В. М. Глушков

ВВЕДЕНИЕ В КИБЕРНЕТИКУ

Издательство АН УССР

АКАДЕМИЯ НАУК УКРАИНСКОЙ ССР
НАУЧНЫЙ СОВЕТ ПО КИБЕРНЕТИКЕ

В. М. ГЛУШКОВ

ВВЕДЕНИЕ В КИБЕРНЕТИКУ

ИЗДАТЕЛЬСТВО АКАДЕМИИ НАУК УКРАИНСКОЙ С
КИЕВ — 1964

В книге собран и обобщен материал, необходимый для построения таких разделов современной кибернетики, как теория электронных цифровых машин, теория дискретных автоматов и теория дискретных самоорганизующихся систем, автоматизацня мыслительных процессов, теория распознавания образов и др. Изложены основы теории булевых функций, теория алгоритмов, логические исчисления и логические сети, основные вопросы теории автоматов, принципы построения электронных цифровых машин и универсальиых алгоритмических языков, основы теории персептронов, некоторые принципиальные вопросы теории самоорганизующихся систем.

Многие фундаментальные результаты по математической логике и теории алгоритмов поданы конспективно, без подробных доказательств, а в ряде случаев и вовсе без доказательств.

Рассчитаиа книга на широкие круги математиков и научных работников многих специальностей, желающих познакомиться с проблемами современной кибернетики.

ПРЕДИСЛОВИЕ

Цель настоящей книги заключается в том, чтобы познакомить читателя с рядом новых научных направлений, составляющих основу кибернетики в ее современном понимании. В наиболее общем плане все эти направления можно подразделить на две большие группы — на общую теорию *преобразования информации* и на теорию и принципы построения различного рода *преобразователей информации*. Однако материал, который можно отнести к указанным большим направлениям, настолько обширен, что его вряд ли возможно изложить даже конспективно в одной книге. Поэтому необходимо было произвести отбор материала в соответствии с теми или иными общими принципами.

Материал для настоящей книги отобран в соответствии с двумя основными принципами. Первый принцип заключается в требовании достаточно строгого оформления материала, позволяющего изложить его в виде некоторой математической теории (хотя и с характерным для кибернетики уклоном в сторону практического моделирования). Второй принцип состоит в том, что автор ограничивается, как правило, дискретными способами представления информации и дискретными преобразователями информации.

В результате отбора в книгу включены следующие основные разделы: *теория алгоритмов* (включая программирование для универсальных электронных цифровых машин и универсальные алгоритмические языки для программирования), *теория дискретных автоматов* (включая теорию булевых функций и понятие о принципах построения универсальных электронных цифровых машин), *теория дискретных самоорганизующихся систем* (включая элементы теории оптимальных решений) и, наконец, *математическая логика* (исчисление высказываний, узкое исчисление предикатов и формальная арифметика), рассматриваемая как база для автоматизации процесса построения дедуктивных (основанных на той или иной системе аксиом) теорий.

Степень подробности изложения материала определяется прежде всего степенью его новизны. Более новые разделы, от-

носящиеся к собственно кибернетике, излагаются более подробно, формулируемые теоремы снабжаются достаточно подробными доказательствами. В то же время в таких разделах, как абстрактная теория алгоритмов и математическая логика, которые развивались в рамках традиционной математики, материал излагается более сжато, доказательства, как правило, опускаются.

Автор старался, однако, дать понятие об основных идеях и методах, с помощью которых устанавливается справедливость таких принципиальных с точки зрения математической логики предложений, как теорема Гёделя о неполноте арифметики или теоремы, устанавливающие алгоритмическую неразрешимость тех или иных проблем.

Книга не претендует на то, чтобы заменить специальные монографии по отдельным включенным в нее разделам. Основное ее назначение состоит в том, чтобы помочь широким кругам математиков и инженеров овладеть тем минимумом знаний, который необходим для работы в области теоретических проблем современной «дискретной» кибернетики. Хорошо известно, что наличие подробных монографий на ту или иную тему далеко не всегда обеспечивает возможность ознакомления с предметом специально не подготовленных читателей. Достаточно убедительным является, например, тот факт, что, несмотря на наличие специальных монографий, такой, имеющий принципиальное значение для всей математики результат, как упоминавшаяся выше теорема Гёделя, остается известным широким кругам математиков разве что понаслышке.

Что же касается настоящей книги, то она предполагает у читателя (да и то лишь одна, четвертая, глава) знание лишь таких элементов математического анализа и теории вероятностей, которые известны практически каждому инженеру, не говоря уже о математиках. Менее широко известные математические результаты, необходимые для понимания основного содержания книги, включены в нее в качестве вспомогательного материала. Примером такого рода вспомогательного материала может служить ряд предложений теории вероятностей, изложенных в гл. IV, § 2.

На случай, если читатель пожелает углубить свои знания в том или ином разделе или познакомиться с подробными доказательствами тех предложений, которые, хотя и помещены в книге, но подробно в ней не доказываются, проведем обзор содержания книги с указанием специальных монографий (на русском языке) по отдельным ее разделам. К сожалению, не по всем разделам книги можно найти такого рода монографии.

В первой главе дано описание основных теоретических универсальных алгоритмических систем (нормальные алгоритмы Маркова, алгоритмическая система Колмогорова — Успенского,

рекурсивные функции, алгоритмы Поста и машины Тьюринга). Приведены также основные принципы доказательств алгоритмической неразрешимости некоторых простейших массовых проблем.

Обобщающей монографии по всей теории алгоритмов в целом в настоящее время нет. Более того, не все из упомянутых выше вопросов сколько-нибудь подробно изложены в монографической литературе. Из капитальных монографий по отдельным алгоритмическим системам назовем следующие: по теории нормальных алгоритмов — «Теория алгорифмов» А. А. Маркова [53]; по теории рекурсивных функций и машин Тьюринга — «Введение в метаматематику» С. К. Клини [42] и «Лекции о вычислимых функциях» В. А. Успенского [76].

Во второй главе излагаются теория булевых функций и ее приложения к теории схем дискретных автоматов. Более подробно эти вопросы изложены в монографии В. М. Глушкова «Синтез цифровых автоматов» [26].

Кроме того, во второй главе излагаются основы теории высказываний. Более подробно с исчислением высказываний можно познакомиться, например, по монографии П. С. Новикова «Элементы математической логики» [61].

Третья глава посвящена абстрактной и структурной теории дискретных (конечных) автоматов. Более детально относящиеся к этой теме вопросы рассмотрены в упоминавшейся выше монографии В. М. Глушкова. Несколько с иных позиций эти вопросы рассматриваются в монографии Н. Е. Кобринского и Б. А. Трахтенброта «Введение в теорию конечных автоматов» [47].

В четвертой главе излагаются основы теории дискретных самоорганизующихся систем. Определяется количественная мера самоорганизации и самообучения, исследуется поведение случайных автоматов и автоматов, работающих в условиях случайных внешних воздействий. Особое место уделяется проблеме распознавания образов и теории одного класса устройств (так называемых α-персептронов), предназначенных для решения этой проблемы. Рассматриваются некоторые вопросы моделирования условных рефлексов, а также процессов обучения распознаванию смысла и выработки новых понятий. В конце главы, в связи с понятием самонастройки и экстремального регулирования, описывается ряд общих методов решения экстремальных задач (метод наискорейшего спуска и его уточнение, симплекс-метод решения задач линейного программирования и так называемый метод последовательного анализа вариантов для решения задач динамического программирования).

Обобщающей монографии по материалам четвертой главы пока нет. Более того, почти все излагающиеся в ней вопросы (за исключением метода наискорейшего спуска и симплекс-метода) до сих пор не освещались в монографической литера-

туре. Ряд вопросов, близких к тем, которые рассматриваются в этой главе (но не вполне тождественных с ними), освещен в «Нейродинамике» Ф. Розенблата, пока еще не переведенной на русский язык. Методам решения экспериментальных задач (исключая метод последовательного анализа вариантов) посвящено большое количество монографий. Однако не будем их перечислять, поскольку эти вопросы не имеют прямого отношения к основному направлению настоящей книги.

В пятой главе излагаются основные принципы построения универсальных электронных цифровых машин и программирования для этих машин. Этому вопросу в настоящее время посвящено так много разнообразных монографий, что перечислить их все было бы крайне затруднительно. В части, касающейся программирования, может быть названа, например, монография Б. В. Гнеденко, В. С. Королюка и Е. Л. Ющенко «Элементы программирования» [31]. Что же касается принципов построения машин, то, несмотря на наличие многих хороших специальных монографий по этому вопросу, подробного изложения материала в нужном для нас плане в них нет: принципы построения электронных цифровых машин излагаются, как правило, в отрыве от общей теории алгоритмов.

Кроме сказанного, в пятой главе подробно описывается универсальный алгоритмический язык АЛГОЛ-60 и приводятся примеры программирования на АЛГОЛе различных задач, прежде всего из теории самоорганизующихся систем. Разобран, в частности, вопрос о программировании процесса обучения персептрона и упрощенной модели процесса биологической эволюции. С монографической литературой по этому вопросу дело также обстоит плохо: «Сообщение об алгоритмическом языке АЛГОЛ-60» (под редакцией П. Наура), изданное Вычислительным центром АН СССР (Москва, 1960 г.), носит справочный характер и мало пригодно для практического обучения языку АЛГОЛ.

В последней (шестой) главе дается конспективное изложение основ узкого исчисления предикатов (включая формальную систему Генцена) и формальной арифметики (включая теорему Гёделя о неполноте арифметики). Подробные доказательства приведенных предложений можно найти в упомянутых выше монографиях С. К. Клини и П. С. Новикова. В эту главу включены также элементы автоматизации доказательств и формулировок теорем в дедуктивных теориях. Затронутые здесь вопросы до сих пор не освещались в монографической литературе.

Как показывает перечень излагаемого в книге материала, в нее не включены многие интересные разделы современной кибернетики. Учитывая приведенные выше критерии отбора материала, можно было бы, например, включить в книгу изложение основ математической лингвистики или элементов теории игр. Однако и без того значительный объем книги заставил автора

отказаться от попыток включения в нее какого-либо дополнительного материала. Вместе с тем содержание книги охватывает круг вопросов, которые в настоящее время принято считать основой теоретической кибернетики (с учетом ограничения на дискретность). Автор надеется поэтому, что книга может оказать помощь в овладении математическим аппаратом кибернетики и подготовке для работы в теоретических областях людей, занимающихся кибернетикой в отдельных ее прикладных аспектах, а также лиц, интересующихся теоретическими проблемами кибернетики.

В настоящей книге широко использованы материалы лекций по различным разделам кибернетики и математической логики, прочитанных автором в Киевском университете и в киевском Доме научно-технической пропаганды в 1959—1962 гг. Часть этих материалов (например, теория алгоритмов) была издана ранее для служебного пользования. Настоящую книгу можно рассматривать как первое достаточно полное пособие для изучающих указанные разделы кибернетики.

Глава I

АБСТРАКТНАЯ ТЕОРИЯ АЛГОРИТМОВ

§ 1. Алфавитные операторы и алгоритмы

Алгоритмами в современной математике принято называть конструктивно задаваемые соответствия между словами в абстрактных алфавитах.

Абстрактным алфавитом называется любая конечная совокупность объектов, называемых *буквами* данного алфавита. Природа этих объектов для нас совершенно безразлична. Буквами абстрактных алфавитов можно считать, например, буквы алфавита какого-либо языка (русского, латинского, греческого и т. п.), цифры, любые значки, рисунки и т. д. При желании можно ввести абстрактный алфавит, буквами которого будут считаться целые слова того или иного конкретного языка (например, русского). Важно лишь, чтобы рассматриваемый алфавит был *конечным*, т. е. состоял из конечного числа букв.

Вводя понятие (абстрактного) алфавита, определим *слова* в этом алфавите как любые конечные упорядоченные последовательности букв. Например, в алфавите $A=A(x, y)$, состоящем из двух букв x и y, словами следует считать любые последовательности $x, y, xx, xy, yx, yy, xxx,...$ Число букв в слове называется обычно длиной этого слова, так что выписанные нами слова в алфавите имеют соответственно длины 1, 1, 2, 2, 2, 2, 3,...

Наряду со словами *положительной длины* (состоящими не менее чем из одной буквы), в ряде случаев целесообразно рассматривать также пустое слово, не содержащее ни одной буквы. В настоящей главе для обозначения пустого слова употребляется малая латинская буква e. Иногда, впрочем, удобно обозначать пустое слово в полном соответствии с его определением, не выписывая на соответствующем этому слову месте ни одной буквы.

Заметим, что при принятом определении понятие слова в русском алфавите будет отличаться от понятия слова, принятого

в обычном языке. При нашем определении словами следует считать любые сочетания букв, в том числе и бессмысленные: сочетания букв «алгоритм», «математика», «ьклт», «дддд» в равной степени должны считаться словами русского алфавита (рассматриваемого как абстрактный алфавит).

При расширении алфавита, т. е. при включении в его состав новых букв, понятие слова может претерпеть существенные изменения. Если, например, расширить русский алфавит «буквами» (« » — кавычки) и (, — запятая), то выписанные только что четыре слова в русском алфавите могут рассматриваться как одно слово в расширенном таким образом алфавите. Пополняя русский алфавит знаками препинания и знаком раздела (пустым местом, оставленным между соседними словами), можно при желании целые фразы, абзацы и даже целые книги рассматривать как отдельные слова.

Точно таким же образом выражение 69+72, представляющее собой два слова (69 и 72) в алфавите A из 10 цифр (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), соединенные знаком суммы, можно рассматривать как *одно слово* в расширении алфавита A, которое получается в результате присоединения к нему новой буквы «+» (знак суммы).

Алфавитным оператором или алфавитным отображением называется всякое соответствие (функция), сопоставляющее словам в том или ином алфавите слова в том же самом или в некотором другом фиксированном алфавите. Первый алфавит называется при этом входным, а второй — выходным алфавитом данного оператора. В случае совпадения входного и выходного алфавитов говорят, что алфавитный оператор задан в соответствующем алфавите.

В дальнейшем речь идет в основном об *однозначных* алфавитных операторах, сопоставляющих каждому *входному слову* (слову во входном алфавите оператора) не более одного *выходного слова (слова в выходном алфавите оператора)*. Если алфавитный оператор не сопоставляет данному входному слову p никакого выходного слова (в том числе и пустого), то говорят, что он *не определен на этом слове*. Совокупность всех слов, на которых алфавитный оператор определен, называется его *областью определения*.

На основании сказанного в дальнейшем под термином «алфавитный оператор» будем всегда понимать (если не оговорено противное) однозначное, вообще говоря, частично определенное отображение множества слов во входном алфавите оператора в множество слов в его выходном алфавите.

Благодаря возможности задавать алфавитные операторы не на всех словах, можно, не нарушая общности, всякий раз считать, что входной и выходной алфавиты оператора совпадают. Для этого достаточно, очевидно, объединить входной и выходной алфавиты данного оператора φ в один общий алфавит A и

рассматривать оператор φ как оператор в этом объединенном алфавите, заданный лишь на тех словах, которые входили в первоначальную область определения оператора φ.

С каждым алфавитным оператором связывается интуитивное представление о его сложности. Наиболее простыми являются операторы, осуществляющие *побуквенные* отображения. Побуквенное отображение состоит в том, что каждая буква x входного слова p заменяется некоторой буквой y выходного алфавита оператора, зависящей только от буквы x, но не от выбора входного слова p. Побуквенное отображение полностью определяется заданием соответствия между буквами входного и выходного алфавитов.

Для последующего изложения большое значение имеют так называемые *кодирующие отображения*, которые для краткости будем называть просто *кодированиями*. В наиболее простом случае слова в одном алфавите, скажем в алфавите A, кодируются словами в другом алфавите — B следующим способом: каждой букве a_i алфавита A сопоставляется некоторая конечная последовательность $b_{i_1}, b_{i_2} \ldots b_{i_k}$ букв в алфавите B, называемая *кодом* соответствующей буквы, так, чтобы различным буквам алфавита A сопоставлялись различные коды.

Для построения искомого кодирующего отображения достаточно теперь заменить все буквы любого слова p в алфавите A соответствующими им кодами. Получаемое таким образом слово в алфавите B назовем *кодом* исходного слова p. Условимся, что кодирующее отображение должно быть непременно *обратимым*. Иначе говоря, различные слова в алфавите A должны иметь различные коды. Условие обратимости кодирования есть не что иное, как условие *взаимной однозначности* соответствующего кодирующего отображения.

Легко видеть, что обратимость кодирования не обеспечивается одним лишь условием, чтобы коды различных букв (слов длины *1*) были различны. Действительно, если букве a_1 сопоставляется код bb, а букве a_2 — код b, то код bbb будет соответствовать, очевидно, как слову a_1a_2, так и словам a_2a_1 и $a_2a_2a_2$.

Нетрудно проверить, что кодирование будет обратимым всякий раз, когда выполняются следующие два условия:

а) *коды различных букв исходного алфавита A различны*;

б) *код любой буквы алфавита A не может совпадать ни с каким из начальных отрезков кодов других букв этого алфавита**.

В самом деле, предположим, что оба эти условия выполнены, и пусть слово $q = b_{i_1} b_{i_2} \ldots b_{i_n}$ является кодом какого-либо слова $p = a_{j_1}a_{j_2} \ldots a_{j_m}$ в алфавите A. Покажем, что по коду q можно однозначно восстановить слово p. В силу условия «б» только один начальный отрезок слова q может совпадать с кодом какой-либо

* Слово p называется начальным отрезком слова q, если слово q имеет вид $q = pr$, где r — любое слово (в том числе, может быть, и пустое).

буквы алфавита A. Ясно, что таким отрезком является код буквы a_{j_1}. Отбрасывая этот отрезок, получим код q_1 слова $p_1 = a_{j_2} \ldots a_{j_m}$. Применяя к нему те же самые рассуждения, однозначно восстановим следующую букву (a_{j_2}) слова p и т. д. Таким способом однозначно восстанавливаются одна за другой все буквы слова p. Следовательно, любому данному коду может соответствовать лишь одно слово в алфавите A, чем доказана обратимость (взаимная однозначность) кодирующего отображения.

Условие «б» выполняется, если коды всех букв исходного алфавита A имеют одинаковую длину. Кодирование в этом случае условимся называть *нормальным*. Использование кодирования позволяет сводить изучение произвольных алфавитных отображений к алфавитным отображениям в каком-либо раз навсегда выбранном стандартном алфавите. Наиболее часто в качестве такого стандартного алфавита выбирается так называемый *двоичный* алфавит, состоящий из двух букв, которые обычно отождествляются с цифрами 0 и 1.

Пусть A — произвольный, а B — стандартный (например, двоичный) алфавиты, состоящие более чем из одной буквы. Если n— число букв в алфавите A, а m — число букв в алфавите B, то всегда можно выбрать число k так, чтобы удовлетворялось неравенство

$$m^k \geqslant n. \tag{1}$$

Поскольку число различных слов длины k в m-буквенном алфавите равно, очевидно, m^k, то неравенство (1) показывает, что можно закодировать все буквы в алфавите A словами длины k в алфавите B так, чтобы коды различных букв были различными. Любое такое кодирование будет нормальным и порождает, в силу сказанного выше, обратимое кодирующее отображение слов в алфавите A в слова в алфавите B. Обозначим это отображение через α, а через α^{-1} будем обозначать обратное ему отображение, которое переводит каждое слово q в алфавите B, являющееся кодом некоторого слова p в алфавите A, в слово p.

Если теперь φ — произвольный алфавитный оператор в алфавите A, то отображение $\psi = \alpha^{-1}\varphi\alpha$, получаемое в результате последовательного выполнения отображений α^{-1}, φ и α, будет представлять собой, очевидно, некоторый алфавитный оператор в стандартном алфавите B. Назовем этот оператор *алфавитным оператором в алфавите B, сопряженным (при помощи кодирования α) с алфавитным оператором* φ.

