью оператора $\mu y_{y\leqslant z}$ получается функция $f(x_1, \ldots, x_n, z)$. 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 $\mu_{y\leqslant z} P(x_1, x_2, y)$ равен наименьшему общему кратному (НОК) x_1 и x_2, если $z \geqslant$ НОК, и равен z, если $z \leqslant$ НОК.

Ограниченный μ-оператор примитивно-рекурсивен:

$$\mu y_{y\leqslant z} P(x_1, \ldots, x_n, y) = \sum_{i=0}^{z} \prod_{j=0}^{i} (1 - \chi_P(x_1, \ldots, x_n, j)).$$

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 $f(x_1, \ldots, x_n, k) = R_n(g, h)$, то для вычисления $f(x_1, \ldots, x_n, k)$ 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 $f(x_1, \ldots, x_n, k)$. 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[1] умножению:

$$[z/x] = \mu y_{y<z}\,(x\,(y+1) > z).$$

б. Целая часть $\sqrt{x}$ — функция $[\sqrt{x}]$ — примитивно-рекурсивная, так как

$$[\sqrt{x}\,] = \mu y_{y<x}\,((y+1)^2 > x).$$

в. $[\log_k x] = \mu y_{y<x}(k^{y+1} > x)$, 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 $f_1, \ldots, f_k$ от $(n+1)$-й переменной, причем значение каждой функции $y+1$ зависит от значения всех функций в точке y:

$$f_1(x_1, \ldots, x_n, 0) = g_1(x_1, \ldots, x_n);$$

. . . . . .

$$f_k(x_1, \ldots, x_n, 0) = g_k(x_1, \ldots, x_n);$$

$$f_1(x_1, \ldots, x_n, y+1) = h_1(x_1, \ldots, x_n, y, f_1(x_1, \ldots, x_n, y), \ldots, f_k(x_1, \ldots x_n, y));$$

. . . . . .

$$f_k(x_1, \ldots, x_n, y+1) = h_k(x_1, \ldots, x_n, y, f_1(x_1, \ldots, x_n, y), \ldots, f_k(x_1, \ldots, x_n, y)).$$

По существу совместная рекурсия дает рекурсивное описание функции-вектора $(f_1, \ldots, f_k)$. 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

[1] 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 $>$.

$\varphi(x_1, ..., x_n, y) = 2^{f_1(x_1,...,x_n,y)} \cdot 3^{f_2} \cdot 5^{f_3} \cdot ... \cdot p_{k-1}^{f_k(x_1,...,x_n,y)}$, 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:

$$f_1(x, 0) = g_1(x);$$
$$f_2(x, 0) = g_2(x);$$
$$f_1(x, y+1) = h_1(x, y, f_1(x, y), f_2(x, y));$$
$$f_2(x, y+1) = h_2(x, y, f_1(x, y), f_2(x, y)).$$

Определим кодирующую функцию F:

$$F(x, 0) = 2^{g_1(x)} \cdot 3^{g_2(x)};$$
$$F(x, y+1) = 2^{h_1(x,y,f_1(x,y),f_2(x,y))} \cdot 3^{h_2(x,y,f_1(x,y),f_2(x,y))}.$$

Тогда $f_1(x, y) = \mu z_{z \leqslant F(x,y)} (\overline{\mathrm{Pd}}_{2^{z+1}}(F(x, y)))$, т. е. равна показателю при 2 в разложении $F(x, y)$ на простые множители; $f_2(x, y) = \mu z_{z \leqslant F(x,y)} (\mathrm{Pd}_{3^{z+1}}(F(x, y)))$, 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, *а* 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: $t=0, 1, 2...$ 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 время 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, *а* 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 обрат-

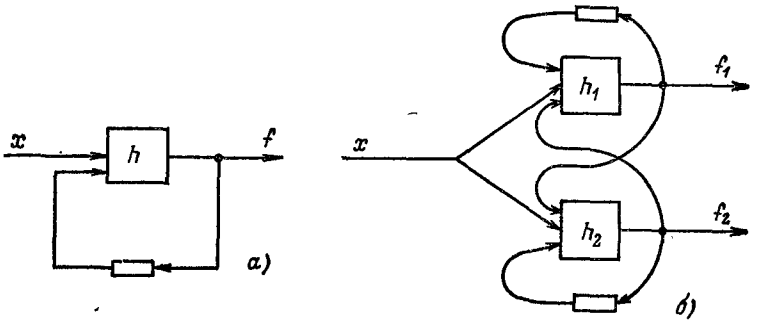

Рис. 5.12

ной 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 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 вычисли-

мых 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 $N \to N$ (т. е. одноместных целочисленных функций) несчетно, тем более это верно для функций типа $N^n \to N$. Каждая примитивно-рекурсивная функция имеет конечное описание, т. е. задается конечным словом в некотором (фиксированном для всех функций) алфавите. Множество всех конечных слов счетно, поэтому примитивно-рекурсивные функции образуют не более чем счетное подмножество несчетного множества функций типа $N^n \to N$. 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 φ_0, φ_1, φ_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:

$$\varphi_1(a, y+1) = a + ay = \varphi_0(a_1, \varphi_1(a, y)); \quad \varphi_1(a, 1) = a;$$

$$\varphi_2(a, y+1) = aa^y = \varphi_1(a, \varphi_2(a, y)); \quad \varphi_2(a, 1) = a.$$

Продолжим эту последовательность, положив по определению

$$\left.\begin{aligned} &\varphi_{n+1}(a, 0) = 1; \\ &\varphi_{n+1}(a, 1) = a; \\ &\varphi_{n+1}(a, y+1)\,\varphi_n = (a_1, \varphi_{n+1}(a, y)). \end{aligned}\right\} \qquad (5.6)$$

Эта схема имеет вид примитивной рекурсии, следовательно, все функции φ_n примитивно-рекурсивные. Растут они крайне быстро; например, $\varphi_3(a, 1)=a$; $\varphi_3(a, 2)=\varphi_2(a, a)=a^a$; $\varphi_3(a, 3)=a^{a^a}$; ...; $\varphi_3(a, k)=a^{a^{\cdot^{\cdot^{a}}}}\}k$ раз. Зафиксируем теперь значение $a: a=2$. Получим последовательность одноместных функций: $\varphi_0(2, y)$, $\varphi_1(2, y)$... 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

$$\left.\begin{aligned} &B(0, y)=2+y;\\ &B(x+1, 0)=\operatorname{sg} x;\\ &B(x+1, y+1)=B(x, B(x+1, y)), \end{aligned}\right\} \qquad (5.7)$$

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 $x \geqslant n$ $A(x)>f(x)$.

Ограничимся наброском доказательства, опустив некоторые выкладки.

1. Вначале доказываются два свойства функции $B(x, y)$:

$$B(x, y+1)>B(x, y) \quad (x, y=1, 2 \ldots); \qquad (5.8)$$

$$B(x+1, y) \geqslant B(x, y+1) \quad (x \geqslant 1, y \geqslant 2). \qquad (5.9)$$

2. Назовем функцию $f(x_1, ..., x_n)$ B-мажорируемой, если существует натуральное m, такое, что для любых $x_1, ..., x_n$, удовлетворяющих усло-

вию $\max(x_1, ..., x_n) > 1$, $f(x_1, ..., x_n) < B(m, \max(x_1, ..., x_n))$. Покажем, что все примитивно-рекурсивные функции B-мажорируемы:

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-мажорируемы 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)

$$(x) = h(g_1(x), g_2(x)) < B(m, \max(g_1(x), g_2(x))) < B(m, B(m+1, x)) =$$
$$= B(m+1, x+1) \leqslant B(m+2, x),$$

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

$$f(k+1) = h(k, f(k)) > B(m_h, \max(k, f(k))). \qquad (5.10)$$

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 $m_h \leqslant m-1$, имеем $f(k+1) < B(m_h, f(k)) < B(m_h, B(m, k)) \leqslant B(m-1, B(m, k)) = B(m, k+1)$, что и завершает индукцию.

Из пп. «а»—«в» следует, что все примитивно-рекурсивные функции B-мажорируемы.

3. Пусть $f(x)$ — примитивно-рекурсивная функция. В силу п. 2 для некоторого m и $x \geqslant 2$ $f(x) < B(m, x)$. 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 $x = m+2$. □

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 $f(x_1,...,x_n)=\mu y(g(x_1,...,x_{n-1}, y)=x_n)$ или $f(x_1,...,x_n)=\mu y(g(x_1, ..., x_n, y)=0)$. Будучи применен к вычислимой функции, μ-оператор снова дает вычислимую функцию (т. е. сохраняет вычислимость). Действительно, для вычисления функции $f(x_1, ..., x_n)=\mu y(g(x_1, ..., x_n, y)=0)$ на наборе $x_1, ..., x_n$ существует следующая простая процедура: вычисляем g на наборах $(x_1, ..., x_n, 0)$, $(x_1, ..., x_n, 1)$, ... до тех пор, пока не получим значение нуль. Однако в отличие от рассмотренных ранее процедур она может не привести к результату: это произойдет в случае, когда на данном наборе $x_1, ..., x_n$ уравнение $g(x_1, ..., x_n, y)=0$ не имеет решения. В таком случае функция $f(x_1, ..., x_n)$ 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 данно-

го набора значений $x_1, ..., x_n$ 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 $f(x_1, ..., x_n) = \mu y(g(x_1, ..., x_n, y) = 0)$ вычисляется с учетом следующего условия: если для $y = 0, 1, ..., i-1$ $g(x_1, ..., x_n, y) \neq 0$, а $g(x_1, ..., x_n, i)$ не определена, то и $f(x_1, ..., x_n)$ 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 $y - 1 = 5$ 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 доопределить 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 [38], задача 42,а). Это обстоятельство выражено в виде *тезиса Черча*, являющегося аналогом [1] тезиса Тьюринга для рекурсивных функций: *всякая функция, вычислимая некоторым алгоритмом, частично-рекурсивна.*

Из сопоставления двух тезисов вытекает утверждение: *функция вычислима 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 вычислима на машине Тьюринга.

План доказательства ясен: сначала доказывается, что простейшие функции вычислимы (это довольно очевидно), а затем — это операторы суперпозиции, примитивной рекурсии и μ-оператор сохраняют вычислимость.

Ограничимся построением машины Тьюринга для опера-

[1] Исторически тезис Черча был сформулирован раньше тезиса Тьюринга.

тора примитивной рекурсии R_n. Для простоты обозначений рассмотрим случай $n=1$. Пусть $f(x, y)=R_1(g, h)$, т. е. определена схемой

$$f(x, 0)=g(x);$$

$$f(x, y+1)=h(x, y, f(x, y)),$$

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 $q_1 1^x * 1^y$, 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 $g(x)$: $\bigcirc 1^x * 1^y \bigcirc \bigcirc q 1^x$.

2. Вычислить $g(x)$ на правой полуленте: $\bigcirc 1^x * 1^y \bigcirc q 1^{g(x)}$.

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 $\bigcirc$):

$$1^{i+1} \bigcirc 1^x * 1^y \bigcirc q 1^x * 1^i * 1^{f(x,i)}.$$

5. Вычислить h на правой полуленте $1^{i+1} \bigcirc 1^x * 1^y \bigcirc q 1^{h(x,i,f(x,i))}$.

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 переменных 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. □

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 $A=\{a_1, ..., a_m\}$ 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 вместо 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 $\varphi_q: Q\times A\to Q$ (функция следующего состояния), $\varphi_a: Q\times A\to A$ (функция печатаемого символа), $\varphi_d: Q\times A\to\{0, 1\}$ (функция сдвига). Если Σ_T содержит команду $q_ia_j\to q_ia_jd_k$, 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 $\alpha q_ia_j\beta$ 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 $q_51\to q_30R$ ранее приведенная 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, каждая из которых зависит от четырех числовых переменных $\tilde{\alpha}$, $\tilde{q}$, $\tilde{a}$, $\tilde{\beta}$. Эти функции довольно простым об-

разом 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 цифр 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

$$\tilde{\alpha}'(\tilde{\alpha}, \tilde{q}_i, \tilde{a}_j, \tilde{\beta}) =$$

$$= \begin{cases} m\tilde{\alpha} + \varphi_a(q_i, a_j), & \text{если } \varphi_d(q_i, a_j) = 1 \text{ (сдвиг вправо);} \\ [\tilde{\alpha}/m], & \text{если } \varphi_d(q_i, a_j) = 0 \text{ (сдвиг влево);} \end{cases} \qquad (5.11а)$$

$$\tilde{q}' = \varphi_q(q_i, a_j); \qquad (5.11б)$$

$$\tilde{a}' = \begin{cases} r(m, \tilde{\beta}), & \text{если } \varphi_d(q_i, a_j) = 1; \\ r(m, \tilde{\alpha}), & \text{если } \varphi_d(q_i, a_j) = 0; \end{cases} \qquad (5.11в)$$

$$\tilde{\beta}' = \begin{cases} m\tilde{\beta} + \varphi_a(q_i, a_j), & \text{если } \varphi_d(q_i, a_j) = 0; \\ [\tilde{\beta}/m], & \text{если } \varphi_d(q_i, a_j) = 1. \end{cases} \qquad (5.11г)$$

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 $q_5 1 \to q_3 0 R$ 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:

$$K_\alpha(0, \tilde{\alpha}_0, \tilde{q}_0, \tilde{a}_0, \tilde{\beta}_0)=\tilde{\alpha}_0;$$

$$K_\alpha(t+1, \tilde{\alpha}_0, \tilde{q}_0, \tilde{a}_0, \tilde{\beta}_0)=\tilde{\alpha}'(K_\alpha(t, \tilde{\alpha}_0, \tilde{q}_0, \tilde{a}_0, \tilde{\beta}_0), \quad (5.12а)$$
$$K_q(t, \tilde{\alpha}_0, \tilde{q}_0, \tilde{a}_0, \tilde{\beta}_0), K_a(t, \tilde{\alpha}_0, \tilde{q}_0, \tilde{a}_0, \tilde{\beta}_0),$$
$$K_\beta(t, \tilde{\alpha}_0, \tilde{q}_0, \tilde{a}_0, \tilde{\beta}_0)).$$

В соотношениях (5.12б)—(5.12г) аргументы $\tilde{\alpha}_0$, $\tilde{q}_0$, $\tilde{a}_0$, $\tilde{\beta}_0$ в функциях K_l для краткости опустим:

$$K_q(0)=\tilde{q}_0;$$

$$K_q(t+1)=\tilde{q}'(K_\alpha(t), K_q(t), K_a(t), K_\beta(t)); \quad (5.12б)$$

$$K_a(0)=\tilde{a}_0;$$

$$K_a(t+1)=\tilde{a}'(K_\alpha(t), K_q(t), K_a(t), K_\beta(t)); \quad (5.12в)$$

$$K_\beta(0)=\tilde{\beta}_0;$$

$$K(t+1)=\tilde{\beta}'(K_\alpha(t), K_q(t), K_a(t), K(t)). \quad (5.12г)$$

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_α, K_q, K_a, K_β (рекурсия ведется по переменной t) с помощью функций $\tilde{\alpha}'$, $\tilde{q}'$, $\tilde{a}'$, $\tilde{\beta}'$, примитивная рекурсивность которых уже доказана. Следовательно, функции K_l также примитивно-рекурсивны.

Остался последний шаг доказательства: рекурсивное описание результата работы машины Тьюринга. Ограничимся для простоты правильно вычисляющими машинами (которые начинают с конфигураций вида $q_1a_0\beta_0$ и останавливаются в конфигурациях вида $q_za_z\beta_z$). 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:

$$f(x)=mK_\beta(\mu t(K_q(t, 0, 1, r(m, x), [x/m])=z),$$
$$0, 1, r(m, x), [x/m])+K_a(\mu t(K_q(t, 0, 1, r(m, x), [x/m])=z),$$
$$0, 1, r(m, x), [x/m]) \qquad (5.13)$$

(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. □

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. □

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[1]. 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

[1] 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.

любой формализации понятия «алгоритм», что не мешает использовать в каждом конкретном рассуждении конкретную алгоритмическую модель, наиболее подходящую для данного случая.

Отступление 5.1. Для читателя этой книги, который в конечном счете интересуется прикладиой стороиой 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. Машиины Тьюринга весьма далеки от современиых ЭВМ, а рекурсивные функции — от языков программирования прежде всего нз-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, прикладиое 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 φ: $N \to Al^*$, взаимно однозначно отображающей в слова в алфавите Al, выбранном для описания алгоритмов, числа натурального ряда. Такая функция $\varphi(n) = A$ называется *нумерацией* алгоритмов, а ее аргумент n — номером алгоритма A при нумерации φ. Из взаимной однозначности отображения φ следует существование обратной функции $\varphi^{-1}(A_n) = n$, восстанавливающей по описанию алгоритма A_n его номер n. Более общие определения нумераций см. в [40].

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 $\varphi^{-1}(A)$ $U(\varphi^{-1}(A), y) = A(y)$; 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^*. □

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. □

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 $A_0, A_1, A_2\ldots$ 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,

$$B(x) = A_{x_B}(x). \tag{5.15}$$

Тогда в точке $x=x_B$ по определению $B(x_B)=A_{x_B}(x_B)+1$, а в силу (5.15) $B(x_B)=A_{xB}(x_B)$. Получаем противоречие, откуда следует, что B не входит в перечисление, порождаемое ψ. □

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:

$$g(0) = \mu y[f(y) = 1];$$
$$g(x+1) = \mu y[y > g(x) \,\&\, f(y) = 1].$$

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 $A_0, A_1, A_2\ldots$ *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. □

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 желательно-

сти такого требования, как результативность алгоритма [1]. 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 $\varphi^{-1}(A_x) = x$. Очевидно, что $f(x)$ вычислима. Пусть теперь g — произвольная общерекурсивная функция и x_g — ее номер: $\varphi^{-1}(g) = x_g$. 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. □

Следовательно, существуют частичные алгоритмы, которые нельзя доопределять 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 ал-

[1] Отметим, что существенность этого требования проявляется в понятии неразрешимости; неразрешимость проблемы истолковывается как отсутствие *всюду* определенного алгоритма, решающего эту проблему.

горитма 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 перечис-

ляющей 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... 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 $\psi_\pi=$ это пример функции, перечисляющей с повторениями. 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, нетрудно извлечь порождающую процедуру. Иногда она содержится в них явно, как, скажем, в определении $(n+1)!=n!(n+1)$, задающем и перечисляющую функцию $\psi(n)=n!$, и способ ее вычисления. Иногда, чтобы получить из такого определения порождающую функцию, нужно ввести дополнительные соглашения. Например, известное индуктивное определение АЛГОЛа «идентификатор — это либо буква, либо идентификатор, к которому справа приписана буква или цифра» порождает все возможные в АЛГОЛе идентификаторы; однако для того, чтобы придать этому процессу порождения вид перечисляющей функции, нужно ввести порядок перечисления. Если в качестве такого порядка принять лексико-графический 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 общерекурсивных фуикций.

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 $\alpha_1, \alpha_2, \ldots, \alpha_n \ldots$ — перечисление всех слов в алфавите A (например, в лексико-графическом 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. □

Обращение теоремы 5.14 неверно.

Теорема 5.15. Существует множество M, которое перечислимо, но неразрешимо.

Докажем, что таковым является множество $M = \{x | A_x(x)$ 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 $\mathrm{St}(x, m)$ — 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 $\mathrm{St}(x, m)$ можно представить 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 $\mathrm{St}(i, j)$.

Таблица 5.3

x	m			
	0	1	2	...
0	St (0,0)	St (0,1)	St (0,2)	...
1	St (1,0)	...	...	...
2	St (2,0)	...	...	...
.				
.	...	...	...	...
.				

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 $\mathrm{St}(i, j) = $ И, то $\psi_M(n) = i$; если $\mathrm{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. □

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 M и $\overline{M}$ перечислимы.

Действительно, если M разрешимо, то $\overline{M}$ также разрешимо: $\psi_{\overline{M}} = 1 - \chi_M$, но тогда перечислимость M и $\overline{M}$ следует из теоремы 5.14.

Пусть теперь M и $\overline{M}$ перечислимы с функциями ψ_M и $\psi_{\overline{M}}$ соответственно. Но тогда для любого элемента a его поиск в обоих списках, точнее, в объединенном списке $\psi_M(0)$, $\psi_{\overline{M}}(0)$, $\psi_M(1)$, $\psi_{\overline{M}}(1)$... обязательно увенчается успехом, так как либо $a \in M$, либо $a \in \overline{M}$. Такой поиск и есть разрешающая процедура для M. □

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. Просматривая накопленный запас алгоритмически неразреши-

мых 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 (теорема Райса). 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 $f_0 \in \overline{M}$. 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 $F(x, y)$ вычислима: 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 $g(x)$ вычислима. 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 $f_a \in \overline{M}$; последующие рассуждения аналогичны, а разрешающая функция для проблемы самоприменимости будет иметь вид $1 - \chi_M(g(x))$. □

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 о 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 при-

ходит, 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 мате-

матике 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 задачи). Задача в более общей постановке имеет больше шансов оказаться неразрешимой.

ГЛАВА ШЕСТАЯ

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 лексико-графического 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. Ньютоном 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 аксиомати-

ческого 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 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 $R(F_1, ..., F_n, G)$ — это вычислимое отношение на множестве формул. Если формулы $F_1, ..., F_n, G$ находятся в отношении R, то формула G называется *непосредственно выводимой* из $F_1, ..., F_n$ по правилу R. Часто правило $R(F_1, ..., F_n, G)$ записывается в виде $\frac{F_1, \ldots, F_n}{G}$. Формулы $F_1, ... F_n$ называются *посылками* правила R, а G — его следствием или *заключением.* Примеры аксиом и правил вывода будут приведены несколько позднее.

Выводом формулы B из формул $A_1, ..., A_n$ называется последовательность формул $F_1, ..., F_m$, такая, что $F_m = B$, а любая $F_i (i = 1, ..., m)$ есть либо аксиома, либо одна из исходных формул $A_1, ..., A_n$, либо непосредственно выводима из формул $F_1, ..., F_{i-1}$ (или какого-то их подмножества) по одному из правил вывода. Если существует вывод B из $A_1, ..., A_n$, то говорят, что B *выводима* из $A_1, ..., A_n$. Этот факт обозначается так: $A_1, ..., A_n \vdash B$. Формулы $A_1, ..., A_n$ 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 $A_1, \ldots, A_n \vdash B$, то $A_1, \ldots, A_n, A_{n+1} \vdash B$ 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 $A_1, \ldots, A_n$, то высказывание «$A_1, \ldots, A_n \vdash B$» является метатеоремой; ее можно рассматривать как дополнительное («произвольное») правило вывода, которое можно присоединить к исходным и использовать в дальнейшем.

Ясно, что общезначимые (тождественно-истинные) высказывания типа $A \vee \bar{A}$ или $\forall x P(x) \to P(y)$, 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 $\vee$, $\&$, $\neg$, $\to$. 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 высказы-

ваний. Исчисление высказываний определяется следующим образом.

1. Алфавит исчисления высказываний состоит из переменных высказываний (пропозициональных букв): A, B, C..., знаков логических связок $\vee$, $\&$, $\neg$, $\rightarrow$ и скобок (,).

2. Формулы:

а) переменное высказывание есть формула;

б) 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;

в) 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. Аксиомы. Приведем здесь две системы аксиом. Первая из них непосредственно использует все логические связки:

Система аксиом I

I1. $A \rightarrow (B \rightarrow A)$;

I2. $(A \rightarrow B) \rightarrow ((A \rightarrow (B \rightarrow C)) \rightarrow (A \rightarrow C))$;

I3. $(A \,\&\, B) \rightarrow A$;

I4. $(A \,\&\, B) \rightarrow B$;

I5. $A \rightarrow (B \rightarrow (A \,\&\, B))$;

I6. $A \rightarrow (A \vee B)$;

I7. $B \rightarrow (A \vee B)$;

I8. $(A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow ((A \vee B) \rightarrow C))$;

I9. $(A \rightarrow B) \rightarrow ((A \rightarrow \neg B) \rightarrow \neg A)$;

I10. $\neg\neg A \rightarrow A$.

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

II1. $A \rightarrow (B \rightarrow A)$;

II2. $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$;

II3. $(\neg A \to \neg B) \to ((\neg A \to B) \to A)$.

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

$$\mathfrak{B}; \quad \frac{\mathfrak{A}(A)}{\mathfrak{A}(\mathfrak{B})};$$

2) правило заключения (Modus Ponens). Если $\mathfrak{A}$ и $\mathfrak{A} \to \mathfrak{B}$ — выводимые формулы, то $\mathfrak{B}$ выводима:

$$\frac{\mathfrak{A},\ \mathfrak{A} \to \mathfrak{B}}{\mathfrak{B}}.$$

В этом описании исчисления высказываний аксиомы являются формулами исчисления (соответствующими определению формулы); формулы же, использованные в правилах вывода ($\mathfrak{A}$, $\mathfrak{A} \to \mathfrak{B}$ и т. д.), это «метаформулы», или *схемы формул*. Схема формул, например $\mathfrak{A} \to \mathfrak{B}$, 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 $\mathfrak{A} \to \mathfrak{B}$ получим формулу $A \to A\&B$.

Использование схем формул можно распространить и на аксиомы. Например, если в системе II переменные (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 $A\to A$ выводима из системы аксиом II:

$$\vdash A\to A.$$

1. $(A\to((A\to A)\to A))\to((A\to(A\to A))\to(A\to A))$ (подстановка в аксиому II2 $A\to A$ вместо B и A вместо C).
2. $A\to((A\to A)\to A)$ (подстановка в II1 $A\to A$ вместо B).
3. $(A\to(A\to A))\to(A\to A)$ (из шагов 2, 1 по правилу заключения).
4. $A\to(A\to A)$ (подстановка в II1 A вместо B).
5. $A\to A$ (из шагов 4, 3 по правилу заключения).

б. $A\vdash B\to A$.

Пусть A выводима. Тогда из A и аксиомы II 1 по правилу заключения получаем $\dfrac{A,\ A\to(B\to A)}{B\to A}$, 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 $\dfrac{\Gamma}{\mathfrak{A}}$, которое можно присоединить к уже имеющимся. Полученную нами выводимость $A\vdash B\to A$ вместе с правилом подстановки можно рассматривать как правило $\dfrac{\mathfrak{A}}{\mathfrak{B}\to\mathfrak{A}}$:

«если формула $\mathfrak{A}$ выводима, то выводима и формула $\mathfrak{B} \to \to \mathfrak{A}$, где $\mathfrak{B}$ — любая формула». Воспользуемся этим правилом в следующем примере.

в. $A \to B,\ B \to C \vdash A \to A \to C$.

1. $B \to C \vdash A \to (B \to C)$ (по новому правилу $\dfrac{\mathfrak{A}}{\mathfrak{B} \to \mathfrak{A}}$).

2. Из $A \to (B \to C)$ и аксиомы II 2 по правилу заключения следует $(A \to B) \to (A \to C)$; следовательно,

$$B \to C \vdash (A \to B) \to (A \to C).$$

3. Из $(A \to B)$ и $(A \to B) \to (A \to C)$ по правилу заключения следует $A \to C$; 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 6.1 (теорема дедукции).

Если Γ, $\mathfrak{A} \vdash \mathfrak{B}$, то $\Gamma \vdash \mathfrak{A} \to \mathfrak{B}$ (Γ — множество формул, $\mathfrak{A}$, $\mathfrak{B}$ — формулы).

Будем исходить из системы аксиом II и рассматривать их как схемы аксиом (т. е. не пользоваться правилом подстановки).

Пусть Γ, $\mathfrak{A} \vdash \mathfrak{B}$. Тогда существует вывод $\mathfrak{B}_1, \ldots, \mathfrak{B}_n$ из Γ, $\mathfrak{A}$, такой, что $\mathfrak{B}_n = \mathfrak{B}$. Докажем по индукции, что для любого $k \leqslant n$ $\Gamma \vdash \mathfrak{A} \to \mathfrak{B}_k$.

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 $\mathfrak{B}_1 \to (\mathfrak{A} \to \mathfrak{B}_1)$ является аксиомой. Если $\mathfrak{B}_1$ — аксиома или содержится в Γ, то по правилу заключения $\mathfrak{A} \to \mathfrak{B}_1$ выводима из Γ. Если же $\mathfrak{B}_1 = \mathfrak{A}$, то из примера 6.7, а имеем $\mathfrak{A} \to \mathfrak{A}$, т. е. $\mathfrak{A} \to \mathfrak{B}_1$. В любом случае получаем $\Gamma \vdash \mathfrak{A} \to \mathfrak{B}_1$.

Предположим теперь, что $\Gamma \vdash \mathfrak{A} \to \mathfrak{B}_i$ 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 заключения; но тогда $\mathfrak{B}_l$ должно иметь вид $\mathfrak{B}_j \to \mathfrak{B}_k$. 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:

$$\Gamma \vdash \mathfrak{A} \to \mathfrak{B}_j \tag{6.1}$$

и $\Gamma \vdash \mathfrak{A} \to \mathfrak{B}_l$, т. е.

$$\Gamma \vdash \mathfrak{A} \to (\mathfrak{B}_j \to \mathfrak{B}_k). \tag{6.2}$$

Подставим в схему аксиом II 2 $\mathfrak{B}_j$ вместо $\mathfrak{B}$ и $\mathfrak{B}_k$ вместо $\mathfrak{C}$. Получим:

$$(\mathfrak{A} \to (\mathfrak{B}_j \to \mathfrak{B}_k)) \to ((\mathfrak{A} \to \mathfrak{B}_j) \to (\mathfrak{A} \to \mathfrak{B}_k)). \tag{6.3}$$

Применив правило заключения к (6.2) и (6.3), получим:

$$\Gamma \vdash (\mathfrak{A} \to \mathfrak{B}_j) \to (\mathfrak{A} \to \mathfrak{B}_k), \tag{6.4}$$

а применив то же правило к (6.1) и (6.4), имеем: $\Gamma \vdash \mathfrak{A} \to \mathfrak{B}_k$. Остается положить $k = n$. □

Отметим, что при построении выводов использовались только аксиомы II1 и II2, которые содержатся и в системе аксиом I. Поэтому приведенное доказательство теоремы дедукции справедливо и для исчисления высказываний, основанного на системе I.

Пример 6.2. а. В качестве первого применения теоремы дедукции покажем, что аксиома II3 выводима из системы аксиом I.

1. Подставим в I 9 $\neg A$ вместо A. Получим:

$$(\neg A \to B) \to ((\neg A \to \neg B) \to \neg\neg A).$$

2. Двойное применение правила заключения к шагу 1 дает:

$$\neg A \to B,\ \neg A \to \neg B \vdash \neg\neg A.$$

3. Так как из аксиомы I 10 следует по правилу заключения, что $\neg\neg A \vdash A$, то по транзитивности выводимости (см. замечание к примеру 6.1, в) получим $\neg A \to B;\ \neg A \to \neg B \vdash A$.

4. Переставим гипотезы в полученной выводимости (их порядок неважен, как видно из определения выводимости):

$$\neg A \to \neg B;\ \ \neg A \to B \vdash A.$$

5. Применив 2 раза к шагу 4 теорему дедукции, получим аксиому II 3:

$$(\neg A \to \neg B) \to ((\neg A \to B) \to A).$$

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 $\Gamma \vdash A \to B$ и $\Gamma \vdash A \to \neg B$. Из этих импликаций и аксиомы I 9 двойным применением правила заключения 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 заключения).

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 заключения 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 $F(A_1, \ldots, A_n)$ и распределение истинностей входящих в нее элементарных высказываний $A_1, \ldots, A_n$, то для выяснения истинности ее нужно вычислить методами, приведенными в гл. 3, как логическую функцию на наборе $(\sigma_1, \ldots, \sigma_n)$, где $\sigma_i = 1$, если A_i истинно, и $\sigma_i = 0$, если A_i ложно. (В этом смысле можно считать, что набор $(\sigma_1, \ldots, \sigma_n)$ задает распределение истинностей.) Если $F(\sigma_1, \ldots, \sigma_n) = 1$, то высказывание F истинно при дан-

ном распределении истинностей $A_1, \ldots, A_n$, если $F(\sigma_1, \ldots, \sigma_n) = 0$, 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 $A_1, \ldots, A_n$ задано распределение истинностей $(\sigma_1, \ldots, \sigma_n)$. 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 $\mathfrak{A}(A_1, \ldots, A_n)$ определяет логическую функцию f от n переменных. Тогда, если $f(\sigma_1, \ldots, \sigma_n) = \sigma$, то в исчислении высказываний $A_1^{\sigma_1}, \ldots, A_n^{\sigma_n} \vdash \mathfrak{A}^\sigma$.

Например, формула $\mathfrak{A}(A_1, A_2, A_3) = (A_1 \to A_2) \to A_3$ на наборе (0, 1, 0) равна нулю. Тогда теорема утверждает, что

$$\neg A_1, A_2, \neg A_3 \vdash \neg((A_1 \to A_2) \to A_3).$$

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 $\neg \mathfrak{B}$, $\mathfrak{B} \to \mathfrak{C}$, $\mathfrak{B} \& \mathfrak{C}$ и $\mathfrak{B} \vee \mathfrak{C}$. Перебор всех случаев опускаем; приведем лишь некоторые.

Пусть $\mathfrak{A} = \neg \mathfrak{B}$ и при заданном $(\sigma_1, \ldots, \sigma_n)$ $\mathfrak{B}$ ложно. По индуктивному предположению $A^{\sigma_1}, \ldots, A^{\sigma_n} \vdash \neg \mathfrak{B}$ и, следовательно, $A^{\sigma_1}, \ldots, A^{\sigma_n} \vdash \mathfrak{A}$, что и требовалось.

Пусть $\mathfrak{A} = \mathfrak{B} \to \mathfrak{C}$ и $\mathfrak{C}$ истинно. Тогда (в силу свойств импликации) $\mathfrak{A}$ тоже истинно и $\mathfrak{A}^\sigma = \mathfrak{A}$. По индуктивному предположению $A^{\sigma_1}, \ldots, A^{\sigma_n} \vdash \mathfrak{C}$. Но тогда по производному правилу из примера 6.1,б $A^{\sigma_1}, \ldots, A^{\sigma_n} \vdash \mathfrak{B} \to \mathfrak{C}$, т. е. $A^{\sigma_1}, \ldots, A_n^{\sigma_n} \vdash \mathfrak{A}$, что и требовалось.

Полный перебор всех случаев для системы аксиом II можно найти в [43]. □

Теоремы исчисления высказываний в терминах истинности характеризуется 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 заключения сохраняет тождественную истинность. Пусть формулы $\mathfrak{A}$ и $\mathfrak{A}\to\mathfrak{B}$ тождественно-истинны, т. е. $\mathfrak{A}\equiv 1$, $\mathfrak{A}\to\mathfrak{B}\equiv 1$.

Так как $A\to B\equiv \neg A\vee B$, то $0\vee\mathfrak{B}\equiv 1$ и $\mathfrak{B}\equiv 1$, т. е. формула $\mathfrak{B}$, выводимая из $\mathfrak{A}$ и $\mathfrak{A}\to\mathfrak{B}$ по правилу заключения, также тождественно-истинна.

Итак, аксиомы тождественно-истинны, а правила вывода сохраняют тождественную истинность; поэтому любая доказуемая формула тождественно-истинна. □

Справедлива и обратная теорема.

Теорема 6.4. Всякая тождественно-истинная формула является теоремой исчисления высказываний.

Пусть $\mathfrak{A}(A_1,\ldots,A_n)$ — тождественно-истинная формула. Тогда по теореме 6.2 $A_1^{\sigma_1},\ldots,A_n^{\sigma_n}\vdash\mathfrak{A}$ для всех 2^n наборов $(\sigma_1,\ldots,\sigma_n)$. Применив n раз теорему дедукции, получим 2^n выводимостей $\vdash A_1^{\sigma_1}\to(A_2^{\sigma_2}\to(\ldots\to(A_n^{\sigma_n}\to\mathfrak{A}))\ldots)$.

Подставим в аксиому I 8 $\neg A$ вместо B. Получим $(A\to\to C)\to((\neg A\to C)\to((A\vee\neg A)\to C))$. Из этой формулы и правила заключения (учитывая, что $A\vee\neg A$ — теорема: см. пример 6.2, в) получаем производное правило: если $\vdash A\to C$ и $\vdash\neg A\to C$, то $\vdash C$. Для любого набора $(\sigma_2,\ldots,\sigma_n)$ имеем при $\sigma_1=1$ $\vdash A_1\to(A_2^{\sigma_2}\to(\ldots\to(A_n^{\sigma_n}\to\mathfrak{A}))\ldots)$, а при $\sigma_1=0$ $\vdash\neg A_1\to(A_2^{\sigma_2}\to(\ldots\to(A_n^{\sigma_n}\to\mathfrak{A}))\ldots)$. Применяя к этим формулам только что доказанное правило, получаем $\vdash(A_2^{\sigma_2}\to\ldots\to(A_n^{\sigma_n}\to\mathfrak{A})\ldots)$, т. е. первая посылка «длинной импликации» удалена. Применяя это удаление еще $n-1$ раз, получаем $\vdash\mathfrak{A}$. □

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 $(f_1 \to f_2) \& (f_2 \to f_1)$. В силу теоремы 6.4 все такие эквивалентности, например $\overline{A \vee B} \sim \bar{A} \& \bar{B}$, 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* $x_1, x_2 \ldots$, *предметных констант* $a_1, a_2 \ldots$, *предикатных букв* $P_1^1, P_2^1, \ldots, P_k^j \ldots$ и *функциональных букв* $f_1^1, f_2^1, \ldots, f_k^j, \ldots$, а также знаков логических связок $\vee$, $\&$, $\neg$, $\to$, *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 $A_1, A_2, \ldots$, либо как 0-местные предикаты $P_1^0, P_2^0, \ldots$, 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 $t_1, \ldots, t_n$ — термы, то $f^n(t_1, \ldots, t_n)$ — терм.

2) Формулы:

а) если P^n — предикатная буква, а $t_1, \ldots, t_n$ — термы, то $P^n(t_1, \ldots, t_n)$ — формула; все вхождения предметных переменных в формулу вида $P^n(t_1, \ldots, t_n)$ называются свободными;

б) если F_1, F_2 — формулы, то формулами являются $\neg F_1$, $(F_1 \to F_2)$, $(F_1 \vee F_2)$, $(F_1 \& F_2)$; 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; вхождения остальных переменных в F остаются свободными.

Функциональные буквы и термы введены «впрок» для целей различных прикладных исчислений предикатов. Чистое исчисление предикатов строится для произвольной предметной области; структура этой области и связи между ее элементами не имеют значения, поэтому в нем фуикциональные буквы и термы не обязательны [1]. 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, определениых на области. Отношениям соответствуют предикатные буквы, функциям — фуикциональиые буквы. Термы — это имена элемеитов предметной области, построенные с помощью фуикций. Они могут быть постоянными (если оии построены из предметных коистаит) и переменными. Формулы — это высказывания о термах. Например, $4+5\cdot 3$ — постояиный терм любого исчисления, содержащего фуикциональные буквы $+$ и $\cdot$, а $x+7$ — переменный терм этого же исчисления. Выражение $4+5\cdot 3=x+7$ — это переменное высказывание, полученное подстаиовкой двух термов в двухместный предикат равенства; его истиниость зависит от значения переменной x.

3. Аксиомы исчисления предикатов делятся на две группы:

1) аксиомы исчисления высказываний (можно взять любую из систем I или II);

2) две следующие предикатные аксиомы:

P1. $\forall x F(x) \to F(y)$;

P2. $F(y) \to \exists x F(x)$.

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 $\forall x \exists y P(y, x) \to \exists y P(y, y)$. Если ее проинтерпретировать на множестве N натуральных чисел с предикатом P «быть больше», то получим высказывание: «если для всякого x найдется y больший, чем

[1] В частности, приводимые далее аксиомы P1 и P2 не учитывают наличия термов в формулах. Для исчисления с термами и функциональными буквами эти аксиомы имеют вид P1′ и P2′.

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 заключения (Modus Ponens) — то же, что и в исчислении высказываний;

2) правило обобщения ($\forall$ — введения):

$$\frac{F \to G(x)}{F \to \forall x G(x)},$$

где $G(x)$ содержит свободные вхождения x, а F их не содержит;

3) правило $\exists$ — введения:

$$\frac{G(x) \to F}{\exists x G(x) \to F}$$

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 $P(x) \to Q(x)$, очевидно, истинно для любого x, однако применение к нему правила обобщения дает высказывание $P(x) \to \forall x Q(x)$, не являющееся всегда истинным. Если же к $P(x) \to Q(x)$ применить правило $\exists$-введения, то получим $\exists x P(x) \to Q(x)$, из которого путем (уже корректного!) применения правила обобщения получим высказывание $\exists x P(x) \to \forall x Q(x)$, ложное на множестве натуральных чисел.

Приведенные здесь аксиомы и правила вывода содержатся в [36, 37]. Возможны и другие системы аксиом и правил (см., например, [39, 43]). В частности, в [43] последовательно проводится принцип минимизации числа логических операторов, который в исчислении высказываний оставляет лишь связки $\neg$ и $\to$ (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 $\exists x P(x)$ 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 чем шла речь в исчислении высказываний) выбрано второе, при ко-

тором правило подстановки отсутствует, а вместо аксиом рассматриваются схемы аксиом. Фактически этот выбор произошел уже тогда, когда аксиомы P1 и P2 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, [15]). 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. $F(x)\to(G\to F(x))$ (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. $G\to F(x)$ (правило заключения к шагам 1 и 2).
4. $G\to\forall xF(x)$ (правило обобщения к шагу 3).
5. $\forall xF(x)$ (правило заключения к G и шагу 4).
6. $F(y)$ (правило заключения к шагу 5 и аксиоме P1).

б. В исчислении предикатов из выводимости $\forall xF(x)$ следует выводимость $\forall yF(y)$, а из выводимости $\exists xF(x)$ — выводимость $\exists yF(y)$ 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. $\vdash \forall xF(x)$ (по предположению).

2. $\forall x F(x) \to F(y)$ (аксиома P1).
3. $\forall(x) F(x) \to \forall y F(y)$ (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. □

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 $(F \to G) \& (G \to F)$). На нем имеет смысл остановиться подробнее. Дело в том, что доказательства 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 $\vdash A \to C \sim B \to C$ и $C \to A \sim C \to B$;
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. Доказательство последних двух правил можно найти, например, в [25].

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. □

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. $\neg \exists x P_2(x, y) \sim \forall x \neg P_2(x, y)$ (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):

$$A \& \forall x F(x) \sim \forall x (A \& F(x)); \tag{6.5}$$

$$A \vee \exists x F(x) \sim \exists x (A \vee F(x)); \tag{6.6}$$

$$A \& \exists x F(x) \sim \exists x (A \& F(x)); \tag{6.7}$$

$$A \vee \forall x F(x) \sim \forall x (A \vee F(x)); \tag{6.8}$$

$$A \to \forall x F(x) \sim \forall x (A \to F(x)); \tag{6.9}$$

$$A \to \exists x F(x) \sim \exists x (A \to F(x)); \tag{6.10}$$

$$\forall x F(x) \to B \sim \exists x (F(x) \to B)); \tag{6.11}$$

$$\exists x F(x) \to B \sim \forall x (F(x) \to B)). \tag{6.12}$$

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 $Q_1x_1Q_2x_2...Q_nx_nF$, где Q_1, ..., 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)].

б. 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 исчислени-

ем 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:

$$\text{P1}'.\ \forall x F(x) \to F(t);$$

$$\text{P2}'.\ F(t) \to \exists x F(x),$$

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. $(x=y)\to(F(x, x)\to F(x, y))$ (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) $\vdash x=y\to y=x$;

3) $\vdash x=y\to(y=z\to x=z)$.

Доказательство.

1. Непосредственно следует из аксиом E1 и P1′ (где $F(x)$ имеет вид $x=x$) по правилу заключения.

2. Из E2 имеем $(x=y)\to(x=x\to y=x)$. Отсюда $x=y$, $x=x\vdash y=x$ (двойное применение правила заключения).

Но так как $\vdash x=x$, то $x=y\vdash y=x$ и, следовательно, по теореме дедукции[1] $x=y\to y=x$.

3. Поменяем местами в E2 x и y, в качестве $F(y, y)$ возьмем $y=z$, а в качестве $F(y, x)$ возьмем $x=z$. Получим $y=x\to(y=z\to x=z)$ и по правилу заключения $y=x\vdash(y=z\to x=z)$. Но так как в силу п. 2 $x=y\vdash y=x$, то по транзитивности выводимости (см. § 6.1) получаем $x=y\vdash(y=z\to x=z)$ и по теореме дедукции $x=y\to(y=z\to x=z)$. □

Три свойства равенства, полученные этой теоремой, верны, как известно, для любого отношения (и, следовательно, соответствующего предиката) эквивалентности (см. гл. 1). Схема аксиом E2 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 $<$:

NE1. $\forall x\neg(x<x)$;

NE2. $\forall x\forall y\forall z(x<y\to(y<z\to x<z))$.

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 P1′ и правилу заключения.

Если к NE1 и NE2 добавить аксиому с предметной константой NE3 $\forall x\neg(x<a)$, то получим теорию частичного порядка с минимальным элементом a.

Пожалуй, наиболее изученной формальной теорией, которая играет фундаментальную роль в основаниях математики, является формальная арифметика. Ее схемы аксиом (см. [36]):

A1. $F(0)\,\&\,\forall x(F(x)\to F(x'))\to F(x)$ (принцип индукции);

A2. $t_1'=t_2'\to t_1=t_2$;

A3. $\neg(t_1'=0)$;

---

[1] 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.

A4. $t_1 = t_2 \to (t_1 = t_3 \to t_2 = t_3)$;

A5. $t_1 = t_2 \to t_1' = t_2'$;

A6. $t_1 + 0 = t_1$;

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 функциональных символа $+$, $\cdot$, $'$, один индивидуальный предикат (предикатная буква) $=$ и одна предметная константа 0. Они придают схемам А2—А9 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 А6—А9 — это просто предикат равенства, в который подставлены термы определенного вида. Схема А1 имеет обычный вид: $F(x)$ — метаобозначение, употребляемое в том же смысле, в каком оно использовалось в чистом исчислении предикатов. Впрочем, схемы А2—А9 можно заменить конкретными аксиомами (т. е. формулами самой арифметики): А2′. $x_1' = x_2' \to x_1 = x_2$ и т. д., из которых любые формулы вида А2—А9 можно получить с помощью правила обобщения и схемы аксиом Р1.

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, А1), либо конкретными аксиомами, т. е. формулами самого исчисления (например, Е1). 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 Е1 и Е2).

2. При «подстановочном» подходе системы аксиом I (или II) и Р1, Р2 являются конечной системой формул самого исчисления. Буква F в аксиомах Р1 и Р2 — это преди-

катная 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 соответствия 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 соответствия 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 отобразятся 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*[1] 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 любая 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 $\forall x_1 \ldots \forall x_n F$, где $x_1, \ldots, x_n$ — свободные переменные формулы F.

---

[1] Некоторые из определяемых здесь понятий уже использовались в § 3.4.

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 mod 11), 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 существо 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 осмыслить 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 форму-

лировал 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 [39] — 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). □

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 [39]) — *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 назвать *аксиоматизируемой*, или

формализуемой, теорией. Из предыдущих результатов следует, что логика высказываний и логика предикатов аксиоматизируемы 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. □

Теорема 6.11 (теорема Черча). 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 [37]) 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. □

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, выражающая непротиворечивость T, недоказуема в T. □

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 исследования 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 единственно 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 не пригоден; 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[1] таких формальных систем были заложены Э. Постом. Эту 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.

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 диаграм-

[1] 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.

ма 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. Лю-

бой 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 — одноместные 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 словах. 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 любой, 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 воз-

можности 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 $R_i(\alpha_1, \ldots, \alpha_n, \beta)$ на множестве A^*, называемых правилами вывода[1]; слово β называется выводимым из $\alpha_1, \ldots, \alpha_n$ по правилу R_i. Понятия вывода, выводимости и доказательства — те же, что и для формальных теорий (см. § 6.1).

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 продукций Поста. Однако уже общего определения формальной системы оказывается достаточным, 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 $\alpha_1, \ldots, \alpha_n$, где α_i — слова в алфавите A. Множество A^{**}, очевидно, перечислимо. Ввиду разрешимости множества аксиом и вычислимости правил вывода по любой последовательности $\alpha_1, \ldots, \alpha_n$ можно эффективно узнать, является она выводом в FS или нет. Поэтому если в процессе перечисления A^{**} выбрасывать все последовательности $\alpha_1, \ldots, \alpha_n$, не являющиеся выводами, то получим пере-

[1] Аксиомы также можно задать правилом вывода (одноместным отношением): $R_i(\beta) \sim$ «β — аксиома».

числение множества выводов. Следовательно, множество последних слов выводов, т. е. слов, выводимых в FS, перечислимо. □

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 $\alpha_i \to \beta_i$, где α_i, β_i — слова, возможно пустые, в A. Подстановка $\alpha_i \to \beta_i$ 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 $\alpha \models \varepsilon_1 \models \varepsilon_2 \models \ldots \models \beta$, 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 никакой 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 $q_z \to \lambda$, 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. □

Ассоциативное исчисление, или *система Туэ*, — это формальная система, определяемая алфавитом A и конечным множеством соотношений $\alpha_i \leftrightarrow \beta_i$, каждое из которых понимается как пара подстановок $\alpha_i \to \beta_i$ (левая подстановка) и $\beta_i \to \alpha_i$ (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 $A_S = A_T \cup \{q_1, \ldots, q_z\}$. Системе команд машины T соответствует система соотношений в $S(T)$ (слева — команды, справа — соотношения):

$$q_i a_j \to q_k a_l R; \quad q_i a_j \longleftrightarrow a_l q_k; \tag{6.13}$$

$$q_i a_j \to q_k a_l L; \quad a_t q_i a_j \longleftrightarrow q_k a_t a_l \text{ для всех } a_t \in A_T \tag{6.14}$$

(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 $\gamma q_z a_k \delta$.

Однако 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 $\alpha q_i a_j \beta \vdash \varepsilon_1 \vdash \ldots \vdash \gamma q_z a_k \delta$. 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 единственно, то любая подстановка $\alpha_i \to \beta_i$ 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 $\alpha q_i a_j \beta \vdash \ldots \vdash \varepsilon_{l-1} \vdash \varepsilon_{l+2} \vdash \ldots \gamma q_z a_k \delta$, 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 $\alpha q_i a_j \beta \to \ldots \to \gamma q_z a_k \delta$, содержащий только левые подстановки, т. е. в точности воспроизводящий последовательность конфигураций машины T. □.

Теорема 6.17 (теорема Маркова—Поста). 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 кон-

фигурациями 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. □

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:

$$ac \longleftrightarrow ca; \quad ad \longleftrightarrow da; \quad bc \longleftrightarrow cb; \quad bd \longleftrightarrow db;$$
$$abac \longleftrightarrow abace; \quad eca \longleftrightarrow ae; \quad edb \longleftrightarrow be.$$

Теорема 6.17′ явилась исторически первым примером доказательства алгоритмической неразрешимости проблемы, не связанной с логикой и основаниями математики.

Системы продукций Поста (канонические системы). *Каноническая система* определяется собственным алфавитом $A=\{a_1, ..., a_m\}$ (алфавитом констант), алфавитом $X=\{x_1, ..., x_n\}$ переменных, конечным множеством аксиом и конечным множеством правил вывода, имеющих вид $\gamma_1, ..., \gamma_k \Rightarrow \delta$ и называемых *продукциями* ($\gamma_1, ..., \gamma_k$, как обычно, называются посылками, δ — следствием). Аксиомы, а также посылки и следствия продукций — 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 слов $\alpha_1, ..., \alpha_n$ применением продукции P с n посылками, если найдется такая подстановка слов вместо переменных P, которая посылки P превратит в слова $\alpha_1, ..., \alpha_n$, 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 $x \to x||$. Если эту продукцию заменить продукцией $x \to xx$, то получим каноническую систему, порождающую степени двойки (в унарном представлении): $|$, $||$, $||||$, ...

б. Множество всех формул исчисления высказываний порождается канонической системой с алфавитами $\{a_1, ..., a_n, \vee, \&, \neg, \to, (,)\}$, $\{x_1, x_2\}$, аксиомами $a_1, ..., a_n$ и про-

дукциями

$$\left.\begin{array}{l} x_1,\ x_2 \Rightarrow (x_1 \vee x_2); \\ x_1,\ x_2 \Rightarrow (x_1 \,\&\, x_2); \\ x_1,\ x_2 \Rightarrow (x_1 \to x_2); \\ x_1 \Rightarrow \neg x_1. \end{array}\right\} \qquad (6.15)$$

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..., 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 (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 $A=\{a, 0, 1, 2, ..., 9, \vee, \&, \neg, \rightarrow, (,)\}$, вспомогательные символы $\{i, f\}$, аксиому ai и 16 продукций:

$$x_1 i \Rightarrow x_1 0i;$$
$$x_1 i \Rightarrow x_1 1i;$$
$$\cdots\cdots\cdots$$
$$x_1 i \Rightarrow x_1 9i;$$
$$x_1 i \Rightarrow x_1 f;$$
$$x_1 f,\ x_2 f \Rightarrow (x_1 \vee x_2) f;$$
$$x_1 f,\ x_2 f \Rightarrow (x_1 \& x_2) f;$$
$$x_1 f,\ x_2 f \Rightarrow (x_1 \rightarrow x_2) f;$$
$$x_1 f \Rightarrow \neg x_1 f;$$
$$x_1 f \Rightarrow x_1.$$

Первые десять продукций порождают идентификаторы; следующие пять продукций формализуют индуктивное определение формулы (из этих пяти четыре последние отличаются от продукций (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, которые за недостатком места в данной книге не рассматриваются. Более подробно об использовании формальных систем такого рода для описания алгоритмических языков говорится в [34], о системах Смальяна — в [46].

Связь канонических систем с системами подстановок

довольно ясна, по крайней мере в одну сторону; очевидно, что всякой подстановке $\alpha_i \to \beta_i$ 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 перечислимо) 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 уж 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 $A_T = \{a_1, \ldots, a_n\}$, первые m букв которого образуют алфавит исходных данных ($m \leqslant n$), множеством состояний $Q = \{q_1, \ldots, q_z\}$ 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 продукции аналогично тому, как это делалось при доказательстве теорем 6.15 и 6.16: например, команде $q_i a_j \to q_k a_l R$ 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 $m+2$ продукции:

$$
\begin{aligned}
& x_1 * \Rightarrow x_1 a_1 *\ ; \\
& \quad\vdots \\
& x_1 * \Rightarrow x_1 a_m *\ ; \\
& x_1 * \Rightarrow q_1 x_1 ; \\
& q_z x_1 \Rightarrow x_1 .
\end{aligned}
$$

Первые m продукций порождают из аксиомы все множество слов в алфавите исходных данных с маркером в конце: $(m+1)$-я продукция (стартовая) порождает из лю-

бого исходного слова начальную конфигурацию; после этого работают продукции, соответствующие командам машины T; наконец, в заключительных конфигурациях (и только в них!) символ состояния выбрасывается и получаются искомые теоремы в алфавите A. □

Итак, в терминах канонических систем можно описать любые перечислимые множества. Правда, используемое определение канонической системы (в частности, вид ее продукций) 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. придание им более простого, «нормального» вида.

Каноническая система называется *нормальной*, если она имеет одну аксиому, а все ее продукции имеют вид $\alpha x \to x\beta$. 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 NS по CS, довольно сложно. Его можно найти, например, в [44], 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 $A=\{a_1, ..., a_n\}$ и S содержит k подстановок $\alpha_i \to \beta_i$. Определим NS как систему с расширенным алфавитом $A'=\{a_1, ..., a_n, a_1', ..., a_n'\}$ и $k+2n$ продукциями:

$$\alpha_i x \Rightarrow x\beta_i' \quad (i = 1, \ \dots \ , k); \tag{6.16}$$

$$a_j (x) \Rightarrow x a_j'; \tag{6.17}$$

$$a_j' (x) \Rightarrow x a_j, \tag{6.18}$$

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 $\alpha_i \to \beta_i$: $\gamma=\gamma_1\alpha_i\gamma_2$ и в S $\gamma_1\alpha_i\gamma_2 \vdash \gamma_1\beta_i\gamma_2$. Но тогда в NS $\gamma_1\alpha_i\gamma_2 \vdash \alpha_i\gamma_2\gamma_1' \vdash \gamma_2\gamma_1'\beta_i' \vdash \gamma_1'\beta_i'\gamma_2' \vdash \vdash \gamma_1\beta_i\gamma_2$. 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 $\gamma \vdash \varepsilon_{i_1} \vdash \varepsilon_{i_2} \vdash \ldots \vdash \delta$, 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 $\gamma_2\gamma_1'$, $\gamma_2'\gamma_1$, γ', 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 продукций (6.16). Для любого вывода, содержащего между словами из A несколько применений продукций (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 $\alpha_1 x \Rightarrow x\beta_1$ и $\alpha_2 x \Rightarrow x\beta_2$, 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 $\gamma=\gamma_1\alpha\gamma_2$, применению продукции (6.16) предшествовало несколько (быть может, ни одного) применений (6.17) и место вывода, где применена продукция (6.16), имеет вид $\alpha\gamma_2\gamma_1' \Rightarrow \Rightarrow \gamma_2\gamma_1'\beta'$. Остаток $\gamma_2\gamma_1'\beta_1' \vdash \varepsilon_{i_1}$ рассматриваемого отрезка не содержит применений (6.16), поэтому в силу свойств продукций (6.17), (6.18) $\varepsilon_{i_1}=\gamma_1\beta\gamma_2$. Но тогда ε_{i_1} получается из γ подстановкой $\alpha \to \beta$, 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 (α_1, β_1), ..., (α_m, β_m) пар слов в алфавите A. Поставим следующую проблему: существует ли последовательность $i_1, i_2, \ldots, i_N$ индексов, такая, что $\alpha_{i_1}\alpha_{i_2}\ldots\alpha_{i_N}=\beta_{i_1}\beta_{i_2}\ldots\beta_{i_N}$?

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[1], 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 продукций, 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 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 [41] — это упорядоченная система подстановок с двумя дополнительными соглашениями: 1) i-я подстановка может быть применена, только если неприменимы $1, \ldots, (i-1)$-я подстановки; 2) если подстановка $\alpha \to \beta$ применима к слову γ, то β подставляет-

[1] Сжатое и блестяще написанное изложение этой концепции имеется в книге Мартин-Лефа [42].

ся 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.

ГЛАВА СЕДЬМАЯ

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. Витгенштейном, Р. Карнапом и другими в 20—30-е гг., 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 «осмысленными» допустимые последовательности знаков. Знаками могут быть буквы алфавита, математические обозначения, звуки, движения брачного танца у животных, ритуальные действия в обрядах различных народов. Наука об осмысленных знаковых системах называется семиотикой. Семиотический подход оказывается весьма плодотворным в раз-

личных 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 сильно развившиеся в последние 30 лет 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 множество 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 разговор в теоретическом 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; $V \cap W = \varnothing$; I — начальный символ (аксиома) грамматики; P — конечное множество правил вида $\xi \to \eta$, 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 $\xi \to \eta$ — правило G. Цепочка β выводима из α, если существует последовательность $\varepsilon_0 = \alpha, \varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n = \beta$, такая, что для всех $i = 0, \ldots, n-1$ $\varepsilon_i \Rightarrow \varepsilon_{i+1}$. 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, порождаемое 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 $A \to a$ ($a \in V$, A — двойник $\cdot a$), а R'' получено из R заменой в каждом правиле всех вхождений терминальных символов вхождениями их двойников. Каждому выводу $I, \varepsilon_1, \ldots, \varepsilon_n$ в G соответствует вывод $I, \varepsilon'_1, \ldots, \varepsilon'_n, \varepsilon'_{n+1}, \ldots, \varepsilon'_{n+m}$ в G', где $\varepsilon'_i (i \leqslant n)$ получено из ε_i заменой всех символов из V их двойниками, а $\varepsilon'_{n+j} (j \leqslant m)$ 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 $L(G) \subseteq L(G_1)$. Поскольку 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, $L(G) = L(G_1)$. □

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 $M=L(G)$. □

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 грамматика — это грамматика, все правила которой имеют вид $\alpha A\beta\to\alpha\omega\beta$, 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, — грамматика, все правила которой имеют вид $A\to\alpha$, где $\alpha\in(V\cup W)^*$.

Грамматика типа 3, или регулярная грамматика, — грамматика, все правила которой имеют вид либо $A\to aB$, либо $A\to a$.

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, $aaaaa$ ..., т. е. язык $\{a^{2n-1}\}$. Он порождается грамматикой $G=\langle V, W, R, I\rangle$, где $V=\{a\}$, $W=\{I, a\}$, а R содержит правила:

$$I\to a;\quad I\to aaI.$$

Из вида правил непосредственно следует, что язык имеет тип 3.

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:

$$I \to alb \text{ и } I \to ab.$$

в. Язык булевых формул с переменными a, b, c порождается КС-грамматикой, где $V=\{a, b, c, \vee, \&, \neg, (,)\}$; $W=\{I\}$; R содержит правила:

$$I \to (I \vee I);$$
$$I \to (I \,\&\, I);$$
$$I \to \neg I;$$
$$I \to a;$$
$$I \to b;$$
$$I \to c.$$

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 более сложной КС-грамматикой:

$$I \to T;$$
$$I \to I + T;$$
$$I \to I - T;$$
$$T \to M;$$
$$T \to T * M;$$
$$T \to T/M;$$

$$M \to (I);$$
$$M \to K;$$
$$K \to a;$$
$$K \to b;$$
$$K \to c.$$

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:

$$I \to aBa;$$
$$B \to aBCa;$$
$$B \to b;$$
$$bC \to BB;$$
$$aC \to Ca.$$

Эта грамматика не является контекстной (из-за последних двух правил), однако можно убедиться, что ей эквивалентна следующая контекстная грамматика:

$$I \to ABA;$$
$$B \to ABCA;$$
$$B \to b;$$
$$bC \to bb;$$
$$AC \to DC;$$
$$DC \to DA;$$
$$DA \to CA;$$
$$A \to a,$$

в которой неконтекстное правило $aC \to Ca$ 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, ... $\beta, \gamma, ..., \beta, ..., \alpha$, т. е. некоторая цепочка повторяется, то, удалив из вывода последовательность γ, ..., 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: $r(n)\leqslant\sum_{i=1}^{n}|V\cup W|^i$, 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 $\leqslant n$, 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).

В действительности справедливо более сильное утверждение — языки, порождаемые неукорачивающими грамматиками, примитивно рекурсивны.

Очевидно, что все контекстные грамматики являются не-

укорачивающими. 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 $A \to \alpha$ 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 $X::=Y_1|...|Y_n$, где X — некоторый текст, заключенный в угловые скобки и называемый металингвистической переменной, а Y_1, ..., 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; металингвистическая 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 АЛГОЛ-60— 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 →, а без связки | можно обойтись, если формулу $X::=Y_1|...|Y_n$ заменить системой формул $X::=Y_1$, ..., $X::=Y_n$. Если указанное преобразование провести для последнего примера, заменив ⟨ арифметическое выражение ⟩ на символ I, ⟨ терм ⟩ — на T, ⟨ 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 последних 20 лет. 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 верши-

ны — 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: $A \to s_{j_1} s_{j_2} \ldots s_{j_{l(i)}}$, 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_{j_1}, s_{j_2}, ..., $s_{j_{l(i)}}$, 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 $\Delta = I$, α_1, α_2, ..., α_n в грамматике G *дерево вывода* определяется индуктивно. Отрезку вывода $\Delta_1 = I$, α_1 соответствует правило $I \to \alpha_1$ 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 $\Delta_i = I$, α_1, ..., α_i, и его кроной является α_i. Дерево $t(\Delta_{i+1})$ для отрезка $\Delta_{i+1} = I$, α_1, ..., α_i, α_{i+1} определяется так. Если α_{i+1} получается из α_i применением правила $r_j : A \to \beta$ 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 кроны 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 $\Delta=I$, α_1, ..., α_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, α_1, ..., α_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).

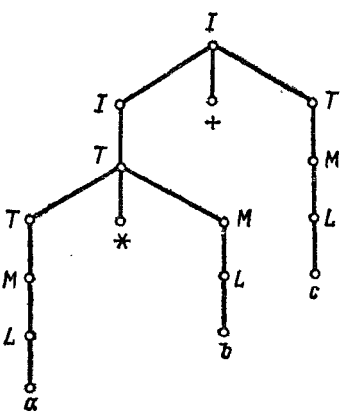

Рис. 7.1

Компактность — не единственное достоинство дерева вывода. Не менее важно то, что дерево вывода сохраняется при некоторых изменениях вывода, что позволяет их считать несущественными. Таким несущественным изменением вывода является изменение порядка применения правил к различным вхождениям нетерминальных символов цепочки. Более точно, если в выводе $\Delta=I$, α_1,.., α_i, α_{i+1}, α_{i+2}, ...,α_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. $\Delta'=I$, α_1, ..., α_i, α_{i+2}, α_{i+1},

α_{i+3}, ..., α_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 имеющих 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 существует взаимно однозначное соответствие. Очевидно, что если левосторонние выводы равны, то равны и деревья. Пусть $\Delta = I$, α_1, α_2... и $\Delta' = I$, β_1, β_2... 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 кроной α_{i-1} будут приклеены 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 $L(G) = \{a^{2^{n-1}}\}$, однозначна. Действительно, в этой грамматике любой вывод имеет вид I, aaI, ..., $a^{2^n}I$..., т. е. в нем $n-1$ раз применяется второе 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 эк-

вивалентных выводов, т. е. различных выводов, имеющих одинаковое дерево). Однако этот же язык можно задать неоднозначной грамматикой $I \to a$, $I \to IaI$, в которой, например, цепочка a^7 имеет несколько различных деревьев (например, рис. 7.2, *а* и *б*) и соответственно несколько левосторонних выводов.

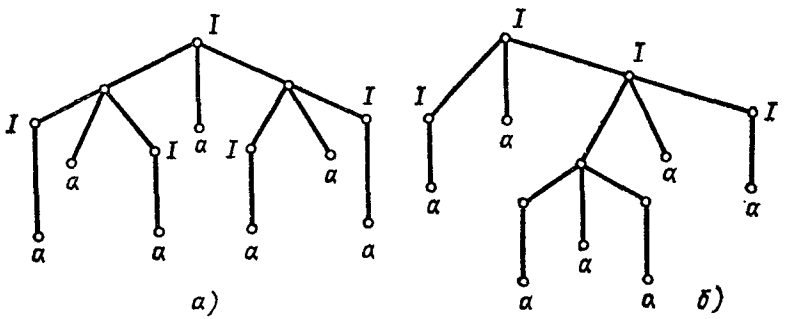

Рис. 7.2

б. Рассмотрим грамматику, полученную из грамматики примера 7.1, г отождествлением всех нетерминальных символов и удалением всех получившихся тривиальных правил $I \to I$:

$$I \to I + I \mid I - I \mid I * I \mid I/I \mid (I) \mid a \mid b \mid c.$$

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 [32].

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 различных 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 вычисле-

ний. 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 умень-

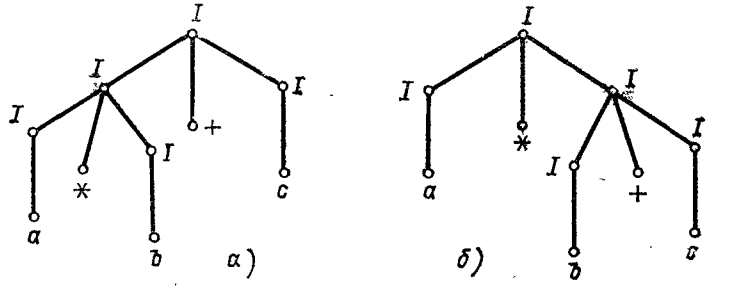

Рис. 7.3

шить число нетерминальных символов и тем самым упростить синтаксическую структуру формулы, определяемую ее деревом. Именно поэтому языки формул исчисления высказываний 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 дости-

жимый 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 $I \to \alpha$. 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 $A \to \alpha$, где $A \in M_i$. Алгоритм останавливается на шаге k, если $M_k = M_{k-1}$; множество достижимых символов совпадает с M_k, причем $k \leqslant |W|$. 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 $A \to \alpha$, 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 $A \to \alpha$, 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 $l \leqslant |W|$.

Теперь приведенная грамматика $G' = \langle V', W', R', I \rangle$, эквивалентная предыдущей, определяется так: W' — это множество существенных символов G': $W' = M_k \cap Q_l$; R' содержит только те правила из R, в которых нет бесполезных символов; V' содержит только такие терминальные 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 $A \to B$, 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 $A \in W$ находит-

ся множество W_A всех нетерминальных символов E, таких, что $A \Rightarrow^* E$; ясно, что $E \in W_A$, если имеется последовательность правил $A \to B$, $B \to C$, ..., $\to E$, т. е. отношение $A \Rightarrow^* E$ является транзитивным замыканием (см. § 1.3) отношения $A \to E$. На втором этапе определяется множество правил R': если $B \to \alpha$ содержится в R и не является цепным, то для любого A, такого, что $B \in W_A$, включаем в R' правило $A \to \alpha$. Полагаем $G' = \langle V, W, R', I \rangle$.

Пример 7.5. Применение описанного алгоритма к грамматике из примера 7.1, г дает грамматику

$$I \to I + T \mid I - T \mid T * M \mid T/M \mid (I) \mid a \mid b \mid c;$$
$$T \to T * M \mid T/M \mid (I) \mid a \mid b \mid c;$$
$$M \to (I) \mid a \mid b \mid c.$$

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 $T \to M$, $M \to L$ и $I \to T$. 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 устранения «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 $\Delta_0 = I$, $\alpha_1, \ldots, \alpha_m$ — 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 $\alpha_k, \ldots, \alpha_l$ применены цепные правила, а α_{l+1} получено из α_l правилом $C \to \gamma$.