Оператор φ однозначно восстанавливается по сопряженному ему оператору ψ и соответствующему кодирующему отображению α

$$\varphi = \alpha\psi\alpha^{-1}. \tag{2}$$

С помощью этой формулы, а также выписанной ранее дуальной ей формулы

$$\psi = \alpha^{-1}\varphi\alpha \tag{3}$$

произвольные алфавитные операторы сводятся к алфавитным операторам в стандартном алфавите. Такое сведение, разумеется, можно произвести бесконечным числом различных способов, поскольку существует бесконечно много различных кодирований слов в любом данном алфавите словами в стандартном алфавите.

Описанное сведение можно осуществлять и в случае алфавитных операторов, у которых входной и выходной алфавиты различны. Например, пусть φ — произвольный алфавитный оператор с входным алфавитом A и выходным алфавитом C, B — стандартный алфавит, α — какое-либо (обратимое) кодирование слов в алфавите A словами в стандартном алфавите, а γ — аналогичное кодирование слов в алфавите C.

Легко видеть теперь, что отображение $\psi = \alpha^{-1}\varphi\gamma$ представляет собой алфавитный оператор в стандартном алфавите B, по которому при условии знания кодирующих отображений α и γ однозначно восстанавливается исходное отображение φ.

Понятие алфавитного оператора является чрезвычайно общим. К нему фактически сводятся или могут быть в некотором смысле сведены любые процессы *преобразования информации*. Под информацией здесь и далее будем подразумевать не только осмысленные сообщения, но и вообще всякие сведения о процессах и состояниях любой природы, которые могут восприниматься органами чувств человека или приборами.

Для некоторых специальных видов информации, например информации лексической или числовой, алфавитный способ задания является самым естественным и постоянно применяющимся. Преобразования этих видов информации сводятся к алфавитным операторам самым непосредственным образом: как входная, так и выходная информации в любом преобразователе информации в этом случае могут быть представлены в виде слов, а преобразование информации сводится к установлению некоторого соответствия между словами. Напомним, что при разумном расширении алфавита словами в лексической информации могут считаться не только обычные слова, но и целые предложения и даже любые последовательности предложений.

Одной из характерных задач преобразования лексической информации является перевод текстов с одного языка на другой. Хорошо известно, что задача перевода не сводится к задаче установления соответствия между словами языков, которыми оперируют при переводе. Если же считать словами целые книги или хотя бы отдельные абзацы книги, то задача перевода полностью сводится к задаче установления соответствия между такими обобщенными словами. Таким образом, процесс перевода с одного языка на другой может трактоваться как процесс реализации некоторого алфавитного оператора.

Следует заметить, впрочем, что вполне качественный и грамотный перевод допускает, как известно, возможность известных модификаций переведенного текста. Поэтому процесс перевода опи-

сывается не обычным *однозначным* алфавитным оператором, а многозначным, или так называемым *вероятностным*, алфавитным оператором. Такой оператор сопоставляет каждому входному слову из области своего определения не одно выходное слово, а целую совокупность выходных слов. При конкретном применении этого оператора к тому или иному входному слову p происходит *случайный выбор* выходного слова из соответствующей слову p совокупности выходных слов.

Кроме алфавитных операторов для перевода с одних языков на другие, можно построить алфавитные операторы, решающие и другие задачи преобразования лексической информации, например задачу редактирования текстов на том или ином языке, задачу составления рефератов статей и т. п. Нетрудно расширить область применения алфавитных операторов, используя алфавитное представление не только для лексической, но и для других видов информации. Например, используя известные приемы шахматной нотации, можно записывать шахматные позиции в виде слов, состоящих из букв русского и латинского алфавитов, цифр и знаков препинания (запятой). В таком случае процесс шахматной игры может быть истолкован как процесс установления соответствия между любой данной позицией и позицией, возникающей из нее после выполнения очередного хода. Таким образом, и в этом случае речь идет о некотором алфавитном операторе (вообще говоря, вероятностном).

Аналогичным образом нетрудно представить в виде процессов реализации алфавитных операторов и многие другие процессы преобразования информации, например оркестровку мелодии, решение математических задач, задачи планирования производства и т. п.

Может показаться сначала, что для характеристики преобразований непрерывной информации (например, зрительных или произвольных слуховых ощущений) понятие алфавитного оператора окажется недостаточным. Однако это не так или, точнее, не совсем так.

Восприятие и преобразование непрерывной информации всегда производятся при помощи неидеальных приборов, не реагирующих на слишком малые изменения характеристик преобразуемой информации. В реальных приборах, воспринимающих и преобразующих непрерывную информацию, всегда существует ряд ограничений, которые позволяют рассматривать эту информацию как алфавитную информацию. Для большей ясности рассмотрим зрительную информацию (то же самое происходит и с другими формами задания непрерывной информации).

Первое ограничение — это ограничение *разрешающей способности* прибора, воспринимающего информацию. Указанное ограничение приводит к тому, что достаточно близкие между собой точки участка пространства, на котором распределена рассматриваемая информация (например, картина или рисунок), воспринимаются прибором (скажем, человеческим глазом) как одна точка.

Отсюда вытекает возможность рассматривать эту информацию как информацию, заданную не в бесконечном, а лишь в конечном множестве точек.

Второе ограничение связано с ограниченной *чувствительностью* прибора, воспринимающего информацию. Это ограничение приводит к тому, что прибор может различать фактически лишь конечное число уровней величины, несущей информацию (например, яркости отдельных точек рисунка).

На основании описанных ограничений приходим к выводу, что прибор, вследствие своей неидеальности, может в каждый данный момент воспринимать лишь одну картину из *конечного* (а не бесконечного, как может показаться без учета указанных ограничений) числа различных картин мгновенного распределения рассматриваемой информации в пространстве.

Вводя для каждой из таких картин специальное буквенное обозначение, придем к *конечному алфавиту* A, который с учетом указанных ограничений вполне достаточен для характеристики информации, поступающей на вход рассматриваемого нами (неидеального) прибора в каждый данный момент времени. Если обозначить буквой n число точек пространства, воспринимаемых прибором как отдельные точки, а буквой m—число различаемых прибором уровней физической величины, несущей информацию, то число букв в алфавите A будет равно, как нетрудно видеть, m^n (для простоты предполагаем число различаемых прибором уровней одинаковым для всех точек пространства).

Разумеется, только что подсчитанное число букв в алфавите A может оказаться чрезвычайно большим (в случае восприятия зрительной информации человеческим глазом оно может быть предположительно оценено как единица с несколькими тысячами нулей). Тем не менее оно все же является конечным, а с абстрактно-теоретической точки зрения существенно только, конечен или бесконечен алфавит A.

Продолжая наше исследование, заметим, что всякий реальный воспринимающий и преобразующий информацию прибор имеет, наряду с указанными двумя ограничениями, еще и третье ограничение. Речь идет об ограниченности *полосы пропускания* прибора, которая не позволяет ему различать слишком быстрые изменения воспринимаемых величин. В силу известного принципа Котельникова [46] ограничение полосы пропускания эквивалентно тому, что при передаче информации вместо обычного непрерывного времени вводится *условное дискретное время*, соседние моменты которого отличаются друг от друга на вполне определенный (хотя обычно и весьма малый) отрезок времени. Грубо говоря, в качестве такого элементарного отрезка времени выбирается максимальный отрезок, в течение которого рассматриваемый прибор оказывается неспособным различить изменения величины, несущей информацию.

После введения такого дискретного времени информация, воспринимаемая нашим прибором за любой конечный отрезок вре-

мени t, естественно представляется в виде слова во введенном выше алфавите A. Число букв в этом слове равняется числу моментов $\tau_1, \ldots, \tau_k$ дискретного времени, укладывающихся в данном отрезке времени t, а его i-я буква ($i = 1, 2, \ldots, k$) представляет собой выраженную в виде буквы алфавита A информацию, воспринимаемую прибором в момент времени τ_i.

Поскольку аналогичные рассмотрения применимы не только ко входной, но и к выходной информации, *любой реальный преобразователь информации может рассматриваться (с учетом указанных выше ограничений) как прибор, реализующий некоторый алфавитный оператор*. Алфавитный оператор, реализуемый прибором, полностью (с точностью до кодирования информации) определяет *информационную сущность* этого прибора, иными словами, — выполняемое этим прибором преобразование информации.

Итак, нами установлена чрезвычайно большая общность понятия алфавитного оператора. Ведь к изучению алфавитных операторов оказалась фактически сведенной теория любых преобразователей информации. А с преобразователями информации человек встречается в своей практике буквально на каждом шагу. Преобразователями информации являются различные приборы и средства автоматического управления. Наконец, одним из наиболее важных и существенных аспектов изучения деятельности самого человека является аспект, связанный с рассмотрением человека как весьма сложного и совершенного преобразователя информации. Все это позволяет считать теорию алфавитных операторов одной из наиболее важных составных частей кибернетики.

Основой теории алфавитных операторов являются *способы их задания*. В случае, если область определения алфавитного оператора конечна, то вопрос о его задании, по крайней мере в теоретическом плане, решается чрезвычайно просто: оператор может быть задан простой *таблицей соответствия*. В левой части такой таблицы выписываются все слова, входящие в область определения рассматриваемого оператора, а в правой части — выходные слова, получающиеся в результате применения оператора к каждому слову из левой части таблицы.

Разумеется, если область определения алфавитного оператора достаточно велика, указанный способ задания может оказаться чрезвычайно громоздким и поэтому неприменимым на практике. Однако не будем принимать пока во внимание соображения подобного рода, ограничиваясь каждый раз установлением лишь принципиальной возможности задания тех или иных алфавитных операторов.

В случае бесконечности области определения алфавитного оператора задание его с помощью простой таблицы соответствия оказывается уже *принципиально невозможным*, поскольку в распоряжении человека нет средств, позволяющих ему фактически выписать или воспринять бесконечное множество слов. Хорошо известно, однако, что человек давно научился задавать операторы

на бесконечных множествах слов, не выписывая *всей* таблицы соответствия. С этой целью достаточно рассмотреть, например, алфавитный оператор, задаваемый формулой

$$\underbrace{xx\ldots x}_{n\text{ раз}}\to\underbrace{yy\ldots y}_{n+1\text{ раз}}\quad(n=1,2,\ldots).\qquad(4)$$

Приведенная формула определяет соответствие на бесконечном множестве слов, достигнутое без фактического выписывания всей таблицы соответствия (что, разумеется, сделать в этом случае невозможно). Вместо самой таблицы соответствия эта формула дает *правило*, с помощью которого за конечное число шагов может быть установлено выходное слово, соответствующее любому наперед заданному входному слову из области определения рассматриваемого алфавитного оператора.

Аналогичная ситуация возникает всякий раз, когда приходится задавать алфавитный оператор с бесконечной областью определения; вместо самой таблицы соответствия задается конечное число правил, позволяющих за конечное число шагов найти наперед заданную строку этой таблицы (значение алфавитного оператора на любом входном слове, входящем в область его определения).

Алфавитные операторы, задаваемые с помощью конечных систем правил, принято называть алгоритмами.

На основании сказанного выше легко понять, что всякий алфавитный оператор, который можно фактически задать, является непременно алгоритмом. Алгоритмами, в частности, будут все алфавитные операторы с конечными областями определения, задаваемые (конечными) таблицами соответствия. Формула (4) также задает некоторый алгоритм.

Нетрудно построить и другие примеры алгоритмов. Сопоставляя каждому целому положительному числу его квадрат, получим алфавитный оператор в алфавите, состоящем из всех цифр системы счисления, принимаемой для представления этих чисел. Поскольку правила возведения в квадрат позволяют за конечное число шагов получить квадрат любого наперед заданного целого числа, указанный оператор можно рассматривать как алгоритм.

Все конкретные алфавитные операторы, рассматриваемые в настоящей главе (в том числе операторы перевода с одного языка на другой, игры в шахматы и т. д.), также можно задать с помощью конечных систем правил и можно, следовательно, рассматривать как алгоритмы.

Необходимо подчеркнуть одно различие, существующее между понятиями алфавитного оператора и алгоритма. В понятии алфавитного оператора существенно лишь само соответствие, устанавливаемое оператором между входными и выходными словами, а не способ, которым это соответствие устанавливается. В понятии алгоритма, наоборот, основной упор делается на *способ задания*

соответствия, устанавливаемого алгоритмом. *Таким образом, алгоритм есть не что иное, как алфавитный оператор вместе с правилами, определяющими его действие.*

В соответствии со сказанным определяется понятие *равенства* для алфавитных операторов и алгоритмов. Два алфавитных оператора считаются равными, если они имеют одну и ту же область определения и сопоставляют любому наперед заданному входному слову из этой области одинаковые выходные слова. Понятие равенства для алгоритмов включает условия равенства для соответствующих им операторов, но предусматривает также совпадение систем правил, задающих действие этих алгоритмов на входные слова. Алгоритмы, у которых совпадают только определяемые ими алфавитные отображения (операторы), но, вообще говоря, не способы задания, будем называть *эквивалентными* алгоритмами.

Обычно в абстрактной теории алгоритмов рассматривают лишь такие алгоритмы, которым соответствуют однозначные алфавитные операторы. Всякий алгоритм A такого рода отличается тем, что любому входному слову p из области своего определения он относит вполне определенное выходное слово $q = A(p)$ независимо от условий, в которых работает алгоритм A. Подобные алгоритмы и соответствующие им алфавитные операторы будем называть *детерминированными*.

В ряде случаев целесообразно расширить понятие алгоритма, вводя в системы правил, описывающих алгоритмы, возможность случайного выбора тех или иных слов или тех или иных правил. При этом вероятности того или иного выбора должны либо фиксироваться заранее, либо определяться в процессе реализации алгоритма. Подобные алгоритмы, будем называть их *случайными*, приводят к многозначным алфавитным операторам. Более точно, для любого входного слова p, входящего в область определения случайного алгоритма A, этот алгоритм однозначно определяет вероятности $\alpha_p(q)$ появления различных выходных слов q в качестве ответа на входное слово p. Вероятности $\alpha_p(q)$ в случае обычного случайного алгоритма не должны изменяться в процессе его функционирования, хотя сам алгоритм при нескольких его применениях к одному и тому же входному слову p может, разумеется, давать различные ответы.

Нам придется рассматривать еще так называемые *самоизменяющиеся* алгоритмы, т. е. такие алгоритмы, которые не только перерабатывают подаваемые на них входные слова, но и сами изменяются в процессе такой переработки. Результат действия самоизменяющегося алгоритма A на то или иное входное слово p зависит не только от этого слова, но и от *истории предыдущей работы* алгоритма, т. е. от (конечной) последовательности входных слов, переработанных алгоритмом A до поступления на его вход рассматриваемого слова p.

Обобщение понятия алгоритма за счет введения возможности самоизменения применимо как к детерминированным, так и к

случайным автоматам. В последнем случае в зависимости от истории предыдущей работы алгоритма меняются вероятности $\alpha_p(q)$ различных выходных слов q, сопоставляемых алгоритмом A любому данному входному слову p. Эта зависимость может, впрочем, также выражаться не детерминированной, а случайной функцией.

Самоизменяющийся алгоритм удобно представлять себе в виде системы двух алгоритмов, из которых первый, так называемый *рабочий*, осуществляет переработку входных слов, а второй, называемый *контролирующим* или *управляющим*, вносит те или иные изменения в первый, рабочий, алгоритм. В гл. IV показано, что свойство самоизменения алгоритма определяется не столько структурой реализующего соответствующий алгоритм устройства, сколько способом дробления входной информации на отдельные слова, который, как отмечалось выше, в случае абстрактных алфавитов до известной степени произволен. Так что в зависимости от выбора этого способа одно и то же устройство может в одних случаях реализовать самоизменяющийся, а в других — несамоизменяющийся алгоритмы.

На протяжении первых трех глав речь идет лишь об обычных (детерминированных, несамоизменяющихся) алгоритмах, и это обстоятельство не оговаривается каждый раз особо. В последующих главах употребляются также введенные выше обобщения понятия алгоритма.

§ 2. Нормальные алгоритмы

В этом и в ряде последующих параграфов исследуются некоторые общие способы задания алгоритмов, характеризующиеся *свойством универсальности*, т. е. такие способы, которые позволяют задать алгоритм, эквивалентный любому наперед заданному алгоритму. В этой главе различные универсальные способы задания алгоритмов излагаются не в исторической последовательности, в которой они фактически возникали, а в таком порядке, который является более удобным с точки зрения задач настоящей книги. Начнем наше изложение с так называемых *нормальных алгоритмов*, предложенных и изученных А. А. Марковым [53].

Всякий общий способ задания алгоритмов назовем алгоритмической системой. *Алгоритмическая система* включает в себя обычно объекты двоякой природы, которые, следуя Л. А. Калужнину [37], будем называть *операторами* (или, более точно, *элементарными операторами*) и *распознавателями* (точнее, *элементарными распознавателями*). Элементарные операторы представляют собой достаточно простые (просто задаваемые) алфавитные операторы, с помощью последовательного выполнения которых реализуются любые алгоритмы в рассматриваемой алгоритмической системе. Распознаватели служат для распознавания наличия тех или иных свойств перерабатываемой алгоритмом информации и для изменения, в зависимости от результатов распознавания, последо-

вательности, в которой следуют друг за другом элементарные операторы.

Для указания набора элементарных операторов и порядка их следования друг за другом при задании того или иного конкретного алгоритма удобно использовать направленные графы особого рода, которые, следуя Л. А. Калужнину [37], будем называть *граф-схемами* соответствующих алгоритмов.

Граф-схема алгоритма представляет собой конечное множество соединенных между собой стрелками кружочков (или других геометрических фигур), называемых узлами граф-схемы. Каждому узлу, кроме двух особых узлов, называемых *входным и выходным*, сопоставляется какой-либо элементарный оператор или распознаватель. Из каждого узла, которому сопоставлен оператор, а также из входного узла выходит точно по одной стрелке; из каждого узла, которому сопоставлен распознаватель, — точно по две стрелки; из выходного узла не выходит ни одной стрелки. Число стрелок, входящих в узел, может быть любым.

Алгоритм, который определяется любой данной граф-схемой, работает следующим образом. Входное слово поступает сначала на входной узел и двигается по направлениям, указанным стрелками, преобразуясь при прохождении *операторных узлов* операторами, сопоставленными этим узлам. При попадании слова в *распознавательный узел* осуществляется проверка сопоставленного этому узлу условия (применение распознавателя условия). При выполнении условия слово выходит из узла по одной из стрелок (отмечаемой обычно знаком «+»), а при невыполнении условия — по другой стрелке (отмечаемой знаком «—»).

В распознавательных узлах слово не изменяется. Если входное слово p, поданное на входной узел граф-схемы, проходя через узлы схемы и преобразуясь, попадает через конечное число шагов в выходной узел, то считается, что алгоритм *применим* к слову p (слово p входит в *область определения* этого алгоритма), причем результатом воздействия алгоритма на слово p будет то слово, которое оказывается в выходном узле схемы. Если после подачи слова p на входной узел граф-схемы его преобразование и движение по граф-схеме продолжается бесконечно долго, не приводя в выходной узел, считается, что алгоритм *не применим* к слову p, иными словами, слово p не входит в область определения алгоритма.

В нормальных алгоритмах используется только один тип элементарных операторов, называемых *операторами подстановки*, и один тип элементарных распознавателей, называемых *распознавателями вхождения*. Опишем эти распознаватели и операторы более подробно. Для этого познакомимся прежде всего с понятием *вхождения* одного слова в другое.

Пусть p и q — два произвольных слова в том или ином алфавите. Говорят, что слово q входит в слово p, если слово p может быть представлено в виде $p = p_1qp_2$, где p_1 и p_2 — некоторые слова,

быть может, и пустые. Найденное вхождение слова q в слово p называется первым слева (или просто первым) вхождением, если в рассмотренном представлении слова p в виде $p = p_1 q p_2$ *слово* p_1 *имеет наименьшую возможную длину среди всех подобных представлений слова p.*

Распознаватель вхождения задается указанием некоторого фиксированного слова q, а смысл его применения состоит в том, что для любого заданного слова p проверяется условие — входит или не входит слово q в слово p. Оператор подстановки задается обычно в виде двух слов, соединенных стрелкой, — $q_1 \to q_2$. Работа оператора состоит в том, что он осуществляет подстановку слова q_2 вместо первого слева вхождения слова q_1 в любое задаваемое слово p. При этом, если выделить явным образом первое вхождение слова q_1 в слово p, записывая слово p в виде $p_1 q_1 p_2$, после применения рассматриваемого оператора оно преобразуется в слово $p_1 q_2 p_2$.