Рассмотрим теперь последовательность $\Delta_1 = I$, $\alpha_1, \ldots, \alpha_k$, α_{l+1}, ..., α_m. В ней переход от α_k к α_{l+1} осуществляется применением правила $C \to \gamma$, 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

$\Delta_n = I, \ldots, \alpha_m$, которая не содержит цепных правил; в ней все переходы к следующей цепочке осуществляются либо нецепными правилами G, либо новыми правилами вида $C \to \gamma$, 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 $\alpha \in L(G)$

$$c_1 |\alpha| \leqslant |\Delta(\alpha)| \leqslant c_2 |\alpha|,$$

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 укорачивающих 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 $|\Delta(\alpha)| \leqslant k_0 |\alpha|$. С другой стороны, $|\alpha| \leqslant k_1 |\Delta(\alpha)|$. Отсюда $|\alpha|/k_1 \leqslant |\Delta(\alpha)|$.

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, порождаемого данной грамматикой типа 1, неразрешимы. Для языков типа 2 эти

две 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 $\alpha=\beta\gamma\omega\delta\varepsilon$, где γ непусто либо δ непусто, $|\gamma\omega\delta|<q$, причем все цепочки вида $\beta\gamma^n\omega\delta^n\varepsilon$ 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 $A_{i_1},\ldots,A_{i_{k+1}}$, в которой хотя бы один символ должен повториться: $A_{i_r}=A_{i_s}=B$, $r<s\leqslant k+1$. 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 $\alpha=\beta\gamma\omega\delta\varepsilon$. 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 $n=1, 2\ldots$ могут быть получены выводы $I\Rightarrow^*\beta B\varepsilon\Rightarrow^*\beta\gamma^n\omega\delta^n\varepsilon$ и, следовательно, $\beta\gamma^n\omega\delta^n\varepsilon\in L(G')=L(G)$. □

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' воз-

можны 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 повториться. 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 $\beta\gamma\omega\delta\varepsilon$ КС-язык обязательно содержит и все слова вида $\beta\gamma^n\omega\delta^n\varepsilon$. 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 $X=(\alpha_1,\ldots,\alpha_m)$ и $Y=(\beta_1,\ldots,\beta_m)$ цепочек в алфавите U решить, существует ли последовательность индексов $i_1,\ldots,i_N$, такая, что $\alpha_{i_1}\ldots\alpha_{i_N}=\beta_{i_1}\ldots\beta_{i_N}$.

Пусть алфавит U и число m зафиксированы. Введем алфавит $U_0=\{b_1,\ldots,b_m\}$, не пересекающийся с U, и обозначим $U_1=U\cup U_0$. Для списка $X=(\alpha_1,\ldots,\alpha_m)$ m цепочек в алфавите U обозначим через $Q(X)$ множество всех цепочек вида $b_{i_N}\ldots b_{i_1}\alpha_{i_1}\ldots\alpha_{i_N}$ для любых последовательностей индексов $i_1,\ldots,i_N$, $N\geqslant 1$. Для языков $Q(X)$ справедливы следующие два утверждения.

1. Если $X=(\alpha_1,\ldots,\alpha_m)$ и $Y=(\beta_1,\ldots,\beta_m)$ — 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 $\gamma=b_{i_N}\ldots b_{i_1}\alpha_{i_1}\ldots\alpha_{i_N}=b_{i_N}\ldots b_{i_1}\ldots\beta_{i_1}\ldots\beta_{i_N}$ и, следовательно, $\alpha_{i_1}\ldots\alpha_{i_N}=\beta_{i_1}\ldots\beta_{i_N}$.

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 $X=(\alpha_1,\ldots,\alpha_m)$, задается системой $2m$ правил $I\to b_iI\alpha_i$, $I\to b_i\alpha_i$ $(i=1,\ldots,m)$. 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. □

Теорема 7.6. Не существует алгоритма, который для произвольной КС-грамматики позволял бы узнать, является ли она однозначной или нет.

Язык $Q(X)$, где $X=(\alpha_1,\ldots,\alpha_m)$, порождается однозначной грамматикой с системой правил R_X: $I\to A$; $A\to b_i\alpha_i$; $A\to b_iA\alpha_i$. Язык $Q(Y)$, где $Y=(\beta_1,\ldots,\beta_m)$, порождается однозначной грамматикой R_Y: $I\to B$; $B\to b_i\beta_i$; $B\to b_iB\beta_i$. 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.

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 $\overline{L}=V^*\setminus L$), 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.

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 $\varnothing$ 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, неприятностей 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 [57].

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 смысле 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 $L_0(a \to L_1)$ языка L_1

в L_0 вместо символа a — это операция, сопоставляющая языкам L_0 и L_1 язык $L=L_0(a\to L_1)$, 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 $L=L_0(a_1\to L_1,\ldots,a_k\to\to L_k)$ нескольких языков $L_1,\ldots,L_k$ в язык L_0, при которой вместо каждого вхождения a подставляется некоторая цепочка соответствующего языка $L_i(i=1,\ldots k)$.

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 $L=L_0(a\to L_1,\ b\to L_1,\ c\to L_1)$. 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 мультипликативной 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: $[L]_a^1=L$, $[L]_a^2=L(a\to L)$, $[L]_a^3=[L_a]^2(a\to L)$ и т. д., а также вместо множества символов $Y=\{a_1,\ldots,a_k\}$: $[L]_Y^2=L(a_1\to L,\ldots,a_k\to\to L)$.

Итерация L^* языка L определяется равенством $L^*=\{e\}\cup L_1\cup\ldots\cup L^n\cup=\{e\}\cup\bigcup\limits_{i=1}^{\infty}L^i$, где e — пустая цепочка.

Важное замечание: не следует смешивать язык $\{e\}$, состоящий из одной пустой цепочки e, с пустым языком ϕ, не содержащим ни одной цепочки. Например, $L\{e\}=\{e\}L=L$; $L\phi=\phi L=\phi$ 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 $L_1L_2=L_{01}(a\to L_1,\ b\to\to L_2)$, $L_1^*=L_{02}(a\to L_1)$, $L_1\cup L_2=L_{03}(a\to L_1,\ b\to L_2)$. Нетруд-

но 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 $\{a_1,..., a_n\}$.

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 $\bigcup_{i=1} [L_0]_a^i$, в которые не входит символ a. Обобщением этой операции является зацикливание $[L_0]_Y^*$ по множеству символов Y, определение которого получается из предыдущего заменой $[L_0]_a^i$ на $[L_0]_Y^i$.

Пример 7.7. Рассмотрим язык сумм $L_S=\{a, (a+a), (a+a+a)...\}$ и язык произведений (термов) $L_T=\{a, aa, aaa...\}$. Подстановка $L_S(a\to L_T)$ дает язык L_{ST} сумм произведений, т. е. язык многочленов, который содержит выражения глубины $\leqslant 2$ (о глубине формул см. § 3.1) над переменной a. Язык $[L_{ST}]_a^n$ — 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, поскольку для любого n любая цепочка $[L_{ST}]_a^n$ 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 $L_{S'}=\{a, b, (a+a), (a+b)...\}$, $L_{T'}=\{a, b, aa, ab...\}$, то $[L_{S'T'}]_a^n$ — это язык выражений глубины $\leqslant 2n$ над переменными a, b, а $[L_{S'T'}]_a^*$ — 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 $A\to\alpha_1|\alpha_2|...|\alpha_k$ — множество всех правил КС-грамматики G с левой частью A, то при порождении $L(G)$ происходит подстановка конечного языка $\{\alpha_1,..., \alpha_k\}$ 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 $L=L_1(a\to L_2)$ порождается грамматикой $G=\langle V, W, R, I\rangle$, которая определяется через G_1 и G_2 следующим образом: $V=(V_1\cup V_2)\setminus$

$\setminus\{a\}$; $W=W_1\cup W_2\cup\{a\}$; $I=I_1$; $R=R_1\cup R_2\cup\{a\to I_2\}$. Иначе говоря, символ a становится нетерминальным, и грамматика G_2 «подставляется» в грамматику G_1 посредством КС-правила $a\to I_2$. Правда, такое определение подстановки грамматики в грамматику корректно только при выполнении условий: 1) $a\notin(V_2\cup W_2)$; 2) $W_1\cap W_2=\varnothing$. Действительно, $a\in V_2$ приводит к противоречию: по определению подстановки языков в этом случае $a\in V$, тогда как по приведенному только что определению грамматики G, порождающей $L=L_1(a\to L_2)$, $a\notin V$; если $W_1\cap W_2\neq\varnothing$, то включение в R правила $a\to I_2$ 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 $a\to I_2$, но и некоторые правила из R_2 с a в левой части, в результате чего может оказаться, что $L(G)\neq L_1(a\to L_2)$. Кроме того, очевидна необходимость условий: 3) $V_1\cap W_2=\varnothing$; 4) $V_2\cap W_1=\varnothing$, 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 необходимо переименовать некоторые символы из V_1 и W_1, т. е. заменить их другими символами 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 $V=(V_1'\cup V_2)\setminus\{a\}$; $W=W_1'\cup W_2\cup\{a\}$; $I=I_1$; $R=R_1'\cup R_2\cup\{a'\to I_2\}$, 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. Нетрудно убедиться, что независимо от того, производится переименование или нет, $L(G)=L(G_1)(a\to L(G_2))$.

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 относительно конкатенации и итерации (поскольку выше была показана представимость этих операций 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: $V = V_1 \setminus \{a\}$; $W = W_1 \cup \{a\}$; $I = I_1$; $R = R_1 \cup \{a \to I_1\}$. 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 $L_1(a \to L_2)$ — 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 $I \Rightarrow \alpha a \beta$, $|\alpha a \beta| \geqslant 2$, 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 $R_S = \{I_1 \to a \mid (A+A), A \to a \mid A+A\}$, и G_T с системой правил $R_T = \{I_2 \to a \mid I_2 a\}$. Подставим G_T в G_S вместо a. Поскольку $a \in V_T$ (т. е. нарушено условие 1), то заменяем в G_S a на B, после чего получаем искомую грамматику G_{ST} с начальным символом I_1 и системой правил $\{I_1 \to B \mid (A+A), A \to B \mid A+A, B \to I_2, I \to a \mid I_2 a\}$, которая порождает язык $L_{ST} = L_S(a \to L_T)$ из примера 7.7.

Устраним теперь из G_{ST} цепное правило $B \to I_2$ и введем в язык L_{ST} вторую переменную b (терминальный символ), добавив правила $I_2 \to b \mid I_2 b$. Получим грамматику G'_{ST} с правилами $\{I_1 \to I_2 \mid (A+A), \ A \to I_2 \mid A+A, \ I_2 \to a \mid b \mid I_2 a \mid I_2 b\}$, порождающую язык $\widetilde{L}_{ST}$ арифметических многочленов (без вычитания и деления) над двумя переменными. Зацикливание грамматики $\widetilde{G}_{ST}$ по символу a дает язык $[\widetilde{L}_{ST}]_a^*$ выражений произвольной глубины над переменной b. Его грамматика, полученная зацикливанием $\widetilde{G}_{ST}$, имеет следующие правила (проведенное здесь переименование a в C необязательно и сделано только для того, чтобы сохранить традиционные обозначения нетерминалов):

$$I_1 \to I_2 \mid (A + A); \quad I_2 \to C \mid b \mid I_2 C \mid I_2 b; \quad A \to I_2 \mid A + A; \quad C \to I_1.$$

Блочные грамматики и сети из языков. Идея использования подстановки языка в правилах грамматики вместо подстановки конечной цепочки, как это делается в прави-

лах обычной КС-грамматики, приводит к понятию *блочной грамматики,* которая определяется как пятерка объектов $G=\langle \mathscr{A}, \mathscr{X}, \mathscr{L}, R, L^0 \rangle$, где $\mathscr{A}=\{A_1, ..., A_m\}$ — система конечных (возможно, пересекающихся) алфавитов; $\mathscr{X}=\{X_1, ..., X_m\}$ — система конечных алфавитов, таких, что $X_1 \subseteq A_1, ..., X_m \subseteq A_m$ и для любых i, j $X_i \cap (A_j \setminus X_j) = \Phi$; $\mathscr{L}=\{L_1, ..., L_m\}$ — конечное множество языков в алфавитах $A_1, ..., A_m$ соответственно; R — конечное множество блочных правил вида $x \to L_j$, где $x \in X$; $X=\bigcup_{k=1}^{m} X_k$; $L_j \in \mathscr{L}$; $L^0 \in \mathscr{L}$ — начальный язык. Алфавит X_i называется входным алфавитом языка L_i $(i=1, ..., m)$; 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 $x \to L_j$ означает подстановку вместо некоторого вхождения x любого слова из L_j. Таким образом, длина блочного вывода не учитывает сложности вывода «внутри» языков $L_1, ..., L_m$. *Язык $L(G)$, порождаемый блочной грамматикой G,* — это множество всех слов в терминальном алфавите G, выводимых в G из L^0. Языки $L_i \in \mathscr{L}$ 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 $\widetilde{W}=\left(\bigcup_{i=1}^{m}\widetilde{W}_i\right)\cup X$; 3) в правилах R_i символы из W_i заменяем символами из $\widetilde{W}_i$; полученную систему правил обозначим R_i. Каждому блочному правилу $x\to L_j$ из R ставим в соответствие обычное правило $x\to I_j$; множество таких правил обозначим R'; полагаем $\widetilde{R}=\left(\bigcup_{i=1}^{m}\widetilde{R}_i\right)\cup\widetilde{R}'$; 4) $\widetilde{I}$ совпадает с начальным символом грамматики G^0 начального языка. Доказательство того, что $L(\widetilde{G})=L(G)$, 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 $x\to L_j$ — 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 приписанным выходному элементу сети. Формально *сеть из языков (Л-сеть)* — это пятерка объектов $\lambda=\langle\mathscr{A},\mathscr{X},\mathscr{L},\gamma,v^0\rangle$, где $\mathscr{A}=\{A_1,\ldots,A_m\}$ — система конечных алфавитов; $\mathscr{X}=\{X_1,\ldots,X_m\}$ — система конечных алфавитов, таких, что $X_1\subseteq A_1,\ldots,X_m\subseteq A_m$, и для любых i,j $X_i\cap(A_j\setminus X_j)\varnothing$; $\mathscr{L}=\{L_1,\ldots,L_m\}$ — конечное множество языков в алфавитах $A_1,\ldots,A_m$ соответственно; γ — связный ориентированный граф с множеством вершин $V=\{v_1,\ldots,v_n\}$; $v^0\in V$ — выделенная вершина, называемая выходом сети. Каждой вершине v_i графа γ приписан язык $L(v_i)\in\mathscr{L}$. 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 $G=\langle \mathscr{A}, \mathscr{X}, \mathscr{L}, R, L^0 \rangle$ к представляющей ее Л-сети $\lambda(G)$ определяется следующим образом. Если $\mathscr{L}=\{L_1, ..., L_m\}$, то $\lambda(G)$ содержит m вершин $v_1, ..., v_m$; $L(v_i)=L_i$, $i=1, ..., m$, а выходом является вершина v^0, для которой $L(v^0)=L^0$. Для каждого правила $x \to L_j$ 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 $G(\lambda)=\langle \mathscr{A}, \mathscr{X}, \mathscr{L}, R, L^0 \rangle$ столь же прост с той разницей, что для непересечения входных алфавитов может понадобиться их переименование: $\mathscr{L}=\{L_1', ..., L_m'\}$ — это множество языков $L_1, ..., L_m$ 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 $x' \to L'(v_j)$, 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: $L_1=\{\alpha_1, ..., \alpha_m\}$, то каждое блочное правило $x \to L_i$ эквивалентно множеству обычных КС-правил $x \to \alpha_1 | \alpha_2 | ... | \alpha_m$, 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 $R=\{a \to L_2,\ a \to L_3,\ b \to L_4,\ c \to L_4,\ c \to L_5\}$; следователь-

но, 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 $b \to L_5$, тогда как соответствующее ребро из L_5 ко входу b языка L_2 в λ_1 отсутствует.

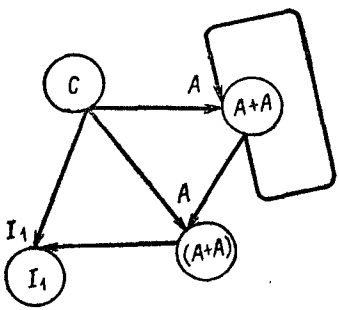

Рис. 7.4

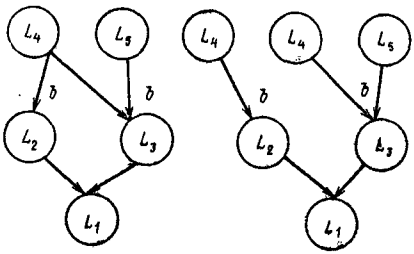

Рис. 7.5

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 (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.

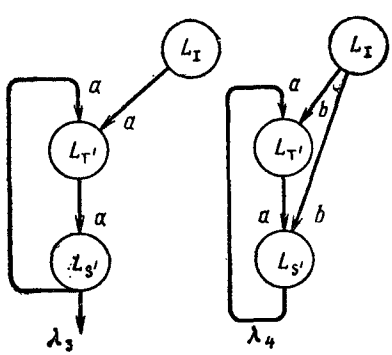

Рис. 7.6

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. Подробнее о блочных грамматиках и сетях из языков см. в [54].

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 искусственного — 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. Кратким знакомством 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 скоб-

кам соответствуют элементарные поддеревья, все вершины которых концевые; вершине v, которой синтаксически соответствует правило ρ_i: $A \to \alpha \circ \beta$ ($\circ$ — знак арифметической операции), семантически соответствует операция, обозначаемая знаком $\circ$; число потомков v_j, равное числу нетерминальных символов в ρ_i, соответствует числу операндов $\circ$; 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 грамматики, а их последовательность — структурой конкретного дерева вывода.

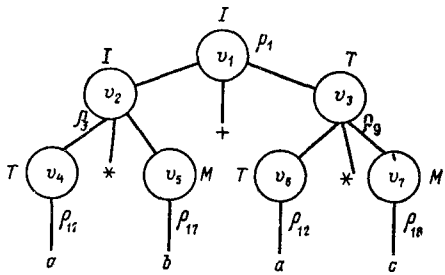

Рис. 7.7

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 $v_1, \ldots, v_7$. Кроме 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:

$$\rho_1: I \to I + T; \quad g_1(x_1, x_2) = x_1 + x_2;$$
$$\rho_3: I \to T * M; \quad g_3(x_1, x_2) = x_1 * x_2;$$
$$\rho_9: T \to T * M; \quad g_9(x_1, x_2) = x_1 * x_2;$$
$$\rho_{12}: T \to a; \quad g_{12} = a,$$

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 осмысленные 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 бессмысленности всего выражения в целом. Это можно сделать, поставив в соответствие правилу ρ_i, кроме арифметической функции g_i, еще и предикат корректности P_i, который 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 [32, 51].

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 программи-

рования, как правило, содержат конструкции, которые не могут быть точно формализованы 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.

ГЛАВА ВОСЬМАЯ

АВТОМАТЫ

8.1. ОСНОВНЫЕ ПОНЯТИЯ

Определения. *Конечным автоматом* (в дальнейшем — просто автоматом) называется система $S=\{A, Q, V, \delta, \lambda\}$, в которой $A=\{a_1, ..., a_m\}$, $Q=\{q_1, ..., q_n\}$, $V=\{v_1, ..., v_k\}$ — конечные множества (алфавиты), а δ: $Q\times A\to Q$ и λ: $Q\times A\to V$ — 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, называемое начальным (обычно будет считаться, что это q_1), то полученный автомат называется *инициальным* 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 $\delta\times\lambda$: $Q\times A\to Q\times V$, 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 изображен 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).

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

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 $\alpha = a_{j_1}a_{j_2}\ldots a_{j_k}$;

$$\delta(q_i, a_{j_1}a_{j_2}\ldots a_{j_k}) = \delta(\delta(\ldots\delta(q_i, a_{j_1}), a_{j_2}), \ldots, a_{j_{k-1}}), a_{j_k}).$$

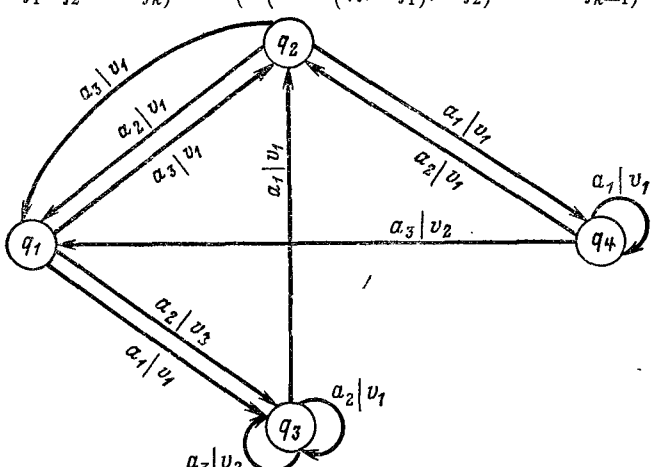

Рис. 8.1

Это традиционное определение «с многоточиями»; намного точнее и лучше читается индуктивное определение, к которому читатель, надеемся, уже привык:

а) $\delta(q_i, a_j)$ задается автоматной таблицей S;
б) для любого слова $\alpha \in A^*$ и любой буквы a_j

$$\delta(q_i, \alpha a_j) = \delta(\delta(q_i, \alpha), a_j). \tag{8.1}$$

С помощью расширенной функции δ определяется (также индуктивно) расширенная функция λ:

$$\lambda(q_i, \alpha a_j) = \lambda(\delta(q_i, \alpha), a_j). \tag{8.2}$$

Зафиксируем в автомате S начальное состояние q_1 и каждому входному слову $\alpha = a_{j_1} a_{j_2} \ldots a_{j_k}$ поставим в соответствие слово ω в выходном алфавите:

$$\omega = \lambda(q_1, a_{j_1})\,\lambda(q_1, a_{j_1} a_{j_2}) \ldots \lambda(q_1, a_{j_1} \ldots a_{j_k}). \tag{8.3а}$$

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:

$$\left.\begin{aligned} &S(q_i, a_j) = \lambda(q_i, a_j); \\ &S(q_i, \alpha a_j) = S(q_i, \alpha)\,\lambda(\delta(q_i, \alpha), a_j). \end{aligned}\right\} \tag{8.3б}$$

Автоматное отображение обладает двумя свойствами, которые следуют непосредственно из (8.3а), (8.3 б): 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* [17], 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 $\alpha = a_{i_1} \ldots a_{i_k}$ ставит в соответствие его отражение, т. е. слово $a_{i_k} \ldots a_{i_1}$; первая буква выходного слова равна здесь последней букве входного слова.

Указанные два свойства были бы достаточными условиями автоматности отображения, если бы речь шла о беско-

нечных 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 $\alpha = a_{i_1} \ldots a_{i_k}$ однозначно определяет путь длины k из этой вершины [обозначим его (q_i, α)], на k ребрах которого написаны соответственно буквы $a_{i_1}, \ldots, a_{i_k}$. 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 $aa \ldots a$.

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 $\sigma\omega\omega \ldots \omega\omega_1$, где ω_1 — начальный отрезок ω; при этом $0 \leqslant |\sigma| \leqslant n$; $1 \leqslant |\omega| \leqslant n$.

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. □

Пример 8.3. Граф автомата, заданного в табл. 8.2, изображен на рис. 8.2 (входные буквы на ребрах опущены; выходной алфавит $V = \{0, 1, 2\}$).

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

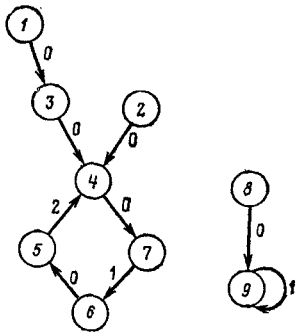

Рис. 8.2

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: $A_S \to A_T$, g: $Q_S \to Q_T$, h: $V_S \to V_T$ 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)]:

$$\begin{aligned} \delta_T(g(q), f(a)) &= g(\delta_S(q, a)); \\ \lambda_T(g(q), f(a)) &= h(\lambda_S(q, a)). \end{aligned} \tag{8.4}$$

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 существует 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, проис-

ходит 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, граф которого приведен на рис. 8.3. Существует гомоморфизм S в T. Соответствующая тройка отображений такова: f тривиально, так как оба входных алфавита состоят из

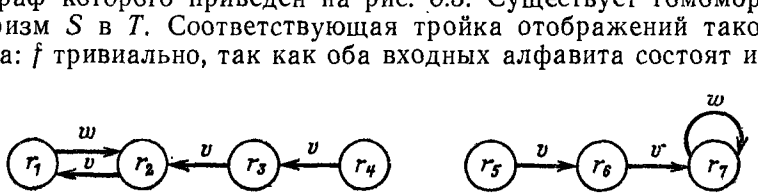

Рис. 8.3

одной буквы, g и h зададим списками (для краткости здесь и в аналогичных случаях вместо $q_i \to r_j$ будем писать $i \to j$); $g = \{1 \to 4,\ 2 \to 3,\ 3 \to 3,\ 4 \to 2,\ 5 \to 1,\ 6 \to 2,\ 7 \to 1,\ 8 \to 6,\ 9 \to 7\}$, $h = \{0 \to v,\ 1 \to w,\ 2 \to w\}$. 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 изоморфен графу T; однако сам автомат T'' не изоморфен 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 то-

го 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 автомату называется эквивалентным преобразованием автомата S. Можно ставить различные задачи о поиске автоматов, эквивалентных данному и обладающих заданными свойствами. 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 $(l \leqslant n)$: $C_1 = \{q_{11}, \ldots, q_{1i_1}\}, \ldots, C_l = \{q_{l1}, \ldots, q_{li_l}\}$, 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 $S_0 = (A_S, Q_{S_0}, V_S, \delta_{S_0}, \lambda_{S_0})$ так: $Q_{S_0} = \{C_1, \ldots$

..., $C_l\}$; 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. □

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 изменяет разбиения, т. е. состо-

яния из одного класса C_{ij} принадлежат одному классу $C_{i+1,j}$, то алгоритм заканчивается и полученное 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.

Последнее разбиение является искомым; минимальный для S автомат имеет шесть состояний. Если найденные классы обозначить по порядку C_1, ..., C_6, то минимальный автомат описывается табл. 8.4а, в которой снова обозначения состояний (классов C_i) заменены их индексами.

Таблица 8.3

q_l	a_1	a_2	a_3
1	2,0	4,1	4,1
2	1,1	1,0	5,0
3	1,1	6,0	5,0
4	8,0	1,1	1,1
5	6,1	4,1	3,0
6	8,0	9,1	6,1
7	6,1	1,1	3,0
8	4,1	4,0	7,0
9	7,0	9,1	7,1

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*.

Предоставляем 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_{kl} 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

Таблица 8.4

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

а)

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

б)

любого слова α. Но состояния q и q' одного класса заведомо принадлежат одному классу C_{1l}, поэтому $\lambda(q, a) = \lambda(q', a)$, 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 $r \leqslant k$. 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 $\delta(q_i, \alpha)$:

а) $\delta(q_i, a_j)$ задана таблицей автомата S;

б) если $\delta(q_i, \alpha)$ определена, то

$$\delta(q_i, \alpha a_j) \cong \delta(\delta(q_i, \alpha), a_j); \qquad (8.5а)$$

в) если $\delta(q_i, \alpha)$ не определена, то $\delta(q_i, \alpha a_j)$ не определена для всех a_j.

Функция $\lambda(q_i, \alpha)$:

$$\lambda(q_i, \alpha a_j) \cong \lambda(\delta(q_i, \alpha), a_j). \qquad (8.5б)$$

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);

б) если $\delta(q_i, \alpha)$ определена, то

$$S(q_i, \alpha a_j) = S(q_i, \alpha)\lambda(\delta(q_i, \alpha), a_j) \qquad (8.6)$$

(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 $S(q_i, \alpha a_j)$ 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 $S(q_i, \alpha)$ «определено, 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_l 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. если область определения q_i и r_j одна и та же и в этой области q_l и r_l 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 автомату 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 примере, о чем идет речь. Рассмотрим автомат, заданный табл. 8.5.

Таблица 8.5

	a_1	a_2	a_3
1	2,0	—	3, —
2	—	1, —	3,0
3	2,1	1, —	3,0

Псевдонеотличимых состояний здесь просто нет. Рассмотрим состояния 2 и 3 (q_2 и q_3). Область определения для q_2, т. е. для отображения $S(q_2, \alpha)$, содержится 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 $\lambda(q_2, a) \cong \lambda(q_3, a)$ и $\delta(q_2, a) \cong \delta(q_3, a)$. 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 соображение 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 нее 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, описанной ранее, покрывает исходный автомат. Отметим, что от-

ношение 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 $C_1, \ldots, C_l$ полной, если $C_1 \cup \ldots \cup C_l = Q$, 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 (теорема Полла — Ангера). Если для час-

тичного автомата S имеется полная и замкнутая система классов совместимости $C_1, ..., C_l$, то существует автомат S', покрывающий S.

Автомат $S' = (A_S, Q_{S'}, V_S, \delta_{S'}, \lambda_{S'})$ строится так: $Q_{S'} = \{C_1, ..., C_l\}$; 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 совместимости 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 совместимости C_i автомата S; следовательно, ввиду полноты системы классов $\{C_i\}$ автомат S' покрывает S. □

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, ..., S_N; соответствующие минимальные автоматы $S_{10}, ..., S_{N0}$ 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 автомату. 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, а, можно

Таблица 8.6

	a_1	a_2
1	1, —	2,0
2	3,0	1,0
3	2,1	1,0

а

	a_1	a_2
C_1	C_2, 0	C_1, 0
C_2	C_1, 1	C_1, 0

б

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 автомату S' с двумя состояниями (табл. 8.6, *б*), 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 заключает-

ся 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 пытаться 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 $\{C_1, \ldots, C_p\}$ 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 книгах Р. Миллера [14] и А. Д. Закревского [13].

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) $t=1, 2, 3...$; 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 программы для ЭВМ). 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}, $a_{i_1}a_{i_2}$, $a_{i_1}a_{i_2}a_{i_3}$ ... 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 видоизменен-

ными 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 [8, 19].

Автоматы Мура. Конечный автомат называется *автоматом Мура*, если его функция выходов зависит только от состояний, т. е. для любых q, a_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 $S=(A, Q, V, \delta, \lambda)$, $A=\{a_1, \ldots, a_m\}$, $Q=\{q_1, \ldots, q_n\}$. Определим автомат Мура $S_м$ следующим образом: $A_м=A$, $V_м=V$, $Q_м$ содержит $mn+n$ состояний: mn состояний $q_{ij}(i=1, \ldots, n; j=1, \ldots, m)$, соответствующих парам (q_i, a_j) автомата S, и n состояний $q_{i0}(i=1, \ldots, n)$. Функции $\delta_м$ и μ определяются так: $\delta_м(q_{i0}, a_k)=q_{ik}$; для $i=1, \ldots, n$ $\delta_м(q_{ij}, a_k)=q_{lk}$, 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 $S(q_i, \alpha)=S_м(q_{i0}, \alpha)$. Это делается индукцией по длине α; проведение индукции предоставляется читателю. □

Рекомендуем в качестве примера построить автомат Мура, эквивалентный автомату 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 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 [62], 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 α $e\alpha = \alpha e = \alpha$; 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 $E = \{\alpha_1, \ldots, \alpha_k\}$ представимо в автомате. Идея построения автомата по конечному множеству слов иллюстрируется графом на рис. 8.4, *а*, где заключительные состояния $q_{n-k}, \ldots, q_{n-1}$ изображены двойным кружком. Для конкретных множеств эта идея модифицируется в связи с тем, что слова могут иметь общие начала (тогда начала соответствую-

щих 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

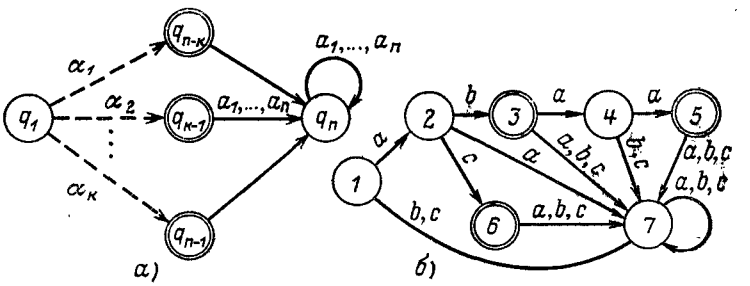

Рис. 8.4

$E=\{ab, ac, abaa\}$ 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 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.

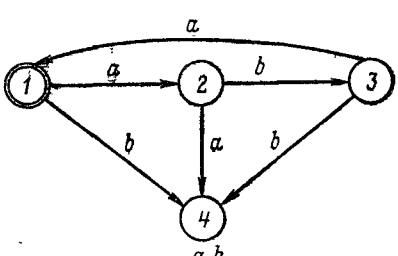

Рис. 8.5

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, 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 $\{e, aba, abaaba, \ldots, (aba)^n\ldots\}$.

События — это множества конечных слов. Однако можно говорить, что автомат распознает бесконечную последовательность букв $\alpha = a_{i_1} a_{i_2} a_{i_3} \ldots$, если он представляет множество $E = \{a_{i_1}, a_{i_1} a_{i_2}, \ldots\}$, состоящее из всех начальных отрезков последовательности α.

Теорема 8.5. Существуют события, непредставимые в конечных автоматах; более конкретно: никакая непериодическая бесконечная последовательность не распознаваема 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 $\alpha = a_{i_1}, a_{i_2} \ldots$ распознаваема автоматом S с n состояниями. Тогда для любого ее начального отрезка $a_{i_1} \ldots a_{i_j}$ $\delta(q_1, a_{i_1} \ldots a_{i_j}) = q_{i_j}$, где q_{i_j} — заключительное состояние; поэтому при переработке этой последовательности автомат проходит последовательность заключительных состояний $q_{i_1} \ldots q_{i_j} \ldots$ Так как Q_S конечно, то некоторое состояние встретится в этой последовательности дважды: $q_{i_j} = q_{i_{j+k}}$ и, следовательно, $\delta(q_{i_j}, a_{i_{j+1}}, \ldots, a_{i_{j+k}}) = q_{i_j}$, причем все состояния, проходимые автоматом, заключительные. Поэтому, если на вход автомата в состоянии q_1 подавать бесконечную периодическую последовательность $\alpha_1 = a_{i_1} \ldots a_{i_j} (a_{i_{j+1}} \ldots a_{i_{j+k}})$, 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. □

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. Рас-

смотрим 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* $E_1^* = e \cup E_1 \cup E_1E_1 \cup \ldots \cup (E_1)^n \cup \ldots = \bigcup_{i=0}^{\infty} E_1^i$.

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

$$E_1(E_2 \cup E_3) = E_1E_2 \cup E_1E_3; \tag{8.7}$$

$$(E_1 \cup E_2)E_3 = E_1E_3 \cup E_2E_3; \tag{8.8}$$

$$(E_1E_2)E_3 = E_1(E_2E_3); \tag{8.9}$$

$$(E^*)^* = E^*; \tag{8.10}$$

$$E^* = e \cup E(E)^*. \tag{8.11}$$

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 $(a_1 \cup \ldots \cup a_n)^*$ действительно задает множество всех слов (включая пустое) в алфавите A.

б. Множество, состоящее из одного слова $a_{i_1}\ldots a_{i_k}$, является регулярным событием, поскольку любое выражение вида $a_{i_1}\ldots a_{i_k}$ регулярно: оно построено из букв с помощью конкатенации. Любое конечное множество слов $E=\{a_1, \ldots, a_k\}$ регулярно и описывается выражением $E=a_1 \cup \ldots \cup a_k$, 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 $a_1 \cup \ldots \cup a_k$; 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^*, кончающиеся 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 $k=1, 2\ldots$; иначе говоря, если $\alpha \in E$, то в E содер-

жатся 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 $abcd \in E$, $aabccdd \in E$ 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, порождаемое 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 единст-

венной 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

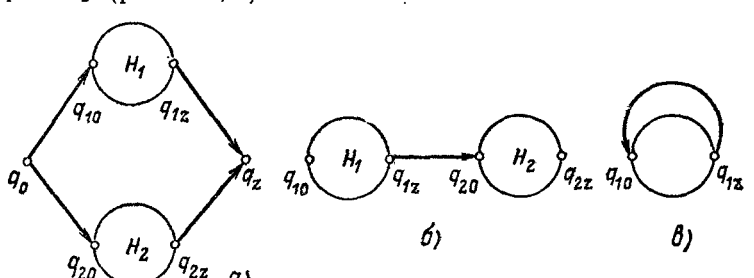

Рис. 8.6

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. $E=E_1E_2$. 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. $E=E_1^*$. 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, $E=E_1E_2$ 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 существуют путь α_1 из q_{10} в q_{1z} и путь α_2 из q_{20} в q_{2z}. Но тогда 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. □

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* [61], 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 тако-

му 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 $\varnothing$, 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 $\leqslant k$, 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 $\delta_G(\tilde{q}_1, \alpha) = \tilde{q}_l$, 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 в q^*. По построению H' из того, что в H' есть ребро a из $\tilde{q}_l$ в $\tilde{q}_j$, сле-

дует, что в H для любой вершины $q \in \tilde{q}_j$ найдется вершина $\tilde{q}_i$, 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 $\tilde{q}_i$ 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 $\alpha \in E$. □

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. □

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

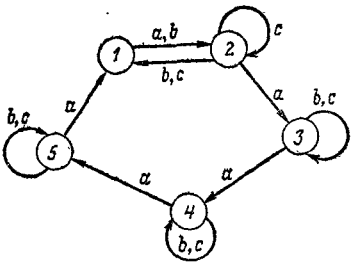

Рис. 8.7

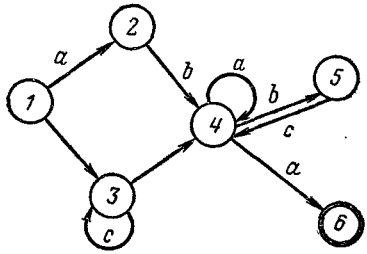

Рис. 8.8

рис. 8.7; доказательство того, что он обладает указанным свойством, можно найти в [17].

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. Сначала 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 ∅, как обычно, обозначено пустое множество); после перенумерации этих подмножеств она приобретает обычный вид таблицы переходов (табл. 8.7,б).

Таблица 8.7

	a	*b*	*c*
1	2, 4, 6	5	3,4
2, 4, 6	4,6	4,5	∅
5	∅	∅	4
3,4	4,6	5	3,4
4,6	4,6	5	∅
4,5	4,6	5	4
∅	∅	∅	∅
4	4,6	5	∅

а)

	a	*b*	*c*
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

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.

Анализ автоматов. Если процесс построения автомата по заданному событию (и вообще по некоторому описанию множества слов во входном алфавите) называется синтезом, то обратный процесс — получения 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 бо́льшими, 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 $M_{ij}^r = E_{ij}^{r-1} \cup \widetilde{M}_{ij}^r$, где $\widetilde{M}_{ij}^r$ — множество всех слов из M_{ij}^r, проходящих через q_r. Если $\alpha \in \widetilde{M}_{ij}^r$ и проходит через q_r s раз, то α представимо в виде $\alpha = \alpha_0\alpha_1 \ldots \alpha_{s-1}\alpha_s$, где $\alpha_0 \in E_{ir}^{r-1}$, $\alpha_s \in E_{rj}^{r-1}$, $\alpha_1 \ldots \alpha_{s-1} \in E_{rr}^{r-1}$ (т. е. содержит $s-1$ цикл, проходящий через q_r). Поэтому

$$\alpha \in E_{ir}^{r-1} \left(E_{rr}^{r-1}\right)^* E_{rj}^{r-1}.$$

И наоборот, если это включение выполнено, то из регулярной формулы его правой части и индуктивного предположения следует, что $\alpha \in \widetilde{M}_{ij}^r$. Таким образом, $\widetilde{M}_{ij}^r = E_{ir}^{r-1} (E_{rr}^{r-1})^* E_{rj}^{r-1}$ и, следовательно, $M_{ij}^r = E_{ij}^{r-1} \cup E_{ir}^{r-1} (E_{rr}^{r-1})^* E_{rj}^{r-1} = E_{ij}^r$, что и доказывает лемму.

Очевидно, что для автомата с n состояниями, начальным состоянием q_1 и заключительными состояниями $q_{z_1}, \ldots, q_{z_p}$ представляемое им событие имеет вид $M_{1z_1}^n \cup \ldots \cup M_{1z_p}^n$. Из леммы следует, что это событие регулярно. □

Доказательство теоремы по существу представляет со-

бой 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 [14], 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 [9].

г. Автомат называется *асинхронным,* если в нем для любых q_i и a $\delta(q_i, aa) = \delta(q_i, a)$. Состояние $q_j = \delta(q, a)$ 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 $S(a)$, таких, что $\delta(q_1, a)$ — 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 представляющей 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-перечислимо, 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-перечислимо, то оно автоматно-разрешимо, однако число состояний разрешающего автомата может экспоненциально возрасти. Аналогия с общим случаем эффективно задаваемых множеств будет 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 $n \leqslant |\alpha| < 2n$. 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 $\alpha x \to \beta x$ (более подробно о связи автоматов с формальными системами см. в [52]).

Автоматы и языки. На естественный вопрос о связи конечных автоматов с языками или, точнее, о том, как конеч-

но-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 $B_i \to aB_j$, если q_j — незаключительное состояние, и два правила $B_i \to aB_j$, $B_i \to a$, 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 $B_i \to aB_j$ соответствует ребро с символом a из q_i в q_j; каждому правилу вида $B_i \to a$ соответствует ребро с символом a из q_i в q_j. Легко видеть, что путь α из начальной вершины в заключительную существует в H_G тогда и только тогда, когда $\alpha \in L(G)$. □

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 эквивалеитна, — 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 бо́льшую 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.

Комбинационные и логические автоматы. Автомат[1] 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 (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.

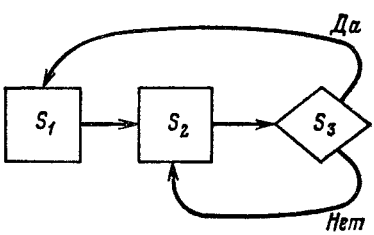

Рис. 8.9

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

[1] Здесь и далее, если не оговорено противное, рассматриваются автоматы общего вида — автоматы Милн.

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, а—в) и рассмотрим два крайних случая, которые возможны для входной информации автоматов. (Выходы для данных рассмотрений несущественны: для определенности будем считать, что они во всех клетках различны.)

Таблица 8.8

q_{11}	q_{11}	q_{1z}	q_{12}
q_{12}	q_{1z}	q_{11}	q_{12}
q_{1z}	—	—	—

а)

q_{21}	q_{23}	q_{22}
q_{22}	q_{2z}	q_{22}
q_{23}	q_{21}	q_{23}
q_{2z}	—	—

б)

q_{31}	q_{31}	q_{3z0}	q_{32}
q_{32}	q_{3z1}	q_{3z1}	q_{31}
q_{3z0}	—	—	—
q_{3z1}	—	—	—

в)

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 автома-

Таблица 8.9

	a	b	c	d	f	g	h	i
q_{11} q_{12}	q_{11} q_{21}	q_{21} q_{11}	q_{12} q_{12}					
q_{21} q_{22} q_{23}				q_{23} q_{31} q_{21}	q_{22} q_{22} q_{23}			
q_{31} q_{32}						q_{31} q_{11}	q_{21} q_{11}	q_{32} q_{31}

а)

	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	—	—	—

б)

тов; 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 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.

Таблица 8.10

	a	b	c
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}

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, ..., 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,....S_k. Автомат S называют суммой S_1, ..., S_k; 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, ..., S_m (компонент сети) понимается вектор $(q_{i_1}, ..., q_{i_m})$, 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 временная последовательность сигналов или импульсов (каждый сигнал соответствует букве); интервал между соседними импульсами равен в точности длине такта[1]. Следовательно, слово

[1] 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.

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: $\delta(q(t), a(t)) = q(t+1)$; 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. Каким 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): *а* — с разделительными входами и алфавитами A_1 и A_2; *б* — с общим входом и алфавитом A.

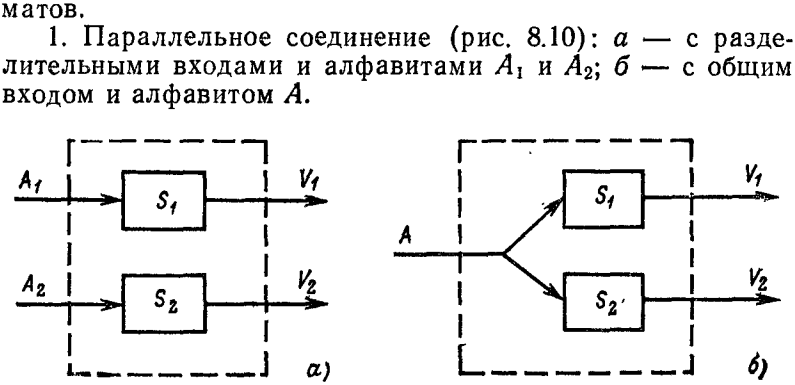

Рис. 8.10

Могут сказать, что соединение *а* — это вообще не соединение; тем не менее пару автоматов $S_1 = (A_1, Q_1, V_1, \delta_1, \lambda_1)$; $S_2 = (A_2, Q_2, V_2, \delta_2, \lambda_2)$ 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 $\lambda(q, a)=(\lambda_1(q^1, a^1), \lambda_2(q^2, a^2))=(v^1, v^2)$, где $v_1\in V_1$; $v^2\in V_2$. Автомат S называется *прямым произведением* автоматов S_1 и S_2.

До сих пор речь шла об автоматах с одним входом и одним выходом. Сеть на рис. 8.10, *а* 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.

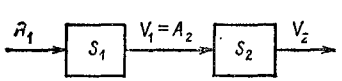

Рис. 8.11

Автомат на рис. 8.10, *б* 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

$$q(t+1)=(q^1(t+1), q^2(t+1))=$$

$$=(\delta_1(q^1(t), a(t)), \delta_2(q^2(t), \lambda_1(q^1(t), a(t)))). \qquad (8.12)$$

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 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 таб-

Таблица 8.11

	0	1
q_0	q_0, 0	q_1, 0
q_1	q_0, 1	q_1, 1

Таблица 8.12

ij	0	1	Выход
00	00	10	0
01	00	10	1
10	01	11	0
11	01	11	1

лице 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 заключа-

ется в том, что выход автомата S_1 присоединяется к одному из его входов.

Рассмотрим случай, когда S_1 — комбинационный автомат без задержки, и предположим, что S_1 реализует логическую функцию x_1x_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 $v(t)=\overline{x_1(t)x_2(t)}=1$, 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 $x_1(t)=1$ $z(t)=\overline{x_1(t)x_2(t)}=0$, т. е. опять

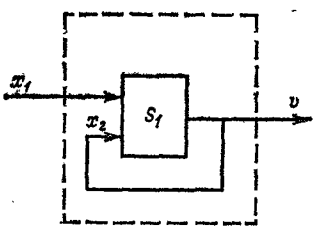

Рис. 8.12

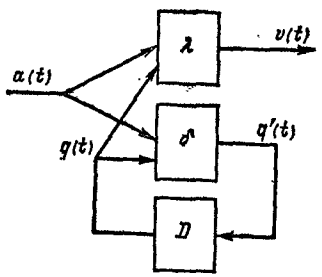

Рис. 8.13

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 задержек, 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.

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 представля-

ет собой автомат Мура, входной и выходной алфавиты которого совпадают и равны $Q=\{q_1, ..., q_n\}$, алфавит состояний $R=\{r_1, ..., r_n\}$ имеет ту же мощность, что и Q, $\lambda_D(r_i)=q_i$, $\delta_D(r_i, q_j)=r_j$ для всех $i, j=1, ..., n$. 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 $\log_2|Q|$. □

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 $x_1, ..., x_m$, выходами $z_1, ..., z_n$ 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 $z=f^1(p_1^1, ..., p_r^1)$, где $p_1^1, ..., p_r^1$ — входы блока. Каждый из них либо является входом сети, т. е. переменной x, либо присоединен к выходу другого блока: $p_i=f_i^2(p_{i1}^2, ..., p_{im_i}^2)$ и, следовательно, $z=f^1(f_1^2(p_{11}^2, ..., p_{1m_1}^2), ..., f_r^2(p_{r1}^2, ..., p_{rm_r}^2))$, 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 из $x_1(t), ..., x_m(t)$ 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. Входа-

ми G_0 служат, во-первых, входы G, а во-вторых, выходы $y_1, ..., y_k$ элементов задержки G; выходы G_0 — это выходы G и входы $Y_1, ..., Y_k$ элементов задержки G (рис. 8.14). Поэтому входной набор G_0 имеет вид $(x_1, ..., x_m, y_1, ..., y_k)$, выходной набор — $(z_1, ..., z_n, Y_1, ..., Y_k)$. Если теперь набор $(x_1(t), ..., x_m(t))$ считать входным сигналом $a(t)$ сети G, набор $(z_1(t), ..., z_n(t))$ — выходным сигналом $v(t)$ сети G, а набор $(y_1(t), ..., y_k(t))$ — состоянием $q(t)$ сети G и учесть, что $(Y_1(t), ..., Y_k(t)) = (y_1(t+1), ..., y_k(t+1)) = q(t+1)$, то получим, что сеть G_0 вычисляет две системы логических

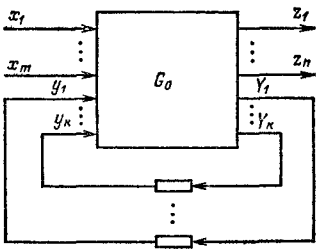

Рис. 8.14

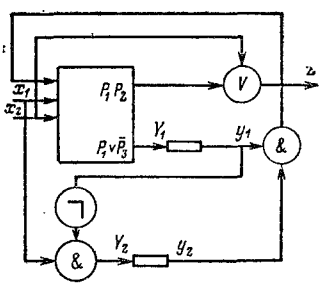

Рис. 8.15

функций от набора $(x_1, ..., x_m, y_1, ..., y_k) = a(t) \times q(t)$ — систему $(Y_1(t), ..., Y_k(t)) = q(t+1)$, т. е. функцию δ, и систему $z_1(t), ..., z_n(t)$, 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. □

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 предоставляется 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: $Y_1 = \bar{x}_2 \vee y_1 y_2$, $Y_2 = x_1 \bar{y}_1$, $z = x_1 y_1 y_2 \vee \bar{x}_2$.

Переходы автомата, реализуемого этой сетью, приведены в табл. 8.13. Заметим, что первые два состояния этого автомата эквивалентны.

Таблица 8.13

$y_1 y_2$	$x_1 x_2$			
	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

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 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 (см., например, [16]).

Описание поведения асинхроных 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 автомату — 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 [14, 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 автоматнополным 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 $F=\varphi(x_{i_1}, ..., x_{i_k})$, где $\varphi\in\Sigma$, $x_{i_1},..., x_{i_k}$ — исходные переменные, то схема G состоит из одного элемента φ, входы которого отождествлены с переменными $x_{i_1},..., x_{i_k}$, а выход — с переменной z; 2) если $F=\varphi(F_1,..., F_k)$, где F_i — переменная x_{i_l} 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 $\overline{x_1x_2x_3} \vee x_2x_3x_4$ соответствует схема на рис. 8.16, *а*.

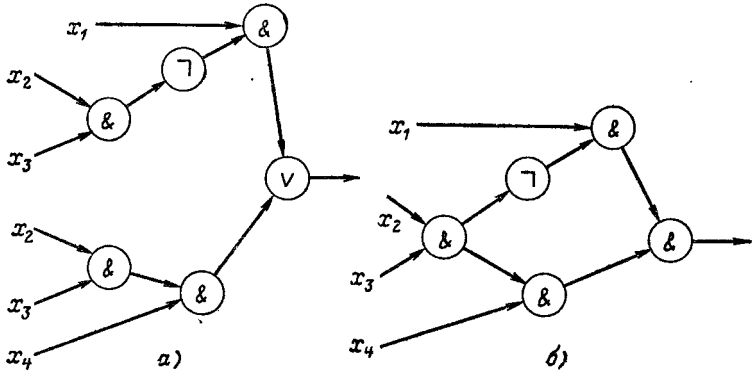

Рис. 8.16

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 G 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 функ-*

циональ но полным в смысле § 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. 8.18, *б* реализует ту же формулу, что и рис. 8.16, *а*, однако схема *б* 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 $L_\Sigma(n) = \max\limits_{f \in P_2(n)} L_\Sigma(f)$, 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 (теорема Шеннона — Лупанова). 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 $L_\Sigma(f) \leqslant (1-\varepsilon) c_\Sigma \frac{2^n}{n}$, стремится к нулю с ростом n. □

Здесь знак $\sim$ обозначает асимптотическое равенство $\left(a(n) \sim b(n)$, если $\lim\limits_{n\to\infty} \frac{a(n)}{b(n)} = 1\right)$. 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 [59].

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 [60], 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 оказались 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 $k_1, ..., k_s$, 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 $b:=f(a_1, ..., a_p)$ (выполнить операцию f над содержимыми ячеек $a, ..., a_p$ 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 $f(a_1, ..., a_p)$ — 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 $k_1, ..., k_s$ называется последовательность шагов $k(1), k(2), ..., k(t)$, 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 П вычисляет (или реализует) логическую функцию $f(x_1, ..., x_n)=y$, если для любого двоичного набора $\sigma=(\sigma_1, ..., \sigma_n)$ при начальном состоянии памяти $x_1=\sigma_1$, $x_2=\sigma_2, ..., x_n=\sigma_n$ (состояние остальных ячеек несущественно) программа через конечное число шагов останавливается и при этом в ячейке y лежит величина $f(\sigma_1, ..., \sigma_n)$.

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 $П$ на наборе σ.

Любой схеме, реализующей функцию f и содержащей N элементов, нетрудно поставить в соответствие программу, реализующую f и состоящую из N команд, следующим образом. Занумеруем элементы схемы числами $1, 2, ..., n$ так, чтобы на любом пути от входа к выходу номера элементов возрастали; 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 $e_{j_1}, ..., e_{j_p}$ (некоторые из них, возможно, являются входами схемы). Поставим элементу e_i в соответствие либо ячейку a_i и команду $a_i=\varphi_i(a_{j_1}, ..., a_{j_p})$, если $i\neq N$; либо ячейку y и команду «$y:=\varphi N(a_{j_1}, ..., a_{j_p})$ 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 $t_{\text{макс}} = t_{\text{ср}} = N$, 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 $2^n/n$. Наоборот, 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 $t_{\text{макс}}=N$ 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^0_{1z} (с командой «$y=0$») и v^1_{1z} (с командой «$y=1$»); G_2 — граф программы для f_2 с началом v_{20} и концами v^0_{2z} и v^1_{2z}. Тогда: 1) программа, граф G которой получен присоединением G_2 к «нулю» G_1 (т. е. отождествлением вершин v^0_{1z} 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^1_{1z} и v_{20}), вычисляет $f=f_1 \& f_2$; 3) программа, граф которой получен из G_1 заменой команд в v^0_{1z} и v^1_{1z} 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 $(x_1 \vee x_2)(\bar{x}_3 x_4 \vee x_5)$ реализуется бинарной программой, граф которой приведен на рис. 8.17.

Описанный метод не гарантирует минимальность получаемых программ по времени и числу команд. Существуют

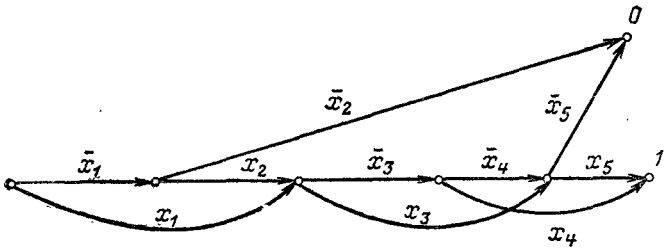

Рис. 8.17

другие методы, которые, в частности, для любой функции n переменных дают бинарную программу с $t_{\text{макс}} \leqslant n$ независимо от длины исходной формулы. В общем же случае сложность бинарных программ характеризуется следующими асимптотическими оценками. Пусть $L_{\text{Б}}(n)$ — функция Шеннона для числа команд бинарных программ (см. аналогичное определение для $L_{\Sigma}(n)$ в конце § 8.3).

Теорема 8.14. 1) $L_{\text{Б}}(n) \sim \frac{2^n}{n}$, причем существует метод синтеза бинарных программ, удовлетворяющих этой оценке, для которых $t_{\text{макс}} \sim n$; 2) для любого $\varepsilon > 0$ доля функций, для которых: а) $L_{\text{Б}}(f) \leqslant (1-\varepsilon)\frac{2^n}{n}$; б) $t_{\text{ср}}(f) \leqslant (1-\varepsilon)n$, стремится к 0 при $n \to \infty$. □.

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 $t_{\text{макс}}$ и $t_{\text{ср}}$. Напомним, что для операторных программ $t_{\text{макс}} = t_{\text{ср}} = N$. Для времени вычисления бинарных программ справедливы следующие утверждения.

1. Для любой функции f, существенно зависящей от всех n переменных, $[\log_2 n+1]\leqslant t_{max}(f)\leqslant n$, причем обе границы достигаются.

2. Существуют последовательности функций $f_1(x_1)$, $f_2(x_1, x_2), ..., f_n(x_1, ..., x_n)...$, для которых $\lim\limits_{n\to\infty} t_{ср}(f_n)=2$; иначе говоря, с ростом n $t_{ср}(f_n)$ практически не увеличивается. Примером такой последовательности является любая последовательность $f_1, ..., f_n...$, где $f_1=x_1,..., f_{n+1}=x_{n+1}\bigcirc f_n$ ($\bigcirc$ — 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, как показано в [55], эта оценка понижается вдвое.

Более подробные сведения об оценках сложности и методах программной реализации логических функций и автоматов можно получить в [53]. Здесь отметим лишь, что логический автомат всегда может быть реализован бинарной программой, не требующей промежуточной памяти.

ГЛАВА ДЕВЯТАЯ

КОМБИНАТОРНЫЕ ЗАДАЧИ И ТРУДОЕМКОСТЬ ВЫЧИСЛЕНИЙ

9.1. ТРУДОЕМКОСТЬ ОТНОСИТЕЛЬНО РАЗНЫХ МАШИН

В классической теории алгоритмов (см. гл. 5) задача считается разрешимой, если существует решающий ее алгоритм. Однако для реализации некоторых алгоритмов при любых разумных с точки зрения физики предположениях о скорости выполнения элементарных шагов может потребоваться больше времени, чем по современным воззрениям существует вселенная. Поэтому возникает потребность конкретизировать понятие разрешимости и придать ему оценочный, количественный характер, введя такие характеристики алгоритмов, которые позволяли бы судить о возможности и целесообразности их практического применения.

Среди различных возможных характеристик алгоритмов наиболее важными с прикладной точки зрения являются две: время и память, расходуемые при вычислении. Физическое время вычисления алгоритма характеризуется произведением $\bar{\tau}t$, где t — число действий (шагов) вычисления, а $\bar{\tau}$ — среднее физическое время реализации одного шага. Число шагов t определяется описанием алгоритма в данной алгоритмической модели, это — величина математическая, не зависящая от особенностей физической реализации модели. Напротив, $\bar{\tau}$ зависит от реализации и определяется скоростью обработки сигналов в элементах, записи и считывания информации и т. д. Поэтому число действий t можно считать математическим временем вычисления алгоритма, определяющим физическое время вычисления с точностью до константы $\bar{\tau}$, зависящей от реализации.

Память как количественная характеристика алгоритма определяется количеством s единиц памяти (ячеек ленты машины Тьюринга или машинных слов в современных ЭВМ), используемых в процессе вычисления алгоритма. Ясно, что эта величина по порядку не может превосходить числа шагов вычисления: $s \leqslant \mu t$, где μ — максимальное число единиц памяти, используемых в данной машине на одном шаге. Напротив, t может существенно превосходить s, например, возможно соотношение $t \geqslant s^c$. С этой точки зрения время более тонко отражает сложность алгоритма, чем память; и это — одна из причин, по которой исследованию временны́х характеристик алгоритма уделяется большее внимание. Существуют и другие, прикладные причины (в частности, то, что, грубо говоря, память дешевле времени), которые здесь обсуждаться не будут. Так или иначе, здесь будет идти речь о трудоемкости[1] алгоритмов и задач, решаемых алгоритмами.

Итак, трудоемкость алгоритма — это число элементарных действий, выполненных при его вычислении.

Полной характеристикой конкретного варианта задачи является его формальное описание. Характеристикой сложности описания можно считать его объем, который иногда называют *размерностью* задачи (например, для изоморфизма графов размерностью задачи можно считать число символов в матрицах смежности графов); тогда исследование

[1] Еще один термин для этого понятия — распространенный, но менее удачный — *временна́я сложность*.

трудоемкости алгоритма рассматривается как исследование зависимости трудоемкости вычисления от размерности задачи, решаемой алгоритмом.

И в математике, и на практике в конечном счете нас интересуют не алгоритмы сами по себе, а задачи, которые они решают. Одна и та же задача может решаться различными алгоритмами и на разных машинах. Если машина M зафиксирована, то трудоемкостью данной задачи относительно машины M называется минимальная из трудоемкостей алгоритмов, решающих задачу на машине M. Задач, трудоемкость которых была бы определена точно, довольно мало; хорошим результатом считается определение ее по порядку, т. е. с точностью до множителя, ограниченного некоторой константой. Чаще удается оценить ее сверху или снизу. Оценку сверху получают, указав конкретный алгоритм решения задачи: по определению, трудоемкость задачи не превосходит трудоемкости любого из решающих ее алгоритмов. Оценки трудоемкости снизу — гораздо более трудное дело; их получают обычно из некоторых общих соображений (например, мощностных или информационных). О них здесь говорить не будем.

Можно ли говорить об инвариантности теории трудоемкости вычислений? Иначе говоря, возможны ли утверждения о трудоемкости вычислений, сохраняющиеся при переходе к любой алгоритмической модели? Что касается прямых количественных оценок, то инвариантами не являются не только константы, но и степени. Например, доказано, что трудоемкость распознавания симметричности слова длины n относительно его середины на машине Тьюринга не меньше, чем cn^2, тогда как для любой ЭВМ, имеющей доступ к памяти по адресу, допускающей операции над адресами, легко написать программу, решающую эту задачу с линейной трудоемкостью.

Таким образом, скорости вычислений на разных моделях различны. Однако строить теорию трудоемкости вычислений, привязываясь к некоторым конкретным моделям, неудобно ни для теории, ни для практики. Для теории — потому, что такая привязка не дает достаточно объективных характеристик трудоемкости задачи, т. е. не позволяет отделить влияние особенностей выбранной модели от специфики самой задачи; для практики — потому, что разнообразие реальных машин растет, и нужны общие понятия и методы оценки трудоемкости решения задач, которые сохраняют свою силу при любых изменениях в мире компью-

теров. Поэтому инвариантная теория трудоемкости нужна, и вопрос не в том, возможна ли она, а в том, как ее построить (т. е. какие инварианты найти). Для того чтобы обсуждать этот вопрос, прежде всего следует посмотреть, как меняется трудоемкость при переходе от одной машины к другой. Это рассмотрение мы начнем с некоторого краткого обзора парка абстрактных машин, о которых будет идти речь.

До сих пор в явном виде была описана только одна абстрактная машина — машина Тьюринга (§ 5.2). Однако в § 8.4 неявно использовалась машина другого типа, гораздо более близкая к современным ЭВМ, в которой возможен доступ к памяти по адресам. Такая машина, называемая машиной с произвольным доступом к памяти, может на следующем шаге переходить к любой ячейке с указанным адресом (команды условного и безусловного переходов) и реализовать команды-операторы (см. § 8.4) вида $b := f(a_1, a_2, ..., a_p)$ (выполнить операцию f над содержимым ячеек $a_1, a_2, ..., a_p$ и результат положить в ячейку b). Возможны различные варианты моделей машины с произвольным доступом к памяти (см., например, [13]); в более сложных вариантах допускаются операции над адресами. Здесь мы не будем рассматривать все эти варианты, ограничившись фиксацией лишь одной простой модели — машины элементарных логических операций, или кратко L-машины. Относительно других моделей абстрактных машин ограничимся констатацией их основных свойств, которых будет достаточно при последующих рассмотрениях. Будем считать, что каждая машина имеет конечное число устройств (головок, устройств управления головками, процессоров — устройств, выполняющих элементарные операции, и т. д.), каждое устройство и каждая ячейка памяти могут находиться в одном из конечного числа возможных состояний (состояние ячейки памяти — это записанный в ней символ), и выполненне любого элементарного действия (шага) зависит от информации из конечного числа ячеек памяти, ограниченного некоторой константой μ. Будем говорить, что все ячейки читаются на данном шаге. Полное состояние машины, т. е. набор состояний устройств, состояний ячеек памяти и указание ячеек, читаемых в настоящий момент, называется, как и в § 5.2, конфигурацией машины.

И наконец, еще одно вступительное замечание. Алгоритм, осуществляемый машиной, может быть реализован двояким образом: он может быть «встроен» в управляю-

щее устройство или записан в памяти машины. В первом случае машина является специализированной и может выполнять только данный алгоритм; чтобы изменить алгоритм, надо поменять управляющее устройство. Таковы машины Тьюринга в примерах 5.2—5.9. Во втором случае запись алгоритма в памяти называется программой, а сама машина — программируемой; алгоритм, встроенный в управляющее устройство, решает задачу исполнения программ, записанных в памяти машины. Такова универсальная машина Тьюринга (см. § 5.2) и все реальные универсальные ЭВМ. В обоих случаях начальная конфигурация машины — состояния всей памяти и всех устройств — полностью определяет процесс вычисления.

Машина элементарных логических операций (L-машина) — это машина с произвольным доступом к памяти, имеющая следующую систему команд:

$$x := 0;$$
$$x := 1;$$
$$x := y;$$
$$x := \neg y;$$
$$x := y \,\&\, z;$$
$$x := y \vee z;$$
„конец“,

где x, y, z — адреса ячеек памяти; $:=$ — знак присвоения (см. § 8.4).

Программа элементарных логических операций (L-программа) с однократной записью — это программа (в смысле § 8.4), состоящая из указанных команд, в которой запись в каждую ячейку памяти производится не более одного раза (читать ячейку можно неоднократно).

Можно было бы дополнить систему команд L-машины командами ввода информации в ячейки памяти для данных, ввода программ и вывода результатов решения задач, однако будем считать, что эти элементы памяти доступны для ввода информации и обозрения извне (трудоемкость этих действий следует считать пропорциональной количеству элементов памяти, куда надо ввести или откуда надо вывести информацию). Пусть программа и данные (значения двоичных переменных) находятся в разных секциях памяти, и каждая секция имеет свою адресацию. Тогда можно так перекодировать программу с однократной записью

(изменить адреса данных), что номера команд программы и адреса ячеек памяти, в которых эти команды производят запись, станут одинаковыми, т. е. будут иметь вид: $i : i : = = U_i (i = 1, 2, ..., t-1)$, где выражения U_i имеют вид либо 0, либо 1, либо j, либо $\neg j$, либо $j \& k$, либо $j \vee k$ (только в конце программы стоит команда «конец»;).

После того как программа с однократной записью кончит работу, переменные $z_1, z_2, ..., z_{t-1}$, хранящиеся в ячейках памяти с адресами 1, 2, ..., $t-1$, будут удовлетворять системе уравнений $z_i = V_i$ $(i = 1, 2, ..., t-1)$, где V_i имеют соответственно вид 0, 1, z_j, $\neg z_j$, $z_j \& z_k$ или $z_j \vee z_k$. Если во всех командах адреса аргументов меньше адресов результатов, т. е. справа от знака $:=$ стоят меньшие числа, чем слева, то предопределены значения всех переменных $z_1, z_2, ..., z_{t-1}$. В противном случае программа читает не то, что сама записала.

Интерпретация[1] **машинных действий.** Конфигурация $\mathrm{Int}(C_{M_2}/M_1)$, моделирующая на некоторой машине M_1 вычисления из конфигурации C_{M_2} машины M_2, называется интерпретатором второй машины на первой: при вычислении из этой конфигурации машина M_1 выдает, т. е. получает определенным образом закодированные результаты, которые выдала бы машина M_2 при вычислении из конфигурации C_{M_2}, и, может быть, дополнительную информацию.

Интерпретаторы, выдающие промежуточные результаты вычисления, полезны при отладке программ (обычно они моделируют машину на самой себе), а при проектировании новой машины они дают возможность заранее приготовить некоторые программы для массовых вычислений и обслуживания работы будущей машины, а также проверить, как будут работать ее устройства в ситуациях, которые удается предвидеть. Не менее важно теоретическое значение интерпретации. Так, описание алгоритма в терминах некоторой алгоритмической модели сводят к его описанию в виде конфигурации машины Тьюринга при помощи построения интерпретатора модели на последней. В частности, приведенное в § 5.2 доказательство возможности решить любую алгоритмически разрешимую задачу на одной и той же универсальной машине Тьюринга U состояло в указании способа интерпретации произвольной машины Тьюринга T

---

[1] Здесь и далее термины «интерпретация» и «интерпретатор» используются в программистском смысле, не имеющем отношения к тому смыслу, в котором эти понятия использовались в гл. 6 и 7.

на машине U. Мы будем заниматься интерпретацией для установления связи между сложностями данной задачи относительно разных машин и разных задач.

Интерпретация элементарных действий. Процесс вычисления из любой конфигурации C_{M_2} машины M_2 является последовательностью шагов $sp(1), sp(2), ..., sp(i)...$ Соответственно процесс интерпретации можно осуществить как последовательность многошаговых операций $step(1)$, $step(2), ..., step(i)...$ интерпретации шагов машины M_2. Если интерпретирующая машина M_1 является машиной с произвольным доступом к памяти, то интерпретацию каждого шага может осуществлять одна и та же подпрограмма $step(i)$, пользующаяся информацией о номере шага и расположении в памяти машины M_1 описания элементарного действия, которое должно выполняться на этом шаге. Блок-схема такого интерпретатора $Int(C_{M_2}/M_1)$ изображена на рис. 9.1.