При применении распознавателей вхождения условимся выделять найденное (первое слева) вхождение распознаваемого слова в заданное слово с помощью скобок. Например, применяя к слову $p = xxyxyxx$ распознаватель вхождения слова $q = xy$, выделяем первое вхождение слова q в слово p следующим образом: $p = x(xy)xyxx$.

Алгоритмы, которые задаются граф-схемами, составленными исключительно из распознавателей вхождения слов и операторов подстановки, назовем обобщенными нормальными алгоритмами. При этом предполагается, что к каждому оператору подстановки вида $q_1 \to q_2$ *подсоединяется только одна стрелка: стрелка со знаком «+», выходящая из распознавателя* q_1.

Пример граф-схемы обобщенного нормального алгоритма изображен на рис. 1. Распознаватели на этом рисунке изображены в виде кружков, а операторы — в виде прямоугольников. Оператор $xy \to$ означает подстановку вместо первого вхождения слова xy пустого слова. В соответствии с принятым в предыдущем параграфе обозначением пустого слова этот оператор может быть записан также в виде $xy \to e$.

Рассматривая работу алгоритма A, заданного граф-схемой рис. 1, заметим, что первый сверху оператор осуществляет перестановку x в левую, а y в правую часть слова до тех пор, пока слово не приобретет вид $xx \ldots xyy \ldots y$ (все x предшествуют всем y). Лишь после приведения слова к указанному виду вступает в действие второй оператор, который уничтожает пары xy до тех пор, пока в слове не останутся одни x либо одни y. Если в первоначально заданном входом слове p было m x-ов и n y-ов, то в результате действия алгоритма A оно преобразуется в слово $q = A(p)$, имеющее длину $|m-n|$ и состоящее из одних x-ов (если $m > n$) или из одних y-ов (если $n > m$).

Рассмотрев обобщенные нормальные алгоритмы, перейдем к характеристике собственно нормальных алгоритмов. *Нормальными алгоритмами называются такие обобщенные нормальные алгоритмы,*

граф-схемы которых имеют некоторый специальный вид. Для того чтобы описать этот вид, заметим, что, в силу приведенного выше определения обобщенных нормальных алгоритмов, в граф-схемы таких алгоритмов всякий оператор $q_1 \to q_2$ входит в паре с распознавателем q_1.

Объединим на граф-схеме каждую такую пару узлов в один узел, сохраняя за ним обозначение соответствующего оператора.

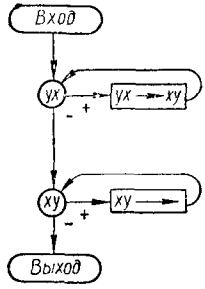

Рис. 1.

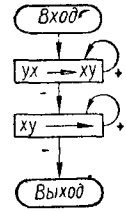

Рис. 2.

Из каждого объединенного узла будут выходить две стрелки: стрелка со знаком «+», по которой направляется слово, подвергнутое действию оператора данного узла, и стрелка со знаком «—», по которой направляется слово в случае, если оператор узла к нему *не применим*. Неприменимость оператора подстановки к слову означает отсутствие вхождений *левой части* оператора (слова q_1 в операторе $q_1 \to q_2$) в данное слово.

Используя описанный прием объединения узлов, граф-схему алгоритма, изображенную на рис. 1, можно представить в виде схемы, показанной на рис. 2. Подобная граф-схема с *объединенными узлами* в случае нормальных алгоритмов должна удовлетворять следующим условиям:

а) *все объединенные (операторно-распознавательные) узлы граф-схемы упорядочиваются с помощью присвоения им последовательных номеров от 1 до n, причем отрицательный выход (стрелка со знаком «—») i-го узла подсоединяется к (i + 1)-му узлу (i = 1, 2, . . ., n — 1), а отрицательный выход n-го узла подсоединяется к выходному узлу граф-схемы*;

б) *положительные выходы (стрелки со знаком «+») всех объединенных узлов подсоединяются либо к первому, либо к выходному узлу граф-схемы. В первом случае подстановка оператора соответствующего узла называется обычной, а во втором — заключительной*;

в) *входной узел подсоединяется стрелкой к первому объединенному (распознавательно-операторному) узлу.*

Эти условия являются необходимыми и достаточными для того, чтобы удовлетворяющая им граф-схема задавала не обобщенный нормальный, а обычный нормальный алгоритм. Легко проверить,

что граф-схема, изображенная на рис. 2, не является граф-схемой нормального алгоритма, поскольку для нее не выполняется второе из только что сформулированных условий (условие «б»).

Нормальные алгоритмы принято задавать не граф-схемами, а просто упорядоченным множеством подстановок всех операторов данного алгоритма, называемых *схемой* данного алгоритма. При этом обычные подстановки записываются, как показано выше, в виде двух слов, соединенных стрелкой ($q_1 \to q_2$), а заключительные подстановки обозначаются стрелкой с точкой ($q_1 \to \cdot\, q_2$).

Порядок выполнения подстановок полностью определяется после этого условиями «*а*», «*б*», и «*в*». Действительно, в силу этих условий произвольная (i)-ая подстановка схемы алгоритма должна выполняться в том и только в том случае, если она является первой из применимых подстановок (все подстановки от 1-ой до (i—1)-ой неприменимы). Процесс выполнения подстановок заканчивается лишь тогда, когда ни одна из подстановок схемы не применима к полученному слову или когда выполнена (первый раз) какая-либо заключительная подстановка.

В качестве примера рассмотрим работу нормального алгоритма A, заданного схемой

$$yyx \to y;$$

$$xx \to y;$$

$$yyy \to \cdot\, x.$$

Пусть нам задано входное слово $p = xyxxxyy$. Первая подстановка алгоритма A к этому слову не применима, для применения второй подстановки выделяем первое вхождение ее левой части (xx) в слово p: $p = xy(xx)xyy$. После выполнения второй подстановки алгоритма получаем слово $p_1 = xyyxyy$, к которому оказывается применимой первая подстановка алгоритма: $p_1 = x(yyx)yy \to$ $\to xyyy = p_2$. К полученному слову применима лишь третья подстановка: $p_2 = x(yyy) \to xx = p_3$, а поскольку она отмечена как заключительная, слово p_3 представляет собой окончательный результат воздействия алгоритма A на исходное слово p, т. е. $p_3 = A(p)$.

Если бы третья подстановка алгоритма A не была заключительной, то процесс подстановок мог бы быть продолжен и вместо слова $p_3 = xx$ в качестве результата действия алгоритма на исходное слово p было бы получено слово $p_4 = y$.

Применение в схемах нормальных алгоритмов наряду с обычными заключительных подстановок является обязательным для возможности реализации в таких схемах произвольных *конструктивных* алфавитных операторов, т. е. таких алфавитных операторов, которые определяются с помощью конечного числа правил. Действительно, любой нормальный алгоритм A, в схеме которого нет ни одной заключительной подстановки, может окончить свою работу только тогда, когда ни одна из его подстановок далее неприменима. Отсюда непосредственно следует, что повторное воз-

действие алгоритма A на слово $A(p)$, полученное в результате его применения к любому входному слову p, уже не может изменить этого слова. Иными словами, для алгоритма A выполняется следующее тождественное (справедливое для любого входного слова p) соотношение (см. А. А. Марков [53]):

$$A(A(p)) = A(p). \tag{5}$$

Этому соотношению удовлетворяет далеко не всякий конструктивный алфавитный оператор. Примером алфавитного оператора, для которого соотношение (5) не имеет места, является оператор B, действие которого на любое слово p заключается в дописывании к этому слову слева какой-либо фиксированной буквы x: $B(p) = xp$. Из сказанного выше ясно, что этот оператор не может быть реализован с помощью нормального алгоритма, в схеме которого нет заключительных подстановок.

В то же время легко проверить, что этот оператор реализуется нормальной схемой, состоящей из одной заключительной подстановки $\to \cdot x$ (или, что то же самое, $e \to \cdot x$). Действительно, в силу принятого выше определения вхождения пустое слово входит во всякое слово p, причем первое его вхождение не будет иметь слева от себя ни одной буквы. Отсюда же непосредственно следует, что применение указанной подстановки к произвольному слову p переведет его в слово xp.

Не менее очевидно, что при построении теории нормальных алгоритмов нельзя ограничиться и одними заключительными подстановками. Действительно, нормальный алгоритм, схема которого состоит лишь из заключительных подстановок, действует на каждое входное слово p не более чем одной из этих подстановок, после чего сразу получается требуемое выходное слово $A(p)$. Ввиду конечности схемы алгоритма модули разностей длин слов p и $A(p)$ ограничены в совокупности (при любом наборе входного слова p) одним и тем же числом N (максимумом модулей разностей длин слов в левых и правых частях подстановок алгоритма A).

Существуют, однако, простые конструктивные операторы, у которых модули разностей длин входных и соответствующих им выходных слов не ограничены в совокупности. Примером таких операторов может служить оператор D удвоения входных слов, действие которого на любое входное слово p определяется равенством $D(p) = pp$. Из сказанного выше ясно, что представление этого оператора в виде нормального алгоритма, в схеме которого содержатся лишь заключительные подстановки, заведомо невозможно.

Итак, если предъявлять к алгоритмической системе, основанной на использовании нормальных алгоритмов, требование универсальности (возможности построения нормального алгоритма, эквивалентного любому наперед заданному алгоритму), то необходимым условием для такой универсальности является использова-

ние обоих видов подстановок — как заключительных, так и обычных. Это условие является также достаточным, т. е. можно сформулировать принцип нормализации (см. [53]).

Принцип нормализации. *Для любого алгоритма (конструктивно задаваемого алфавитного отображения) в произвольном конечном алфавите A можно построить эквивалентный ему нормальный алгоритм над алфавитом A.*

Употребляемое в формулировке принципа нормализации понятие нормального алгоритма *над алфавитом* означает следующее. В ряде случаев не удается построить нормальный алгоритм, эквивалентный данному алгоритму (в алфавите A), если использовать в подстановках алгоритма только буквы алфавита A. Однако можно построить требуемый нормальный алгоритм, добавляя к алфавиту A некоторое количество новых букв или, как обычно говорят, производя *расширение* алфавита A. В этом случае принято говорить, что построенный (нормальный) алгоритм является алгоритмом *над алфавитом* A. Условимся, однако, что, несмотря на расширение алфавита, алгоритм по-прежнему будет применяться лишь к словам в исходном алфавите A.

Как показали А. А. Марков [53] и Н. М. Нагорный [58], если можно построить нормальный алгоритм, эквивалентный данному алгоритму в алфавите A, присоединяя к алфавиту A какое-нибудь (быть может, очень большое) конечное число букв, то можно построить эквивалентный ему нормальный алгоритм, присоединяя к алфавиту A лишь одну дополнительную букву.

Невозможно дать строгое математическое доказательство принципа нормализации, поскольку понятие *произвольного* алгоритма не является строго определенным математическим понятием. Поэтому к его обоснованию следует подходить так, как подходят к обоснованию всякого естественнонаучного закона или принципа. Обоснование, которое можно дать в этом плане принципу нормализации, дает возможность считать этот принцип в высшей степени правдоподобным. Укажем основные пути такого обоснования. Для упрощения формулировок условимся вслед за А. А. Марковым [53] называть тот или иной алгоритм *нормализуемым*, если можно построить эквивалентный ему нормальный алгоритм (используя при этом, быть может, расширение алфавита), и ненормализуемым — в противном случае. Принцип нормализации может быть теперь высказан в несколько видоизмененной форме.

Все алгоритмы нормализуемы.

Справедливость этого принципа основана прежде всего на том, что все известные в настоящее время алгоритмы являются нормализуемыми. Поскольку в течение долгой истории развития точных наук было придумано немало различных алгоритмов, приведенное соображение является убедительным само по себе.

В действительности оказывается нечто большее. Можно показать, что все известные в настоящее время способы *композиции алгоритмов*, позволяющие строить новые алгоритмы из уже извест-

ных, не выводят за пределы класса нормализуемых алгоритмов. Иными словами, если исходные алгоритмы были нормализуемы, то нормализуемой будет и любая композиция этих алгоритмов (из числа известных в настоящее время видов композиции). Из сказанного вытекает, что для построения примера ненормализуемого алгоритма необходимо применять приемы, качественно отличные от всего того, с чем математики сталкивались до сих пор.

Однако и этого мало. Целым рядом ученых предприняты специальные попытки построения алгоритмов наиболее общего вида, и все эти попытки не вывели за пределы класса нормализуемых алгоритмов. С одной из таких попыток (алгоритмическая схема Колмогорова — Успенского) познакомимся ниже. Неудача этих попыток как раз и является наиболее ярким свидетельством в пользу справедливости принципа нормализации.

Итак, принцип нормализации следует считать достаточно обоснованным, хотя это обоснование и не исключает полностью возможности его опровержения в будущем (построения примера ненормализуемого алгоритма). Во всяком случае, нормализуемые алгоритмы охватывают значительную часть алгоритмов (если не все), поэтому систему нормальных алгоритмов можно считать практически универсальной алгоритмической системой.

Рассмотрим теперь некоторые из употребительных видов композиции алгоритмов, которые упоминались выше. Будем определять не композицию самих алгоритмов, а композицию соответствующих им алфавитных отображений, однако, как отмечалось выше, возможность нормализации результата композиции нормальных алгоритмов создает возможность (по крайней мере, в классе нормальных алгоритмов) доопределить композицию отображений до композиции самих алгоритмов.

Одним из наиболее распространенных видов композиции алгоритмов (отображений) является *суперпозиция алгоритмов*. При суперпозиции двух алгоритмов A и B выходное слово первого алгоритма (A) рассматривается как входное слово второго алгоритма (B), так что результат суперпозиции алгоритма A и B можно представить в виде $D(p) = B(A(p))$. Это определение распространяется на суперпозиции какого угодно конечного числа алгоритмов.

Суперпозицию обобщенных нормальных алгоритмов можно рассматривать как обобщенный нормальный алгоритм. Для этого достаточно выходной узел граф-схемы каждого предыдущего алгоритма совместить с входным узлом последующего алгоритма. Нормализация суперпозиции нормальных алгоритмов требует уже известных ухищрений, однако тоже может быть всегда осуществлена [53].

Укажем еще некоторые виды композиции алгоритмов.

Объединением алгоритмов A и B в одном и том же алфавите $\mathfrak{A}$ называется алгоритм C в том же алфавите, преобразующий любое входное слово p, содержащееся в пересечении областей определения алгоритмов A и B, в записанные рядом слова $A(p)$ и $B(p)$; на

всех остальных входных словах этот алгоритм считается не определенным.

Разветвление алгоритмов представляет собой композицию трех алгоритмов A, B и C. Обозначая результат этой композиции буквой D, будем считать, что область определения алгоритма D совпадает с пересечением областей определения всех трех алгоритмов A, B и C, а для любого слова p из этого пересечения $D(p) = A(p)$, если $C(p) = e$, и $D(p) = B(p)$, если $C(p) \neq e$.

Повторение (итерация) представляет собой композицию двух алгоритмов A и B. Обозначая результат этой композиции через P, определим, что для любого входного слова q соответствующее ему выходное слово $P(q)$ определяется следующим условием: существует такой ряд слов $q=q_0, q_1, q_2, \ldots, q_n=P(q)$, что для всех $i=1, 2,\ldots, n$ $q_i=A(q_{i-1})$, для всех $i=1, 2,\ldots, n-1$ $B(q_i) \neq e$, а $B(q_n) = e$. Иными словами, алгоритм A применяется последовательно несколько раз до тех пор, пока не получится слово, перерабатываемое алгоритмом B в пустое слово e (можно, разумеется, вместо пустого слова выбрать любое другое фиксированное слово).

Все описанные способы композиции нормальных алгоритмов приводят к нормализуемым алгоритмам [53].

Очень важное значение для нормальных алгоритмов, как и для всякой универсальной алгоритмической системы, имеет задача построения так называемого *универсального алгоритма*. Рассмотрим универсальный алгоритм применительно к нормальным алгоритмам.

Необходимо построить нормальный алгоритм, который бы выполнял работу любого нормального алгоритма, если задана *схема* (набор подстановок) этого последнего алгоритма.

Точная постановка задачи об универсальном алгоритме может быть осуществлена различными способами. Опишем один из самых естественных способов такой постановки. Для этого фиксируем прежде всего некоторый стандартный алфавит $\mathfrak{A}$ (например, двоичный). Для всех других возможных алфавитов фиксируем некоторый определенный способ кодирования букв этих алфавитов в выбранном стандартном алфавите. В случае двоичного стандартного алфавита это может быть сделано, например, следующим образом: буквы любого данного алфавита нумеруют последовательными натуральными числами, после чего i-ой букве присваивают двоичный код, начинающийся и кончающийся нулем и имеющий между этими нулями ровно i единиц. Если общее число букв в данном алфавите равно n, то вводят еще дополнительные ($(n+1)$-ую, $(n+2)$-ую и т. д.) буквы для обозначения знаков, употребляющихся в схемах нормальных алгоритмов (стрелок, точек, знака раздела между формулами), а также специального *концевого* знака, ставящегося в начале и в конце схемы алгоритма.

Записывая схему алгоритма одним словом и кодируя буквы этого слова принятым выше способом, получим слово в стандарт-

ном алфавите, которое называется *изображением* данного алгоритма. Например, для нормального алгоритма, задаваемого схемой

$$xy \to x$$

$$y \to .,$$

изображение A^u алгоритма A в двоичном алфавите может быть получено следующим путем: фиксируем нумерацию букв, считая x— первой, y — второй, стрелку — третьей, точку — четвертой, знак раздела — пятой, а концевой знак — шестой буквой. Тогда изображение A^u алгоритма A запишется: 060 010 020 030 010 050 020 030 040 060. Для краткости здесь вместо выписывания подряд любого положительного числа n единиц выписано само это число n.

Наряду с изображением алгоритма A, может быть получено также с помощью описанного выше кодирования в стандартном алфавите $\mathfrak{A}$ изображение p^u любого входного слова p этого алгоритма.

Справедлива следующая теорема *об универсальном нормальном алгоритме* (см. А. А. Марков [53]).

Существует такой нормальный алгоритм U, называемый универсальным нормальным алгоритмом, который для любого нормального алгоритма A и любого входного слова p из области определения этого последнего алгоритма переводит слово $A^u p^u$, полученное приписыванием изображения слова p к изображению алгоритма A, в слово, являющееся изображением соответствующего выходного слова $A(p)$, в которое алгоритм A перерабатывает слово p. Если же слово p выбирается так, что алгоритм A к нему не применим, то универсальный алгоритм U оказывается не применимым к слову $A^u p^u$.

Эта теорема имеет огромное значение, поскольку из нее вытекает возможность построения машины, которая может выполнять работу любого нормального алгоритма, а значит, в силу принципа нормализации,— работу произвольного алгоритма. Для этой цели в машину достаточно вложить программу, т. е. изображение того нормального (нормализованного) алгоритма, работу которого машина должна выполнять.

Однако, хотя в принципе доказана возможность нормализации для всех известных в настоящее время алгоритмов, фактическое выполнение нормализации даже для относительно простых алгоритмов (например, для алгоритма умножения двух целых чисел) является весьма нелегким делом. Это означает, что программирование для машины, моделирующей универсальный нормальный алгоритм, было бы чрезвычайно громоздко и непрактично. Поэтому на практике машины, дающие возможность реализовать работу любого алгоритма, строятся на основании использования других алгоритмических систем, отличных от системы нормальных алгоритмов. Эти системы описаны в гл. V.