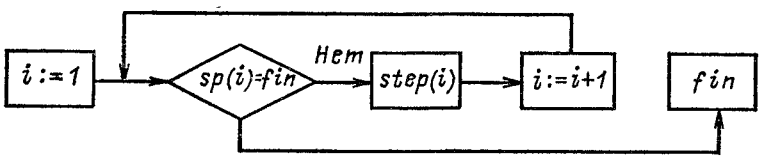

Рис. 9.1

Нас, в частности, будет интересовать интерпретация работы произвольной машины M на машине L: для любых конфигураций C_M машины M и натурального числа t будем строить интерпретатор $Int_t(C_M/L)$ вычисления первых t шагов, причем вычисления машины M будут полностью проинтерпретированы, если на t-м шаге или раньше M останавливается.

Интерпретатор $Int_t(C_M/L)$ можно конструировать в виде последовательности *макрооператоров* — подпоследовательностей логических операций. Эти макрооператоры выполняют функции блока $step(i)$ из блок-схемы 9.1 при $i = 1, 2, ..., t$. Программа $Int_t(C_M/L)$ — это последовательность макрооператоров $Int_t(C_M/L)$: $step(1)$; $step(2)$; ... ..., $step(t)$; fin. Макрооператоры $step(1)$, $step(2)$, ..., $step(t)$ осуществляют интерпретацию шагов вычисления, fin состоит из одной команды «конец».

Блок $step(i)$ из блок-схемы 9.1, в свою очередь, состоит из подблоков, интерпретирующих «микродействия», выпол-

няемые во время одного шага. Аналогичным образом каждый макрооператор $step(i)$ в программе $Int_t(C_M/L)$ элементарных логических операций ($i=1, 2, ..., t$) состоит из «более мелких» макрооператоров, имеющих те же функции (не пользуясь языком макрооператоров, практически невозможно описывать программы элементарных логических операций). Одно из представлений блока имеет вид $step(i)$: : $R(i)$; $comp(i)$; $W(i)$.

Блок $R(i)$ имитирует чтение информации из памяти машины L, необходимой для выполнения i-го шага, и настраивает следующие блоки на интерпретацию собственно элементарного действия и запись его результата; блок $comp(i)$ осуществляет это действие; $W(i)$ записывает результаты последнего по вычисленным ранее адресам.

Частичные конфигурации и их кодирование. Если интерпретируемая машина M из конфигурации C_M кончает работу через t шагов или раньше или интерпретируются только t первых шагов работы машины M, то можно полностью не описывать конфигурацию C_M, а ограничиться только состояниями читаемых на этих шагах s ячеек памяти ($s \leqslant \mu t$). Это основано на следующем очевидном утверждении.

Лемма 9.1. Пусть конфигурации C_M и C'_M машины M отличаются только состояниями устройств, не используемых, и ячеек памяти, не читаемых в процессе вычисления из первой конфигурации. Тогда машина M «не заметит разницы» между C_M и C'_M, т. е. в процессах вычисления из них перед шагами с одинаковыми номерами состояния всех устройств M будут одинаковыми, будут прочитаны одни и те же ячейки и произведены одни и те же действия.

Доказательство. На первом шаге для конфигураций C_M и C'_M читаются одни и те же ячейки, выполняются одни и те же действия. Следовательно, одинаковым образом меняются состояния одних и тех же ячеек памяти машины M и ее устройств. Таким образом, перед вторым шагом состояния «задействованных» на нем ячеек памяти и устройств для обеих конфигураций будут одинаковы. □

Назовем частичной конфигурацией C^{π}_M, определяющей данный процесс вычисления π, набор начальных состояний устройств и ячеек памяти, используемых машиной M в процессе π. Рассмотренным в лемме конфигурациям C_M и C'_M соответствует одна и та же конфигурация C^{π}_M.

Для интерпретации процесса π вычисления из частичной конфигурации $C^{\pi}_{M_2}$ машины M_2 на машине M_1, продолжаю-

щегося не более t шагов, или первых t шагов процесса, в машину M_1 нужно ввести описания состояний sd_1, sd_2, ..., sd_k устройств машины M_2 и sm_1, sm_2, ..., sm_s ее ячеек памяти, где $s \leqslant \mu t$, μ — максимальное число ячеек памяти машины M_2, которые могут понадобиться при выполнении одного ее шага. Пусть l_j $(j=1, 2, ..., k)$ — количества возможных состояний этих устройств и l_m — количество возможных состояний ячеек памяти, $l = \max\limits_{j=1,2,...,k} (l_j, l_m) + 2$. Упорядочим некоторым образом устройства машины M_2 и ее ячейки памяти и рассмотрим описания номеров команд и адресов данных в виде слов алфавита, состоящего не более чем из $l-2$ символов. Состояние каждого устройства и каждой ячейки памяти можно обозначить символом такого алфавита. Добавим еще разделители «;» для состояний устройств и ячеек памяти и «:» для номеров команд, адресов данных и параметров для вычисления номеров или адресов. Каждое описание адреса должно встретиться среди возможных состояний устройств, определяющих чтение из памяти; значит, его длина не превосходит k и оно должно быть повторено либо начальным состоянием устройств, либо данными, записанными в читаемых ячейках памяти. Частичная конфигурация $C^{\pi}_{M_2}$ может быть описана словом, имеющим длину не более чем $2(k+s)$, в алфавите, состоящем не более чем из l разных символов:

$$sd_1;\ sd_2;\ \ldots\ ;\ sd_k;\ A_1 : A_2 : \ldots : sm_1\, sm_2 \ldots ;$$
$$A_g : \ldots : sm_h \ldots sm_s.$$

Будем называть это слово описанием частичной конфигурации $C^{\pi}_{M_2}$ в самой машине M_2, а число символов в нем — длиной описания $C^{\pi}_{M_2}$ и обозначать $|C^{\pi}_{M_2}|$.

Можно считать, что интерпретирующая машина M_1 «понимает» символы 0 и 1. Каждому символу sym_g $(g=1, 2, ..., l)$, используемому для описаний частичных конфигураций $C^{\pi}_{M_2}$ в машине M_2, можно поставить в соответствие слово $\underbrace{0\,0\,\ldots\,0}_{g-1}\,1\,\underbrace{0\,\ldots\,0}_{l-g}$ длины l в алфавите 0,1. Заменим каждый символ таким подсловом и получим слово в том же двухсимвольном алфавите, также определяющее частичную конфигурацию $C^{\pi}_{M_2}$ — $\underbrace{c_1 c_2 \ldots c_l}\ \underbrace{c_{l+1} \ldots c_{2l}}\ \underbrace{c_{2l+1} \ldots \ldots}\ \ldots \underbrace{c_{rl}}$. Есть более экономный способ кодирования, в котором g-му символу соответствует изображение натурального числа g в двоичной позиционной системе с добавлением, если потребуется, сле-

ва символов 0 так, чтобы длины всех подслов были одинаковыми, но порядок длины описания конфигурации $C^{\pi}_{M_2}$ от этого не меняется.

Итак, длина описания конфигураций $C^{\pi}_{M_2}$ в алфавите 0,1, «понятном» любой интерпретирующей машине M и, в частности, машине элементарных логических операций L, имеет длину rl, где $r = |C^{\pi}_{M_1}| \leqslant 2(k+s) \leqslant 2(k+1)\mu t = A't$.

Полиномиальная интерпретируемость. Пусть для любых частичной конфигурации $C^{\pi}_{M_2}$ машины M_2 и натурального числа $t=1, 2...$ возможна интерпретация первых t шагов вычисления из частичной конфигурации $C^{\pi}_{M_2}$ интерпретатором $Int_t(C^{\pi}_{M_2}/M_1)$, состоящим из блоков (макрооператоров) $step(i)$, fin, и при каждом $i=1, 2, ..., m$ интерпретация i-го шага $sp(i)$ машины M_2 производится не более чем за $B'i^{C-1}$ шагов, где B' и C — положительные константы, причем $C \geqslant 1$. Тогда машина M_2 называется полиномиальным образом со степенью C, интерпретируемой на машине M_1.

Теорема 9.1. Пусть машина M_2 полиномиальным образом со степенью C интерпретируема на машине M_1, и задача p решается на машине M_2 из частичной конфигурации $C^{\pi}_{M_2}$ не более чем за t шагов, после чего машина останавливается. Тогда задача p может быть решена на машине M_1 не более чем за Bt^C шагов, где B — положительная константа, т. е. ее сложность относительно машины M_1 не больше Bt^C.

Доказательство. Будем производить интерпретацию процесса π решения задачи p на машине M_1 интерпретатором $Int_t(C^{\pi}_{M_2}/M_1)$, о котором идет речь в определении понятия полиномиальной интерпретируемости. Работа блоков (макрооператоров) $step(i)$ при $i=1, 2, ..., t$ требует не более чем

$$B' \cdot 1^{C-1} + B' \cdot 2^{C-1} + \ldots + B' t^{C-1} \leqslant$$

$$\leqslant B' \int_1^{t+1} \tau^{C-1} d\tau = \frac{B'}{C}((t+1)^C - 1) <$$

$$< \frac{B'}{C}(2t)^C = \frac{B' \cdot 2^C}{C} t^C = B'' t^C$$

шагов. Блок (макрооператор) fin состоит из конечного количества действия D (одного действия может не хватить:

у современных машин конец работы программы — это ряд действий).

Общее количество шагов работы интерпретатора $Int_t(C^{\pi}_{M_2}/M_1)$ не больше чем $B''t^c+D\leqslant(B''+D)t^c=Bt^c$, так как $C\geqslant 1$ и $t\geqslant 1$. Работа машины M_2 из частичной конфигурации $C^{\pi}_{M_2}$ продолжается не более t шагов, после чего машина останавливается. Значит, она будет полностью проинтерпретирована, и не более чем за Bt^c шагов на машине M_1 будет решена рассматриваемая задача p. Таким образом, сложность последней относительно машины M_1 не больше, чем Bt^c. □

Перейдем к конкретным машинам M_1 и M_2. Прежде всего рассмотрим интерпретацию произвольной машины Тьюринга T элементарными логическими операциями. При этом будут продемонстрированы основные приемы конструирования интерпретаторов.

Ассемблер элементарных логических операций. Пусть T — произвольная машина Тьюринга. Интерпретаторы $Int_t(C^{\pi}_T/L)$ — это программы, а при создании любых программ легче сначала определить, какие надо производить действия, а потом — адреса ячеек памяти, где будут находиться данные для них и куда будут записываться их результаты. Таким ячейкам будут сначала даны имена. Постараемся выбирать их так, чтобы они были выразительными, т. е. напоминали, о чем идет речь. Программу $Int_t(C^{\pi}_T/L)$ будем писать на ассемблере или автокоде элементарных логических операций, а затем укажем, как ее закодировать, т. е. перейти к указанному выше способу изображения элементарных логических операций. Ассемблер элементарных логических операций отличается от широко применяемых программистами ассемблеров для реальных машин следующим.

1. Одно и то же имя в разных командах программы можно заменять при кодировании разными адресами.

2. Алфавит символов, из которых состоят слова-имена, для большей выразительности имен не фиксирован в отличие от ассемблеров реальных машин.

3. Будем неформально пользоваться аналогиями, например заменять многоточием части программы, если из неопущенных команд видно, что оно заменяет.

Указанные отличия не имеют принципиального значения и введены ради удобства изложения.

Описание машины Тьюринга T. Как было указано

в § 5.2, машина Тьюринга T определяется следующими параметрами: количеством внутренних состояний n, количеством m символов алфавита, употребляемого для изображения информации в ячейках ленты машины T, и таблицей переходов Tab, которую изобразим несколько иначе, чем в § 5.2. Она будет состоять из элементов $Tab_{j,k,l}$ ($j=1, 2, ..., n$; $k=1, 2, ..., m$; $l=1, 2, ..., n+m+3$), принимающих значения 0 или 1. Если в таблице, приведенной в § 5.2, есть переход $st_j\ sym_k \to st_f\ sym_g\ move_h$ ($move_1=Lf$, $move_2=Im$, $move_3=Rh$), то $Tab_{j,k,f}=Tab_{j,k,n+g}=Tab_{j,k,n+m+h}=1$, а при остальных значениях l $Tab_{j,k,l}=0$.

Время работы интерпретатора Int_t (C_T^{π}/L), который мы будем конструировать, — Bt^2, причем константа B — не лучшая, а степень полиномиальной интерпретируемости машины Тьюринга на машине элементарных логических операций L равна 2. Сложнее устроен интерпретатор, выполняющий свою работу за $O(t \ln t)$ действий L-машины. Представление параметров процесса вычисления π машины T, принятое нами при описании интерпретатора на языке ассемблера, аналогично предложенному выше представлению таблицы переходов.

1. Представление ленты. Занумеруем ячейки ленты машины Тьюринга T так, чтобы в начале вычисления читающая и пишущая головки находились над ячейкой с номером 0. Так как на каждом шаге она либо переходит к ближайшей ячейке, находящейся справа или слева, либо остается на месте, на i-м шаге ($i=1, 2, ..., t$) головка находится над ячейкой с номером ν, где $-i+1 \leqslant \nu \leqslant i-1$, а на первых t шагах она находится над ячейками с номерами от $\nu_{min}(t)$ до $\nu_{max}(t)$, где $-t < \nu_{min}(t) \leqslant 0$, $0 \leqslant \nu_{max}(t) < t$. Существенной для работы интерпретатора Int_t (C_T^{π}/L) части ленты соответствуют идентификаторы $Ban_{j,k}$, где $j=0, \pm 1, \pm 2, ..., \pm(t-1)$, а $k=1, 2, ..., m$. Если в ячейке с номером j записан k-й символ sym_k, то $Ban_{j,h}=0$ при $h=1, 2, ..., k-1, k+1, ..., m$ и $Ban_{j,k}=1$. Можно иметь информацию только о «нетривиальной» части ленты, вне которой во всех ячейках стоят одинаковые символы пробела λ, это важно для конструирования «бесконечного» интерпретатора Int (T/L), но тогда макрооператоры $step$ (i) должны быть более сложными. Мы не будем заниматься такой интерпретацией.

2. Представление внутреннего состояния машины T. Текущее внутреннее состояние описывается

переменными stp_ν, а внутреннее состояние на следующем шаге, вычисляемое при помощи таблицы переходов Tab, — переменными stf_j ($j=1, 2, ..., n$). Если машина T находится в j-м состоянии, то $stp_1=stp_2=...=stp_{j-1}=stp_{j+1}=...$ $...=stp_n=0$, $stp_j=1$. Аналогичные значения имеют переменные stf_j.

3. Представления символов, читаемых машиной Тьюринга из ячеек ленты, над которой находится головка машины, и записываемых в эти ячейки в конце j-го шага. Рассматриваемые символы описываются соответственно переменными $sym\ r_j$ и $sym\ w_j$ ($j=1, 2, ..., m$) с аналогичными значениями, например, если прочитан j-й символ, то $sym\ r_1=sym\ r_2=...=sym\ r_{j-1}=sym_{j+1}=...=sym\ r_m=0$ и $sym\ r_j=1$.

4. Представление номера ячейки ленты, над которой расположена головка (текущего номера ν). Длина представления зависит от номера шага. На i-м шаге текущий номер ν описывается переменными ν_{-i+1}, $\nu_{-i+2}, ..., \nu_{-1}$, ν_0, $\nu_1, ..., \nu_{i-2}$, ν_{i-1}. Все они должны быть равны 0, кроме одной переменной ν_k, соответствующей ячейке машины T с номером k, над которой расположена головка. Так как перед началом счета головка стоит над нулевой ячейкой и после каждого шага сдвигается на одну ячейку вправо, влево или остается на месте, $-i+1 \leqslant k \leqslant i-1$.

5. Представление движения головки после шага. Оно описывается переменными Lf, Im, Rh. Если головка должна двинуться влево, то $Lf=1$, остаться на месте — $Im=1$, двинуться вправо — $Rh=1$. Остальные переменные равны 0.

Макрооператоры $step\,(i)$. Шаг работы машины Тьюринга T состоит в том, что она читает символ из ячейки ленты, определяет новое внутреннее состояние машины, символ на ленте и движение головки, пишет в ячейку ленты новый символ, производит движение головки и переходит в новое состояние. Аналогичные действия, но уже многошаговые, производят макрооператоры $step\ (i)$:

$$step\,(i): R\,(i);\quad comp\,(i);\quad W\,(i);\quad A\,(i+1).$$

Рассмотрим макрооператоры, выполняющие отдельные функции шага:

$$R\,(i): sym\, r_k := Ban_{j_\nu, k},\ k = 1, 2, ..., m.$$

Как без условных действий найти информацию о нужной

ячейке на ленте — переменные $Ban_{j,k}$ с тем значением j, для которого $v_j = 1$? Применим прием, который и в дальнейшем будет применяться многократно: нужные значения умножим на 1, ненужные — на 0 и все сложим (умножение и сложение могут быть арифметические — $\times$, $+$ — или логические — &, $\vee$; машина элементарных логических операций L, естественно, предпочтет вторые). Таким образом, нам предстоят вычисления по формулам:

$$sym\ r_k := \bigvee_{j=-i+1}^{i-1} (Ban_{j,k} \,\&\, v_j),\ k = 1, 2, \ldots, m.$$

Эту запись можно считать определением макрооператора $R(i)$ в нашем ассемблере. Она расписывается на элементарные логические операции следующим образом:

$$R(i): sym\ r_k := \bigvee_{j=-i+1}^{i-1} (Ban_{j,k} \,\&\, v_j): sym\ r_1 := Ban_{-i+1};$$

$$(j = -i+1,\ k = 1)$$

$$\left.\begin{array}{l} \rho := Ban_{-i+2,1} \,\&\, v_{-i+2}; \\ sym\ r_1 := sym\ r_1 \vee \rho; \end{array}\right\} (j = -i+2,\ k = 1)$$

$$\ldots$$

$$\ldots$$

$$\left.\begin{array}{l} \rho := Ban_{-1,1} \,\&\, v_{-1}; \\ sym\ r_1 := sym\ r_1 \vee \rho; \end{array}\right\} (j = -1,\ k = 1)$$

$$\left.\begin{array}{l} \rho := Ban_{0,1} \,\&\, v_0; \\ sym\ r_1 := sym\ r_1 \vee \rho; \end{array}\right\} (j = 0,\ k = 1)$$

$$\left.\begin{array}{l} \rho := Ban_{1,1} \,\&\, v_1; \\ sym\ r_1 := sym\ r_1 \vee \rho; \end{array}\right\} (j = 1,\ k = 1)$$

$$\ldots\ldots\ldots\ldots$$

$$\left.\begin{array}{l} \rho := Ban_{i-1,1} \,\&\, v_{i-1}; \\ sym\ r_1 := sym\ r_1 \vee p; \end{array}\right\} (j = i-1,\ k = 1)$$

$$sym\ r_2 := Ban_{-i+1,2} \,\&\, v_{-i+1};\quad (j = -i+1,\ k = 2)$$

$$\left.\begin{array}{l} \rho := Ban_{-i+2,2} \,\&\, v_{-i+2}; \\ sym\ r_2 := sym\ r_2 \vee \rho; \end{array}\right\} (j = -i+2,\ k = 2)$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

. . . . . .

$$\left.\begin{array}{l}\rho := Ban_{i-1,2} \,\&\, v_{i-1};\\ sym\ r_2 := sym\ r_2 \vee \rho;\end{array}\right\} (j = i-1,\ k = 2)$$

$$\left.\begin{array}{l}sym\ r_3 := Ban_{-i+1,3} \,\&\, v_{-i+1};\\ \rho := Ban_{-i+2,3} \,\&\, v_{-i+2};\end{array}\right\} (j = -i+1,\ k = 3)$$

. . . . . .

$$\left.\begin{array}{l}\rho := Ban_{i-1,m} \,\&\, v_{i-1};\\ sym\ r_m := sym\ r_m \vee \rho.\end{array}\right\} (j = i-1,\ k = m)$$

Формулы, которыми мы будем пользоваться в дальнейшем, расписываются аналогичным образом. Поэтому не давая такого развернутого описания макрооператоров, будем иногда выписывать только типичный «внутренний цикл» и указывать, как изменяются индексы. Нужно еще подсчитать число элементарных логических операций в макрооператоре $R(i)$. Оно равно $m(1+2\ (2i-2)) = 4mi-3m$.

Макрооператоры $comp(i)$, $W(i)$ и $A(i+1)$. Собственно элементарное действие машины Тьюринга T состоит в определении внутреннего состояния машины на следующем шаге, символа, который надо записать в ячейку ленты, и движения головки. Макрооператор $comp(i)$ интерпретирует определение этих параметров при помощи таблицы Tab. Пусть на i-м шаге вычисления из частичной конфигурации C_T^{π} машина Тьюринга T находится в j'-м состоянии ($1 \leqslant j' \leqslant n$) и читает с ленты k'-й символ ($1 \leqslant k' \leqslant m$). Тогда в процессе работы макрооператора $step\ (i)$ переменные $stp_j = 0 (j \neq j')$, $stp_{j'} = 1$, а после работы макрооператора $R(i)$ переменные $sym\ r_k = 0 (k \neq k')$, $sym\ r_{k'} = 1$. Если машина T должна перейти в f-е состояние, записать на ленту g-й символ и произвести h-е движение головки ($h = -1$ означает движение влево, $h = 0$ — неподвижность, $h = 1$ — движение вправо), то $Tab_{j',k',f} = Tab_{j',k',n+g} = Tab_{j'k',n+m+h+2} = 1$, а при остальных значениях l $Tab_{j',k'l} = 0$.

Внутреннее состояние машины T на следующем $(i+1)$-м шаге описывается переменными $stf_l = (l = 1, 2, ..., n)$, символ, который надо записать, — переменными $sym\ w_l (i = 1, 2, ..., m)$, движение головки — Lf, Im, Rh. Перенести в них табличные значения можно при помощи логических действий:

$$stf_l := \bigvee_{j=1}^{n} \bigvee_{k=1}^{m} (Tab_{j,k,l} \,\&\, stp_j \,\&\, sym\ r_k) = Tab_{j',k',l};$$

$$(l = 1, 2, \ldots, n);$$

$$sym\ w_l := \bigvee_{j=1}^{n} \bigvee_{k=1}^{m} (Tab_{j,k,n+l} \,\&\, stp_j \,\&\, sym\ r_k) = Tab_{j',k',n+l};$$

$$(l = 1, 2, \ldots, m);$$

$$Lf := \bigvee_{j=1}^{n} \bigvee_{k=1}^{m} (Tab_{j,k,n+m+1} \,\&\, stp_j \,\&\, sym\ r_k) = Tab_{j',k',n+m+1};$$

$$Im := \bigvee_{j=1}^{n} \bigvee_{k=1}^{m} (Tab_{j,k,n+m+2} \,\&\, stp_j \,\&\, sym\ r_k) = Tab_{j'k'n+m+2};$$

$$Rh := \bigvee_{j=1}^{n} \bigvee_{k=1}^{m} (Tab_{j,k,n+m+3} \,\&\, stp_j \,\&\, sym\ r_k) = Tab_{j',k',n+m+3}.$$

Расписать эти операции в виде последовательности элементарных логических действий можно образом, аналогичным примененному при конструировании макрооператора $R(i)$. Укажем здесь действия внутреннего цикла при данных значениях j, k и l. Удобно, чтобы l был внутренним индексом. Если j — самый внешний, то при изменении k выполняется действие

$$\rho 1 := stp_j \,\&\, sym\, r_k;$$

а затем для $l = 1, 2, \ldots, n, n+1, \ldots, n+m+3$

$$\rho 2 := Tab_{j,k,l} \,\&\, \rho 1;$$

$$Res_l := Res_l \,\&\, \rho 2.$$

Здесь при $l = 1, 2, \ldots, n$ Res_l — это stf_l, при $l = n+1, n+2, \ldots, n+m$ — $sym\ w_{l-n}$; $Res_{n+m+1} = Lf$, $Res_{n+m+2} = Im$, $Res_{n+m+3} = Rh$. При $j = 1$ и $k = 1$ во внутреннем цикле можно иметь одну команду —

$$Res_l := Tab_{1,1,l} \,\&\, \rho 1; \quad (l = 1, 2, \ldots, n + m + 3).$$

Общее количество элементарных логических операций в макрооператоре $comp\ (i)$ равно

$$(nm - 1)(2(n + m + 3) + 1) + n + m + 4 =$$

$$= nm(2n + 2m + 7) - n - m - 3,$$

т. е. зависит только от параметров машины Тьюринга T.

Макрооператор $W(i)$ состоит из элементарных логических операций:

$$\left.\begin{array}{l}\rho 1 := \neg\, v_j;\\ Ban_{j,k} := Ban_{j,k} \,\&\, \rho 1;\\ \rho 2 := sym\, w_k \,\&\, v_j;\\ Ban_{j,k} := Ban_{j,k} \vee \rho 2;\end{array}\right\}(k = 1, 2, \ldots, m)\left\{\begin{array}{c}(j = -i+1, -i+\\ +2, \ldots, -1,\\ 0, 1, \ldots, i-1).\end{array}\right.$$

Если на i-м шаге вычислений из конфигурации C_T^π головка машины Тьюринга T стоит над j'-й ячейкой ленты, то в процессе интерпретации этого шага $v_{j'}=1$ и $v_j=0 (-t+1 \leqslant j \leqslant t-1,\ j \neq j')$. Поэтому в результате действий макрооператора $W(i)$ все значения $Ban_{j,k}$ не изменятся, кроме $Ban_{j',k}$, которые станут равными $sim\ w_k$ $(k=1, 2, \ldots, m)$. Количество элементарных логических операций равно $2i-1+(2i-1)3m=(2i-1)(3m+1)=(6m+2)i-3m-1$.

Макрооператор $A(i+1)$ устанавливает новое внутреннее состояние, т. е. изменяет переменные stp_j $(j=1, 2, \ldots, n)$ и двигает головку, т. е. изменяет переменные $v_j (j=-i, -i+1, \ldots, i)$. Отметим, что на предыдущих шагах переменных v_{-i} и v_i не было. Значения stp_j просто делаются равными stf_j:

$$stp_j := stf_j; \quad (j = 1, 2, \ldots, n).$$

Если должен произойти сдвиг головки влево, то новые значения v_j должны стать равными старым v_{j+1}; если головка неподвижна, то значения v_j не изменяются, если она движется вправо, то v_j станут равны старым значениям v_{j-1}. Однако необходимо учитывать «краевые эффекты» — отсутствие на предыдущем шаге переменных $v_{\pm i}$. Учитывая еще значения переменных Lf, Im и Rh, можно доказать, что новые значения v_j определяются формулами:

$$v_j := (v_{j+1} \,\&\, Lf) \vee (v_j \,\&\, Im) \vee (v_{j-1} \,\&\, Rh); \quad (j = -i+2, -i+3, \ldots, i-2);$$

$$v_{-i+1} := (v_{-i+2} \,\&\, Lf) \vee (v_{-i+1} \,\&\, Im);$$

$$v_{-i} := (v_{-i+1} \,\&\, Lf);$$

$$v_{i-1} := (v_{i-1} \,\&\, Im) \vee (v_{i-2} \,\&\, Rh);$$

$$v_i := (v_{i-1} \,\&\, Rh).$$

Расписать эти операции в виде последовательности элементарных логических действий можно, например, следую-

щим образом:

$$v_{-i} := v_{-i+1} \,\&\, Lf;$$
$$v_{-i+1} := v_{-i+1} \,\&\, Im;$$
$$\delta_{-i+2} := v_{-i+1} \,\&\, Rh;$$

$$\left.\begin{array}{l}\beta_{j-1} := v_j \,\&\, Lf;\\ \gamma_j := v_j \,\&\, Im;\\ \delta_{j+1} = v_j \,\&\, Rh;\end{array}\right\} (j = -i+2, -i+3, \ldots, -1, 0, 1, \ldots, i-2)$$

$$\beta_{i-2} := v_{i-1} \,\&\, Lf;$$
$$\gamma_{i-1} := v_{i-1} \,\&\, Im;$$
$$v_i := v_{i-1} \,\&\, Rh;$$

$$\left.\begin{array}{l}v_j := \beta_j \vee \gamma_j;\\ v_j := v_j \vee \delta_j;\end{array}\right\} (j = -i+2, -i+3, \ldots, -1, 0, 1, \ldots, i-2)$$

$$v_{i-1} := \gamma_{i-1} \vee \delta_{i-1}.$$

Их количество равно $1+2+3(2i-3)+2+1+2(2i-3)+1=10i-7$. В макрооператоре $A(2)$ действия несколько иные —

$$v_{-1} := v_0 \,\&\, Lf;$$
$$v_0 := v_0 \,\&\, Im;$$
$$v_1 := v_0 \,\&\, Rh.$$

Однако их 3, т. е. тоже $10i-7$.

Общее количество элементарных логических операций в макрооператоре $A\ (i+1)$ равно $n+10i-7$, а во всем макрооператоре *step* $(i) - (4mi-3m) + (nm(2n+2m+7) - n - m - 3 + ((6m+2)i - 3m - 1) + 10i - 7 = (10m+12)i + nm(2n+2m+7) - 7m - n - 11 = B'i + C'$, где B' и C' — константы.

Кодирование программы элементарных логических действий. Мы могли бы занумеровать все переменные, которые вычисляются при работе интерпретатора $Int_t\ (C_T^{\pi}/L)$, и получить программу, которую машина элементарных логических операций «поняла» бы и выполнила. При этом в каждой команде либо некоторой переменной присваивалось яв-

но указанное значение, либо оно вычислялось в результате операции с переменными, значения которых были ранее определены. Таким образом, составленная нами программа предопределяет результаты своих вычислений. Однако она не удовлетворяет требованию однократной записи в ячейки памяти. Мы уже указывали, что такую программу можно перекодировать так, чтобы запись информации в каждую ячейку памяти производилась не более одного раза. Сейчас мы укажем способ перекодирования.

Наша программа до перекодирования имеет вид

$$i: y_i := U_i; \quad (i = 1, 2, \ldots, \tau - 1)$$

$$\tau: \text{«конец»};$$

где U_i — это выражения 0, 1, u_i, $\neg u_i$, $u_i \& v_i$, $u_i \vee v_i$, y_i, u_i, v_i — номера введенных нами переменных (в том числе вспомогательных — ρ, ρ1, ρ2, используемых для промежуточных результатов); $\tau \leqslant Bt$ — длина программы, на 1 большая, чем количество элементарных логических операций в ней. Для каждых номера шага i работы машины Тьюринга T от первого до t-го и номера переменной $y = 1, 2, \ldots, N$ определим величину $\psi(i, y)$ как номер шага, предшествовавшего i-му, когда в последний раз было присвоено значение переменной y:

$$\psi(i, y) = \max_{j<i} (j \mid j : y := U_j).$$

Если до j-го шага переменной y не было присвоено никакого значения, то $\psi(i, y) = 0$. Не важно, сколько действий нужно для определения функции $\psi(i, y)$, надо только установить, что «правильный» интерпретатор $Int_t(C_T^{\pi}/L)$ может быть записан.

Правила перекодирования определяются следующей таблицей, где слева приведен вид команды до перекодирования, а справа — после него:

$$i_1 : y := 'c'; \to i_1 ! i_1 := 'c';$$

$$i_2 ! y := u; \quad \to i_2 : i_2 ! = \psi(i_2, u);$$

$$i_3 : y := \neg u; \quad \to i_3 ! i_3 ! = \neg \psi(i_3, u);$$

$$i_4 : y := u \& v; \quad \to i_4 : i_4 ! = \psi(i_4, u) \& \psi(i_4, v);$$

$$i_5 : y := u \vee v; \quad \to i_5 : i_5 ! = \psi(i_5, u) \vee \psi(i_5, v).$$

Так как в правых частях (после знака присваивания :=) команд исходной программы имеются только номера переменных y, встречавшиеся в левых частях предыдущих

команд (перед знаком присваивания :=, т. е. им уже были присвоены какие-нибудь значения), после перекодирования в новых правых частях команд все номера будут меньше номеров самих команд (и равных им номеров переменных в левой части), но больше 0.

Итак, мы закончили конструирование макрооператора $Int_t(C_T^{\pi}/L)$. Читателю предоставляется доказать по индукции, что элементарные логические операции исходной и перекодированной программ с одинаковыми номерами дают одинаковые результаты, хотя они записываются в разные ячейки памяти (для доказательства справедливости индуктивного предположения, когда команды имеют номер 1, нужно воспользоваться тем, что это — команды вида $y:=$ $='c'$).

Теорема 9.2. Если трудоемкость задачи p относительно машины Тьюринга T не больше t, то ее можно решить при помощи не более чем $Bt^2+Ct=0$ (t^2) элементарных логических операций[1].

Доказательство. Если трудоемкость задачи p относительно машины Тьюринга T не больше t, то из некоторой частичной конфигурации C_T^{π} машина T не более чем за t шагов решит задачу p и остановится. При доказательстве теоремы 9.1 уже говорилось, что в этом случае машина L при помощи интерпретатора $Int_t(C_T^{\pi}/L)$ тоже решит задачу p. В частности, машина элементарных логических операций L решит ее при помощи построенного выше интерпретатора $Int_t(C_T^{\pi}/L)$. Подсчитаем количество элементарных логических операций (последнее действие машины L — выполнение команды τ: «конец»; — мы не считаем логической операцией):

$$(B'\cdot 1 + C') + (B'\cdot 2 + C') + \ldots + (B'\cdot t + C') =$$

$$= \frac{B'\,t\,(t+1)}{2} + C'\,t = \frac{B'}{2}\,t^2 + \left(\frac{B'}{2} + C'\right)t =$$

$$= Bt^2 + Ct = O\,(t^2),$$

где $B=B'/2$, $C=B'/2+C'$. □

[1] Напомним смысл обозначений $O(f)$ и $o(f)$, принятых в анализе: $g(n)=O\,(f(n))$, если $\sup\limits_{n=1,2,\ldots}\left|\dfrac{g(n)}{f(n)}\right|=\text{const}>0$. В этом случае говорят, что $g(n)$ — величина того же порядка, что и $f(n)$, $g(n)=o(f(n))$, если $\lim\limits_{n\to\infty}\dfrac{g(n)}{f(n)}=0$. В этом случае $g(n)$ называют величиной более низкого порядка, чем $f(n)$.

Полиномиальная интерпретируемость машин друг на друга. Аналогичным образом доказывается, что машина с произвольным доступом к памяти и «полным» набором команд — конечным набором операторов вычисления функций от конечного же количества двоичных переменных, условными и безусловными переходами и индексированными командами — полиномиальным образом интерпретируема на машине элементарных логических операций. Однако в этом случае степень больше трех (сложнее всего интерпретация индексированных команд, особенно адресов, которые могут появиться в результате индексации). Обратная полиномиальная интерпретируемость машины элементарных логических операций L на машине с произвольным доступом к памяти тривиальна, так как в системе команд последней машины есть все команды первой.

Так как машина с произвольным доступом к памяти полиномиальным образом интерпретируема на машине элементарных логических операций L, для доказательства ее полиномиальной интерпретируемости на некоторой машине Тьюринга PU достаточно доказать полиномиальную интерпретируемость машины L на машине PU. Кроме того, этим будет доказана полиномиальная интерпретируемость произвольных машин Тьюринга T на машине PU.

Теорема 9.3. L-машина полиномиальным образом интерпретируется на машине Тьюринга. Для доказательства нужна определенная техника программирования машин Тьюринга, и мы его опускаем.

9.2. КЛАССЫ ТРУДОЕМКОСТИ КОМБИНАТОРНЫХ ЗАДАЧ

Размерность комбинаторных задач. Вопрос о размерности комбинаторных задач рассмотрим на примере задачи *PERT* (задачи сетевого или календарного планирования), которая формулируется следующим образом.

Комплекс работ описывается сетью — ориентированным графом с двумя отмеченными вершинами — полюсами (вообще говоря, количество полюсов в сети может быть любым, но для «классической» задачи *PERT* их два). Каждая вершина v сети N соответствует определенному событию в процессе выполнения работ из данного комплекса: первый полюс — началу процесса, второй его концу, каждой работе соответствуют события ее начала и конца, могут быть и другие события. Естественно, из первого полюса до-

стижимы все вершины сети событий N, а второй полюс достижим из всех вершин.

Пусть $v_1, v_2, \ldots, v_n$ — вершины сети N, причем v_1 — начало процесса, а v_n — его конец. Если v_i — начало некоторой работы, v_j — ее конец, а t_{ij} — продолжительность, то сроки событий v_i и v_j связаны одним из соотношений: $T_j = T_i + t_{ij}$, если продолжительность работы строго задана, или $T_j \geqslant T_i + t_{ij}$, если работа не может продолжаться меньше, чем t_{ij} единиц времени, или T_i — не начало работы, а событие, состоящее в том, что последнюю можно начинать. Если v_j — начало некоторой работы, а v_i — конец работы, которая обязательно должна быть окончена, чтобы первую можно было начать, то $T_j \geqslant T_i$. В дальнейшем равенства $T_j = T_i + t_{ij}$ заменим неравенствами $T_j \geqslant T_i + t_{ij}$ и $T_i \geqslant T_j - t_{ij}$.

Может случиться, что следующую работу нельзя начинать сразу после конца предыдущей, например, после укладки бетона надо дать ему застыть. Тогда обозначим t_{ij} — время ожидания, и связь между сроками событий принимает уже привычный вид $T_j \geqslant T_i + t_{ij}$. К такому же виду можно привести связь между концом предшествующей работы v_i и началом следующей v_j, когда времени ожидания нет: надо только положить $t_{ij} = 0$ (к тому же виду приводятся некоторые другие условия выполнения комплекса работ, причем значения t_{ij} могут даже оказаться отрицательными). Итак, каждому неравенству $T_j \geqslant T_i + t_{ij}$ соответствует ориентированное ребро (v_i, v_j) сети N. Обычно еще задается срок начала процесса T_1, и требуется найти сроки T_j $(j = 2, \ldots, n)$, удовлетворяющие всем неравенствам и минимальности срока конца процесса T_n.

Мы описали общие условия массовой задачи. Каждый ее вариант задается параметрами: число событий n, списком ребер (v_i, v_j) сети N и соответствующими длительностями t_{ij}. Эти варианты имеют разную размерность, с ростом которой естественно ждать увеличения сложности задачи. Однако понятие размерности определяют по-разному. Проще всего считать размерностью число вершин n в сети N, но с увеличением количества дуг сложность задачи тоже возрастает, да и увеличение количества значащих цифр в параметрах t_{ij} приводит к росту числа элементарных логических операций при интерпретации действий с ними.

Необходим глобальный параметр размерности, который к тому же годился бы для любых массовых задач. Если «содержательный» смысл такой задачи известен, то любой

ее вариант можно описать одним словом: нужно алгоритмически определить порядок параметров и их идентификаторы, если без последних нельзя обойтись, выбрать способы описания параметров и их идентификаторов в виде подслов, состоящих из символов некоторого алфавита, добавить к последнему символы-разделители, наконец, записать значения всех параметров с необходимыми идентификаторами и разделителями в порядке, который был алгоритмически определен. С учетом информационных возможностей многосимвольного алфавита, длина полученного слова множится на логарифм числа символов в нем по основанию 2. Произведение естественно называть *информационной сложностью* варианта массовой задачи. Как было указано в § 9.1, его иногда называют размерностью варианта.

Таким образом, размерность варианта массовой задачи зависит от способа его описания. Дать независимое определение затруднительно. Кроме того, метод описания конкретных вариантов следует считать существенной частью задачи: от него зависит трудоемкость решения, например, существуют описания, в которых явно указан ответ.

Как описать вариант задачи *PERT*? Сеть N можно задать любым способом, которым задаются графы (см. § 4.1), но проще всего — квадратной матрицей смежности $\|\varepsilon_{ij}\|_{i=1}^{n}{}_{j=1}^{n}$. Кроме того, нужно задать соответствующие длительности. Будем описывать их тоже квадратной матрицей $\|t_{ij}\|_{i=1}^{n}{}_{j=1}^{n}$ с элементами, принимающими числовые значения. Если $\varepsilon_{ij}=0$, т. е. ориентированного ребра (v_i, v_j) в сети N нет, то значение t_{ij} не играет роли, но для упрощения формул, которые понадобятся в дальнейшем, мы рассматриваем такую переменную, принимающую числовые значения. Длительности t_{ij} $(i, j=1, 2, ..., n)$ будем считать целыми числами: их измеряют с точностью до некоторой единицы времени (чаще всего — рабочего дня). Порядок вершин сети N, определяющий матрицы $\|\varepsilon_{ij}\|$ и $\|t_{ij}\|$, — почти произвольный; важно только, что v_1 — это начало процесса, а v_n — его конец. Количество n вершин сети N можно определить по числу остальных параметров; его следует задать лишь для возможности контроля.

Как описать решение рассматриваемой задачи? Оно состоит из сроков событий T_2, T_3, ..., T_n. Срок начала процесса T_1 можно не указывать: он должен быть равен 0, но опять же для упрощения формул будем считать, что реше-

ние задачи описывается последовательностью чисел T_1, T_2, ..., T_n. Так как мы рассматриваем целые значения t_{ij}, можно потребовать, чтобы сроки T_i тоже были целыми числами: если какое-нибудь решение задачи *PERT* существует, то существует и решение с целыми значениями сроков.

Однако решения может и не быть. Когда в сети N есть ориентированный цикл Z $(v_{i_1}, ..., v_{i_s}, v_{i_1})$ с положительной суммой длительностей $\sum_{k=2}^{s} t_{i_{k-1}i_k} + t_{i_s i_1}$, не существует сроков $T_{i_k} (k=1, ..., s)$, удовлетворяющих всем неравенствам

$$T_{i_k} \geqslant T_{i_{k-1}} + t_{i_{k-1}i_k} (k = 2, 3, \ldots, s);$$
$$T_{i_1} \geqslant T_{i_s} + t_{i_s i_1}.$$

Действительно, складывая эти неравенства, получаем

$$\sum_{k=1}^{s} T_{i_k} \geqslant \sum_{k=1}^{s} T_{i_k} + \sum_{k=2}^{s} t_{i_{k-1}i_k} + t_{i_s i_1},$$

откуда следует

$$\sum_{k=2}^{s} t_{i_{k-1}i_k} + t_{i_s i_1} \leqslant 0,$$

что противоречит положительности суммы длительностей, соответствующих ребрам цикла Z. Кроме того, если даже существуют значения $T_1, T_2, \ldots, T_n$, удовлетворяющие неравенствам $T_j \geqslant T_i + t_{ij}$ при $\varepsilon_{ij}=1$ $(i, j=1, 2, \ldots, n)$, но в сети N нет ориентированной цепи $L(v_1, \ldots, v_n)$, соединяющей начало процесса v_1 с его концом v_n, то и при условии $T_1 = 0$ значение T_n не имеет минимума.

Отметим еще, что решение может быть не единственным: пусть сеть N состоит из трех вершин v_1, v_2, v_3 и трех ориентированных ребер (v_1, v_2), (v_1, v_3) и (v_2, v_3), которым соответствуют неравенства: $T_2 \geqslant T_1 + 3$; $T_3 \geqslant T_1 + 8$; $T_3 \geqslant T_2 + 2$. Тогда сроки T_1 и T_3 определяются однозначно — $T_1 = 0$, $T_3 = 8$, но T_2 может иметь любое значение от 3 до 6 включительно.

Обозначим y_0 двоичную переменную, которая будет равна 1, если решение рассматриваемого варианта задачи *PERT* существует, и 0 в противном случае. Значение этой переменной должно быть указано в ответе для варианта. Если $y_0=0$, то переменные $T_1, T_2, \ldots, T_n$ не имеют смысла. Им могут быть присвоены любые значения или даже не

присвоены никакие (во многих алгоритмах решения задачи *PERT* какие-то значения всем или по крайней мере некоторым переменным T_i присваиваются всегда). При $y_0=1$ значения $T_1, T_2, \ldots, T_n$ должны удовлетворять условиям задачи.

Как описать задачу *PERT* в целом? Ее условия являются предикатами, зависящими от введенных нами переменных:

$$P_{ij}(\varepsilon_{ij}, t_{ij}, T_i, T_j) = [\varepsilon_{ij}=0] \vee [T_j \geqslant T_i + t_{ij}],$$
$$(i, j = 1, 2, \ldots, n);$$
$$P_1(T_1) = [T_1 = 0];$$
$$P_n(\varepsilon_{11}, \varepsilon_{12}, \ldots, \varepsilon_{1n}, \varepsilon_{21}, \ldots, \varepsilon_{nn}, t_{11}, \ldots$$
$$\ldots, t_{1n}, t_{21}, \ldots, t_{nk}, T_n) = P_n(\{\varepsilon_{ij}, t_{ij}\}_{i=1\,j=1}^{n\quad n}, \{T_i\}_{i=1}^{n}) =$$
$$= P_1(T_1) \;\&\; \mathop{\&}_{i=1}^{n} \mathop{\&}_{j=1}^{n} P_{ij}(\varepsilon_{ij}, t_{ij}, T_i, T_j);$$
$$P_{n,min}(\{\varepsilon_{ij}, t_{ij}\}_{i=1\,j=1}^{n\quad n}, T_n) = \forall\, \theta_1 \theta_2 \ldots \theta_n \in R$$
$$P(\{\varepsilon_{ij}, t_{ij}\}_{i=1\,j=1}^{n\quad n} \{\theta_i\}_{i=1}^{n}) \rightarrow [\theta_n \geqslant T_n]$$

— это условие минимальности продолжительности $T_n - T_1 = T_n$ выполнения всего комплекса работы;

$$P_{n,ex}(\{\varepsilon_{ij}, t_{ij}\}_{i=1\,j=1}^{n\quad n}) = \exists\, T_1, T_2, \ldots$$
$$\ldots, T_n \in R \quad P_n(\{\varepsilon_{ij}, t_{ij}\}_{i=1\,j=1}^{n\quad n} \{T_i\}_{i=1}^{n})$$

— это условие существования допустимого календарного плана.

Таким образом, P_n, $P_{n,ex}$ и $P_{n,min}$ — это не предикаты, у которых количество аргументов должно быть фиксировано, а системы предикатов. Их арность, т. е. количество аргументов, определяется целым параметром n. Его и считают семантической размерностью варианта массовой задачи *PERT*. Система предикатов $P_n(\{\varepsilon_{ij}, t_{ij}\}_{i=1\,j=1}^{n\quad n}, \{T_i\}_{i=}$ $(n=1, 2, \ldots)$ описывает саму массовую задачу.

В общем случае имеются два параметра семантической размерности n_1, n_2 (отличающейся от ранее введенной размерности — приведенной длины описания варианта задачи). Массовая задача — это система предикатов $P_v^{n_1,n_2}(\{X_i\}_{i=1}^{\varphi(n_1)}, \{Y_j\}_{j=1}^{\psi(n_2)})$. Совокупность переменных $\bar{X}(X_1, \ldots, X_{\varphi(n_1)})$ является описанием варианта данной массовой задачи, $\bar{Y}(Y_1, \ldots, Y_{\psi(n_2)})$ — описание гипотетического ответа,

и если $P_v^{n_1 n_2}$ $(\bar{X}, \bar{Y})$ истинно, то Y — в самом деле ответ. Количества $\varphi(n_1)$ переменных X_i, описывающих вариант, и $\psi(n_2)$ переменных Y_j, описывающих ответ, — заданные функции семантических размерностей n_1 и n_2. В зависимости от них заданы также области значений каждой переменной и сами предикаты $P_v^{n_1 n_2}(X, Y)$.

Значения переменных $n_1, X_1, \ldots, X_{\varphi(n_1)}, n_2, Y_1, \ldots, Y_{\psi(n_2)}$ описываются словами некоторого алфавита. Добавим к последнему разделители $'$, $'$ и $';'$. Тогда вариант данной массовой задачи можно описать словом $n_1, X_1, X_2, \ldots, X_{\varphi(n_1)}$, состоящим из подслов, описывающих переменные, и разделителей. Произведение длины этого слова на двоичный логарифм количества l символов употребляемого алфавита (в их число входят разделители) мы назвали раньше размерностью варианта данной задачи. В отличие от семантической размерности ее можно назвать информационной размерностью. Аналогично конструируется слово $n_2, Y_1, \ldots, Y_{\psi(n_2)}$ — гипотетический или настоящий ответ.

Комбинаторные задачи перебора. Будем рассматривать такие комбинаторные задачи, у которых для любого описания варианта длина описаний (т. е. размерность) возможных ответов ограничена значением $S(n_1, n_2)$, где S — это рекурсивная функция. Можно считать, что длины описания варианта и ответа ограничены одним и тем же числом S. Если l — мощность алфавита, то этому ограничению удовлетворяет не более чем $l+l^2+\ldots+l^S=\frac{l^{S+1}-l}{l-1}$ возможных ответов. В этом случае при заданном варианте ответы $\bar{Y}$ можно перебирать по очереди и для каждого из них выяснять, истинен ли предикат $P_v^{n_1 n_2}$ $(\bar{X}, \bar{Y})$. Задача называется *задачей перебора,* если для любого ее варианта размерность ответа ограничена и предикат $P_v^{n_1 n_2}$ $(\bar{X}, \bar{Y})$ разрешим. Очевидно, что задача перебора всегда разрешима, хотя, быть может, и с очень высокой сложностью.

Комбинаторная проблема Поста (§ 6.4) не является задачей перебора: ее предикат $P_v^{n_1, n_2}$ разрешим, но размерность ответа не ограничена. Для проблемы остановки ответ можно понимать по-разному: как двоичную переменную («да» или «нет») или как протокол (последовательность конфигураций) работы машины. В первом случае предикат $P_v^{n_1, n_2}$ неразрешим; во втором случае он разрешим, но не ограничена размерность ответа.

Займемся приведением массовых задач перебора к некоторому стандартному виду. Пусть S фиксировано. Можно уравнять длины описаний вариантов и ответов, добавив к алфавиту пустой символ * и поставив в начале слишком коротких слов недостающее количество таких символов. Затем мы перейдем к двухсимвольному алфавиту 0,1 способом, который уже не раз рассматривался в этой книге. Информационная размерность почти не изменится: вместо $S \log_2 l$ она станет равной $m = S\lfloor \log_2(1+l) \rfloor$, где $\lfloor u \rfloor$ — наименьшее целое число, удовлетворяющее неравенству $\lfloor u \rfloor \geqslant u$. Таким образом, допустимые варианты и гипотетические ответы можно считать последовательностями m двоичных переменных — соответственно $\bar{x} = (x_1, ..., x_m)$ и $\bar{y}(y_1, ..., y_m)$ можно рассматривать только массовые задачи перебора, определяемые предикатами $P(x_1, x_2, ..., x_m, y_1, y_2, ..., y_m)$, зависящими от двоичных переменных.

Примеры. Один пример — задача *PERT* — уже был рассмотрен. Другой пример — определение натурального числа $n+1$, следующего за данным числом n. Если число n изображено числом единиц: 1 — один, 11 — два, 111 — три и т. д., то к такой последовательности надо прибавить 1. Однако размерность задачи при таком способе изображения переменных слишком велика. Обычно ее рассматривают для чисел n, изображенных в двоичной позиционной системе. Предикаты массовой задачи перебора $N^m(x_1, x_2, ..., x_m, y_1, y_2, ..., y_m)$, где двоичные переменные $x_1, x_2, ..., x_m$ представляют данное m-разрядное число n, а двоичные переменные $y_1, y_2, ..., y_m$ — искомое число $n+1$, можно определить индуктивно вместе с предикатами тождества

$$E^m(x_1, x_2, ..., x_m, y_1, y_2, ..., y_m):$$

$$E^1(x_1, y_1) = (x_1 \& y_1) \vee (\neg x_1 \& \neg y_1); \quad N^1(x_1, y_1) = \neg x_1 \& y_1;$$

$$E^m(x_1, x_2, ..., x_m, y_1, y_2, ..., y_m) =$$

$$= E^{m-1}(x_1, x_2, ..., x_{m-1}, y_1, y_2, ..., y_{m-1}) \&$$

$$\& ((x_m \& y_m) \vee (\neg x_m \& \neg y_m));$$

$$N^m(x_1, x_2, ..., x_m, y_1, y_2, ..., y_m) =$$

$$= (\neg x_m \& y_m \& E^{m-1}(x_1, x_2, ..., x_{m-1}, y_1, y_2, ...$$

$$... , y_{m-1})) \vee (x_m \& \neg y_m \& N^{m-1}(x_1, x_2, ..., x_{m-1}, y_1, y_2, ...$$

$$... , y_{m-1})) \quad (m = 2, 3, ...).$$

То, что в этих формулах по существу использовано понятие следующего натурального числа (за $m-1$ следует m), не создает порочного круга: при $m=2, 3$ можно выписать явные формулы, а $m \geqslant 4$ изображаются числами более короткой длины $\lfloor \log_2 m \rfloor$, для которых понятие следующего натурального числа было ранее определено. Отметим еще, что при заданных $x_1, x_2, \ldots, x_m$ существуют единственные значения $y_1, y_2, \ldots, y_m$ — ответ рассматриваемой массовой задачи, кроме случая, когда $x_1=x_2=\ldots=x_m=1$. Увеличив размерность задачи на 1 и произведя стандартное отображение варианта в вариант большей размерности: $x_1=0$; $x_i=x_{i-1}$ $(i=2, 3, \ldots, m+1)$, можно решить задачу и в этом случае.

Следующие примеры будем рассматривать на «содержательном» уровне, не увлекаясь излишней формализацией.

Задача о камнях. Даны n камней с целыми весами $x_1, x_2, \ldots, x_n$. Можно ли выбрать некоторое количество из них так, чтобы сумма их весов была равна данному числу M? Введем двоичные переменные $y_1, y_2, \ldots, y_n$. Наша задача — это уравнение

$$x_1 y_1 + x_2 y_2 + \ldots + x_n y_n = M.$$

Таким образом,

$$P_v^n(x_1, x_2, \ldots, x_n, M, y_1, y_2, \ldots, y_n) = [x_1 y_1 + x_2 y_2 + \ldots + x_n y_n = M].$$

У варианта задачи о камнях либо нет решения, либо размерность решения меньше размерности описания варианта.

Многомерная задача о камнях. Пусть камни имеют несколько параметров: веса $x_1^1, x_2^1, \ldots, x_n^1$, объемы $x_1^2, x_2^2, \ldots, x_n^2$ и цены в разных валютах или в разные времена $x_1^i, x_2^i, \ldots, x_n^i$ $(i=3, \ldots, k)$. Требуется найти значения двоичных переменных $y_1, y_2, \ldots, y_n$, чтобы выполнялись равенства

$$x_1^i y_1 + x_2^i y_2 + \ldots + x_n^i y_n = M_i \, (i = 1, 2, \ldots, k).$$

Значения x_j^i и M_i — целые числа, обычно неотрицательные.

Часто рассматривают задачу с неравенствами

$$\sum_{j=1}^{n} x_j^i y_j \leqslant M_i \; (i \in P^{<} \subset \{1, 2, \ldots, l\});$$

$$\sum_{j=1}^{n} x_j^i y_j \geqslant M_i \left(i \in P^{\geqslant} = \{1, 2, \ldots, l\} \setminus P^{\leqslant}\right)$$

(если не требовать неотрицательности x_j^i, то все условия можно привести к виду $\sum_{j=1}^{n} x_j^i y_j \leqslant M_i$). Увеличив размерность задачи (можно показать, что полиномиальным образом), ее можно свести к задаче с равенствами. Добавим двоичные переменные $v_{ij} (i=1, 2, \ldots, l, \ j=n+1, \ldots, n+L\lfloor \log_2 M_i \rfloor)$ и для $j=n+1$ положим $x_j^i = 2^{j-n-1}$. Тогда наша система с неравенствами эквивалентна задаче

$$\sum_{j=1}^{n} x_j^i y_j + \sum_{j=n+1}^{n+L\lfloor \log_2 M \rfloor} x_j^i y_{ij} = M_i \left(i \in P^{\leqslant}\right);$$

$$\sum_{j=1}^{n} x_j^i y_j + \sum_{j=n+1}^{n+L\lfloor \log_2 M \rfloor} x_j^i v_{ij} = M_i + \sum_{j=n+1}^{n+L\lfloor \log_2 M \rfloor} x_j^i \left(i \in P^{\geqslant}\right).$$

Действительно, из неравенства $\sum_{j=1}^{n} x_j^i y_j \leqslant M_i$ следует, что $\sum_{j=1}^{n} x_j^i y_i = M_i - Q_i$, где Q_i — неотрицательное число, не превосходящее M_i. Пусть $\varepsilon_{i, \lfloor \log_2 M_i \rfloor}$, $\varepsilon_{i, \lfloor \log_2 M_i \rfloor - 1}$, ..., ε_2, ε_1 — представление Q_i в двоичной позиционной системе. Это значит, что

$$Q_i = \varepsilon_{i1} + 2\varepsilon_{i2} + 2^2 \varepsilon_{i3} + \ldots + 2^{\lfloor \log_2 M_i \rfloor - 1} \varepsilon_{i, \lfloor \log_2 M_i \rfloor}$$

$$(i \in P^{\leqslant}).$$

Положим $v_{ij} = \varepsilon_{i(j-n)}$, $x_j^i = 2^{j-n-1}$ $(n < j \leqslant n + \lfloor \log_2 M_i \rfloor)$. Тогда будут выполняться соответствующие равенства. Для остальных неравенств $\sum_{j=1}^{n} x_j^i y_j = M_i + Q_i$, причем $Q_i \leqslant \sum_{j=1}^{n} x_j^i$ (предполагаем, что $M_i \geqslant 0$). Рассмотрим аналогичное представление числа Q_i и положим $v_{ij} = 1 - \varepsilon_{i(j-n)} (j = n+1, \ldots, n + \left\lfloor \log_2 \sum_{j=1}^{n} x_j^i \right\rfloor)$. Тогда будут выполняться оставшиеся равенства. Тот факт, что из выполнения этих равенств следует выполнение исходных неравенств, доказывается без труда.

Задача о покрытии множества подмножествами. Пусть

дано конечное множество $V(v_1, v_2, \ldots, v_n)$ и система его подмножеств $W_i = \{v_{i,j_1} v_{i,j_2}, \ldots, v_{i,j_{k(i)}}\} (i = 1, 2, \ldots, l)$. Эту систему можно задать матрицей $\|x_{ij}\|_{i=1,j=1}^{l,n}$, где

$$x_{ij} = \begin{cases} 1, & \text{если } v_j \in W_i; \\ 0, & \text{если } v_j \notin W_i. \end{cases}$$

Требуется выяснить, существуют ли k подмножеств W_i, покрывающих все множество V. Введем двоичные переменные $y_i (i = 1, 2, \ldots, L)$.

Значение $y_i = 1$ соответствует выбору подмножества W_i в состав покрытия. Рассматриваемая задача эквивалентна системе условий

$$\sum_{i=1}^{l} x_{ij} y_i \geqslant 1 \; (j = 1, 2, \ldots, n); \quad \sum_{i=1}^{l} y_i = k.$$

Выполнимость конъюнктивной нормальной формы. Пусть $x_1, x_2, \ldots, x_n$ — двоичные переменные,

$$\Phi(x_1, x_2, \ldots, x_n) = \mathop{\&}_{i=1}^{l} (\xi_{i1} \vee \xi_{i2} \vee \ldots \vee \xi_{is(i)}),$$

где каждое ξ_{ij} — это одна из переменных x_h или ее отрицание $\neg x_h (1 \leqslant h \leqslant n)$. Требуется найти значения переменных $x_1, x_2, \ldots, x_n$, для которых $\Phi(x_1, x_2, \ldots, x_n) = 1$, или доказать, что данная конъюнктивная нормальная форма $\Phi(x_1, x_2, \ldots, x_n)$ тождественно равна 0. В крайнем случае достаточно только узнать, существуют ли такие значения $x_1, x_2, \ldots, x_n$, что $\Phi(x_1, x_2, \ldots, x_n) = 1$, а самих значений не указывать.

Если задача об определении значения предиката Φ имеет полиномиальную трудоемкость, то трудоемкость задачи о выполнимости конъюнктивной нормальной формы тоже полиномиальна. Действительно, пусть последняя задача имеет трудоемкость t (она разрешима, как всякая массовая задача перебора). Положим $x_1 = 1$. Если в выражении $(\xi_{i1} \vee \ldots \vee \xi_{is(i)})$ есть некоторое $\xi_{i'j} = x_1$, то такое выражение будет равно 1; если в нем есть $\xi_{ij} = \neg x$, то оно равно дизъюнкции $(\xi_{i1}, \vee \ldots \vee \xi_{i,j-1} \vee \xi_{i,j+1} \vee \ldots \vee \xi_{is(i)})$, в которой ξ_{ij} исключена. Таким образом, при $x_1 = 1$ $\Phi(x_1, x_2, \ldots, x_n)$ будет равна конъюнктивной нормальной форме $\Phi'(x_2, \ldots, x_n)$. Решим задачу для нее. Если существуют значения $x_2, \ldots, x_n$, для которых $\Phi'(x_2, \ldots, x_n) = 1$, то можно выбрать $x_1 = 1$. Если же таких значений нет, то, так как $\Phi(x_1, x_2, \ldots, x_n)$ может быть равно $1, x_1$ нужно положить равным 0 и рассмотреть аналогичную конъюнктивную нормальную

форму $\Phi^0(x_2, \ldots, x_n)$. Таким же образом можно выбрать затем допустимые значения $x_2, \ldots, x_n$. Понадобится решить n задач о выполнимости конъюнктивных нормальных форм все более низких размерностей.

Задача о выполнимости конъюнктивной нормальной формы сводится к многомерной задаче о камнях. Рассмотрим $2n$ двоичных переменных $y_1, y_2, \ldots, y_n, y_{n+1}, \ldots, y_{2n}$ и систему условий

$$y_i + y_{i+n} = 1 \; (i = 1, 2, \ldots, n);$$

$$\sum_{h=1}^{2n} x_h^i y_h \geqslant 1 \; (i = 1, 2, \ldots, l),$$

где при $1 \leqslant h \leqslant n$ $x_h^i = 1$ в том и только в том случае, когда среди ξ_{ij} есть значение, равное x_h, а при $h > n$ — равное $\neg x_{h-n}$. Легко видеть, что сконструированы условия задачи о камнях, и они выполняются тогда и только тогда, когда выполняется условие $\Phi(x_1, x_2, \ldots, x_n) = 1$, причем $y_i = x_i$ при $i = 1, 2, \ldots, n$ и $y_i = \neg x_{i-n}$ при $i = n+1, \ldots, 2n$.

Классы трудоемкости задач. Задача принадлежит классу P, если существует машина Тьюринга, на которой она решается с трудоемкостью, полиномиально зависящей от параметров ее размерности, т. е. с трудоемкостью, не превосходящей $C_0 m^{C_1} n_1^{C_2} n_2^{C_3}$, где C_0, C_1, C_2, C_3 — константы (свои для каждой задачи).

Классу P принадлежат задачи определения следующего натурального числа $x+1$, суммы $y = x_1 + x_2$, произведения $y = x_1 x_2$ и т. д. К ним относится и задача *PERT*. Только задачи класса P можно считать эффективно решаемыми, да и то при условии, что показатели степени C_1, C_2, C_3 не слишком велики.

Задача принадлежит классу PC (классу задач с полиномиальной трудоемкостью проверки), если ее предикат $P^{m, n_1, n_2}(\bar{x}, \bar{y})$ (или $P^m(\bar{x}, \bar{y})$) вычисляется на некоторой машине Тьюринга с полиномиальной трудоемкостью. Задачи о камнях, о покрытии множества подмножествами, о выполнимости конъюнктивной нормальной формы, вообще об определении значений двоичных переменных $y_1, y_2, \ldots, y_m$, удовлетворяющих условиям

$$\sum_{j=1}^{n} x_j^i y_j \lesseqgtr M_i \; (i = 1, 2, \ldots, l),$$

относятся к этому классу.

Действительно, проверка выполнения условий требует $k(n+1)$ арифметических операций, а арифметические — не более чем $8m$ логических, где m — сумма длин всех чисел x_j^i и M.

Естественно, задачи класса P являются также задачами класса PC, но существуют ли задачи класса PC, не принадлежащие классу P, пока не известно. Мы еще вернемся к этому вопросу.

Отметим в соответствии с результатами, полученными в § 9.1, если задача решается с полиномиальной трудоемкостью на машине одного класса, она решается с полиномиальной трудоемкостью и на машине другого класса. Поэтому принадлежность задачи классу P или PC не зависит от выбора машины, для которой оценивалась трудоемкость вычисления этой задачи; иначе говоря, классы P и PC инвариантны относительно выбора детерминированных машин.

Другой подход к классификации задач по сложности основан на понятии недетерминированной машины Тьюринга.

Недетерминированная машина Тьюринга моделирует алгоритмы с некоторой «свободой выбора», причем нас интересует, сколько времени понадобится, если с выбором «всегда будет везти». По крайней мере некоторые команды недетерминированной машины Тьюринга NT при одной и той же истории ее работы и, значит, при одном и том же ее полном состоянии могут выполняться разными способами. Для каждого внутреннего состояния машины и читаемого с ленты символа эти способы заданы.

У машины Тьюринга NT также имеется конечный набор внутренних состояний $\{st_i\}_{i=1}$, и алфавит символов на ленте $\{sym_j\}$ $(j=1, 2, \ldots, m)$ тоже конечен. Однако правила действий имеют вид списков:

$$st_i\, sym_j \to \begin{cases} st_{i_1}\, sym_{j_1}\, move_{k_1}; \\ st_{i_2}\, sym_{j_2}\, move_{k_2}; \\ \cdots\cdots\cdots \\ st_{i_l}\, sym_{j_l}\, move_{k_l}, \end{cases}$$

где l — это максимальное число вариантов выполнения действия. Если для некоторых внутренних состояний st_i и читаемых символов sym_j вариантов меньше, то последний будем дублировать так, чтобы их стало l.

Перед выполнением очередного действия из одной ма-

шины возникает l машин. Записи на их лентах одинаковы — такие, какие были у их «предка», но с этого момента их пути расходятся: каждая машина выполняет свой вариант действия из списка. Процесс работы детерминированной машины Тьюринга T (и любой рассмотренной раньше машины) может быть изображен в виде линейной последовательности действий, а процесс работы недетерминированной машины NT — в виде дерева, вершинами которого являются выполняемые действия, а ребрами — переходы от одного действия к следующим (рис. 9.2). Таким образом, она одновременно решает задачу разными способами, чтобы не упустить тот, при котором «везет». Каждый способ — это последовательность машинных операций, которой соответствует цепь дерева работы машины NT с началом в корне.

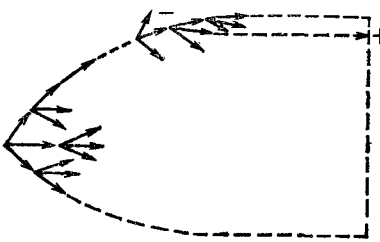

Рис. 9.2

Если по некоторой цепи действия заканчиваются, то возможны два исхода: задача решена или решение не найдено. В первом случае все остальные машины, размножившиеся к этому моменту, тоже кончают работу и превращаются в одну машину. Во втором — они продолжают работать. Если же по всем цепям произойдет окончание вычислений с отрицательными результатами, то все машины тоже сливаются в одну: задача не имеет решения. Других обменов информацией между различными вариантами процесса работы машины не происходит.

Класс *NP*. Что значит решить данную массовую задачу перебора при значениях параметров размерностей m, n_1, n_2 или только m? Каждому варианту $x(x_1, x_2, \ldots, x_m)$ такой задачи надо уметь поставить в соответствие ответ $y(y_0, y_1, \ldots, y_m)$. Если $y_0=1$, то вариант имеет решение, и значения $y_1, y_2, \ldots, y_m$ — это описание некоторого из них (решение может быть не единственным). Если $y_0=0$, то задача не имеет решения, и значения $y_1, y_2, \ldots, y_m$ могут быть какими угодно. В дальнейшем будем предполагать, что учитывается только информационная размерность m. Пусть имеется недетерминированная машина Тьюринга NT, которая при любом варианте x данной массовой задачи не позднее чем

через $C_0 m^{C_1}$ шагов останавливается и дает ответ, положительный ($y_0=1$) или отрицательный ($y_0=0$), причем C_0 и C_1 — константы. Тогда рассматриваемая задача принадлежит классу NP.

Так как обычную машину Тьюринга T можно считать частным случаем недетерминированной с количеством разветвлений $l=1$, все задачи класса P являются задачами класса NP.

Теорема 9.4. Все задачи класса PC принадлежат классу NP.

Доказательство. Пусть машина Тьюринга T не более чем за $C_0 m^{C_1}$ шагов вычисляет предикат $y_0=P^m(x_1, x_2, \ldots, x_m, y_1, y_2, \ldots, y_m)$, т. е. проверяет, является ли набор переменных $y(y_1, y_2, \ldots, y_m)$ решением данного варианта $x(x_1, x_2, \ldots, x_m)$ массовой задачи перебора, причем читающая и пишущая головка должна находиться над левой ячейкой информативной части ленты, а запись на последней имеет вид, приведенный на рис. 9.3. Сначала идет опи-

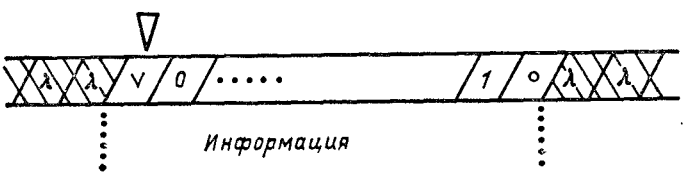

Рис. 9.3

сание задачи, заканчивающееся знаком конца, например, "$.$". Затем идут значения $x_1, x_2, \ldots, x_m$, снова "$.$", $y_1, y_2, \ldots, y_m$ и последний раз "$.$", на чем информативная часть ленты кончается.