§ 3. Алгоритмическая схема Колмогорова — Успенского

В настоящем параграфе излагается предложенный А. Н. Колмогоровым и В. А. Успенским [43] способ определения алгоритмов наиболее общего вида. Для построения соответствующей алгоритмической схемы избирают такой путь, чтобы опираться лишь на такие свойства, которые, безусловно, присущи любой алгоритмической схеме, и воплощать эти свойства в те или иные конкретные формы, не допуская при этом никакой потери общности.

При построении подобной общей алгоритмической схемы полезно представлять себе в качестве наглядной модели человека, производящего вычисления или другую обработку информации в соответствии с той или иной точно предписанной системой правил. Человек выполняет роль *преобразователя* информации, сама же преобразуемая информация находится вне человека. Будем предполагать для определенности, что эта информация записывается на листках бумаги, причем в распоряжении человека — неограниченный запас чистых листов и неограниченный резерв места для хранения исписанных листов. Преобразование информации, реализуемое человеком, разбивается на отдельные *дискретные шаги*. На каждом таком шаге человек обозревает некоторое число исписанных листов, и, в зависимости от содержания этих записей, по находящейся в его памяти *строго определенной* и не *меняющейся с течением времени* системе правил он производит те или иные изменения в обозреваемой информации. Эти изменения могут быть трех видов: *стирание* (уничтожение) всей обозреваемой информации или некоторой ее части, *запись* на обозреваемых листах новой информации, *изменение* совокупности обозреваемых листов.

На первый взгляд кажется, что требование неизменности во времени системы правил, по которым производится переработка информации, значительно сужает круг рассматриваемых задач по сравнению с задачами, которые в действительности могут решаться человеком, поскольку человек способен изменять правила в процессе работы. В действительности указанное ограничение не существенно, поскольку характер изменения информации на каждом шаге преобразования зависит не только от правил преобразования, но и от самой этой информации. В связи с этим оказывается возможным в случае необходимости варьировать характер преобразования информации с течением времени, вносить соответствующие изменения в саму информацию, а не в правила, хранящиеся в памяти преобразователя, иными словами, записывать в правилах на листках бумаги необходимые изменения, а не запоминать их.

Совершенно необходимым ограничением при построении любой алгоритмической системы является способность преобразователя информации воспринимать в каждый данный момент лишь ограниченное количество информации. Если общий объем перерабатываемой информации превосходит объем этой *активной зоны*

преобразователя, то информация должна вовлекаться в преобразование постепенно, шаг за шагом.

После предварительных замечаний перейдем непосредственно к описанию схемы Колмогорова — Успенского. Информация в этой схеме, как и вообще в случае алфавитных преобразований, записывается с помощью конечного числа символов — букв, которые будем обозначать T_0, T_1, ..., T_n. Для достижения возможно большей общности установим еще некоторые связи между символами, принадлежащие к одному из типов R_1, R_2, ..., R_m. Для каждого типа связи R_i фиксируем число k_i связываемых символов (букв). Максимальное среди чисел k_1, ..., k_m обозначим K. Связи между символами вводятся с той целью, чтобы учесть случай сложных букв, обозначающих, например, целые фразы в обычном языке. В таком случае в состав фразы (буквы) могут входить указания относительно расположения информации (других букв), имеющей непосредственное отношение к рассматриваемой букве (скажем, такой, которая должна быть вовлечена в рассмотрение на следующем шаге алгоритма). Ограничение числа связываемых символов зависит от ограниченности информации, содержащейся в каждой букве (в противном случае буква не могла бы помещаться целиком в активной зоне и ее пришлось бы расчленить на отдельные части).

Предположим, что все связи, в которые может входить любая данная буква, упорядочены тем или иным способом и занумерованы, а общее число таких связей ограничено одним и тем же числом s. Условимся обозначать буквы кружками, вводя в случае необходимости нумерацию этих кружков числами, которые будем писать рядом с соответствующими кружками. Эти номера не имеют никакого отношения к типу символа (буквы), обозначаемого данным кружком. В случае необходимости символ соответствующей буквы пишется внутри обозначающего его кружка.

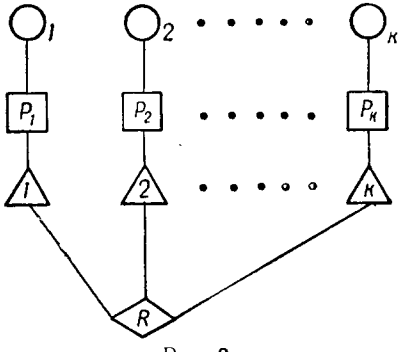

Рис. 3.

Произвольная связь между символами (буквами) может быть изображена теперь так, как показано на рис. 3.

Индексы p_1, p_2, ..., p_k на этом рисунке показывают, какие места занимает рассматриваемая связь в упорядоченном наборе связей для соответствующих (обозначенных занумерованными кружочками) букв. Эти индексы (независимо от выбора букв и вида связи R) могут принимать лишь значения 1 ,2, 3, ..., s.

Можно существенно упростить запись информации в рассматриваемой схеме, добавив к числу букв T_0, T_1, ..., T_n еще $s + K + m$ букв: s букв для обозначения номеров связи в любом

данном элементе (квадраты на рис. 3), K букв для обозначения номеров связей с буквами для любой данной связи R (треугольники на рис. 3) и m букв для обозначения самих связей R_1, R_2, ..., R_m. Если обозначать все новые буквы кружочками, то информация примет вид набора кружочков, соединенных между собой парными связями. При этом не требуется какой-либо специальной нумерации для порядка вхождения буквы в те или иные связи, поскольку, как показано на рис. 3, все буквы, связанные с какой-либо одной буквой, будут непременно различными. Тем самым связи, в которые входит данный символ (буква), нумеруются автоматически — номерами символов (букв), с которыми связывается данный символ.

Таким образом, окончательно информация в записываемой алгоритмической схеме изображается произвольным конечным множеством M, элементами которого являются фиксированные буквы T_0, T_1, ..., T_N ($N \geqslant 1$), причем в множество M каждая из букв T_2, T_3, ..., T_N может входить любое число раз, и, кроме того, каждый раз в него входит одна и только одна из букв T_0 или T_1. На этом множестве устанавливается парное отношение (некоторые буквы «соединяются» попарно между собой) так, что удовлетворяется следующее условие «а»: все буквы, соединенные с какой-либо одной буквой множества M, попарно различны между собой.

Иными словами, информация представляется в виде некоторого одномерного комплекса (линейного ненаправленного графа), вершины которого (обозначаемые кружочками) отождествляются с буквами T_0, T_1, ..., T_N, а (ненаправленные) *отрезки*, соединяющие некоторые пары вершин, — с описанными выше парными связями между буквами.

Требование вхождения в рассматриваемый комплекс (множество M) *одной и только одной* вершины, отождествленной либо с буквой T_0, либо с буквой T_1, связано с необходимостью установления *начала отсчета* информации (центра активной зоны), причем одна из этих букв (примем, что ею является буква T_0) употребляется для комплексов, обозначающих информацию, переработка которой еще не закончена, а вторая (в данном случае буква T_1) — для комплексов, обозначающих заключительную информацию, из которой должен быть извлечен окончательный результат работы алгоритма.

Вершину информационного комплекса S, отождествленную с буквой T_0 или T_1, назовем начальной вершиной комплекса. *Активной зоной* комплекса S называется подкомплекс комплекса S, состоящий из вершин (букв) и отрезков (связей), принадлежащих содержащим начальную вершину цепям длины $\lambda \leqslant P$, где P — определенное, фиксированное для данного алгоритма число. *Цепью* здесь и ниже называем такую любую конечную последовательность вершин B_1, B_2, ..., B_p, что произвольные две соседние в этой последовательности вершины соединены между собой отрезком; число всех этих отрезков (равное $p - 1$) называется длиной

цепи, а сами эти отрезки также включаются в рассматриваемую цепь.

Границей активной зоны информационного комплекса называется совокупность всех его вершин, соединимых с начальной вершиной цепями длины P и не соединимых с нею цепями меньшей длины. Комплекс называется *связным*, если любые две его вершины можно соединить цепью. Совокупность вершин и отрезков, лежащих за пределами активной зоны комплекса S, называется внешней частью комплекса.

Два комплекса называются изоморфными между собой, если между их вершинами можно установить взаимно однозначное соответствие, при котором соответствующие друг другу вершины обозначены одинаковыми буквами T_0, T_1, ..., T_N, а соответствующие друг другу пары вершин являются либо одновременно соединенными, либо одновременно не соединенными между собой. Изоморфные между собой комплексы по существу одинаковы и различаются разве лишь способом своего изображения (например, расположением вершин на плоскости).

Ввиду ограниченности общего числа вершин в активной зоне информационного комплекса любого данного алгоритма и ограниченности числа букв T_0, T_1, ..., T_N для любого данного алгоритма A существует лишь конечное число различных (попарно неизоморфных) активных зон U_1, U_2, ..., U_r. Исходя из этого, правила их переработки можно задать простой таблицей соответствия $U_i \to W_i$ ($i = 1, 2, ..., r$).

Комплексы, стоящие в правой части этой таблицы, должны иметь подкомплексы, изоморфные границам соответствующих активных зон U_i, причем эти изоморфизмы должны быть раз и навсегда фиксированными. Иными словами, каждой вершине, лежащей на границе $L(U_i)$ активной зоны U_i, должна быть сопоставлена вполне определенная вершина комплекса W_i ($i = 1, 2, ..., r$). Каждый из комплексов W_i должен удовлетворять всем условиям, наложенным выше на информационные комплексы; он должен, в частности, иметь одну и только одну начальную вершину, обозначенную буквой T_0 или буквой T_1.

С помощью построенной таблицы соответствия определяем оператор R_A, осуществляющий непосредственную переработку информационного комплекса на каждом шаге работы данного алгоритма A. В рассматриваемом информационном комплексе S (начальном и промежуточном) находим начальную вершину. Проводя от нее все возможные цепи длины P, строим активную зону и определяем ее границу $L(U)$.

Далее, находим ту (единственную) активную зону из левой части таблицы соответствия, которая изоморфна найденной активной зоне U. Вследствие определенных выше свойств информационных комплексов (в частности, свойства «а») и связности обоих комплексов U_i и U между ними возможен только один изоморфизм. Это позволяет однозначно отождествить вершины, лежащие на

границе $L(U)$ активной зоны U, с соответствующими им при изоморфизме вершинами, лежащими на границе $L(U_i)$, активной зоны U_i, а используя отождествление вершин, принятое в таблице соответствия, — и с некоторыми вершинами комплекса U_i.

Теперь легко убрать всю внутреннюю, т. е. не лежащую на границе $L(U)$, часть активной зоны U и подменить ее подкомплексом W_i' комплекса W_i, включающим все элементы этого комплекса, кроме ранее отождествленных его вершин. Таким образом, производится «вклейка» в рассматриваемый информационный комплекс вместо внутренней части его активной зоны нового комплекса W_i' при сохранении в неизменности границы активной зоны.

Поскольку в комплексе W_i' начальная вершина занимает новое положение по отношению к границе прежней активной зоны, то новая активная зона, определяемая после вклейки, будет иметь другую границу. Получившийся после подобной «вклейки» новый информационный комплекс S' и будет представлять собой результат применения оператора непосредственной переработки R_A рассматриваемого алгоритма A к исходному информационному комплексу S. Оператор непосредственной переработки применяется к полученному информационному комплексу до получения комплекса, начальная вершина которого обозначена буквой T_1, а не буквой T_0.

Подобный комплекс носит название *заключительного*, а его максимальный связный подкомплекс, содержащий начальную вершину T_1, считается *решением*, т. е. информационным комплексом, полученным в результате воздействия алгоритма A на начальный (входной) информационный комплекс S_0. Если же алгоритм продолжает свою работу без конца, не получая ни на каком шаге заключительного комплекса, то, как и в случае нормальных алгоритмов, принимается, что к данному начальному комплексу S_0 рассматриваемый алгоритм не применим.

Можно расширить определение алгоритма таким образом, чтобы допускать в правой части таблицы соответствия комплексы без начальной вершины. Применение подстановки с такой правой частью приводит к естественному окончанию алгоритмического процесса, поскольку определение активной зоны и дальнейшие подстановки оказываются невозможными.

Однако так как заключительный комплекс (в определенном выше смысле) не возникает, то и в этом случае алгоритмический процесс следует считать закончившимся безрезультатно, а алгоритм — не применимым к соответствующему начальному информационному комплексу.

Возможен еще один тип безрезультатной остановки алгоритмического процесса, когда таблица соответствия содержит не все возможные для данного алгоритма виды активных зон. В случае достижения информационным комплексом состояния, при котором хотя и есть начальная вершина, обозначенная буквой T_0, но ни одна из подстановок таблицы соответствия неприменима. также

считается, что алгоритм не применим к соответствующему начальному информациоиному комплексу.

Необходимо сделать еще одно замечание, касающееся характера подстановок в таблице соответствия. Если не принять специальных мер, то в результате подстановки может быть нарушено введенное выше условие «а», которому должны удовлетворять все рассматриваемые информационные комплексы. Для того чтобы избежать подобного искажения информации, достаточно, очевидно, предполагать, что любая вершина произвольного комплекса W_i из правой части таблицы соответствия, которая в процессе «вклейки» отождествляется с какой-либо вершиной q границы активной зоны, в комплексе W_i может быть соединена отрезками лишь с начальной вершиной и с вершинами, обозначенными теми же буквами, что и вершины, с которыми соединена отрезками в комплексе U_i вершина, соответствующая вершине q.

Подобное условие (назовем его «б») не нарушает общности наших рассмотрений. Граница, с помощью которой производится операция «вклейки», определена достаточно условно. Если бы мы включили в границу не только те вершины, которые отстоят от начальной вершины на расстоянии P (соединены с ней цепями длины P, но не цепями меньшей длины), но и вершины, которые отстоят от нее на расстоянии $P - 1$, то, устанавливая изоморфизм границ в комплексах U_i и W_i, мы получили бы, как нетрудно видеть, более сильное ограничение на таблицу соответствия, чем ограничение, накладываемое условием «б».

Внимательный анализ описания алгоритмической схемы Колмогорова—Успенского показывает, что эта схема по форме в весьма значительной степени напоминает работу, фактически выполняемую человеком, когда он преобразовывает задаваемую ему извне информацию в соответствии с теми или иными правилами запоминаемого им алгоритма. Авторы этой схемы предпринимали специальные меры, чтобы не потерять общности в характере выполняемых преобразований. Тем не менее они показали, что описанная ими схема дает возможность строить лишь нормализуемые алгоритмы. Этот результат можно рассматривать как подтверждение сформулированного в § 2 принципа нормализации.

§ 4. Другие теоретические алгоритмические системы

Исторически первой алгоритмической системой, получившей достаточно полное и всестороннее развитие, была система, основанная на использовании конструктивно определяемых арифметических (целочисленных) функций, получивших специальное название *рекурсивных* функций. Применение подобных функций в теории алгоритмов основано на идее нумерации слов в произвольном алфавите последовательными натуральными числами. Наиболее просто такую нумерацию можно осуществить, располагая слова в порядке возрастания их длин, а слова, имеющие одинаковую дли-

ну, — в произвольном (например, в лексикографическом) порядке.

После нумерации входных и выходных слов в произвольном алфавитном операторе этот оператор превращается в функцию $y=f(x)$, в которой как аргумент x, так и сама функция y принимают неотрицательные целочисленные значения. Функция $f(x)$, разумеется, может быть определенной не для всех, а лишь для некоторых значений аргумента x, составляющих область определения этой функции. Подобные частично определенные целочисленные и целозначные функции для краткости называют обычно *арифметическими функциями.*

Среди арифметических функций выделим следующие особо простые функции, которые будем называть *элементарными арифметическими функциями:* функцию, *тождественно равную нулю* (определенную для всех целых неотрицательных значений аргументов); тождественные функции $f(x_i)=x_i$, повторяющие значения своих аргументов; функцию *непосредственного следования* $f(x)=x+1$, также определенную для всех целых неотрицательных значений своего аргумента.

Используя в качестве исходных функций перечисленные элементарные арифметические функции, можно с помощью небольшого числа общих конструктивных приемов строить все более и более сложные арифметические функции. В теории рекурсивных (конструктивных арифметических) функций особо важное значение имеют три операции: операции *суперпозиции, примитивной рекурсии* и *наименьшего корня.*

Операция суперпозиции функций заключается в подстановке одних арифметических функций вместо аргументов других арифметических функций. Таким образом, из уже известных функций можно строить новые арифметические функции. Например, осуществляя суперпозицию функций $f(x) = 0$ и $g(x) = x + 1$, придем к функции $h(x) = 1$. При суперпозиции функции $g(x)$ с самой собой возникает функция $p(x) = x + 2$ и т. д.

Операция примитивной рекурсии позволяет строить n-местную арифметическую функцию (функцию от n аргументов) по двум заданным функциям, одна из которых является $(n - 1)$-местной, а другая — $(n + 1)$-местной. Способ такого построения определяется следующими двумя соотношениями:

$$f(x_1, \ldots, x_{n-1}, 0) = g(x_1, \ldots, x_{n-1}); \tag{6}$$

$$f(x_1, x_2 \ldots, x_{n-1}, x_n + 1) = h(x_1, x_2, \ldots, x_n, y), \tag{7}$$

где $y = f(x_1, \ldots, x_{n-1}, x_n)$; f определяемая, а g и h — заданные функции.

Для правильного понимания операции примитивной рекурсии необходимо заметить, что всякую функцию от меньшего числа переменных можно рассматривать как функцию от любого большего

числа переменных. В частности, функции-константы, которые естественно рассматривать как функции от нуля аргументов, можно при желании рассматривать как функции от любого конечного числа аргументов.

В качестве примера применения операции примитивной рекурсии покажем, каким образом при помощи этой операции из элементарных арифметических функций можно построить двухместную функцию суммирования $f(x, y) = x + y$. Эта функция определяется с помощью тождественной функции $g(x) = x$ и функции непосредственного следования $h(x) = x + 1$

$$f(x, 0) = x = g(x);$$
$$f(x, y + 1) = (x + y) + 1 = h(f(x, y)).$$

Аналогично можно построить функции произведения, показательную, степенную и другие широко известные арифметические функции.

Функции, которые могут быть построены из элементарных арифметических функций с помощью операций суперпозиции и примитивной рекурсии, примененных любое (конечное) число раз в произвольной последовательности, называются примитивно рекурсивными функциями.

Большинство арифметических функций относится к примитивно рекурсивным функциям. Тем не менее примитивно рекурсивные функции не охватывают всех арифметических функций, которые могут быть определены конструктивно. При построении всех этих функций используются другие операции, в частности операция наименьшего корня.

Операция наименьшего корня позволяет определять новую арифметическую функцию $f(x_1, \ldots, x_n)$ от n переменных с помощью ранее построенной арифметической функции $g(x_1, \ldots, x_n, y)$ от $n+1$ переменных. Для любого заданного набора значений переменных $x_1 = \alpha_1, \ldots, x_n = \alpha_n$ в качестве соответствующего значения $f(\alpha_1, \alpha_2, \ldots, \alpha_n)$ определяемой функции $f(x_1, x_2, \ldots, x_n)$ принимается наименьший целый неотрицательный корень $y = \beta$ уравнения $g(\alpha_1, \ldots, \alpha_n, y) = 0$. В случае несуществования целых неотрицательных корней у этого уравнения функция $f(x_1, x_2, \ldots, x_n)$ считается неопределенной при соответствующем наборе значений переменных. Обычно принимается также, что функция $f(x_1, x_2, \ldots, x_n)$ не определена на наборе $x_1 = \alpha_1, x_2 = \alpha_2, \ldots, x_n = \alpha_n$, если при существовании наименьшего корня $y = \beta$ уравнения $g(\alpha_1, \alpha_2, \ldots, \alpha_n, y) = 0$ хотя бы при одном целом неотрицательном значении $y = \gamma$, удовлетворяющем соотношению $0 \leqslant \gamma \leqslant \beta - 1$, функция $g(\alpha_1, \alpha_2, \ldots, \alpha_n, y)$ оказывается неопределенной.

Арифметические функции, которые могут быть построены из элементарных арифметических функций с помощью операций суперпозиции, примитивной рекурсии и наименьшего корня, называются частично рекурсивными функциями. Если такие функции оказываются к тому же всюду определенными, то они называются общерекурсивными функциями.

В этом определении, как и в определении примитивно рекурсивных функций, предусматривается возможность выполнения всех допустимых операций в любой последовательности и любое конечное число раз. Существует, однако, принадлежащий С. Клини [41] результат, позволяющий получать произвольную частично рекурсивную функцию из двух примитивно рекурсивных функций с помощью последовательного применения к ним одной операции наименьшего корня и одной операции суперпозиции. Более точно этот результат можно сформулировать так:

для любой частично рекурсивной функции $f(x_1, \ldots, x_n)$ *существуют две примитивно рекурсивные функции* $g(x_1, \ldots, x_n, y)$ и $h(x)$ *такие, что функция* $f(x_1, \ldots, x_n)$ *может быть получена из них в виде* $f(x_1, \ldots, x_n) = h(\mu_y[g(x_1, \ldots, x_n, y) = 0])$, *где* μ_y *— оператор наименьшего корня. При этом функция* $h(x)$ *может быть выбрана раз и навсегда, независимо от выбора функции* f.

Частично рекурсивные функции представляют собой наиболее общий класс конструктивно определяемых арифметических функций. Они охватывают, в частности, все арифметические функции, которые можно задать в виде конечных рекурсивных схем произвольного вида. Под конечной рекурсивной схемой здесь подразумевается любая конечная система равенств $r = s$, где r и s — любые конечные (содержащие конечное число символов) выражения, построенные из известных примитивно рекурсивных функций неизвестных функций с числовыми и буквенными аргументами, причем значения неизвестных функций для любых заданных значений аргументов должны определяться однозначно за конечное число шагов (зависящее от выбора значений аргументов) в результате применения двух правил. Первое правило (правило подстановки) состоит в *подстановке* в какое-нибудь из заданных равенств вместо одного из аргументов какого-либо его числового значения. Второе правило (правило замены) позволяет использовать некоторое равенство вида $x = f(x_1, x_2, \ldots, x_n)$, где $x, x_1, x_2, \ldots, x_n$— числа для *замены* некоторого вхождения величины $f(x_1, x_2, \ldots, x_n)$ в одно из равенств $r = s$ величиной x.

Оказывается, что подобным образом могут быть заданы все общерекурсивные функции и только такие функции. Это обстоятельство позволяет, следуя Эрбрану и Гёделю, определять общерекурсивные функции как функции, задаваемые конечными рекурсивными схемами описанного выше вида.

Если, сохранив условие однозначности, не требовать определимости значений входящих в схему функций для *всех* значений аргументов, можно задавать подобными схемами частично рекурсивные функции. Существенно, что *никакими* рекурсивными определениями (с помощью конечных схем) не удается выйти за пределы класса частично рекурсивных функций.

После осуществления нумерации входных и выходных слов любой нормальный алгоритм может быть реализован в виде частично рекурсивной функции. Наоборот, любой алгоритм, реализуемый с

помощью частично рекурсивной функции, оказывается эквивалентным некоторому нормальному алгоритму. Таким образом, можно сделать следующий важный вывод.

Алгоритм тогда и только тогда нормализуем, когда он может быть реализован с помощью частично рекурсивной функции.

Сформулированное предложение показывает, что и на базе арифметического (числового) подхода к теории алгоритмов не происходит выхода из класса нормализуемых алгоритмов.

Изучим еще два подхода к теории алгоритмов, предложенные в 1936 г. Э. Постом [63] и А. Тьюрингом [73].

В алгоритмической системе, предложенной Постом, входная и выходная информации представляются в стандартном двоичном алфавите, а алгоритм — в виде конечного упорядоченного набора правил, называемых приказами. Для записи входной, выходной и промежуточной информации используется гипотетическая бесконечная *информационная лента*, разделенная на отдельные ячейки, в каждой из которых можно поместить лишь одну букву (цифру 0 или 1). Те ячейки, в которых записаны единицы, называются *отмеченными*, а те, в которых записаны нули, — *неотмеченными*. В любой момент работы алгоритма лишь конечное число ячеек может быть отмеченным.

Работа алгоритма осуществляется дискретными шагами, на каждом из которых выполняется один из составляющих алгоритм приказов. Каждому шагу соответствует определенная *активная* ячейка на информационной ленте. Для первого приказа алгоритма фиксируется в качестве активной некоторая *начальная* ячейка. Дальнейшие изменения месторасположения активной ячейки на ленте должны быть предусмотрены в самом алгоритме. Составляющие алгоритм приказы могут принадлежать к одному из следующих шести типов.

Первый тип. Отметить активную ячейку ленты (записать в нее единицу) и перейти к выполнению i-го приказа (i может быть любым числом из чисел, использованных для нумерации приказов алгоритма).

Второй тип. Стереть отметку активной ячейки (записать в нее нуль) и перейти к выполнению i-го приказа.

Третий тип. Сместить активную ячейку на один шаг вправо и перейти к выполнению i-го приказа.

Четвертый тип. Сместить активную ячейку на один шаг влево и перейти к выполнению i-го приказа.

Пятый тип. Если активная ячейка отмечена (если в ней записана единица), то перейти к выполнению j-го приказа, а если активная ячейка не отмечена (если в ней записан нуль), то перейти к выполнению i-го приказа.

Шестой тип. Остановка, окончание работы алгоритма.

Алгоритмы, составляемые из любого конечного числа правил описанных типов, называются *алгоритмами Поста*. Доказано, что алгоритмы Поста сводятся к алгоритмам, реализуемым с помощью

частично рекурсивных функций, и, наоборот, любая частично рекурсивная функция может быть представлена алгоритмом системы Поста. Таким образом, можно сформулировать следующее предложение.

Класс всех алгоритмов, эквивалентных алгоритмам Поста, совпадает с классом всех нормализуемых алгоритмов.

К описанной алгоритмической схеме очень близка схема, предложенная одновременно с Э. Постом А. Тьюрингом [73]. В схеме Тьюринга, которую принято называть *машиной* Тьюринга, также производится запись информации на бесконечной в обе стороны информационной ленте, разделенной на отдельные ячейки. Однако, в отличие от алгоритма Поста, здесь для записи информации употребляется произвольный конечный алфавит. Каждая ячейка информационной ленты служит для записи одной буквы. Эта буква может обозреваться специальным чувствительным элементом так называемой *головки* машины Тьюринга, способной перемещаться вдоль информационной ленты в обе стороны. Головка машины Тьюринга может находиться в конечном числе различных состояний $q_1, q_2, \ldots, q_n$, печатать в обозреваемую ячейку любую букву $x_1, x_2, \ldots, x_m$ и сдвигаться вправо или влево вдоль информационной ленты на одну ячейку.

Запись алгоритма, реализуемого машиной Тьюринга, производится с помощью программы работы этой машины, представляющей собой набор пятерок символов вида $x_i\ q_j\ x_k\ q_r\ s_p$. Выписанная пятерка символов означает, что головка машины Тьюринга, находясь в состоянии q_j и воспринимая записанную на ленте букву x_i, печатает на месте этой буквы новую букву x_k (которая может в частном случае совпасть с ранее записанной буквой x_i), переходит в новое состояние q_r (которое также может совпасть с прежним состоянием) и осуществляет сдвиг вдоль ленты на величину s_p, равную ± 1.

Первоначальная схема машины Тьюринга была предназначена для выписывания значений, принимаемых произвольной одноместной частично рекурсивной функцией при значениях аргумента, равных 0, 1, 2, ... При этом, разумеется, машина Тьюринга должна работать бесконечно долго. Можно построить машину Тьюринга, вычисляющую значения любой наперед заданной частично рекурсивной функции. Целесообразно, однако, видоизменить описанную выше первоначальную схему машины Тьюринга. Примем, что последний символ s_p пятерок символов, описывающих работу машины Тьюринга, может принимать, кроме введенных выше значений ± 1, и третье значение—«остановка машины». С таким дополнением машина Тьюринга превращается в обычную алгоритмическую систему. Она либо преобразует первоначально записанное на ленту входное слово p бесконечно долго, либо после конечного числа шагов преобразований остановится. В первом случае принимается, как обычно, что реализуемый машиной алгоритм не применим ко входному слову p. Во втором случае остающаяся

на ленте в момент остановки машины информация принимается за выходное слово, в которое машина преобразует заданное входное слово p. При этом, разумеется, необходимо иметь в алфавите, используемом для записи информации на ленте, специальную *пустую* букву для обозначения тех ячеек, в которые не записана никакая информация.

Можно показать, что все алгоритмы, реализуемые с помощью описанных видоизмененных машин Тьюринга, нормализуемы и, наоборот, любой нормализуемый алгоритм может быть реализован с помощью специально построенной для этой цели машины Тьюринга. Используя запись программ работы машин Тьюринга и их входных слов в некотором стандартном алфавите, можно построить *универсальную* машину Тьюринга точно таким же путем, как и универсальный нормальный алгоритм (§ 2). Задавая универсальной машине Тьюринга *изображение* программы любой данной машины Тьюринга M и изображение любого ее входного слова p, получим изображение выходного слова q, в которое машина M переводит входное слово p. Если же алгоритм, реализуемый машиной M, не применим к слову p (машина M работает над его преобразованием бесконечно долго), то алгоритм, реализуемый универсальной машиной Тьюринга, также не применим к слову, образованному из изображения слова p и программы машины M.

Таким образом, несмотря на большое качественное различие, все описанные алгоритмические системы приводят, по существу (с точностью до эквивалентности), к одному и тому же классу алгоритмов. Этот вывод является еще одним подтверждением, что современная теория алгоритмов охватывает чрезвычайно широкий класс (если не все) конструктивно определяемых алфавитных операторов.

§ 5. Понятие об алгоритмически неразрешимых проблемах

Всякий алгоритм представляет собой способ решения некоторой *массовой проблемы*, формулируемой в виде проблемы переработки не одного, а целого *множества* входных слов в соответствующие им выходные слова. Поскольку как условие, так и решение любой задачи может быть выражено в виде отдельных слов, всякий алгоритм можно рассматривать как некоторое универсальное средство для решения *целого класса* задач.

Детальный анализ показывает, что существуют такие классы задач, для решения которых нет и не может быть единого *универсального* приема. Проблемы решения такого рода задач называют *алгоритмически неразрешимыми проблемами*. Однако алгоритмическая неразрешимость проблемы решения задач того или иного класса вовсе не означает невозможности решения любой конкретной задачи из этого класса. Речь идет о невозможности решения *всех* задач данного класса *одним и тем же* приемом.

Для лучшего понимания проблемы алгоритмической неразр-

решимости приведем примеры алгоритмически разрешимой и алгоритмически неразрешимой проблем.

Типичным примером алгоритмически разрешимой проблемы является проблема доказательства тождеств в обычной алгебре. Для простоты ограничимся случаем, когда тождества строятся из рациональных чисел и букв (обозначений переменных) с помощью действий сложения, вычитания и умножения. Из школьного курса алгебры хорошо известен следующий общий прием решения указанной проблемы: используя распределительный закон для умножения, раскрывают скобки в правой и в левой частях любого данного тождества и выполняют приведение подобных членов в соответствии с хорошо известными правилами. После осуществления всех этих преобразований как левая, так и правая части исходного тождества превращаются в полиномы. Тождество будет справедливым в том и только в том случае, когда эти полиномы тождественно совпадают между собой. Иначе говоря, справедливость тождества означает, что после перенесения всех членов преобразованного тождества в одну часть эти члены взаимно уничтожаются, в результате чего тождество превращается в тривиальное тождество $0 = 0$.

Таким образом, проблема тождества в элементарной алгебре алгоритмически разрешима — существует единый конструктивный прием, позволяющий за конечное число шагов решить, является ли любое заданное соотношение тождественным. Можно, однако, построить примеры таких алгебраических систем, в которых проблема тождества является алгоритмически неразрешимой проблемой. В качестве таких алгебраических систем могут быть, например, выбраны полугруппы или группы, заданные системами образующих элементов и определяющих соотношений. Примеры для полугрупп с неразрешимой проблемой тождества впервые были найдены Э. Постом [64], а соответствующие примеры для групп — П. С. Новиковым [60].

Не выписывая явно определяющих соотношений, поясним суть указанных примеров. Пусть $x_1, x_2, \ldots, x_n$ — буквы какого-нибудь конечного алфавита. Множество всех слов в этом алфавите, включая пустое слово e, называется *свободной полугруппой* с образующими элементами $x_1, x_2, \ldots, x_n$, если для произвольных пар слов p, q введена операция умножения, заключающаяся просто в приписывании одного слова к другому. Условимся обозначать свободную полугруппу с образующими элементами $x_1, x_2, \ldots, x_n$ через $F(x_1, x_2, \ldots, x_n)$, а результат умножения слова p на слово q — через pq.

В свободной полугруппе можно ввести любое множество *определяющих соотношений*, представляющих собой формальные равенства между двумя неодинаковыми словами: $p_i = q_i$ ($i = 1, 2, \ldots$). Два слова в свободной полугруппе F с заданной системой S определяющих соотношений называются тождественными, или эквивалентными, между собой, если одно из них может быть

получено из другого в результате произвольного числа подстановок во второе слово правых частей определяющих соотношений вместо левых, и, наоборот, левых вместо правых. Например, в полугруппе с системой образующих (x, y) и одним определяющим соотношением $xy = yx$ слова $p = xxy$ и $q = yxx$ являются тождественными между собой, поскольку первое слово может быть получено из второго в результате двух подстановок описанного выше вида: $q = yxx \to xyx \to xxy = p$. При обратиой подстановке выписанная цепь подстановок может быть прочитана и в обратном направлении, что дает возможность преобразования не только слова q в слово p, но и слова p в слово q.

Проблема тождества слов для полугрупп фурмулируется так. *Пусть в произвольной свободной полугруппе F с конечным числом образующих задана любая система определяющих соотношений S, состоящая из конечного числа соотношений. Требуется найти единый конструктивный прием, позволяющий за конечное число шагов решить, являются любые два заданных слова полугруппы F с системой определяющих соотношений S тождественными или нетождественными.*

Для некоторых систем определяющих соотношений сформулированная проблема разрешима, однако, как показал Э. Пост [64], существуют и такие системы определяющих соотношений, для которых проблема тождества слов *алгоритмически неразрешима*. Это не означает, разумеется, невозможности установления тождественности или нетождественности любой *фиксированной* конкретной пары слов. Не существует *единого* приема для установления тождественности *любой* пары слов, подобно описанному выше приему для доказательства справедливости или несправедливости любого соотношения в элементарной алгебре.

Проблема тождества слов для групп в основных чертах совпадает с соответствующей проблемой для полугрупп. Свободная группа G с образующими элементами $x_1, x_2, \ldots, x_n$ строится как совокупность слов, составленных из букв $x_1, x_2, \ldots, x_n$ и «обратных» им букв $x_1^{-1}, x_2^{-1}, \ldots, x_n^{-1}$. При этом две рядом стоящие обратные друг другу буквы взаимно уничтожаются (оказываются эквивалентными пустому слову)

$$x_i x_i^{-1} = x_i^{-1} x_i = e. \tag{8}$$

При определении тождественности двух слов в группе с системой определяющих соотношений S следует учитывать не только соотношения, входящие в эту систему, но и соотношения вида (8). Как и для полугрупп, проблема тождества слов для групп, заданных конечным числом образующих и определяющих соотношений, в общем случае оказывается алгоритмически неразрешимой. Примеры групп с неразрешимой проблемой тождества слов были построены впервые П. С. Новиковым [60].

Каким образом может быть доказана алгоритмическая неразрешимость той или иной проблемы? Классическим примером подоб-

ной неразрешимой проблемы является проблема распознавания самоприменимости алгоритмов. Для точной формулировки этой проблемы будем оперировать лишь нормальными алгоритмами в алфавитах, состоящих не менее чем из двух букв. При этом предположении можно, не нарушая общности, условиться о том, чтобы какие-либо буквы алфавита любого алгоритма, о котором будет идти речь, отождествлять с двумя буквами (0 и 1) стандартного двоичного алфавита. По принятому условию для любого рассматриваемого алгоритма A его изображение A^{μ} в стандартном двоичном алфавите можно рассматривать как входное слово этого алгоритма. Если слово A^{μ} входит в область определения алгоритма A, то алгоритм называется *самоприменимым*, в противном случае — *несамоприменимым*.

Существуют как самоприменимые, так и несамоприменимые алгоритмы. Примером самоприменимого (нормального) алгоритма является так называемый *тождественный* алгоритм в любом алфавите $\mathfrak{X}$, содержащем две или более двух букв. По определению, этот алгоритм применим к любому слову p в алфавите $\mathfrak{X}$ и перерабатывает любое входное слово в себя. Примером несамоприменимого алгоритма является так называемый *нулевой* алгоритм в любом конечном алфавите $\mathfrak{Y}$. Этот алгоритм задается схемой, содержащей единственную подстановку $\rightarrow y$ (где y — любая буква алфавита $\mathfrak{Y}$). По самому своему определению он не применим ни к какому входному слову, а значит, и к своему изображению.

Проблема распознавания самоприменимости алгоритмов состоит в том, чтобы найти единый конструктивный прием, позволяющий за конечное число шагов по схеме любого данного алгоритма A в какой-либо фиксированной алгоритмической системе (например, в системе нормальных алгоритмов) узнать, является алгоритм A самоприменимым или несамоприменимым.

Если считать справедливым сформулированный в § 2 принцип нормализации, можно предполагать, что единый конструктивный прием, о котором идет речь, является не чем иным, как нормальным алгоритмом B, определенным на любом слове p, которое является изображением произвольного нормального алгоритма A, и переводящим это слово в два различных фиксированных слова q_1 и q_2 в зависимости от того, будет алгоритм A самоприменимым или несамоприменимым (слово q_1 — код слова «самоприменим», а q_2 — код слова «несамоприменим»).

На любом входном слове l, не являющемся изображением какого-либо (нормального) алгоритма, алгоритм B также должен быть определен. Действительно, в противном случае, не получив никакого результата после какого-либо (достаточно большого) числа шагов работы алгоритма, неизвестно было бы, изображением какого алгоритма — самоприменимого или несамоприменимого — является слово l. Ясно также, что результат применения алгоритма B к любому слову, не являющемуся изображением алгоритма, должен быть отличным как от слова q_1, так и от слова q_2.

Предположим, что алгоритм B с указанными свойствами существует. В таком случае существует нормальный алгоритм C в том же самом алфавите $\mathfrak{A}$, что и алгоритм B, определенный на всех тех и только тех словах в алфавите $\mathfrak{A}$, которые являются изображениями несамоприменимых алгоритмов (напомним, что по самому определению алгоритма B алфавит $\mathfrak{A}$ включает в себя стандартный двоичный алфавит).

Действительно, построим нормальный алгоритм D в алфавите $\mathfrak{A}$, область определения которого состоит из одного лишь слова q_2. Такой алгоритм может быть задан, например, в виде (нормализованной) суперпозиции двух нормальных алгоритмов D_1 и D_2, первый из которых задается схемой, состоящей из одной подстановки $q_2 \to \cdot$, а второй — схемой, состоящей из подстановок вида $x_i \to x_i$, где x_i пробегает все буквы алфавита $\mathfrak{A}$. Ясно, что первый алгоритм переводит в пустое слово только слово q_2, а область определения второго алгоритма состоит только из пустого слова. Поэтому область определения суперпозиции D алгоритмов D_1 и D_2 будет состоять только из слова q_2, что нам и требуется.

Построив алгоритм D, образуя суперпозицию с ним алгоритма B и нормализуя эту суперпозицию, придем к нормальному алгоритму C в алфавите $\mathfrak{A}$, область определения которого состоит из всех тех и только тех слов в алфавите $\mathfrak{A}$, которые являются записями несамоприменимых алгоритмов. Однако подобное свойство алгоритма C внутренне противоречиво, поскольку к своему собственному изображению C^{μ} алгоритм C не может быть ни применимым, ни не применимым.