На ленте недетерминированной машины Тьюринга NT в начале работы находится такая же запись, причем $y_1=y_2=\ldots=y_m=0$. У нее есть переходы в новое состояние с порождением двух ветвей и переходы без разветвления (формально $l=2$, переходами без разветвления мы считаем переходы, если внутренние состояния, символы, которые пишутся в ячейку ленты, и движения читающей и пишущей головки на обеих ветвях одинаковы). Начальное внутреннее состояние st_{pp} (пройти описание задачи) определяется строками таблицы переходов:

$$st_{pp} \urcorner . \rightarrow st_{pp}\, NWRh;$$
$$st_{pp} . \rightarrow st_{px}\, NWRh.$$

Здесь NW означает — не записывать нового символа в ячейку, а оставить тот, который был.

Следующее состояние st_{px} (пройти описание варианта) тоже не производит ветвления:

$$st_{px} \urcorner . \rightarrow st_{px} NWRh;$$
$$st_{px} . \rightarrow st_{gy} NWRh.$$

Теперь происходит порождение всех вариантов ответов

$$y_1, y_2, \ldots, y_m:$$
$$st_{gy}\, 0 \rightarrow \begin{cases} st_{gy}\, 0\, Rh; \\ st_{gy}\, 1\, Rh. \end{cases}$$

В течение m шагов машина NT читает на ленте символы 0, и в одной ветви оставляют его без изменения, а в другой заменяют на 1. После этого возникнут экземпляры машины NT со всеми вариантами ответа

$$\underbrace{00\ldots00}_{m\text{ раз}},\ \underbrace{00\ldots01}_{m-1\text{ раз}},\ \underbrace{00\ldots10}_{m-2\text{ раз}},\ \ldots,\ \underbrace{11\ldots11}_{m\text{ раз}}.$$

Наконец будет прочтен символ '.':

$$st_{gy} \urcorner 0 \rightarrow st_{If} NWLf.$$

Во всех ветвях, отличающихся друг от друга виртуальными ответами $\overline{y} = (y_1, y_2, \ldots, y_n)$, надо вернуться к началу информативной части ленты. Предполагается, что внутри нее нет символов пробела λ. Тогда годятся переходы

$$st_{If} \urcorner \lambda \rightarrow st_{If} NWLf;$$
$$st_{If}\, \lambda \rightarrow st_T NWRh.$$

Пишущие и читающие головки всех экземпляров машины NT оказываются над начальными ячейками информативных частей лент, после чего эти машины начинают работать, как машина Тьюринга T, т. е. вычисляют значения предиката $y_0 = P^m(x_1, x_2, \ldots, x_m, y_1, y_2, \ldots, y_m)$ при всех допустимых значениях $y(y_1, y_2, \ldots, y_m)$.

Для этого машина NT имеет внутренние состояния, аналогичные состояниям машины T: у них такие же таблицы переходов (одинаковые на обеих ветвях). В частности, состояние st_T, в котором находятся все экземпляры машины NT в рассматриваемый момент, аналогично начальному состоянию машины Тьюринга T. Если данный вариант $x(x_1, x_2, \ldots, x_n)$ массовой задачи имеет хотя бы одно ре-

шение, то не позднее чем через $C_0\, m^{c_1}$ шагов после предварительной работы некоторые экземпляры машины NT вычислят $y_0 = P^m(x_1, x_2, \ldots, x_m, y_1, y_2, \ldots, y_m)$, остановятся и дадут положительный сигнал. Работа остальных экземпляров прекратится. В противном случае, тоже не позднее чем через $C_0\, m^{c_1}$ шагов, от всех экземпляров машины NT будут приняты отрицательные сигналы, все они остановятся, т. е. будет обнаружено, что вариант $\bar{x}(x_1, x_2, \ldots, x_m)$ не имеет допустимого решения.

Остается оценить количество шагов предварительной работы, т. е. порождения вариантов. Читающие и пишущие головки всех экземпляров машины NT сначала проходят над всеми ячейками информативной части ленты слева направо, затем — справа налево, делают еще шаг влево и шаг вправо и переходят к вычислениям предикатов $y_0 = P^m(\bar{x}, \bar{y})$. Таким образом, на порождение вариантов уходит $L - -1 + L + 1 = 2L$ шагов, где L — длина информативной части ленты машины NT (и T) в начале вычисления. Эта часть состоит из постоянного количества ячеек, не зависящих от варианта и предполагаемого ответа, и $(2m+2)$ ячеек, занятых описанием $\bar{x}(x_1, x_2, \ldots, x_m)$ и $\bar{y}(y_1, y_2, \ldots, y_m)$, если считать символы концов описания — $'.'$. Значит, полное количество шагов не больше, чем $2L + C_0\, m^{c_1} = 2(\mathrm{const} + +2m+2) + C_0\, m^{c_1} \leqslant C_0' m^{c_1'}$ при $C_0' = C_0 + 2\,\mathrm{const} + 8$, $C_1' = \max(C_1, 1)$. Оно полиномиальным образом зависит от размерности варианта задачи. □

Интерпретация ветви процесса вычисления недетерминированной машины Тьюринга NT элементарными логическими операциями отличается от интерпретации процесса вычисления детерминированной машины Тьюринга только макрооператорами $comp(i)$: вместо

$$stf_i := \bigvee_{j=1}^{n} \bigvee_{k=1}^{m} Tab_{i,j,k} \,\&\, stp_j \,\&\, sym\ r_k \quad (i = 1, 2, \ldots, n);$$

$$sym\ w_i := \bigvee_{j=1}^{n} \bigvee_{t=1}^{m} Tab_{t+n,j,k} \,\&\, stp_j \,\&\, sym\ r_k \quad (i = 1, 2, \ldots, m);$$

$$move_i := \bigvee_{j=1}^{n} \bigvee_{t=1}^{m} Tab_{t+n+m+1,j,k} \,\&\, stp_j \,\&\, sym\ r_k \quad (i = -1, 0, 1)$$

надо вычислить

$$stf_i := \bigvee_{j=1}^{n} \bigvee_{k=1}^{m} \bigvee_{h=1}^{l} Tab_{i,j,k,h} \,\&\, stp_j \,\&\, sym\ r_k \,\&\, q_h \quad (i = 1, 2, \ldots, n);$$

$$sym\ w_i := \bigvee_{j=1}^{n} \bigvee_{k=1}^{m} \bigvee_{h=1}^{l} Tab_{i+n,j,k,h} \& stp_j \& sym\ r_k \& q_h (i=1, 2, \ldots, m);$$

$$move_i := \bigvee_{j=1}^{n} \bigvee_{k=1}^{m} \bigvee_{h=1}^{l} Tab_{i+n+m+1,j,k,h} \& stp_j \& sym\ r_k \& q_h$$

$$(i = -1, 0, 1).$$

Здесь n — количество внутренних состояний экземпляра машины; m — число символов алфавита, которые могут быть записаны в ячейках лент; l — количество разветвлений при выполнении шага недетерминированной машины Тьюринга NT.

Однако имеется принципиальное различие: при интерпретации детерминированной машины Тьюринга результаты предопределены, так как каждая операция либо производится с ранее вычисленными переменными, либо состоит в присвоении некоторой переменной значения 0 или 1, записанного в команде машины элементарных логических операций; при интерпретации недетерминированной машины то же можно сказать обо всех переменных, кроме $q_h (h=1, 2, \ldots, l)$, которые на каждом шаге определяются выбором ветви. Таким образом, результаты зависят от того, «как повезет» с выбором значений этих переменных на каждом шаге.

NP-полные задачи. Уже говорилось, что после конца процесса элементарных логических операций результаты вычислений, записанные в ячейках памяти машины с номерами $1, 2, \ldots, \tau-1$ (τ — длина программы машины элементарных логических операций) и являющиеся значениями переменных $z_1, z_2, \ldots, z_{\tau-1}$, соответственно удовлетворяют одной из следующих систем логических уравнений:

1) $z_i = c_i$,

если i-я команда имеет вид $i : i := c_i$;

2) $z_i = z_j$,

если она имеет вид $i : i := j$;

3) $z_i = \neg z_j$,

когда это — операция отрицания $i : i := \neg j$;

4) $z_i = z_j \& z_k$;

$z_i = z_j \vee z_k$,

если это соответственно операции $i : i := j \& k$, или $i : i := j \vee k$;.

Если программа элементарных логических операций предопределяет результаты, то значения всех переменных

$z_i (i=1, 2, \ldots, \tau-1)$ однозначно находятся из этой системы, но при интерпретации ветви вычислений недетерминированной машины Тьюринга результаты не определены однозначно, так как переменные z_i, которым соответствуют выборы ветвей $q_h (h=1, 2, \ldots, l)$ на 1-м, 2-м, ..., $(\tau-1)$-м шагах, не определяются командами программы (им даже нельзя присвоить номера по предложенному выше правилу, приходится нумеровать их отдельно, и они получают номера, большие чем номера команд, в правых частях которых они упоминаются). Некоторая известная переменная $z_{j'}$ равна значению сигнала о том, что найдено решение $y(y_1, y_2, \ldots, y_m)$ рассматриваемой массовой задачи перебора. Некоторые переменные $z_{j_1}, z_{j_2}, \ldots, z_{j_m}$ равны соответственно $y_1, y_2, \ldots y_m$ — решению варианта задачи, который находит машина NT.

Добавим к указанным выше уравнениям условие положительного результата процесса решения и допустимости выборов $q_{h,i}$ $(h=1, 2, \ldots, l;\ i=1, 2, \ldots, \tau-1)$, состоящие в том, что для каждого номера шага i одна из переменных $q_{h,i}$ равна 1, а остальные — 0:

$$y_0 = z_{j'} = 1;\quad \bigvee_{h=1}^{l} q_{h,i} = 1;\quad \bigvee_{h_1=1}^{l-1} \bigvee_{h_2=h_1+1}^{l} (q_{h_1} \& q_{h_2}) = 0$$

$$(i = 1, 2, \ldots, \tau - 1).$$

Если полученная система имеет решение, то значения $q_{h,i} (h=1, 2, \ldots, l;\ i=1, 2, \ldots, \tau-1)$ определяют ветвь процесса вычисления недетерминированной машины Тьюринга NT, дающей положительный результат $y_0=z_{j'}=1$. Остальные результаты вычисления, в том числе допустимый ответ $y_1=z_{j_1}$, $y_2=z_{j_2}$, ..., $y_m=z_{j_m}$, определяются через $q_{h,i}$ однозначно. В противном случае ни одна из ветвей не приводит к положительному результату, и если во всех них происходит остановка, то с отрицательным результатом.

Пусть данная массовая задача перебора принадлежит классу NP, т. е. решается на некоторой недетерминированной машине Тьюринга NT за $t \leqslant C_0 m^{C_1}$ шагов, где m — информационная размерность варианта. Интерпретация произвольной ветви процесса вычисления машины NT требует $C_0' t^2 = C_0' C_0^2 m^{2C_1}$ элементарных логических операций, опре-

деляющих $C_0'' m^{C_1'} = C_0' C_0^2 m^{2C_1'}$ двоичных переменных z_j. К ним надо добавить $l(t-1)$ переменных $q_{h,i}(h=1, 2, \ldots, l;$ $i=1, 2, \ldots, t-1)$. Каждой переменной z_j соответствует уравнение системы, а переменной $z_{j'}$ — еще одно: $z_{j'}=1$. Переменным $q_{h,t}$ соответствуют $2t-2$ уравнений $\bigvee_{h=1}^{l} q_{h,i} = 1$ и $\bigvee_{h_1=1}^{l-1} \bigvee_{h_2=h_1+1}^{l} (q_{h_1} \& q_{h_2}) = 0$. Таким образом, чтобы найти решение данного варианта $\bar{x}(x_1, x_2, \ldots, x_m)$ массовой задачи перебора из класса NP, достаточно найти решение системы логических уравнений, в которой количества переменных $C_0'' m^{C_1'} + 2t-2 = C_0'' m^{C_1'} + lC_0 m^{C_1'} - l \leqslant C_0''' m^{C_1'}$ и уравнений $C_0'' m^{C_1'} + 2C_0 m^{C_1'} - 2 \leqslant C_0''' m^{C_1'}$ ограничены степенным образом (можно считать, что $C_0'' = C_0' + lC_0$).

Если любую систему логических уравнений можно решить на машине элементарных логических операций за время, степенным образом зависящее от количеств переменных и уравнений, то любая массовая задача перебора класса NP принадлежит классу P. Действительно, по процессу решения задачи на недетерминированной машине Тьюринга NT, имеющему полиномиальную трудоемкость $t \leqslant C_0 m^{C_1}$, можно построить систему из $Q_1 \leqslant C_0''' m^{C_1'}$ логических уравнений с $Q_2 \leqslant C_0''' m^{C_1''}$ двоичными переменными, а затем решить ее на любой из рассматриваемых нами детерминированных машин (машине элементарных логических операций, машине с произвольным доступом к памяти или достаточно мощной машине Тьюринга T) за время, степенным образом зависящее от Q_1 и Q_2, и, значит, степенным образом только с большим показателем степени C_1'' — от m.

Легко видеть, что массовая задача решения системы из логических уравнений с Q_1 двоичными переменными является массовой задачей перебора, так как длина ответа равна Q_2, т. е. не больше описания варианта задачи. Она принадлежит классам PC и, значит, NP. Действительно, для вычисления предиката $P'''(\bar{x}, \bar{y})$, равного 1, когда все Q_1 уравнений удовлетворяются, достаточно произвести указанные в уравнениях элементарные логические операции и проверки равенств, т. е. число действий не больше, чем информационная размерность системы m — длина ее описания.

Задача класса NP называется NP-полной, если из предположения о том, что она принадлежит классу P, следует совпадение классов NP и P. Таким образом, задача о решении системы логических уравнений является NP-полной. До сих пор не удалось ни доказать, ни опровергнуть, что классы P и NP совпадают, но неудача многочисленных попыток найти алгоритмы решения NP-полных задач на детерминированных машинах с полиномиальной оценкой сложности склоняет к гипотезе о том, что класс P является собственной частью класса NP.

Напомним еще раз, что в процессе автоматического решения задачи количество физических устройств, занятых его реализацией, не может неограниченно возрастать. Поэтому реально решать задачи при помощи алгоритмов, рассчитанных на недетерминированную машину, нельзя. Однако существуют алгоритмы «со свободой выбора», которые в зависимости от актов выбора могут реализовать любую ветвь процесса вычисления недетерминированной машины.

Человек часто находит решение комбинаторной задачи, не перебирая многих вариантов ответа. Различные плохо формализуемые соображения позволяют ему определять порядок генерирования гипотетических решений и корректировать этот порядок на основе анализа причин неудовлетворительности уже просмотренных гипотетических решений. Такие соображения называются эвристиками. Иногда их удается, хотя бы частично, формализовать и применить для определения порядка перебора. Еще реже удается сформулировать и доказать теоремы о качестве эвристик, т. е. средней длине перебора до получения ответа.

При этом характерной является такая ситуация. Если решение данного варианта задачи существует, то, как правило, эвристики позволяют сравнительно быстро получить ответ, однако нет эвристик, позволяющих быстро доказать, что решений нет. Задачи дискретной оптимизации тоже являются массовыми задачами перебора, причем эвристики нередко позволяют довольно быстро найти экстремальное или близкое к нему решение, но для доказательства экстремальности приходится продолжать перебор.

NP-полнота других задач. Система логических уравнений, порожденная интерпретацией ветви процесса вычисления недетерминированной машины Тьюринга, состоит из уравнений, соответствующих элементарным логическим операциям интерпретатора, и условий положительности ре-

зультата и выбора ветви. У первых уравнений в левой части стоят какие-либо изолированные переменные, а в правой — другие такие же переменные, константы или элементарные логические операции, и условие положительного результата $z_{j'}=1$ имеет один из перечисленных выше видов (слева переменная, справа константа). Остальные уравнения можно привести к аналогичному виду, введя дополнительные переменные, но не увеличивая существенно информационной размерности вариантов.

Действительно, эти уравнения имеют вид:

$$\bigvee_{h=1}^{l} q_{h,i}=1; \quad \bigvee_{h_1=1}^{l-1}\bigvee_{h_2=h_1+1}^{l}(q_{h_1,i}\&q_{h_2,i})=0 \quad (i=1,2,\ldots,t-1).$$

Они преобразуются следующим образом:

$$v_{h,1}=q_{h,1};$$

$$v_{h,i}=v_{h-1,i}\,\&\,q_{h,i}\,(h=2,\ 3,\ \ldots\ ,\ l-1);$$

$$v_{l,i}=1;$$

$$w_{h_1,h_2,i}=q_{h_1,i}\,\&\,q_{h_2,i};$$

$$w_{h_1,h_2,i}=0\,(h_1=1,\ 2,\ \ldots\ ,\ l-1,\ h_2=h_1+1,\ \ldots\ ,\ l,$$
$$i=1,\ 2,\ \ldots\ ,\ t-1).$$

В этих уравнениях столько же знаков &, сколько их в уравнениях, которые они заменяют; знаков $\vee$ меньше, и каждому уравнению со знаком соответствует не более одного уравнения без знака и не более одной переменной. Общее число новых уравнений и неизвестных не больше, чем $(t-1)\ (l+1+2((l-1)+(l-2)+\ldots+1))=(t-1)\ (l^2+1)<Ct$, т. е. степенным образом со степенью 1 зависит от сложности t варианта задачи относительно недетерминированной машины Тьюринга и степенным же образом с некоторой степенью C_1 — от информационной размерности m варианта задачи. Итак, NP-полной является задача решения системы уравнений, содержащих не менее трех переменных и имеющих один из видов:

$$\zeta_j=C_j; \quad \zeta_j=\zeta_k\,\&\,\zeta_h;$$
$$\zeta_j=\zeta_k; \quad \zeta_j=\zeta_k\vee\zeta_h.$$
$$\zeta_j=\neg\,\zeta_k;$$

Каждое из них эквивалентно таким логическим условиям:

$$\zeta_j=\zeta_k\sim(\zeta_j\vee\neg\,\zeta_k)\,\&\,(\neg\,\zeta_j\vee\zeta_k);$$

$$\zeta_j = \neg\zeta_k \sim (\zeta_j \vee \zeta_k) \,\&\, (\neg\zeta_j \vee \neg\zeta_k);$$

$$\zeta_j = \zeta_k \,\&\, \zeta_h \sim (\neg\zeta_j \vee \zeta_k) \,\&\, (\neg\zeta_j \vee \zeta_h) \,\&\, (\zeta_j \vee \neg\zeta_k \vee \neg\zeta_h);$$

$$\zeta_j = \zeta_k \vee \zeta_h \sim (\zeta_j \vee \neg\zeta_h) \,\&\, (\zeta_k \vee \neg\zeta_h) \,\&\, (\neg\zeta_j \vee \zeta_k \vee \zeta_h).$$

Переменную, приравненную константе, можно в этих формулах заменить константой; выражения вида $\zeta_{j_1} \vee \zeta_{j_2} \vee \ldots \ldots \zeta_{j_r} \vee \zeta_{j'_1} \vee \neg\zeta_{j'_2} \vee \ldots \vee \neg\zeta_{j'_s}$ будем называть дизъюнктами.

Каждое уравнение заменяется одним, двумя или тремя дизъюнктами, и все они должны выполняться. Поэтому рассматриваемая задача о решении системы логических уравнений эквивалентна задаче о выполнимости конъюнктивной нормальной формы (КНФ)

$$\Phi(\zeta_1, \zeta_2, \ldots, \zeta_n) = \mathop{\&}_{i=1}^{m} \left(\bigvee_{j=1}^{k_i} \zeta_{i,j} \right),$$

где каждое $\xi_{i,j}$ равно одной из переменных ζ_h или ее отрицанию $\neg\zeta_h$. Требуется определить такие значения $\zeta_1, \zeta_2, \ldots, \zeta_m$, чтобы КНФ $\Phi(\zeta_1, \zeta_2, \ldots, \zeta_m)$ было равно 1, или доказать, что $\Phi(\zeta_1, \zeta_2, \ldots, \zeta_m)$ тождественно равно 0. Следовательно, задача о выполнимости конъюнктивной нормальной формы тоже является *NP*-полной. *NP*-полна и обратная задача о тождественной истинности дизъюнктивной нормальной формы:

$$\neg\Phi(\zeta_1, \zeta_2, \ldots, \zeta_n) = \bigvee_{i=1}^{m} \left(\mathop{\&}_{j=1}^{k_i} \neg\xi_{ij} \right) \equiv 1.$$

Система логических уравнений эквивалентна системе линейных уравнений для булевых переменных, т. е. двоичных переменных, рассматриваемых как целые числа, причем информационные размерности соответствующих вариантов по порядку — те же. Добавить переменные $\eta_j = \neg\zeta_j$ $(j = 1, 2, \ldots, n)$ и переменные $\vartheta_{i,s}$, соответствующие некоторым логическим уравнениям. Уравнение $\zeta_j = C_j$ можно с одинаковым основанием считать либо логическим (C_j равно 0 или 1), либо линейным; переменные ζ_j и η_j связаны условиями $\zeta_j + \eta_j = 1$ $(j = 1, 2, \ldots, n)$.

Уравнения $\zeta_j = \zeta_k$ и $\zeta_j = \neg\zeta_k = \eta_k$ можно заменить эквивалетными линейными уравнениями

$$\zeta_j + \eta_k = 1; \quad \zeta_j + \zeta_k = 1,$$

уравнения $\zeta_j = \zeta_k \& \zeta_h$ — сначала неравенствами

$$\zeta_j \geqslant \zeta_k; \quad \zeta_j \geqslant \zeta_h; \quad \zeta_k + \zeta_h \geqslant \zeta_j,$$

затем неравенствами

$$\zeta_j + \eta_k \geqslant 1; \quad \zeta_j + \eta_h \geqslant 1; \quad \zeta_k + \zeta_h + \eta_j \geqslant 1$$

и, наконец, равенствами

$$\zeta_j + \eta_k + \vartheta_{i,1} = 2; \quad \zeta_j + \eta_h + \vartheta_{i,2} = 2; \quad \zeta_k + \zeta_h + \eta_j + \vartheta_{i,3} + \vartheta_{i,4} = 3.$$

Действительно, если $\zeta_j + \eta_k \geqslant 1$, то при некотором значении булевой переменной $\vartheta_{i,1}$, где i — номер рассматриваемого логического уравнения в исходной системе логических уравнений, $\zeta_j + \eta_k + \vartheta_{i,1} = 2$: если $\zeta_j = 1$, $\eta_k = 1$, то $\vartheta_{i,1} = 0$, если $\zeta_j = 1$, $\eta_k = 0$, то $\vartheta_{i,1} = 1$, а других значений эта сумма иметь не может. Аналогичным образом заменяется неравенство $\zeta_j + \eta_k \geqslant 1$. Сумма $\zeta_k + \zeta_h + \eta_j$ может быть равна 1, 2 или 3. Соответственно $\vartheta_{i,3} = \vartheta_{i,4} = 1$, $\vartheta_{i,3} = 1$ и $\vartheta_{i,4} = 0$ (или $\vartheta_{i,3} = 0$, $\vartheta_{i,4} = 1$) или $\vartheta_{i,3} = \vartheta_{i,4} = 0$. Наконец, логическое уравнение $\zeta_j = \zeta_k \vee \zeta_h$ можно заменить эквивалентным $\eta_j = \neg\zeta_j = \neg(\zeta_k \vee \zeta_h) = \neg\zeta_k \& \neg\zeta_h = \eta_k \& \eta_h$, а последнее — равенствами:

$$\eta_j + \zeta_k + \vartheta_{i,1} = 2; \quad \eta_j + \zeta_h + \vartheta_{i,2} = 2;$$
$$\eta_k + \eta_h + \zeta_j + \vartheta_{i,3} + \vartheta_{i,4} = 3.$$

Таким образом, задача решения системы линейных уравнений для булевых переменных NP-полна (легко показать, что она принадлежит классу PC и, значит, классу NP). Следовательно, NP-полна многомерная задача о камнях, частным случаем которой она является. Покажем теперь, что NP-полна и одномерная задача о камнях

$$\sum_{j=1}^{n} b_j \zeta_j = M; \quad \zeta_j \in \{0, 1\} \ (j = 1, 2, \ldots, n).$$

Переименуем все переменные в NP-полной задаче о решении системы линейных уравнений с булевыми переменными, эквивалентной системе логических уравнений. Тогда она будет иметь вид

$$\sum_{j=1}^{n} a_{ij} \varphi_j = b_i \ (i = 1, 2, \ldots, N = 2n + 4f),$$

где n — число переменных $\zeta_1, \zeta_2, \ldots, \zeta_n$, а f — число уравнений в исходной системе логических уравнений; все коэффициенты a_{ij} равны 0 или 1, причем количество равных 1 коэффициентов в каждом уравнении не больше 5, а их правые части равны 0, 1, 2 или 3. Рассмотрим задачу

о камнях

$$\sum_{j=1}^{n} A_j \varphi_j = B,$$

где $A_j = \sum_{i=1}^{n} a_{ij} \cdot 6^{i-1}$ $(j = 1, 2, \ldots, n)$; $B = \sum_{i=1}^{N} b_i \cdot 6^{i-1}$, и докажем, что она эквивалентна системе линейных уравнений с булевыми переменными и имеет такую же по порядку информационную размерность m'.

Введем обозначения:

$$A_j = \sum_{i=1}^{N} a_{ij} \cdot 6^{i-1} = A_j' = 6 \sum_{i=2}^{N} a_{ij} \cdot 6^{i-2} + a_{ij} = 6A_j^2 + a_{ij};$$

$$A_j^{i'} = \sum_{i=i'}^{N} a_{ij} \cdot 6^{i-i'} = 6 \sum_{i=i'+1}^{N} a_{ij} \cdot 6^{i-i'-1} +$$

$$+ a_{i'j} = 6A_j^{i'+1} + a_{i'j} \quad (i' = 2, 3, \ldots, N-1, \; j = 1, 2, \ldots, n);$$

$$B = \sum_{i=1}^{N} b_i \cdot 6^{i-1} = 6 \sum_{i=2}^{N} b_1 \cdot 6^{i-2} + b_1 = B_1 = 6B_2 + b_1;$$

$$B_{i'} = \sum_{i=i'}^{N} b_i \cdot 6^{i-i'} = 6 \sum_{i=i'+1}^{N} b_i \cdot 6^{i-i'-1} + b_{i'} =$$

$$= 6B_{i'+1} + b_{i'} \; (i = 2, 3, \ldots, N-1);$$

$$A_j^{N+1} = B_{N+1} = 0 \quad (j = 1, 2, \ldots, n).$$

Пусть рассматриваемая задача о камнях имеет решение $\varphi_1, \varphi_2, \ldots, \varphi_4$. Тогда

$$\sum_{j=1}^{n} A_j \varphi_j = \sum_{j=1}^{n} A_j^1 \varphi_j = 6 \sum_{j=1}^{n} A_j^2 \varphi_j +$$

$$+ \sum_{j=1}^{n} a_{1j} \varphi_j = B = B_1 = 6B_2 + b_1;$$

$$0 \leqslant \sum_{j=1}^{n} a_{ij} \varphi_j < 6;$$

$$0 \leqslant b_1 < 6;$$

$$6\sum_{j=1}^{n} A_j^2 \varphi_j \text{ делится на } 6;$$

$$6B_2 \text{ делится на } 6.$$

Следовательно,

$$\sum_{j=1}^{n} a_{1j}\varphi_j = b_1, \quad \sum_{j=1}^{n} A_j^2 \varphi_j = B_2.$$

Аналогичным образом доказывается, что

$$\sum_{j=1}^{n} a_{ij}\varphi_i = b_j \,(i = 2, 3, \ldots, N).$$

Если же $\varphi_1, \varphi_2, \ldots, \varphi_n$ — решение системы линейных уравнений

$$\sum_{j=1}^{n} a_{ij} = b_i \,(i = 1, 2, \ldots, N), \text{ то}$$

$$\sum_{j=1}^{n} A_j \varphi_j = \sum_{i=1}^{N} 6^{i-1} \sum_{j=1}^{n} a_{ij}\varphi_j = \sum_{i=1}^{N} 6^{i-1} b_i = B.$$

Таким образом, доказана эквивалентность рассматриваемых задач.

При самом коротком описании системы линейных уравнений нужно воспользоваться тем, что в каждом уравнении не более пяти коэффициентов a_{ij} равны 1, а остальные равны 0. Значит, достаточно указать номера коэффициентов, отличных от 0. Эти номера не меньше 1 и не больше n. Для их описания в двоичной позиционной системе достаточно $\lfloor \log_2 n \rfloor$ символов на каждый, а всего — не больше $5\lfloor \log_2 n \rfloor$. Однако, так как нужны еще разделители, то либо в алфавите должны быть символы, отличные от 0 и 1, либо цифры в изображении номеров j, для которых $a_{ij}=1$, в двоичной позиционной системе придется изображать парами символов, например, вместо 0 писать 00, а вместо 1 — 01.

Итак, для описания коэффициентов матрицы $\|a_{ij}\|$ достаточно $N\lfloor \log_2 n \rfloor$ символов алфавита {0, 1}. На каждую правую часть, равную 1, 2 или 3 (в двоичной позиционной системе 01, 10 или 11), достаточно двух символов. Значит, размерность задачи о булевых решениях системы линейных уравнений

$$m \leqslant c\,(N \lfloor \log_2 n \rfloor + 2n) \leqslant c' N \lfloor \log_2 n \rfloor.$$

Каждая переменная исходной системы логических уравнений один раз входит в правую часть, кроме переменных $z_{i'}$, $v_{h,l}$ и w_{h_1,h_2}, которые входят в правые части двух уравнений (правда, в одном из них они приравниваются некоторым константам, но для простоты мы не станем этого учитывать: все равно порядок размерности вариантов не изменится). Значит, количество уравнений в системе равно $c''n'$, где n' — количество неизвестных и $1 \leqslant c'' \leqslant 2$. При переходе к системе линейных уравнений увеличиваются количества переменных и уравнений: добавляются переменные $\eta_j = \neg\zeta_j$ (их n' штук) и не более четырех переменных $\vartheta_{i,k}$ на каждую исходную переменную. Таким образом, количество переменных n не меньше, чем $2n'$, и не больше, чем $2n' + 4c''n' = c'''n'$, т. е. имеет тот же порядок, что и n'. Каждому уравнению системы логических уравнений соответствует не менее одного и не более четырех уравнений системы линейных уравнений. Значит, количество последних N не меньше, чем $c'n'$, и не больше $4c'n'$. Поэтому $N = cn$, где $c''/c''' \leqslant c \leqslant 2c''$ и $m \leqslant c'cn \lfloor \log_2 n \rfloor$.

Коэффициенты $A_j (j = 1, 2, \ldots, n)$ и правая часть B задачи о камнях — целые числа, не превосходящие 6^N. Для их изображения в данной позиционной системе нужно $N \lfloor \log_2 6 \rfloor$ двоичных символов на каждое число, с учетом необходимости разделительных знаков — всего $\tilde{c}Nn \leqslant c\tilde{c}n^2 \leqslant \tilde{\tilde{c}}m^2$. Поэтому алгоритм решения одномерной задачи о камнях, имеющий полиномиальную сложность степени C_k относительно некоторой детерминированной машины, может быть применен для решения системы логических уравнений, причем его сложность не будет превосходить C_k^2. NP-полнота одномерной задачи о камнях доказана.

В последнее время была доказана NP-полнота очень многих задач, имеющих прикладное значение. Мы еще остановимся только на доказательстве NP-полноты задачи о покрытии множества системой подмножеств. Рассмотрим исходную систему логических уравнений (с добавленными переменными $v_{h,l}$ и w_{h_1,h_2}) и произведем следующие преобразования.

1. Добавим переменные $\eta_j = \neg\zeta_j$.

2. Исключим переменные ζ_j, если в систему входит уравнение $\zeta_j = c_j$, и, естественно, исключим это уравнение. В других уравнениях заменим такие переменные соответствующими константами.

3. Произведем замены переменных и исключения уравнений, соответствующих уравнениям $\zeta_j=\zeta_h$ и $\zeta_j=\neg\zeta_h=\eta_h$, исключив переменные с бо́льшими номерами.

4. В результате таких преобразований останутся уравнения следующих видов:

$$\zeta_j=\xi_h\,\&\,\xi_k;\quad 1=\xi_h\,\&\,\xi_k;$$
$$\zeta_j=\xi_h\vee\xi_k;\quad 0=\xi_h\,\&\,\xi_k;$$
$$\zeta_j=\xi_h\,\&\,0=0;\quad 1=\xi_h\vee\xi_k;$$
$$\zeta_j=\xi_h\,\&\,1=\xi_h;\quad 0=\xi_h\vee\xi_k;$$
$$\zeta_j=\zeta_h\,\&\,\eta_h=0;\quad \eta_j=\neg\zeta_j,$$
$$\zeta_j=\xi_h\vee 0=\xi_h;$$
$$\zeta_j=\xi_h\vee 1=1;$$
$$\zeta_j=\zeta_h\vee\eta_h=1;$$

где ξ_h — это либо ζ_h, либо η_h. Некоторые из этих уравнений дают возможность исключить переменные, применяя преобразования 2 и 3; уравнение $\zeta_j=\xi_h\,\&\,0$ равносильно уравнению $\zeta_j=0$; $\zeta_j=\zeta_h\,\&\,\eta_h$ тоже равносильно уравнению $\zeta_j=0$; $\zeta_j=\xi_h\,\&\,1$ равносильно уравнению $\zeta_j=\xi_h$; $\zeta_j=\xi_h\vee 0$ — такому же уравнению, $\zeta_j=\xi_h\vee 1$ и $\zeta_j=\zeta_h\vee\eta_h$ равносильны уравнениям $\zeta_j=1$; из уравнения $1=\xi_h\,\&\,\xi_k$ следует, что $\xi_h=1$ и $\xi_k=1$, а из уравнения $0=\xi_h\vee\xi_k$ — что $\zeta_h=0$ и $\xi_k=0$. Процесс исключения переменных продолжается, пока имеется возможность.

Можно показать, что он имеет степенную трудоемкость относительно информационной размерности m исходной системы логических уравнений. Действительно, каждый цикл просмотра уравнений и исключения некоторых из них имеет трудоемкость $O(N)$, где N — число не исключенных к данному моменту уравнений; число таких циклов не больше N_0 — начального числа уравнений. Таким образом, общее число действий $O(N_0^2)\leqslant O(m^2)$, где m — информационная размерность исходной задачи. В начале исключения переменных $N_0\leqslant 2m$, значит, и в конце его $N\leqslant 2m$. Процесс может закончиться исключением всех переменных. Тогда их значения определяются, т. е. задача будет решена. Однако в большинстве случаев количество оставшихся уравнений и переменных будет иметь такой же порядок, что и до исключения.