В самом деле, в первом случае алгоритм C был бы применим к своему изображению и являлся бы поэтому *самоприменимым*. Но это противоречило бы тому, что алгоритм C, в силу своего построения, должен быть применимым только к несамоприменимым алгоритмам. Во втором случае, будучи неприменимым к своему изображению, алгоритм C принадлежал бы к числу несамоприменимых алгоритмов. Но тогда по определению алгоритм C должен был бы быть применимым к своему изображению, поскольку он применим к изображению *всех* несамоприменимых алгоритмов. Следовательно, алгоритм C является самоприменимым.

Таким образом, предположение об алгоритмической разрешимости проблемы распознавания самоприменимости приводит к логическому противоречию и является поэтому неверным. Тем самым показана алгоритмическая неразрешимость этой проблемы.

Вывод обоснован нами лишь при условии, что справедлив принцип нормализации алгоритмов. Однако природа противоречия, использованного при доказательстве алгоритмической неразрешимости проблемы распознавания самоприменимости алгоритмов, является в действительности более глубокой. Читатель, знакомый с парадоксами теории множеств и математической логики, легко заметит, что указанное противоречие имеет ту же природу, что и противоречие в известном парадоксе Рассела, устанавливающем

внутреннюю противоречивость понятия «множество всех множеств, не содержащее себя в качестве своего элемента».

Это обстоятельство приводит к выводу, что алгоритмическая неразрешимость проблемы распознавания самоприменимости не является следствием узости современного точного понятия алгоритма. Если бы удалось построить точное понятие алгоритма, включающее в себя некоторые ненормализуемые алгоритмы, то проблема распознавания самоприменимости алгоритмов по-прежнему оставалась бы алгоритмически неразрешимой.

Из алгоритмической неразрешимости проблемы распознавания самоприменимости алгоритмов выводится алгоритмическая неразрешимость целого ряда других проблем. Общий метод для подобных выводов состоит в том, что из предложения о существовании алгоритма, решающего ту или иную проблему Q, выводится существование алгоритма, решающего проблему распознавания самоприменимости алгоритмов. Поскольку последнее невозможно, то невозможно и существование алгоритма, решающего проблему Q.

С помощью общего метода была доказана алгоритмическая неразрешимость множества различных проблем, в том числе рассмотренных выше общих проблем тождества слов для групп и полугрупп. Назовем еще некоторые алгоритмически неразрешимые проблемы, неразрешимость которых установлена этим же методом. Это проблема *распознавания применимости* какого-либо алгоритма к тому или иному слову. *Может быть построен алгоритм A, действующий в некотором алфавите $\mathfrak{X}$, для которого не существует алгоритма в алфавите $\mathfrak{X}$ и в любом его расширении, перерабатывающего в некоторое фиксированное слово те и только те слова, к которым алгоритм A не применим.*

Проблема построения алгоритма, перерабатывающего в фиксированное слово p все слова, к которым любой данный алгоритм A *применим*, является, как нетрудно видеть, алгоритмически разрешимой — для ее решения достаточно построить алгоритм B, переводящий в слово p все слова в алфавите алгоритма A, и образовать суперпозицию алгоритмов A и B. Условимся, что некоторый алгоритм *аннулирует* те или иные слова, если он переводит их в пустоте слово e. Проблема *распознавания аннулирования* для любого данного алгоритма A состоит в построении алгоритма B (в том же, что и алгоритм A, алфавите), аннулирующего все те и только те слова, которые алгоритм A не аннулирует. Эта проблема в общем случае оказывается алгоритмически неразрешимой, а именно: можно так выбрать алгоритм A, что алгоритм B с указанными свойствами для него построить невозможно.

Очень часто при доказательстве алгоритмической неразрешимости тех или иных проблем используют доказанную Э. Постом [64] алгоритмическую неразрешимость следующей проблемы, получившей название *комбинаторной проблемы Поста*. Пусть в произвольном конечном алфавите $\mathfrak{X}$ заданы любые конечные системы S пар слов $(p_1, q_1), \ldots, (p_n, q_n)$. Требуется построить единый кон-

структивный прием, позволяющий для любой такой системы S за конечное число шагов ответить, можно ли построить такое слово $p_{i_1} p_{i_2} \ldots p_{i_k}$ из первых элементов пар системы S, чтобы оно совпало со словом $q_{i_1} q_{i_2} \ldots q_{i_k}$, построенным из соответствующих им вторых элементов той же системы пар.

Алгоритмически неразрешимой оказывается также *проблема представимости* для матриц. Для формулировки этой проблемы условимся, что некоторая матрица *представима* через матрицы $U_1, U_2, \ldots, U_n$, если для некоторой конечной последовательности (вообще говоря, с повторениями) $U_{i_1} U_{i_2} \ldots U_{i_k}$ этих матриц произведение $U_{i_1} U_{i_2} \ldots U_{i_k}$ всех матриц, входящих в данную последовательность, совпадает с заданной исходной матрицей U. *Проблема представимости состоит в том, чтобы найти общий конструктивный прием, с помощью которого за конечное число шагов для любой матрицы U и любой конечной системы S матриц можно было бы узнать, представима матрица U через матрицы системы S или непредставима.*

Напомним, что алгоритмическая неразрешимость всех указанных проблем доказывается в предположении справедливости принципа нормализации, однако, как отмечалось выше, природа этой неразрешимости оказывается более глубокой и в некотором смысле независимой от этого принципа.

Глава II

БУЛЕВЫ ФУНКЦИИ И ИСЧИСЛЕНИЕ ВЫСКАЗЫВАНИЙ

§ 1. Понятие о булевых функциях

Булевыми (или *переключательными*) функциями принято называть такие функции, у которых все аргументы, как и сами функции, могут принимать лишь два различных значения.

Роль булевых функций в кибернетике определяется двумя основными обстоятельствами. Во-первых, булевы функции представляют собой удобный аппарат для описания схем многих преобразователей информации, построенных по дискретному принципу, поскольку для современной техники гораздо проще строить дискретные элементы, работающие именно в двоичном, а не в каком-нибудь другом алфавите. Во-вторых, булевы функции широко применяются в математической логике, представляющей собой одну из основ, на которых зиждется автоматизация сложных мыслительных процессов.

Применение наряду с обычными переменными, принимающими числовые значения, булевых переменных, имеющих лишь два возможных значения, играет существенную роль при построении различного рода практических алгоритмических систем для программирования на электронных вычислительных машинах. Булевы функции можно с успехом применять и для решения некоторых общих вопросов теории алгоритмов, например при уточнении понятия сложности алгоритма. Два возможных значения переменных, которые фигурируют в определении булевых функций, можно обозначать произвольно. На практике, однако, применяются в основном две системы обозначений. Первая из них (при применениях булевых функций к теории схем автоматов) закрепляет за возможными значениями булевых переменных обозначения 0 и 1. Будем называть введенные символы, как и в случае чисел, нулем и единицей, учитывая что нуль и единица выступают здесь пока не в качестве чисел, а лишь в качестве удобных обозначений для букв абстрактного двоичного алфавита. В дальнейшем при-

дадим этим символам ряд свойств, которые позволят рассматривать их (с одним исключением) как обычные числа (в этом как раз и состоит удобство рассматриваемой системы обозначений). Но все подобные свойства должны точно определяться перед использованием. Нельзя, в частности, пока пользоваться свойствами нуля и единицы, вытекающими из наличия операций сложения и умножения для чисел, поскольку указанные операции для этих символов не были еще нами определены.

Во второй системе обозначений в качестве обозначений для двух возможных значений булевых переменных служат слова «истина» и «ложь». Эта система обозначений используется в математической логике и прежде всего в той ее части, которая называется *исчислением высказываний*. Ее применение связано с тем обстоятельством, что в исчислении высказываний булевы переменные интерпретируются как переменные высказывания, рассматриваемые под углом зрения своей истинности и ложности.

В данном и трех последующих параграфах будем пользоваться первой системой обозначений, не оговаривая этого каждый раз. При необходимости перехода от одной системы обозначений к другой условимся, что единица соответствует истине, а нуль — лжи (можно было бы, разумеется, принять и прямо противоположное соответствие).

Рассмотрим булевы функции от любого конечного числа аргументов. Если число аргументов равно n, то соответствующую функцию принято называть *n-местной*. Благодаря тому, что каждая булева переменная может принимать лишь два значения, область определения любой булевой функции будет непременно конечной. Легко видеть, что область определения n-местной булевой функции может состоять максимум из 2^n различных элементов, представляющих собой всевозможные *наборы значений* n ее аргументов. Будем обычно упорядочивать аргументы заданной булевой функции, присваивая им номера 1, 2, ..., n. В таком случае набор значений аргументов отождествляется с некоторым *кортежем* (конечной упорядоченной последовательностью) нулей и единиц. Например, набор значений $x_1 = 1$, $x_2 = 0$, $x_3 = 0$ аргументов трехместной булевой функции $f(x_1, x_2, x_3)$ может быть кратко записан в виде кортежа 100, а набор $x_1 = 0$, $x_2 = 0$, $x_3 = 1$ — в виде кортежа 001. В дальнейшем будем часто называть указанные кортежи просто наборами (аргументы при этом нумеруются всегда в определенном порядке — в том порядке, в каком они встречаются в обозначении $f(x_1, x_2, \ldots, x_n)$, соответствующем булевой функции). Термин *булев* применительно к кортежу (набору) означает, что соответствующий кортеж составлен из нулей и единиц.

Каждый кортеж длины n, составленный из нулей и единиц (булев кортеж), может быть отождествлен с некоторой вершиной n-мерного единичного куба, имеющей соответствующие координаты. Для двумерного случая, когда n-мерный куб сводится к квадрату, способ отождествления булевых кортежей с вершинами

указан на рис. 4. В силу возможности подобного отождествления булевы наборы (кортежи) будут иногда называться также *точками*.

В настоящей главе ограничимся (за исключением особо оговариваемых случаев) рассмотрением лишь таких булевых функций, область определения которых включает все наборы значений ее аргументов. Таким образом, n-местная булева функция должна быть определена в 2^n различных точках. Если не исключается случай, когда та или иная булева функция может быть не определена хотя бы на одном из наборов, то она называется *частичной булевой функцией*. Рассмотрение частичных булевых функций полезно при синтезе схем дискретных автоматов. В теоретическом плане интерес представляют именно всюду определенные булевы функции, тем более, что в случае необходимости всякая частичная булева функция может быть *доопределена* (вообще говоря, произвольным способом) на тех наборах, на которых она первоначально не была определена. Поэтому, говоря о булевых функциях, в дальнейшем (если специально не оговорено противное) будем подразумевать под ними именно всюду определенные функции.

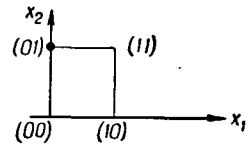

Рис. 4.

Заметим еще, что при рассмотрении той или иной булевой функции будем считать заданным число ее аргументов. Необходимость этой оговорки обусловливается возможностью трактовки всякой n-местной функции как $(n+1)$-местной, $(n+2)$-местной и, вообще, как $(n+k)$-местной функции для любого натурального числа k. Действительно, например, функцию-константу (равную тождественно нулю или единице) можно при желании считать функцией любого числа аргументов, от которых она, однако, в действительности не зависит. Аналогично можно к любой функции $f(x_1, x_2, \ldots, x_n)$ добавить какое угодно число новых аргументов $x_{n+1}, \ldots, x_{n+k}$, от которых значения функции фактически не будут зависеть. Для этого достаточно положить, что для всех наборов значений переменных $x_1, x_2, \ldots, x_{n+k}$ справедливо равенство

$$f(x_1, x_2, \ldots, x_n, x_{n+1}, \ldots, x_{n+k}) = f(x_1, x_2, \ldots, x_n).$$

Описанную операцию превращения n-местной функции в $(n+k)$-местную будем называть *операцией формального приписывания аргументов*. Подобная операция применима, очевидно, к произвольным функциям (а не только к булевым).

Как отмечалось выше, имеется точно 2^n булевых набора (кортежей) длины n. Эти наборы можно рассматривать как записи некоторых целых чисел в двоичной системе счисления, так что набор $\alpha_1, \alpha_2 \ldots \alpha_n$ отождествляется с двоичной записью числа $\alpha_1 \cdot 2^{n-1} + \alpha_2 \cdot 2^{n-2} + \ldots + \alpha_{n-1} \cdot 2 + \alpha_n$ (здесь булевы значения 0 и 1 рассматриваются просто как обычные числа 0 и 1). Это число будем называть *номером* соответствующего набора. Номера наборов ме-

няются от нуля (для набора, состоящего из одних нулей), до $2^n - 1$ (для набора, состоящего из одних единиц). Номером набора 010 будет число $0 \cdot 2^2 + 1 \cdot 2 + 0 = 2$, номером набора 101 — число $1 \cdot 2^2 + 0 \cdot 2 + 1 = 5$ и т. д.

Располагая наборы в столбик один за другим в порядке роста их номеров и помещая рядом с каждым набором значение булевой функции на этом наборе, получим *таблицу значений* булевой функции. Поскольку на каждом наборе функция может принять любое из двух значений (0 или 1), независимо от ее значений на остальных наборах, то при m наборах можно определить ровно 2^m различные (отличающиеся друг от друга своими значениями хотя бы на одном наборе) булевы функции. Имея в виду определенное выше общее число наборов для n переменных (равное 2^n), приходим к выводу, что *число различных булевых функций от n аргументов, которое будем обозначать* $B(n)$, *определяется формулой*

$$B(n) = 2^{2^n}. \tag{9}$$

При $n = 1$ величина $B(n)$ равна 4, а при увеличении на 1 эта величина возводится в квадрат: $B(n+1) = (B(n))^2$. Так что если число одноместных булевых функций равно всего 4, то число различных двухместных (булевых) функций будет равно 16, трехместных — 256, четырехместных — $256^2 \approx 65$ тыс., пятиместных — $256^4 \approx 4$ млрд., шестиместных — примерно 16 триллионов ($16 \cdot 10^{18}$) и т. д. Реальные возможности перебора всех булевых функций ограничиваются, таким образом, трехместными или в лучшем случае четырехместными функциями.

Хотя всякая булева функция может быть задана в виде таблицы своих значений, в большинстве случаев практического применения теории булевых функций подобный способ задания оказывается неудобным. Поэтому одной из основных задач наших дальнейших построений будет разработка иных, более удобных способов задания булевых функций. В этой связи особое значение приобретают булевы функции от одного и двух аргументов, поскольку, как будет показано далее, с их помощью можно представлять произвольные булевы функции. Поэтому изучим более подробно одноместные и двухместные булевы функции.

Из четырех одноместных функций $\varphi(x)$, которые вообще возможно построить, две функции представляют собой константы 0 и 1, не зависящие явно от x. Еще одна функция просто повторяет значения своего аргумента $\varphi(x) = x$ и поэтому также не представляет интереса. Последняя, четвертая функция, для которой введем специальное обозначение $\bar{x}$ или $\neg x$, имеет всегда противоположное по сравнению со своим аргументом значение: $\bar{0} = 1$ и $\bar{1} = 0$. Эта функция называется *инверсией*, или *отрицанием*. Выражение $\bar{x}$ (как и выражение $\neg x$) читается как «*отрицание x*», или «*не x*». В теории булевых функций, а также в приложениях этой теории к синтезу схем автоматов, следуя традиции, будем пользоваться обозначением $\bar{x}$. В математической логике (конец

настоящей главы и начало шестой главы), а также в практических аспектах теории алгоритмов (конец пятой главы) по ряду соображений более удобно обозначение $\neg x$.

Из числа 16 различных двухместных булевых функций $f(x, y)$, которые вообще возможно построить, шесть функций сводятся к функциям от меньшего числа аргументов. Это, во-первых, снова две функции константы (0 и 1), во-вторых, — две функции, повторяющие значения какого-нибудь аргумента (x или y), и, в-третьих, — две функции, представляющие собой отрицания каждого из аргументов ($\bar{x}$ и $\bar{y}$).

Десять оставшихся функций $f(x, y)$, действительно зависящих от обоих своих аргументов, можно разбить на пары так, что вторая функция пары представляет собой отрицание первой функции (т. е. имеет на каждом наборе противоположное по сравнению с первой функцией значение). В этом случае фактически используется одноместная булева функция $\bar{x}$ для построения *одноместной операции отрицания* на множестве всех булевых функций. Применение операции отрицания к любой булевой функции g можно трактовать как *подстановку* функции вместо аргумента x в функцию $\bar{x}$. Подобной подстановкой одних булевых функций вместо аргументов других булевых функций (называемой *суперпозицией* этих функций) будем широко пользоваться и далее для образования различных операций на множестве булевых функций (булевых операций). Для обозначения строящихся таким образом операций используют обычно обозначения порождающих эти операции булевых функций. В нашем случае обозначением для отрицания произвольной булевой функции g будет служить $\bar{g}$ (или $\neg g$).

Описанное выше разделение двухместных булевых функций на пары $(g, \bar{g})$ позволяет фактически ограничиться описанием лишь пяти функций, которые выберем в качестве первых элементов указанных пар.

Начнем описание с *конъюнкции*, называемой также (логическим) *произведением*, или *операцией логического «и»*. Конъюнкцию переменных x и y в математической логике принято обозначать $x \,\&\, y$ или $x \wedge y$ (примем второе из этих обозначений). Конъюнкция $x \wedge y$ по определению тогда и только тогда равна единице, когда оба ее аргумента x и y равны единице.

Для равенства конъюнкции $x \wedge y$ нулю достаточно, чтоб хотя бы один из ее аргументов (x или y) обращался в нуль. Указанные свойства конъюнкции совершенно аналогичны всем свойствам, которые имело бы произведение xy, если бы составляющие его сомножители могли принимать лишь два *численных* значения — 0 и 1. Это обстоятельство наводит на мысль считать *булевы постоянные* (0 и 1) своеобразными «псевдочислами», для которых определена операция умножения, обладающая всеми свойствами операции обычного умножения для *чисел* 0 и 1:

$$0 \cdot 0 = 0, \quad 0 \cdot 1 = 0, \quad 1 \cdot 0 = 0, \quad 1 \cdot 1 = 1.$$

В теории булевых функций и в ее приложениях к теории автоматов оказывается удобным принять именно такую точку зрения. Более того, операцию конъюнкции в этих случаях будем просто отождествлять с умножением как по названию, так и по форме записи. Иными словами, вместо обозначения $x \wedge y$ будем употреблять обозначение $x \cdot y$, или xy, а также пользоваться терминами «произведение», «сомножитель» и всеми свойствами умножения из обычной элементарной алгебры. Легко понять, что в силу совпадения определений умножение в нашем случае будет иметь все общие (выполняющиеся тождественно) свойства умножения в обычной алгебре (коммутативность, ассоциативность и т. п.). Вместе с тем ограничение на множество возможных значений величин приводит к появлению у рассматриваемого нами логического умножения таких свойств, которых обычное умножение не имеет. Например, в случае логического умножения тождественное соотношение $x \cdot x = x$ становится неверным, если вместо значений 0 и 1 подставлять в это соотношение другие *численные* значения величины x.

Как и в случае отрицания, умножение (конъюнкцию) можно рассматривать не только как функцию, но и как *операцию* на множестве всех булевых функций. Для этой цели достаточно вместо независимых переменных x и y подставлять в произведение xy две произвольные булевы функции f и g: $p = fg$. Аналогично любая другая двухместная булева функция $b(x, y)$ определяет *двухместную*, или, как обычно принято говорить в алгебре, *бинарную*, *операцию* на множестве всех булевых функций, которую будем называть и обозначать так же, как и соответствующую функцию $b(x, y)$. Разумеется, независимые переменные x и y при этом заменяются произвольными булевыми функциями f и g. В дальнейшем будем пользоваться описанным приемом введения новых бинарных операций на множестве булевых функций без подробных разъяснений.

Возможность интерпретации конъюнкции как обычного умножения наводит на мысль поискать также булев аналог для обычного (численного) сложения. В отличие от умножения полной аналогии здесь быть, разумеется, не может, поскольку равенство $1 + 1 = 2$ в случае обычного сложения вводит третью величину (двойку), отличную как от нуля, так и от единицы. При ограничении лишь булевым (двоичным) алфавитом прямая интерпретация указанного факта, конечно, невозможна. Поэтому можно определить два различных (но неполных) аналога численного сложения для булевых величин, полагая «сумму» двух единиц равной либо единице, либо нулю.

Операция (двухместная булева функция), возникающая при первом допущении, носит название *дизъюнкции*, *логического сложения*, а также *логического* (так называемого *неразделительного*) *«или»*. Для обозначения этой операции фиксируем специальный символ (знак дизъюнкции) $\vee$. Таким образом, дизъюнкция двух величин x и y (независимых переменных или функций) будет обо-

значаться $x \vee y$. Сами величины x и y называются при этом *логическими слагаемыми*, или чаще — *дизъюнктивными членами*.

Система соотношений, полностью определяющих операцию дизъюнкции, записывается в виде $0 \vee 0 = 0$, $0 \vee 1 = 1$, $1 \vee 0 = 1$, $1 \vee 1 = 1$. Первые три соотношения точно такие же, как и в случае обычного (численного) сложения, и лишь четвертое соотношение отличает логическое сложение от обычного. В силу приведенных соотношений дизъюнкция двух величин x и y тогда и только тогда равна нулю, когда обе эти величины обращаются в нуль. Если хотя бы одна из указанных величин принимает значение 1, то это же самое значение 1, независимо от значения другого дизъюнктивного члена, принимает и сама дизъюнкция.

Более удачный аналог обычного (численного) сложения получается в том случае, когда «сумма» двух единиц полагается равной нулю. Возникающую таким образом операцию (двухместную булеву функцию) называют обычно операцией *неравнозначности, разделительного «или»*, а также *сложением по модулю два*. Последнее название связано с тем, что вводимая операция совпадает с определяемым в теории чисел сложением по модулю два, если нуль и единицу рассматривать как обычные числа.

Для краткости будем называть эту операцию просто сложением и употреблять по аналогии с обычным сложением такие термины, как сумма и слагаемые. Для обозначения операции сложения по модулю два будем пользоваться обычным знаком $(+)$. Чтобы подчеркнуть, что речь идет не об обычном сложении, иногда будем обводить этот знак кружком.

Операция сложения булевых величин определяется следующими четырьмя соотношениями: $0 + 0 = 0$, $0 + 1 = 1$, $1 + 0 = 1$, $1 + 1 = 0$. Первые три из них — точно такие же, как и в случае обычного (численного) сложения (и такие же, как в случае логического сложения — дизъюнкции), так что специфика введенной операции определяется прежде всего четвертым соотношением. С тем же самым соотношением связано употребительное в математической логике название для операции сложения — разделительное «или». Если интерпретировать единицу как истину, а нуль — как ложь, то сумма двух булевых величин будет истинна тогда и только тогда, когда истинна *или* первая, *или* вторая величина, но не тогда, когда они *обе* истинны. В случае логической суммы (неразделительного «или») сумма оказывается истинной и тогда, когда истинны оба слагаемые (дизъюнктивные члены) вместе. «Или» в этом случае не исключает одновременной истинности обоих членов, *не разделяет* вопрос об истинности суммы на два исключающих друг друга случая, с чем и связан термин «неразделительное» применительно к «или» в логической сумме (дизъюнкции).

Еще две двухместные булевы функции порождаются одной и той же бинарной операцией, называемой *импликацией*, или *операцией логического следования*. Для обозначения этой операции фиксируем символ $\supset$. Импликация определяется следующими че-

тырьмя соотношениями: $0 \supset 0 = 1$, $0 \supset 1 = 1$, $1 \supset 0 = 0$, $1 \supset 1 = 1$. В импликации $x \supset y$, в отличие от умножения, дизъюнкции и сложения, существенное значение имеет порядок, в котором располагаются ее члены. При изменении этого порядка на обратный значение импликации изменяется так, что $x \supset y$ и $y \supset x$ представляют собой две *различные* булевы функции.

Если обозначать двухместную булеву функцию $f(x, y)$ кортежем $(\alpha_0\alpha_1\alpha_2\alpha_3)$, где α_i — значение, принимаемое этой функцией на наборе с номером i ($i = 0,1, 2, 3$), то импликации $x \supset y$ будет соответствовать кортеж (1101), а импликации $y \supset x$ — кортеж (1011). Заметим одновременно, что произведение, дизъюнкция и сумма переменных x и y, *независимо от порядка*, в котором записаны эти переменные, представляются соответственно кортежами: (0001), (0111) и (0110).

Из рассмотрения всех приведенных кортежей вытекает, кстати, что все пять определенных нами двухместных булевых функций (произведение, дизъюнкция, сумма и две импликации) являются попарно различными. Легко понять, что кортеж для отрицания любой булевой функции получается из кортежа для самой этой функции заменой всех единиц нулями и всех нулей единицами. Применяя это правило, можно определить кортежи для отрицания произведения $\overline{xy}$, отрицания дизъюнкции $\overline{x \vee y}$, отрицания суммы $\overline{x+y}$ и двух отрицаний для импликации $\overline{x \supset y}$ и $\overline{y \supset x}$. Этими кортежами будут соответственно (1110), (1000), (1001), (0010) и (0100).

Легко проверить, что вместе с введенными ранее пятью функциями пять новых функций (отрицаний предыдущих пяти) составляют систему десяти попарно различных двухместных булевых функций. Все они отличаются также от функций-констант 0 и 1 и функций x, y, $\overline{x}$, $\overline{y}$, рассматриваемых как функции двух переменных x и y, поскольку последние функции характеризуются соответственно кортежами (0000), (1111), (0011), (0101), (1100), (1010). Таким образом оказываются перечисленными все шестнадцать двухместных булевых функций, которые вообще можно построить.

Сделаем еще несколько замечаний относительно введенных выше функций. Функция $\overline{xy}$ (отрицание произведения), характеризуемая кортежем (1110), и определяемая ею бинарная операция называются обычно штрихом Шеффера. Легко проверить (используя определения отрицания и дизъюнкции), что штрих Шеффера может быть представлен не только в виде отрицания произведения $\overline{xy}$, но и в виде дизъюнкции отрицаний $\overline{x} \vee \overline{y}$.

Отрицание дизъюнкции $\overline{x \vee y}$ — так называемая *функция Пирса*, — характеризуемое кортежем (1000), может быть представлено также в виде произведения отрицаний переменных x и y, т. е. в виде $\overline{x} \cdot \overline{y}$. Легко видеть, что как штрих Шеффера, так и функция Пирса, подобно произведению, дизъюнкции и сумме, являются

симметричными функциями, т. е., не меняют своих значений при перестановке аргументов.

Подобным же свойством обладает и отрицание суммы $\overline{x+y}$, называемое *операцией равнозначности*, или *логической эквивалентностью*. Для обозначения этой функции, как и для определяемой ею бинарной операции, употребляются специальные символы $\sim$ или $\cong$ (знак эквивалентности). Функция $\overline{x+y} = x \sim y$ характеризуется кортежем (1001). Термины «равнозначность» и «неравнозначность» применительно к функциям $x \sim y$ и $x + y$ соответственно подчеркивают тот факт, что первая функция равна единице тогда и только тогда, когда значения ее аргументов равны между собой, а вторая, — когда значения ее аргументов оказываются неравными. Термин «эквивалентность» имеет тот же смысл, что и термин «равнозначность».

Функция (бинарная операция) импликации может быть выражена дизъюнкцией и отрицанием. Легко проверить, что $x \supset y = \bar{x} \vee y$, а $y \supset x = x \vee \bar{y}$. Отрицание импликации, называемое также *функцией запрета*, легко выражается произведением и отрицанием: $\overline{x \supset y} = x \cdot \bar{y}$, $\overline{y \supset x} = \bar{x} \cdot y$. Как импликация, так и функция запрета представляют собой примеры *несимметричных* булевых функций, поскольку они изменяют свои значения при перестановке аргументов.

Отметим в заключение, что при чтении формул знак конъюнкции $\wedge$ (или &) произносится как «и», знак дизъюнкции $\vee$ — как «или», знак суммы $+$ (или $\oplus$)—как «плюс», знак импликации $\supset$ — как «влечет», знак эквивалентности $\sim$ (или $\cong$) — как «эквивалентно» и знак отрицания (или $\neg$) — как «не».

Всем перечисленным десяти двухместным булевым функциям отвечают соответствующие двухместные булевы операции, которые будем обозначать и называть точно так же, как и определяющие их функции.

§ 2. Булева алгебра

Булевой алгеброй будем называть множество всех (конечноместных) булевых функций, рассматриваемое вместе с заданными на нем операциями отрицания, дизъюнкции и умножения (конъюнкции).

Условимся буквами u, v, w, ... (с индексами или без индексов) обозначать произвольные элементы булевой алгебры, т. е., иными словами, произвольные булевы функции. Одной из основных задач булевой алгебры является установление *тождественных соотношений* вида $A(u, v, w, \ldots) = B(u, v, w, \ldots)$, где через $A(u, v, w, \ldots)$ и $B(u, v, w, \ldots)$ обозначены *формулы*, т. е. выражения булевой алгебры, построенные из конечного числа букв u, v, w, ..., знаков трех операций алгебры, булевых констант (0 и 1) и скобок для обозначения порядка выполнения операций.

Формулы должны быть построены *правильно*. Иными словами, они должны приводиться к вполне определенным булевым функциям после выбора тех или иных булевых функций в качестве значений входящих в эти формулы букв u, v, w, ... Можно дать

строго формальное определение *правильно построенной* формулы, вводимое рекуррентно по правилу: все буквы $u, v, w, \ldots$ (с индексами или без них) и константы 0 и 1 являются правильно построенными формулами. Если A и B — правильно построенные формулы, то $(\bar{A})$, $(A) \vee (B)$ и $(A) \cdot (B)$ — также правильно построенные формулы. Множество правильно построенных формул считается совпадающим с множеством всех формул, которые могут быть получены в результате последовательного (вообще говоря, многократного) применения указанного правила.

Введение каждой дополнительной операции в формулу сопровождается появлением одной или двух пар скобок. Для того чтобы избежать чрезмерной громоздкости формул, несколько расширяют понятие правильно построенной формулы, позволяя опускать некоторые скобки по аналогии с тем, как это делается в школьном курсе алгебры. С этой целью вводят правило о старшинстве операций: первыми при прочих равных условиях должны выполняться отрицания, затем умножения, затем дизъюнкции. При необходимости выполнения действий в ином порядке употребляются скобки. Кроме того, знак отрицания, записываемый чертой над всем выражением, обычно употребляется без скобок, в которые должно было бы быть заключено это выражение. В дальнейшем будет установлено также, что порядок, в котором выполняются одноименные операции, следующие в формуле непосредственно друг за другом, безразличен, так что в этом случае скобки также оказываются излишними и их можно опускать. Напомним, наконец, что знак умножения между буквами может быть опущен.

Все полученные в результате описанных расширений правильно построенные формулы будем в дальнейшем называть просто формулами, допуская употребление в них кроме букв u, v, w любых других букв латинского алфавита.

Существует весьма простое общее правило для проверки правильности тождественных соотношений в булевой алгебре. Суть этого правила состоит в следующем.

Каждая формула $A(u, v, w, \ldots)$ булевой алгебры может рассматриваться как *представление* некоторой булевой функции от переменных $u, v, w, \ldots$. Действительно, если придать этим переменным некоторые постоянные значения ($\bar{0}$ и 1), то, применяя соотношения, определяющие операции отрицания, дизъюнкции и умножения (т. е. соотношения вида $\bar{0} = 1$, $0 \vee 1 = 1$ и т. д.), можно через конечное число шагов найти *значение* (0 или 1) формулы $A(u, v, w, \ldots)$ при выбранных значениях переменных $u, v, w, \ldots$. А это и означает как раз, что наша формула представляет собой некоторую всюду определенную булеву функцию переменных $u, v, w, \ldots$.

Легко понять, что (тождественное) соотношение $A(u, v, w, \ldots) = B(u, v, w, \ldots)$ справедливо в том и только в том случае, когда формулы $A(u, v, w, \ldots)$ и $B(u, v, w, \ldots)$ представляют собой одну и ту же булеву функцию переменных $u, v, w, \ldots$. Для проверки же

факта указанного равенства двух представлений достаточно проверить, совпадают значения этих представлений на всех наборах значений переменных u, v, w, ... или не совпадают.

Тем самым построен общий алгоритм, годный для проверки правильности *любых* тождественных соотношений в булевой алгебре, поскольку ввиду конечности числа наборов значений для любого конечного множества булевых переменных описанная проверка всегда заканчивается через конечное число шагов.

Вместе с тем становится ясным, что тождественные соотношения в булевой алгебре достаточно устанавливать для случая, когда все входящие в эти соотношения буквы рассматриваются как *независимые* (булевы) переменные. В случае же необходимости вместо этих переменных могут быть подставлены любые булевы функции.

Независимые переменные будем обозначать буквами x, y, z (с индексами или без них). Эти же буквы будем употреблять и для записи тождественных соотношений булевой алгебры. Проверку указанных соотношений будем производить с помощью подстановки в них всех возможных наборов значений входящих в эти соотношения переменных (букв).

В качестве примера рассмотрим *соотношение коммутативности для умножения*

$$xy = yx. \tag{10}$$

Чтобы убедиться в правильности этого соотношения, достаточно заметить, что его левая и правая части равны нулю на наборах (00), (01), (10) и равны единице на наборе (11). Ввиду тривиальности подобной проверки, не будем ее повторять в дальнейшем и ограничимся простым выписыванием необходимых нам соотношений, которые будем называть также *законами* или *правилами*.

Кроме соотношения (закона) коммутативности, для умножения существует также так называемый *закон ассоциативности*, выражаемый равенством

$$x(yz) = (xy)z. \tag{11}$$

Умножение удовлетворяет еще одному закону, называемому обычно *законом идемпотентности*,

$$xx = x. \tag{12}$$

В силу этого закона понятия степени и возведения в степень для булевой алгебры фактически не имеют значения.

Законы коммутативности, ассоциативности и идемпотентности распространяются и на операцию дизъюнкции. Соответствующие соотношения записываются

$$x \vee y = y \vee x; \tag{13}$$

$$x \vee (y \vee z) = (x \vee y) \vee z; \tag{14}$$

$$x \vee x = x. \tag{15}$$

Умножение и дизъюнкция связываются между собой первым и вторым дистрибутивными законами, которые могут быть выражены соотношениями

$$x(y \vee z) = xy \vee xz; \tag{16}$$

$$x \vee yz = (x \vee y)\cdot(x \vee z). \tag{17}$$

Заметим, что в силу принятых соглашений о старшинстве операций правая часть соотношения (16) представляет собой упрощение (за счет выбрасывания излишних скобок и знака умножения) формулы $(x\cdot y) \vee (x\cdot z)$, а левая часть соотношения (17) является упрощением формулы $(x) \vee (y\cdot z)$.

Для умножения и дизъюнкции оказываются справедливыми так называемые *правила поглощения*, выражаемые соотношениями

$$x \vee xy = x; \tag{18}$$

$$x(x \vee y) = x. \tag{19}$$

Для операции отрицания существенное значение **имеет** *закон двойного отрицания*

$$\bar{\bar{x}} = x. \tag{20}$$

В силу этого закона любое четное число отрицаний, выполненное подряд, не меняет результата, а любое нечетное число их эквивалентно выполнению одного отрицания.

При различных преобразованиях в булевой алгебре часто приходится пользоваться так называемыми *правилами де Моргана*, связывающими воедино все три операции алгебры,

$$\overline{xy} = \bar{x} \vee \bar{y}; \tag{21}$$

$$\overline{x \vee y} = \bar{x}\cdot\bar{y}. \tag{22}$$

Отметим еще ряд соотношений, включающих константы 0 и 1:

$$x \vee \bar{x} = 1; \tag{23}$$

$$x\bar{x} = 0; \tag{24}$$

$$x\cdot 0 = 0; \tag{25}$$

$$x\cdot 1 = x; \tag{26}$$

$$x \vee 0 = x; \tag{27}$$

$$x \vee 1 = 1; \tag{28}$$

$$\bar{1} = 0; \tag{29}$$

$$\bar{0} = 1. \tag{30}$$

Соотношение (23) называется *законом исключенного третьего*, соотношение (24) — *законом противоречия*. Соотношения (25) и (28) можно рассматривать как частные случаи правил поглощения.

Рассмотрим некоторые следствия из приведенной системы соотношений. Из законов коммутативности и ассоциативности для дизъюнкции и умножения вытекает возможность произвольного порядка выполнения действий при нахождении значений произведения и дизъюнкции для любого конечного числа членов. Отсюда вытекает отмечавшаяся выше возможность записи формул вида $x_1 \vee x_2 \vee \ldots \vee x_n$ и $x_1 x_2 \ldots x_m$ без скобок, не вызывая при этом никаких двусмысленностей, за счет вариаций порядка выполнения действий.

Заметим еще , что, как следует из соотношений (25) и (28), наличия хотя бы одной единицы в дизъюнкции вида $x_1 \vee x_2 \vee \ldots \vee x_n$ достаточно для обращения всей дизъюнкции в единицу, как и наличие хотя бы одного нулевого сомножителя в произведении $x_1 x_2 \ldots x_m$ обращает все это произведение в нуль. В то же время соотношения (26) и (27) показывают, что в любой дизъюнкции могут быть опущены члены, равные нулю, а в любом произведении — члены, равные единице.

В силу соотношения (20) любое количество подряд выполняемых отрицаний сводится либо к одному-единственному отрицанию, либо вообще к отсутствию каких-либо отрицаний. Условимся через $\tilde{x}$ (читается как «x с волной») обозначать выражение, которое может быть равным любому из двух выражений x или $\bar{x}$. Следуя установленному выше правилу для проверки тождественных соотношений в булевой алгебре, формулы, представляющие одну и ту же булеву функцию от входящих в них переменных, будем называть *равными*, или *эквивалентными*, между собой. Хотя равенство или неравенство любых двух формул булевой алгебры может быть в принципе проверено путем перебора всех возможных комбинаций значений входящих в них переменных, такой путь при увеличении числа переменных становится чрезмерно громоздким и практически непригодным. Поэтому одной из основных задач булевой алгебры является разработка более экономных способов установления различного рода соотношений, существующих в этой алгебре.

Для решения указанной задачи можно воспользоваться уже выведенными соотношениями (10) — (30), применяя их многократно и в различных комбинациях. Например, двухкратное применение соотношения (12) позволяет установить справедливость соотношения $xxx = x$, многократное использование соотношений (10) и (13) позволяет распространить законы коммутативности для дизъюнкции и произведения на какое угодно число дизъюнктивных членов и, соответственно, сомножителей. Таким образом, возникает возможность доказывать различные соотношения в булевой алгебре, преобразуя их левые и правые части с помощью соотношений (10) — (30). Если при этом удается *привести* левую

и правую части какого-либо соотношения к одной и той же формуле, то справедливость соответствующего соотношения оказывается тем самым установленной.

Априори неясно, удается ли подобным путем вывести *все* соотношения, существующие в булевой алгебре? Однако в действительности такой вывод оказывается всегда возможным. Для установления этого факта определим некоторый стандартный тип формул, к которому будем стремиться приводить все формулы булевой алгебры. При приведении той или иной формулы A булевой алгебры к стандартному виду всегда будем фиксировать некоторое конечное множество M булевых переменных $x_1, x_2, \ldots, x_n$, включающее обязательно все переменные, которые входят в состав рассматриваемой формулы. Всякое произведение переменных или их отрицаний (т. е. произведение вида $\tilde{x}_{i_1}\tilde{x}_{i_2}\ldots\tilde{x}_{i_k}$) назовем *элементарным произведением*, если в нем каждая буква встречается не более одного раза.