В результате нужно будет решить систему, состоящую из уравнений вида

$$\zeta_j = \xi_h \,\&\, \xi_k; \quad 1 = \xi_h \vee \xi_k;$$
$$\zeta_j = \xi_h \vee \xi_k; \quad \eta_j = \neg\, \zeta_j.$$
$$0 = \xi_h \,\&\, \xi_k;$$

Их можно заменить системой неравенств. Вместо $\zeta_j = \xi_h \& \xi_k$, как и выше, рассмотрим неравенства $\xi_h + \eta_j \geqslant 1$; $\xi_k + \eta_j \geqslant 1$; $\eta_j + \neg\xi_h + \neg\xi_k \geqslant 1$, вместо $\zeta_j = \xi_h \vee \xi_k$ — неравенства $\neg\xi_h + \zeta_j \geqslant 1$; $\neg\xi_k + \zeta_j \geqslant 1$, и $\eta_j + \xi_h + \xi_k \geqslant 1$; $0 = \xi_h \& \xi_k$ равносильно неравенству $\neg\xi_h + \neg\xi_k \geqslant 1$; $1 = \xi_h \vee \xi_k$ — неравенству $\xi_h + \xi_k \geqslant 1$, и только условия $\eta_j = \neg\, \zeta_j$ эквивалентны равенствам $\zeta_j + \eta_j = 1$.

Перенумеруем оставшиеся переменные ζ_j и η_j, их будет одинаковое количество $n' < n$, где n — число переменных в исходной системе уравнений, имеющее тот же порядок, что и число уравнений и информационная размерность m. Равенства $\zeta_j + \eta_j = 1$ можно заменить неравенствами $\zeta_j + \eta_j \geqslant 1$ и одним равенством $\sum_{j=1}^{n'} (\zeta_j + \eta_j) = n'$. Каждому неравенству полученной системы поставим в соответствие точку v_i некоторого конечного множества V, а каждой переменной ζ_j или η_j — подмножество W'_j или W''_j соответственно, состоящее из точек v_i, для которых i-е неравенство содержит эту переменную. Легко видеть, что наша система равносильна задаче о покрытии множества V системой из n' подмножеств, часть которых — некоторые W'_j, а остальные — некоторые W''_j. Значит, и задача о покрытии NP-полна.

Некоторые итоги. Проведенные нами исследования показывают, что искомым инвариантом теории трудоемкости вычислений является понятие полиномиальной трудоемкости. Переход от одной машины к другой изменяет трудоемкость задачи полиномиальным образом; поэтому, если задача имеет полиномиальную трудоемкость вычисления, то это ее свойство не зависит от выбора машины. Напротив, значение степени в оценке трудоемкости, как мы видели, при смене машины может измениться, и более тонкие характеристики трудоемкости могут не быть инвариантными. Поэтому все, что «инвариантно», можно сказать о «хорошей» (с точки зрения трудоемкости) задаче, — это то, что она не более чем полиномиально трудоемка. Если же задача «плохая», т. е. имеет экспоненциальную или даже большую трудоемкость (см. примеры таких задач в [22]), то это свойство также инвариантно.

Второе важное понятие, до конца еще не исследованное, — это понятие полиномиальной трудоемкости проверки и связанный с ним класс PC. Если верно, что $P \neq PC$, то это означает существование обширного класса задач, для которых проверить правильность предъявленного ответа существенно проще, чем найти ответ.

И наконец, понятие NP-полноты выделяет обширный класс задач (среди которых много известных комбинаторных задач), которые в инвариантных терминах эквивалентны по трудоемкости и являются самыми трудоемкими среди задач классов PC и NP. Можно доказать, что либо $P = PC = NP$, либо $P \subset PC \subset NP$, где оба включения — строгие.

9.3. МЕТОД ВЕТВЕЙ И ГРАНИЦ

Исследование проблем перебора в § 9.1 показывает, что среди разрешимых задач могут быть хорошо и плохо реализуемые. Однако подобно тому, как в неразрешимой задаче возможны разрешимые частные случаи, так и в «плохой» задаче возможны «хорошие» частные случаи. Как правило, это объясняется тем, что при некоторых конкретных значениях исходных данных задачи удается полный перебор заменить перебором с отсечениями; иначе говоря, найти такие «хорошо» вычислимые условия, которые позволяют отбрасывать некоторые подмножества вариантов, удовлетворяющие этим условиям (т. е. не выполнять для них перебор). Наиболее типичным вычислительным методом сокращения перебора является метод ветвей и границ, к рассмотрению которого мы переходим. Этот метод нельзя назвать алгоритмом, поскольку его основные блоки (и прежде всего, вычисление условий отсечения) зависят от конкретной задачи; скорее, это вычислительная схема. Эффективность же метода может зависеть от конкретных данных задачи и в «плохих» случаях привести к тому же полному перебору.

Ветвление. Пусть имеется конечное множество M ситуаций или вариантов и функция F, принимающая различные значения в зависимости от выбранного варианта. Требуется среди вариантов найти оптимальный, т. е. такой, на котором функция F принимает минимальное значение (или максимальное в зависимости от условия задачи).

Метод ветвей и границ отыскания оптимального варианта состоит из ветвления и отсечений. Рассмотрим сначала ветвление.

Принимаем какой-либо принцип разбиения множества M на подмножества M_i такие, что $\bigcup_i M_i = M$, $M_i \cap M_i = 0$, $i \neq j$.

Затем, пользуясь этим же принципом, разбиваем полученные множества на части и т. д. После некоторого шага разбиения каждое множество содержит по одному варианту.

На каждом шаге вариант, оптимальный для всего множества M, принадлежит одному из M_i и является для него оптимальным. Поэтому его достаточно искать среди оптимальных вариантов для подмножеств M_i, составляющих множество M. Этим самым решение задачи для всего множества M сводится к решению задач для составляющих его множеств M_i и последующему отысканию оптимальных среди найденных для них решений.

Процесс разбиения множества M на подмножества становится наглядным, если его изобразить с помощью ориентированного графа следующим образом. Каждому множеству M_i, полученному при последовательном разбиении множества M на части, ставится в соответствие вершина графа. Обозначим вершины графа так же, как и множества, которым они соответствуют, т. е. M_i. От вершины M_i к вершине M_j проводим направленное ребро, если множество M_j получено непосредственным разбиением на части множества M_i. Полученный граф является деревом, поскольку в каждую его вершину входит единственное ребро и граф имеет одну начальную вершину [1].

Построение дерева с помощью разбиения множества вариантов на подмножества называется *ветвлением*.

Построенное указанным ранее способом дерево имеет число конечных вершин, равное числу элементов (вариантов) в множестве M: для каждого варианта одна конечная вершина. Такое дерево назовем деревом полного перебора.

Пример 9.1. Построить дерево путей из v_1 в v_6 в графе рис. 9.4.

Решение. Таких путей в рассматриваемом графе шесть:

$$L_1 = (v_1, v_3, v_6); \quad L_2 = (v_1, v_4, v_6); \quad L_3 = (v_1, v_2, v_4, v_6);$$

[1] В более общем случае может быть рассмотрено разделение множества на пересекающиеся подмножества. Кроме того, в некоторых задачах производится разбиение одного и того же множества на части несколько раз различными (независимыми) способами. В общих случаях граф, изображающий процесс разбиения, не является деревом.

$$L_4 = (v_1, v_2, v_4, v_5, v_6); \quad L_5 = v_1, v_4, v_5, v_6);$$
$$L_6 = (v_1, v_2, v_5, v_6).$$

Ветвление производим по принципу: множество M_i путей, проходящих через вершину v_i, разбиваем на непересекающиеся подмножества M_j' путей, содержащих ребра (v_i, v_j). На рис. 9.5 изображено дерево всех путей (полного перебора). В вершинах римскими цифрами указаны номера путей, образующих соответствующие множества.

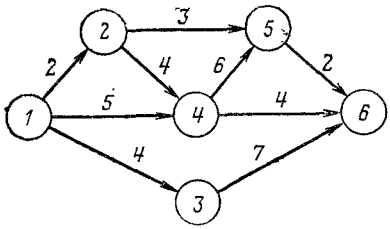

Рис. 9.4

Пусть маршрут — это путь в дереве от его корня к какой-нибудь конечной вершине. Функция F определена на конечных вершинах дерева полного перебора. Каждой конечной вершине такого дерева соответствует один и только один маршрут с концом в этой вершине. Поэтому функция F — это функция от маршрута.

Оценочные функции. Оценки. Разбиение множества на подмножества и независимое их рассмотрение приобретает смысл лишь в случае, если на некоторых этапах построе-

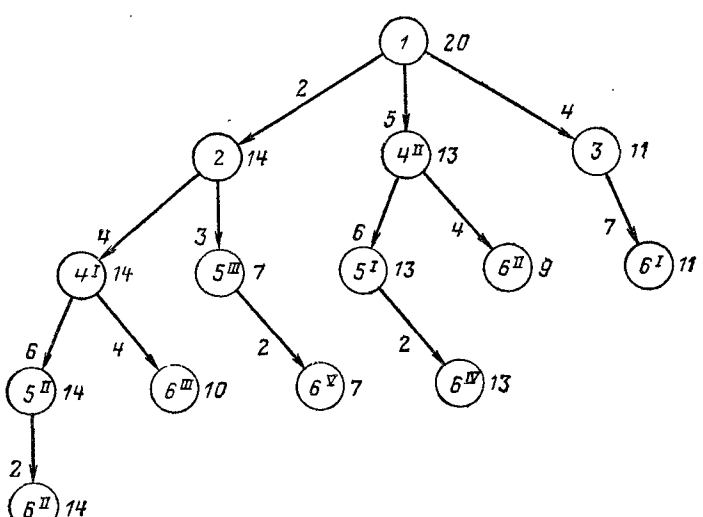

Рис. 9.5

ния дерева полного перебора удается установить, что в каком-то подмножестве нет варианта, оптимального для всего множества. Дальнейшее ветвление в этой вершине не происходит. От дерева полного перебора отсекаются ветви с корнем в ней (отсекаются вершины и ребра, следующие за ней).

Исключение множеств вариантов из рассмотрения производится с помощью оценочных функций. Оценочная функция — это функция $m(M_i)=m_i$, заданная на вершинах дерева полного перебора, возможно, исключая корень, и равная в его конечных вершинах соответствующим значениям функции F, а в остальных вершинах M_i дающая нижнюю или верхнюю границу значений функции F для вариантов, входящих в множество M_i.

Оценочная функция, дающая нижнюю границу, в процессе разбиения множества на части не убывает, а дающая верхнюю границу — не возрастает. Действительно, пусть $m(M_i)=m_i$ — оценочная функция, дающая нижнюю границу. Это значит, что все значения вариантов из множества M_i не меньше числа m_i. Тогда для любого подмножества M_i нижняя граница значений вошедших в него вариантов не может быть меньше m_i. Поэтому если для некоторой части M_i не находится значение оценочной функции, большее m_i, считаем его равным m_i.

Следует различать оценочные функции, определенные на любых подмножествах вариантов, и оценочные функции, определенные лишь на некоторых таких подмножествах. В первом случае при построении дерева можно использовать для ветвления произвольный принцип разбиения множеств вариантов на части. Во втором случае при ветвлении следует делить множества вариантов лишь на такие части, на которых определена рассматриваемая оценочная функция. Поэтому дерево полного перебора зависит, вообще говоря, от выбора оценочной функции.

Рассмотрим несколько примеров оценочных функций.

Пример 9.2. Пусть каждому ребру (M_k, M_l) дерева полного перебора поставлено в соответствие некоторое число $d_{kl}\geqslant 0$. Требуется среди маршрутов L найти такой, для которого величина $F(L)=\sum\limits_{(M_k,M_l)\in L} d_{kl}$ принимает минимальное значение.

Функция $m_i=\sum\limits_{1}^{i} d_{kl}$, где суммирование производится лишь по ребрам, входящим в путь от корня до вершины M_i,

является оценочной функцией, дающей нижнюю границу.

Пример 9.3. Если в дереве полного перебора примера 9.2 все конечные вершины имеют ранг n, то оценочной функцией, дающей нижнюю границу, является и функция

$$m_i = \sum_{1}^{i} d_{kl} + \sum_{i+1}^{n} \min d_{kl}, \qquad (9.1)$$

где минимумы ищутся среди значений d_{kl} для ребер M_i-ветви, начала которой имеют одинаковый ранг, знак $\sum_{i+1}^{n}$ означает суммирование по рангам начал этой ветви (отсчитываемым от M_i).

Аналогично функция

$$n_i = \sum_{1}^{i} d_{kl} + \sum_{i+1}^{n} \max d_{kl} \qquad (9.2)$$

является оценочной функцией, дающей верхнюю границу.

Пример 9.4. Если конечные вершины дерева полного перебора имеют неодинаковые ранги, то в примере 9.3 оценочные функции m_i и n_i принимают вид:

$$m_i = \sum_{1}^{i} d_{kl} + \sum_{i+1}^{s} \min d_{kl}; \quad n_i = \sum_{1}^{t} d_{kl} + \sum_{i+1}^{t} \max d_{kl}, \quad (9\,3)$$

где знак $\sum_{i+1}^{s}$ означает суммирование по рангам начал M_i-ветви, в которых представлены все пути, выходящие из M_i, а знак $\sum_{i+1}^{t}$ — суммирование по рангам ребер, в которых имеется хотя бы один путь, выходящий из M_i.

В примерах 9.2—9.4 для построения оценочной функции были использованы конкретные свойства изучаемых в задаче объектов (неотрицательных слагаемых d_{kl} и оценка их значений по рангам). Вообще основным принципом получения оценочной функции является индивидуальное изучение задачи.

Проиллюстрируем это следующим примером.

Пример 9.5. Рассмотрим задачу об отыскании одиого радиоактивного шарика среди n шариков, одинаковых в остальных отношениях. При этом используется прибор, позволяющий устанавливать, находится ли искомый шарик в рассматриваемой группе шариков: если да, при-

бор показывает единицу, если нет — нуль. Оценочной функцией любого подмножества шариков будем считать показание прибора при проверке этого подмножества. Такая оценочная функция определена на любом подмножестве шариков и тем самым на любом подмножестве вариантов размещения радиоактивного шарика. Всякое подмножество, для которого значение оценочной функции равно нулю, исключается из дальнейшего рассмотрения. Поэтому при каждой проверке исключается одно из подмножеств: либо проверяемое, либо его дополнение до множества шариков, оставшихся после исключений при предыдущих проверках.

Обозначим $k=k(n)$ минимальное число проб (вопросов), достаточное для выделения одного радиоактивного шарика из n. Можно доказать, что $k(n)$ не убывает при возрастании n. Оценим $k(n)$. Пусть мы спрашиваем о множестве M_1, содержащем n_1 шариков, $n_1<n$. Если значение оценочной функции $m(M_1)=1$, то на выделение радиоактивного шарика из n_1 шариков требуется $k(n_1)$ вопросов, при $m(M_1)=0$ требуется $k(n-n_1)$ вопросов да один вопрос (о подмножестве M_1) уже задан. Поэтому

$$k(n)=\min_{n_1<n}\max\{1+k(n_1);\quad 1+k(n-n_1)\}. \tag{9.4}$$

Из (9.4) следует, что минимальное число проб получается, если каждый раз задавать вопрос о половине рассматриваемых шариков, т. е. когда $n_1=\left\lceil\frac{n-1}{2}\right\rceil+1$, $n_2=n-n_1=\lceil n/2\rceil$.

Методом индукции для $k(n)$ получаем оценки

$$(s-1)\leqslant k(n)\leqslant s, \tag{9.5}$$

где $2^{s-1}<n\leqslant 2^s$. Отсюда

$$\lceil\log_2 n\rceil\leqslant k(n)\leqslant[\log_2(n-1)]+1. \tag{9.6}$$

Действительно, непосредственно убеждаемся в том, что (9.5), (9.6) верно для $n=2$ и $n=3$. Далее из предположения, что (9.5), (9.6) верно при $2^{s-3}<n\leqslant 2^{s-2}$, и учитывая значения n_1 и n_2, получаем, что эти оценки верны и при $2^{s-1}<n\leqslant 2^s$.

Из (9.6) следует, что минимальное число k проб, достаточное для выделения из n шариков одного радиоактивного, меньше числа проб при полном переборе. Это уменьшение получается с помощью оценочной функции: как было указано ранее, подмножества шариков, на которых она равна нулю, каждый раз отсекаются.

Значение оценочной функции для данного множества M_i назовем его оценкой и обозначим $\text{оц}M_i=m_i$. Она может не быть точной границей значений вариантов из M_i. Чем ближе оценка к точной границе, тем эффективнее применение метода ветвей и границ, ибо число отсекаемых вершин де-

рева полного перебора зависит, в частности, от силы оценки. Действительно, пусть в задаче отыскивается минимум и на некоторых подмножествах определены две нижние оценочные функции m_i и m_i', причем m_i' — более точная, т. е. $m_i' \geqslant m_i$. Пусть найдено значение F_0 для некоторого варианта. Если для какой-то вершины M_{i_0} имеем $m(M_{i_0}) \geqslant F_0$, то и $m'(M_{i_0}) \geqslant F_0$ и множество M_{i_0} не следует далее рассматривать, так как оно не содержит оптимального варианта. Но при $m(M_{i_0}) < F_0$ может оказаться $m'(M_{i_0}) \geqslant F_0$. Тогда при использовании оценочной функции m_i множество M_{i_0} нужно изучать дальше, а при использовании m_i' этого делать не следует. К сожалению, обычно чем точнее оценка, тем больших по объему вычислений требует ее отыскание.

Так как точность оценки зависит, в частности, от того, насколько целесообразен принятый принцип разбиения на части рассматриваемого множества вариантов, то при выборе способа разбиения из нескольких возможных следует отдать предпочтение тому из них, при котором оценки полученных подмножеств более точные. Не менее важно, насколько сильно на этих подмножествах различаются между собой значения оценочной функции. Лучшим является тот из способов разбиения (и такая оценочная функция), при котором разность между оценками подмножеств наибольшая.

Отсечение вариантов (ветвей). Можно указать следующие основные принципы отсечения ветвей.

А. Отсечение по сравнению с уже найденным значением функции F. Пусть ищется минимальное значение F и получены оценки снизу оцM_i для части вершин дерева и значение $F = F_0$ для некоторого варианта. Тогда в вершинах, где оц$M_i \geqslant F_0$, ветвление прекращается.

Количество отсечений по этому способу тем больше, чем меньше F_0.

Б. Отсечение по сравнению двух оценок. Его можно производить, когда для M_i строятся оценка снизу (ноцM_i) и оценка сверху (воцM_i). Если при этом для некоторых M_i и M_j оказывается, что ноц$M_i \geqslant$ воцM_j, то при отыскании минимума F ветвление из вершины M_i прекращается.

В. Ветвление в данной вершине прекращается, если известно, что соответствующее ей подмножество не содержит оптимального варианта или известен оптимальный среди принадлежащих ему вариантов.

Порядок продолжения ветвления может быть выбран различными способами. Укажем основные из них.

1. Ветвление по минимальной нижней границе (при отыскании минимума F). Сначала строятся (все или некоторые) ребра, выходящие из корня. Затем в каждый момент ветвление производится в вершине с минимальной из уже найденных оценок снизу. Для всех вершин, в которых производится ветвление, можно строить все ребра, выходящие из них в дереве полного перебора, либо только часть этих ребер.

2. Развитие дерева по ветвям (последовательное построение ветвей). Сначала строим один маршрут, включая его конечную вершину. При этом получаем некоторое значение F_0, которым можно пользоваться при отсечении множества вариантов. Пусть M_k — первая от конца этого маршрута вершина, такая, что еще не построено выходящее из нее ребро (M_k, M_i) и при этом оц$M_i < F_0$, т. е. множество M_i не отсекается (отыскивается минимум). Строим ребро (M_k, M_i) и достраиваем какой-нибудь из проходящих через нее маршрутов до конца либо до вершины, в которой происходит отсечение. Далее строим ребро (M_j, M_l) в первой от конца этого маршрута (от конца построенной его части) вершине M_j, такой, что выходящее из нее ребро (M_j, M_l) еще не построено и при этом множество M_l не отсекается и т. д.

При таком способе продолжения ветвления в построенном дереве обычно больше вершин, чем при ветвлении по минимальной нижней границе, но запоминать приходится данные о меньшем количестве построенных вершин.

Никаких точных оценок числа отсечений при различных способах продолжения ветвления указать нельзя, все соображения здесь носят эвристический характер.

Пример 9.6. Используя оценку типа (9.3), найти путь максимальной длины из p_1 в p_6 в графе рис. 9.2.

На рис. 9.6 приведено решение при ветвлении по максимальной верхней границе, на рис. 9.7, *а* и *б* — при развитии дерева по ветвям. Количество построенных при этом вершин дерева зависит от очередности развития ветвей.

На рисунках слева от вершин проставлены оценки, найденные по формулам типа (9.3), справа — номера вершин в порядке появления при построении дерева.

3. Еще один способ продолжения ветвления сочетает принципы рассмотренных ранее двух способов. Зададим некоторое число $\delta > 0$. Пусть M_i — вершина, в которую мы

попали при очередном шаге ветвлений. Если оцM_i превосходит минимальную из ранее найденных оценок не более чем на δr_i, где r_i — ранг рассматриваемой вершины, то ветвление продолжаем из этой вершины. В противном слу-

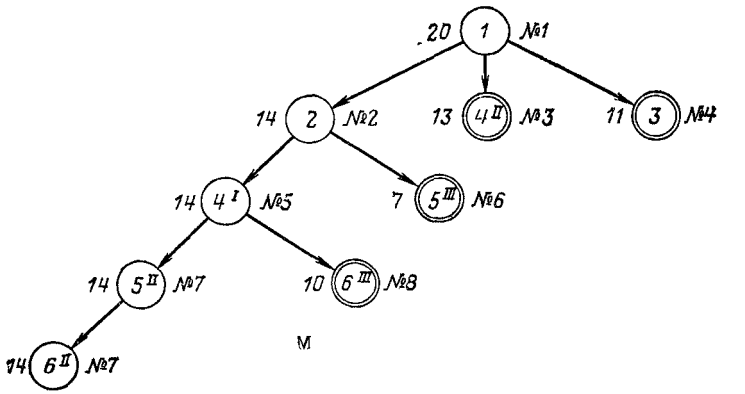

Рис. 9.6

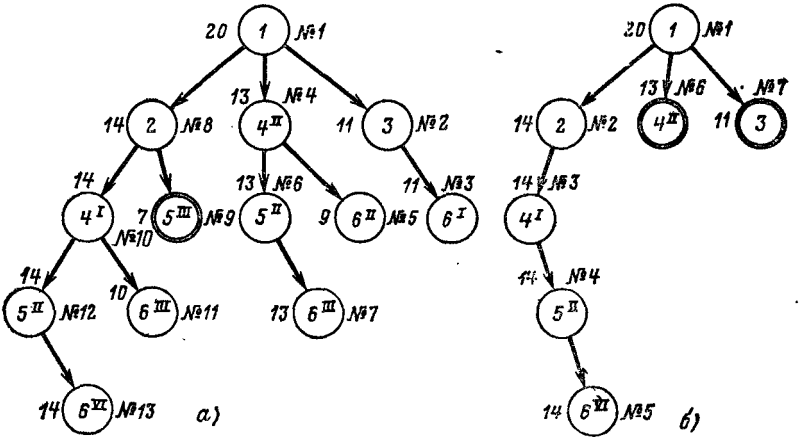

Рис. 9.7

чае производим ветвление в вершине с минимальной нижней границей.

Отметим, наконец, что могут возникнуть дополнительные отсечения, если отыскивается не минимум функции F, а лишь выясняется, существует ли значение F, удовлетворяющее условию $F(M_i) \geqslant C$.

Задача об очередности выполнения работ. Постановка задачи. Пусть сеть комплекса работ задана рис. 9.8, где каждому ребру соответствует работа. Заданы постоянные продолжительности d_i всех работ сети (табл. 9.1).

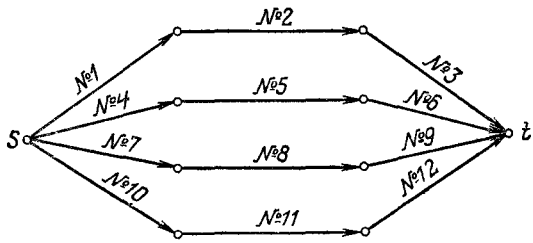

Рис. 9.8

Таблица 9.1

Номер работы i	1	2	3	4	5	6	7	8	9	10	11	12
Продолжительность d_i	10	5	3	12	7	10	28	4	6	23	7	3

Имеется нескладируемый ресурс в количестве 1 (например, башенный кран), который необходим при выполнении работ № 2, 5, 8, 11. Для остальных работ он не нужен; потребность в других видах ресурса не учитывается (считается, что их всегда достаточно). Требуется указать такой порядок выполнения работ № 2, 5, 8, 11 (впредь будем их называть ресурсными), чтобы задержка выполнения комплекса работ по сравнению с критическим временем $T_{кр}=38$ был минимальной.

Обозначим: $t_i^{р}$ — ранний срок начала i-й работы (т. е. момент, раньше которого она не может начаться), $t_i^{н}$ — время начала i-й работы, t_i^{0} — время окончания i-й работы; $\tau_i=d_{i,1}$. Учитывая вид сети, для i-й ресурсной работы имеем $t_i^{р}=d_{i-1}$.

Ветвление. Всего имеется 24 варианта последовательности выполнения ресурсных работ (4!=24). Разобьем их на подмножества вариантов, в которых первой выполняется одна и та же ресурсная работа. Ее будем считать фиксированной. Это варианты с одинаковым началом. Таких подмножеств четыре (по шесть вариантов в каждом).

Далее полученные подмножества разбиваем на части, в которых фиксирован порядок выполнения уже двух ресурсных работ и т. д.

Отсечение. При отыскании оценок снизу будем опираться на следующий факт: пусть задан некоторый момент времени t и календарный план выполнения ресурсных работ из некоторого их множества R найден с помощью формул

$$t_j^{\text{н}} = t + \sum_{\tau_i < \tau_j} d_i,\ i,\ j \in R;$$

$$t_i^0 = t_i^{\text{н}} + d_j,\ j \in R;$$

если в этом плане для всех i выполнено неравенство $t_i^{\text{н}} \geqslant t_i^{\text{р}}$, то он минимизирует величину $\max\limits_{i \in R} (t_i^{\text{н}} + d_i + \tau_i - t)$.

При решении задачи время t принимается равным сроку окончания последней из ресурсных работ, фиксированных в данной группе вариантов. Вычисления производятся с помощью таблиц такого типа, как табл. 9.2: $t_i^{\text{н}}$ — срок начала

Таблица 9.2

i	$t_i^{\text{р}}$	d_i	τ_i	$t_i^{\text{н}}$	$T_i = t_i^{\text{н}} + d_i + \tau_i$	
2	10	5	3	10	18	$T_{max} = 36$
5	12	7	10	15	32	
8	28	4	6	22	32	
11	23	7	3	26	36	

i-й ресурсной работы при соответствующем в таблице порядке выполнения ресурсных работ; T_i — длина пути (т. е. суммарная продолжительность) от начальной вершины к конечной, проходящего через i-ю ресурсную работу.

В табл. 9.2 приведены значения T_i для подмножества вариантов, в которых зафиксировано положение одной ресурсной работы — работы № 2. Имеем $t_2^{\text{н}} = 10 = t_2^{\text{р}}$; $t_5^{\text{н}} = t_2^{\text{н}} + d_2 = 10 + 5 = 15$, $t_8^{\text{н}} = t_5 + d_5 = 15 + 7 = 22$; $t_{11}^{\text{н}} = t_8 + d_8 = 22 + 4 = 26$, причем $t_8^{\text{н}} = 22 < 28 = t_8^{\text{р}}$. Из таблицы получаем нижнюю границу $T = \max\{T_2; T_5; T_8; T_{11}\} = 36$ продолжительности для всех вариантов рассматриваемого подмножества.

В первой строке табл. 9.2 выписаны данные для ресурс-

ных работ, положение которых в рассматриваемой вершине уже зафиксировано. Для них время начала $t_i^н$ допустимое, т. е. $t_i^н \geqslant t_i^р$. Остальные ресурсные работы выписываются в порядке убывания τ_i. Для каждой из них $t_i^н$ равно сумме времени начала и продолжительности записанной перед ней (выше нее) ресурсной работы.

Заполнив таблицу, находим $T = \max T_i$. Если для ресурсных работ, положение которых в рассматриваемой вершине не зафиксировано, оказалось $t_i^н \geqslant t_i^р$, то найденное T — минимальное время выполнения комплекса работ для множества вариантов, соответствующих этой вершине, причем минимум достигается при выполнении ресурсных работ в порядке, указанном таблицей. Если же окажется, что для какой-либо ресурсной работы, положение которой не зафиксировано, $t_i^н < t_i^р$, то найденное время T является нижней границей для этого подмножества вариантов. Действительно, T получено без учета того, что работы нельзя начинать ранее их раннего срока начала.

При фиксации положения одной ресурсной работы № 8 получаем табл. 9.3.

Таблица 9.3

i	$t_i^р$	d_i	τ_i	t_i	T_i	
8	28	4	6	28	38	$T_{min} = 52$
5	12	7	10	32	49	
2	10	5	3	39	47	
11	23	7	3	44	52	

Поскольку здесь все $t_i^н \geqslant t_i^р$, число $T = \max T_i = 52$ равно минимальной продолжительности комплекса работ для множества вариантов, в которых первой из ресурсных работ выполняется работа № 8. Это время достигается при следующем порядке выполнения ресурсных работ: № 8, 5, 2, 11. Независимо от результатов дальнейших вычислений соответствующую ветвь графика развивать не следует.

Одновременно с подсчетом оценок строим дерево, например, развивая ветвь с наименьшей нижней границей (рис. 9.9). Последовательность рассмотрения вершин указана номерами возле них на рисунке.

В данном примере числа 42 и 40 в вершинах № 8 и 9 являются значениями соответствующих вариантов. В вер-

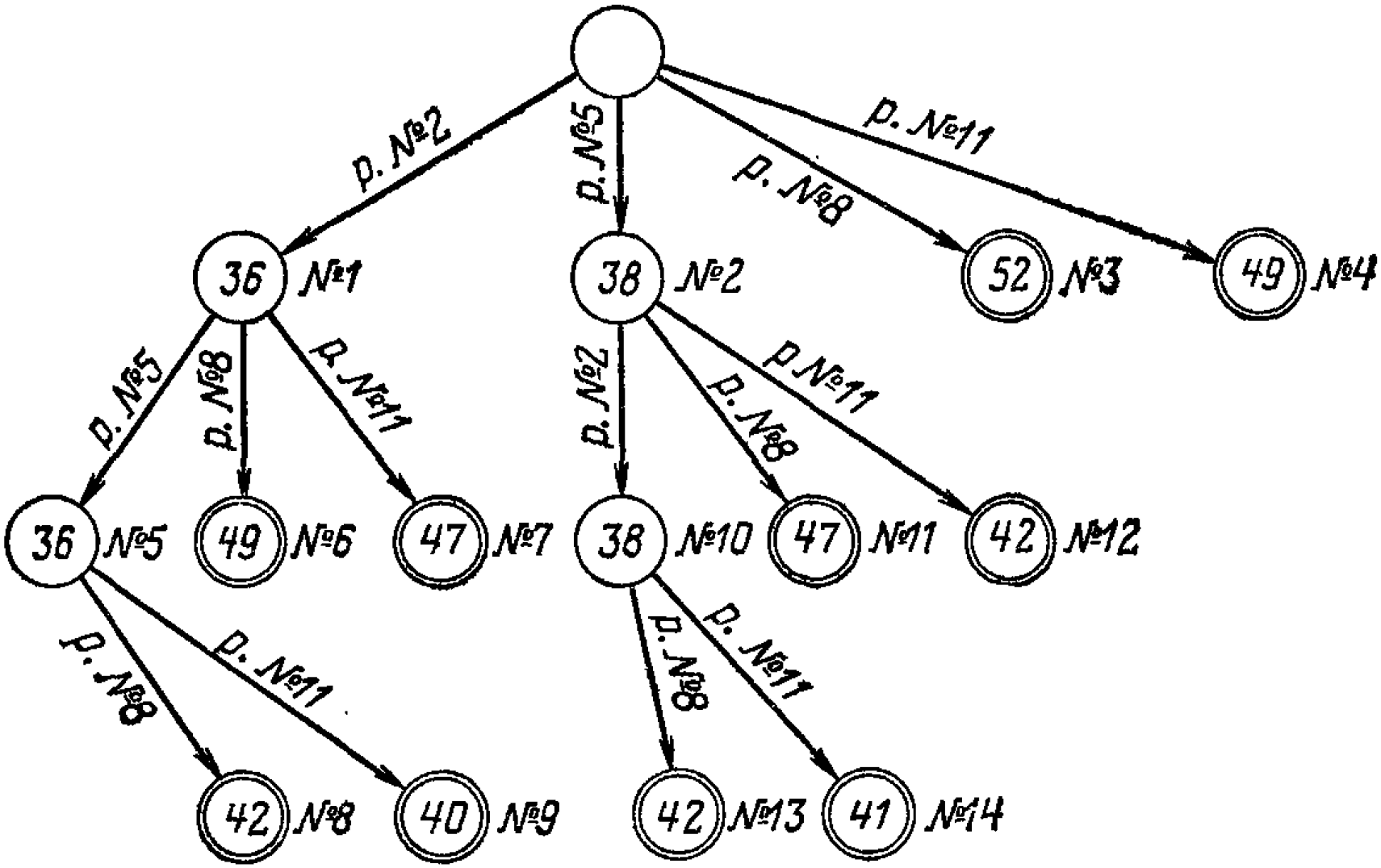

Рис. 9.9

шинах № 3, 4, 6, 7, 11, 12 ветвление производить не следует по двум причинам: так как их оценки (на рисунке записаны в кружках) — минимумы для соответствующих групп вариантов и так как каждая из них больше уже полученного времени $t=40$. Любой одной из этих причин достаточно для прекращения ветвления.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

ОГЛАВЛЕНИЕ

Производственное издание

Кузнецов Олег Петрович
Адельсон-Вельский Георгий Максимович

ДИСКРЕТНАЯ МАТЕМАТИКА ДЛЯ ИНЖЕНЕРА

Редактор *С. В. Петров*
Редактор издательства *З. И. Михеева*
Художник переплета *Ю. С. Шлепер*
Художественный редактор *Т. А. Дворецкова*
Технический редактор *Г. В. Преображенская*
Корректор *М. Г. Гулина*
ИБ № 1361

Сдано в набор 30.12.86. Подписано в печать 15.02.88. Т-08009. Формат 84×108$^{1}/_{32}$. Бумага типографская № 2. Гарнитура литературная. Печать высокая. Усл. печ. л. 25,2. Усл. кр.-отт. 25,2. Уч.-изд. л. 27,03. Тираж 30 000 экз. Заказ 750. Цена 1 р. 80 к.

Энергоатомиздат, 113114, Москва, М-114, Шлюзовая наб., 10

Владимирская типография Союзполиграфпрома при Госкомиздате СССР
600000, г. Владимир, Октябрьский проспект, д. 7