Например, произведения $\bar{x}_1x_2$ или $\bar{x}_1x_2x_3$ элементарны, а произведения $x_1\bar{x}_1$ или $x_3\bar{x}_2x_3$ — неэлементарны. Условимся причислять к элементарным произведениям сами переменные x_i и их отрицания $\bar{x}_i$, рассматривая их как произведения, состоящие из одного сомножителя. Удобно также считать, что константа 1 представляет собой элементарное произведение — произведение нуля (пустого множества) сомножителей. Число сомножителей в произведении называется его *длиной*. Элементарные произведения для выбранного множества M переменных могут иметь, таким образом, любую длину от 0 до n включительно.

Элементарные произведения максимальной длины (в данном случае длины n) принято называть *конституэнтами единицы* для выбранного множества (M) переменных. Легко видеть, что всякая конституэнта единицы содержит все переменные множества M (либо в прямом виде, либо в виде отрицания) в точности по одному разу, а общее число всех таких конституэнт равно 2^n.

Дизъюнкция любого числа элементарных произведений, не содержащая двух одинаковых произведений, называется дизъюнктивной нормальной формой. Дизъюнктивная нормальная форма, состоящая исключительно из конституэнт единицы, называется совершенной дизъюнктивной нормальной формой.

Как и в случае произведений, в приведенном определении не исключается, что дизъюнкция, о которой идет речь, может состоять из одного-единственного члена (дизъюнкция длины 1) и даже из пустого множества членов (дизъюнкция длины 0). В последнем случае дизъюнкция по определению принимается равной нулю. Таким образом, формулы 0, x_1, $x_1 \vee x_2\bar{x}_3$, 1 можно рассматривать как дизъюнктивные нормальные формы. Первая из них состоит из пустого множества членов, вторая — из одного члена, третья — из двух членов, а четвертая — снова из одного члена, представляющего собой элементарное произведение нулевой длины.

Заменив во всех определениях дизъюнкции произведениями, произведения — дизъюнкциями, (булеву) константу 0—(булевой) константой 1 и наоборот, получим соответственно определения *элементарных дизъюнкций, конституэнт нуля, конъюнктивной нормальной формы и совершенной конъюнктивной нормальной формы.*

В булевой алгебре, вследствие того что при замене нуля единицей, а единицы нулем дизъюнкция превращается в конъюнкцию и наоборот, возникает своеобразная двойственность свойств дизъюнции и конъюнкции (умножения). Осуществляя такую замену, можно автоматически для любого выведенного впоследствии свойства (соотношения) получать двойственное ему свойство (соотношение). В частности, всем свойствам дизъюнктивных нормальных форм можно сопоставить, пользуясь указанным законом двойственности, соответствующие им свойства конъюнктивных нормальных форм. Поскольку такое сопоставление осуществляется каждый раз почти автоматически, ограничимся в дальнейшем рассмотрением лишь дизъюнктивных нормальных форм.

Используя соотношения (10), (11), (13) — (16), (23) и (26), можно любую дизъюнктивную нормальную форму преобразовать в эквивалентную ей *совершенную* дизъюнктивную нормальную форму. Рассмотрим процесс такого преобразования на примере дизъюнктивной нормальной формы от трех переменных $x \vee \bar{y}z \vee \bar{x}yz$, которую для краткости будем обозначать одной буквой f.

Третий член этой формулы представляет собой конституэнту единицы и не нуждается поэтому ни в каких преобразованиях. Второму члену для того, чтобы быть конституэнтой единицы, не хватает множителя $\tilde{x}$ (т. е. x или $\bar{x}$), а первому члену — множителей $\tilde{y}$ и $\tilde{z}$. На основании соотношений (23), (26) можем написать, что $f = x(y \vee \bar{y})(z \vee \bar{z}) \vee \bar{y}z(x \vee \bar{x}) \vee \bar{x}yz$. Используя первый дистрибутивный закон (соотношение (16)) и соотношения (10), (11), (13)—(15), последовательно приведем нашу форму к виду $f = (xy \vee x\bar{y})(z \vee \bar{z}) \vee \bar{y}zx \vee \bar{y}z\bar{x} \vee \bar{x}yz = xyz \vee x\bar{y}z \vee xyz \vee \bar{x}yz \vee xyz \vee \bar{x}yz \vee \bar{x}yz = xyz \vee xy\bar{z} \vee x\bar{y}z \vee \bar{x}\bar{y}z \vee \bar{x}yz$. Последнее выражение в этой цепи равенств представляет собой искомую совершенную дизъюнктивную нормальную форму. Установим теперь следующий важный результат.

Теорема 1. *С помощью соотношений* (10) — (30) *любая формула булевой алгебры может быть приведена к совершенной дизъюнктивной нормальной форме.*

В самом деле, используя несколько раз правила де Моргана (21) и (22), закон двойного отрицания (20), а также соотношения (29) и (30), любую формулу $A(x_1, x_2, \ldots, x_n)$ булевой алгебры нетрудно привести к эквивалентной ей формуле $B(x_1, x_2, \ldots, x_n, \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$, не содержащей никаких отрицаний, кроме отрицаний, относящихся непосредственно к буквам $x_1, x_2, \ldots, x_n$. Необходимые здесь преобразования легко уяснить из примера

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

Описанный прием последовательного опускания знаков отрицания применим к любой формуле булевой алгебры.

Формула $B(x_1, x_2, \ldots, x_n\ \bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$ строится из приведенных в ее обозначении букв (с отрицаниями или без них) с помощью одних лишь операций умножения и дизъюнкции. Соотношения же (10), (11), (13), (14) и (16) показывают, что подобные выражения можно точно так же, как в обычном школьном курсе алгебры (считая дизъюнкцию сложением), преобразовать с целью раскрытия всех скобок и группировки всех одинаковых членов. После такого преобразования с последующим учетом соотношений (25), (26) и (27) наша формула превратится в дизъюнкцию некоторых произведений букв $x_1, x_2, \ldots, x_n$ и их отрицаний. С помощью соотношений (10), (12), (24) и (25) все эти произведения могут быть преобразованы в эквивалентные им элементарные произведения или нули. Используя теперь формулы (27) и (15), приведем нашу формулу к совершенной дизъюнктивной нормальной форме. Пример этого был разобран выше. Тем самым теорема полностью доказана.

Ясно, что получившаяся в результате совершенная дизъюнктивная нормальная форма эквивалентна исходной формуле, поскольку на каждом из описанных выше шагов мы пользовались эквивалентными преобразованиями.

Заметим, что все выполненные шаги обратимы, так что с помощью соотношений (10) — (30) может быть осуществлен и *обратный переход* от построенной совершенной дизъюнктивной нормальной формы к исходной формуле $A(x_1, x_2, \ldots, x_n)$.

Теорема 2. *Для произвольной булевой функции f от любого конечного числа переменных $x_1, x_2, \ldots, x_n$ может быть построена одна и с точностью до перестановки дизъюнктивных членов и сомножителей только одна равная ей совершенная дизъюнктивная нормальная форма с тем же самым множеством переменных.*

Каждому набору $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ значений переменных $x_1, x_2, \ldots, x_n$ соответствует точно одна конституэнта единицы $\tilde{x}_1\,\tilde{x}_2 \ldots \tilde{x}_n$, обращающаяся на этом наборе в единицу. Эта конституэнта однозначно определяется условием $\tilde{x}_i = x_i$, если $\alpha_i = 1$, и $\tilde{x}_i = \bar{x}_i$, если $\alpha_i = 0$ $(i = 1, 2, \ldots, n)$. Все остальные конституэнты для данного набора значений переменных имеют нулевые значения. Поскольку в дизъюнкции члены, равные нулю, могут опускаться, то становится ясным, что дизъюнкция g конституэнт единицы, соответствующих всем тем наборам, на которых значения функции f равны единице, представляет собой совершенную дизъюнктивную нормальную форму, равную (как булева функция) функции f. Ясно также, что всякое изменение в составе конституэнт единицы, входящих в форму g, непременно изменит таблицу ее значений и, естественно, нарушит установленное равенство. Следовательно, форма g определяется функцией f однозначно, что и требовалось доказать.

Ввиду указанной однозначности определения форму g принято называть совершенной дизъюнктивной нормальной формой рассматриваемой функции f.

Из теорем 1 и 2 непосредственно вытекают еще два важных результата.

Теорема 3. *Любая булева функция может быть представлена в виде формулы булевой алгебры.*

Теорема 4. *С помощью соотношений* (10) — (30) *всякую формулу булевой алгебры можно преобразовать в любую другую эквивалентную ей (т. е. представляющую ту же самую булеву функцию) формулу.*

Действительно, в качестве формулы, представляющей любую данную булеву функцию f, можно выбрать ее совершенную дизъюнктивную нормальную форму. Преобразовать же какую-либо формулу A в эквивалентную ей формулу B всегда можно через совершенную дизъюнктивную нормальную форму g, которая, в силу теоремы 2, будет общей для формул A и B. Цепочка преобразований, переводящая формулу A в g, и взятая в обратном порядке цепочка, приводящая B к g (такие цепочки существуют в силу теоремы 1), составят цепочку преобразований, переводящих формулу A в формулу B.

Заметим, что в преобразованиях из доказательства теоремы 1 использованы не все из выписанных выше соотношений (10) — (30) (не использовано, например, соотношение (17)). Так что при желании систему соотношений (10) — (30) можно сократить так, что теоремы 2 и 4 будут по-прежнему справедливы.

Второе замечание касается того, что способ преобразования формулы A в эквивалентную ей формулу B через общую для них совершенную дизъюнктивную нормальную форму g нужен был нам лишь для установления принципиальной возможности перехода от A к B. На практике этот способ оказывается обычно слишком громоздким, в результате чего ищут, как правило, более прямые пути перехода от A к B (хотя, разумеется, существенно более короткого пути от A к B, чем указанный выше «окольный» путь, может иногда и не существовать).

Важной задачей, решаемой в рамках булевой алгебры, является *задача минимизации формул*. Смысл этой задачи состоит в том, чтобы найти общий прием (алгоритм), позволяющий для любой формулы булевой алгебры находить эквивалентную ей формулу, имеющую минимальную возможную *сложность*.

Под сложностью формулы наиболее естественно подразумевать количество входящих в эту формулу операций, так что, например, сложностью формулы $\bar{x}$ будет число 1, а сложностью формулы $(\bar{x} \vee y)\ (\bar{y} \vee z)$ — число 5 (два отрицания, две дизъюнкции и одно умножение). Однако, следуя традиции, установившейся в большинстве работ по проблеме минимизации, будем пользоваться другим критерием, подразумевая под сложностью формулы суммарное количество входящих в нее букв. Речь идет здесь о количестве *вхождений* букв (в том числе, быть может, и одинаковых

букв), а не о количестве *различных* букв в формуле. Так что, например, в силу определенного нами критерия сложностью формулы $(x \vee y)(x \vee \bar{y})$ следует считать 4, а не 2.

Нетрудно понять, что множество $M(A)$ различных формул булевой алгебры, сложность которых не превосходит сложности любой фиксированной формулы A, будет непременно конечным. Поэтому сформулированная выше задача минимизации формул может быть в принципе решена за счет перебора всех формул множества $M(A)$ в порядке возрастания их сложности до тех пор, пока не будет найдена формула, эквивалентная формуле A. Однако алгоритм, основанный на указанном переборе, столь громоздок, что практически оказывается непригодным.

Задача построения более экономных алгоритмов минимизации формул в булевой алгебре в общем виде пока еще не решена. Поэтому на практике ограничиваются, как правило, задачей нахождения минимальной формулы в том или ином классе формул и прежде всего *в классе всех дизъюнктивных нормальных форм*. Эта задача называется обычно задачей минимизации булевых функций, что, разумеется, не совсем точно, поскольку речь идет не о минимизации *функции* (остающейся в процессе минимизации неизменной), а о минимизации представляющих ее формул (в данном случае— дизъюнктивных нормальных форм).

Все методы минимизации в классе дизъюнктивных нормальных форм основываются на понятии *простой импликанты. Импликантой* булевой функции f будем называть всякую булеву функцию g, обращение которой в единицу возможно лишь на тех наборах значений переменных, на которых обращается в единицу функция f. Условимся, что импликанта g *накрывает* своими единицами некоторые единицы функции f. Из свойств дизъюнкции вытекает, что дизъюнкция любого (конечного) множества импликант $g_1, g_2, \ldots, g_n$ функции f снова будет ее импликантой. Если при этом единицы импликант $g_1, g_2, \ldots, g_n$, рассматриваемые в *совокупности*, накрывают *все* единицы функции f, то эта дизъюнкция просто совпадает с функцией f: $g_1 \vee g_2 \vee \ldots \vee g_n = f$.

Ясно и обратное: любой член дизъюнкции, совпадающей с функцией f, является импликантой этой функции, а единицы всех членов указанной дизъюнкции в совокупности накрывают все единицы функции f. В частности, всякую дизъюнктивную нормальную форму g булевой функции f можно рассматривать как *накрытие* этой функции множеством всех членов формы g, каждый из которых представляет собой импликанту функции f. В этом случае в роли импликант выступают элементарные произведения.

Заметим, что при уменьшении длины элементарного произведения (за счет отбрасывания части сомножителей) количество накрываемых им единиц увеличивается. Элементарное произведение максимальной длины (конституэнта единицы) для n переменных обращается в единицу лишь в одной точке, а элементарное произведение длины $n - k$ — в 2^k точках. Выгодно поэтому на-

крывать любую заданную функцию f элементарными произведениями минимально возможной длины, т. е. такими элементарными произведениями, что сами они являются импликантами функции f, и никакие их собственные части импликантами этой функции уже не являются. Подобные элементарные произведения принято называть *простыми импликантами* рассматриваемой булевой функции.

Множество всех простых импликант любой булевой функции f накрывает все ее единицы. Поэтому дизъюнкция g всех простых импликант функции f, называемая *сокращенной дизъюнктивной нормальной формой* этой функции, является одним из возможных ее представлений. Однако такое представление обычно не бывает самым экономным, поскольку некоторые простые импликанты могут накрывать единицы, уже накрытые остальными импликантами. Выбрасывая из формы g *все* подобные *лишние* импликанты, превращаем ее в так называемую *тупиковую дизъюнктивную нормальную форму* рассматриваемой функции f.

Булева функция может иметь, вообще говоря, не одну, а несколько тупиковых дизъюнктивных нормальных форм. Например, функция трех переменных x, y, z, обращающаяся в нуль только на наборах (000) и (111), а на всех остальных наборах равная единице, имеет пять различных тупиковых дизъюнктивных нормальных форм. В то же время можно показать, что любая двухместная булева функция имеет единственную тупиковую дизъюнктивную нормальную форму.

Легко понять, что среди тупиковых дизъюнктивных нормальных форм любой булевой функции f обязательно содержатся все ее *минимальные* дизъюнктивные нормальные формы (их может быть несколько), т. е. такие дизъюнктивные нормальные формы функции f, которые содержат наименьшее число букв по сравнению со всеми остальными дизъюнктивными нормальными формами этой функции.

Можно построить достаточно экономные алгоритмы для нахождения всех простых импликант и всех тупиковых дизъюнктивных нормальных форм любой булевой функции. Однако для выделения из числа тупиковых дизъюнктивных нормальных форм минимальных форм нет в общем случае существенно более простого метода, чем метод последовательного перебора и сравнения всех тупиковых дизъюнктивных нормальных форм (см. Ю. И. Журавлев [36]).

Одним из наиболее эффективных алгоритмов нахождения простых импликант и тупиковых дизъюнктивных нормальных форм является алгоритм, предложенный Блейком [8]. Суть алгоритма Блейка заключается в следующем. Нетрудно установить, что в булевой алгебре выполняется тождественное соотношение вида

$$AB \vee \bar{A}C = AB \vee \bar{A}C \vee BC. \tag{31}$$

Если в этом соотношении считать A буквой, а B и C — эле-

ментарными произведениями, то из соотношения (31) выводится правило тождественного преобразования дизъюнктивных нормальных форм, позволяющее при наличии в них двух членов вида xp и $\bar{x}q$ дополнять их новым членом (элементарным произведением) pq. Возможно, правда, что этот член обратится в нуль или совпадет с одним из имевшихся уже в форме дизъюнктивных членов. Легко понять, что ввиду конечности общего числа элементарных произведений (заданных переменных) через конечное число шагов применения указанного правила новые члены получаться уже не будут. Результат Блейка состоит в том, что *преобразуемая нами форма f после достижения подобной «стабилизации» будет содержать все простые импликанты представляемой ею булевой функции.*

После получения дизъюнктивной нормальной формы g, содержащей все свои простые импликанты, ее нетрудно освободить от всех членов, не являющихся простыми импликантами. Действительно, если какое-либо элементарное произведение P из g не является простой импликантой, то, будучи во всяком случае импликантой функции g, оно включает в себя некоторую простую импликанту p этой функции и, следовательно, может быть представлено в виде $P=pq$. Поскольку в q есть дизъюнктивный член p, то его можно употребить для исключения из q члена $P=pq$ с помощью соотношения (18): $p \vee pq = p$. Подобное исключение называют обычно операцией *элементарного поглощения.* Ее применение к дизъюнктивной нормальной форме g обеспечит через конечное число шагов уничтожение всех членов, не являющихся простыми импликантами, и превращение, таким образом, формы g в сокращенную дизъюнктивную нормальную форму g_0.

Чтобы перейти от формы g_0 к какой-нибудь тупиковой дизъюнктивной нормальной форме, можно найти лишние члены в g_0 тем же методом Блейка: если некоторый член в форме g_0 (или в любой другой дизъюнктивной нормальной форме, состоящей из простых импликант) может быть получен с помощью применения (быть может, неоднократного) соотношения (18) из остальных членов, то этот член является лишним и его можно исключить.

Применяя процесс подобного исключения многократно, приведем форму g_0 к какой-нибудь тупиковой дизъюнктивной нормальной форме g_1. Действительно, в силу приведенного выше результата Блейка, с помощью соотношения (30) из формы g_1 все еще можно получить все простые импликанты, входящие в g_0. Дальнейшее же исключение членов формы g_1 оказывается невозможным. В самом деле, при попытке такого исключения хотя бы одна из единиц функции g_1 окажется ненакрытой. Ясно, что накрывавшая эту единицу простая импликанта (исключенная из g_1) уже не может быть восстановлена из дизъюнкции оставшихся членов *никакими* эквивалентными преобразованиями, в частности и с помощью применения тождества (31).

Для получения всех тупиковых дизъюнктивных нормальных форм описанный способ исключения следует применить несколько раз с изменением порядка, в котором совершаются попытки исключения различных членов. Как отмечалось выше, нахождение минимальных дизъюнктивных нормальных форм требует громоздкой работы по перебору всех тупиковых дизъюнктивных нормальных форм (которые могут быть различной сложности). Поэтому на практике обычно решение задачи минимизации ограничивают нахождением какой-нибудь одной, наугад выбираемой тупиковой дизъюнктивной нормальной формы.

В качестве примера применения алгоритма Блейка покажем процесс минимизации дизъюнктивной нормальной формы $f = \bar{x}yz \vee \vee xy \vee x\bar{y}\bar{z}$.

Применяя соотношение (31) к парам, составленным из первого члена со вторым и из второго члена с третьим, приведем заданную форму f к виду $f_1 = \bar{x}yz \vee xy \vee x\bar{y}\bar{z} \vee yz \vee x\bar{z}$. Применение соотношения (31) к любым парам членов формы f_1 не приводит к появлению новых членов. Следовательно, в форме f_1 содержатся все простые импликанты функции f (функции, представляемой формой f).

Применение операции элементарного поглощения к форме f_1 приводит к сокращенной дизъюнктивной нормальной форме $f_2 = xy \vee yz \vee x\bar{z}$. Первый член формы f_2 может быть получен с помощью соотношения (31) из остальных двух членов и является, таким образом, лишним. Исключая его, приходим к форме $f_3 = yz \vee \vee x\bar{z}$, уже не содержащей лишних членов и являющейся, следовательно, искомой тупиковой дизъюнктивной нормальной формой. В данном случае тупиковая дизъюнктивная нормальная форма является единственной и совпадает в силу этого с минимальной дизъюнктивной нормальной формой.

Более подробные сведения о различных способах минимизации формул булевой алгебры можно получить в специальных монографиях по теории синтеза схем дискретных автоматов (см., например, В. М. Глушков [26]). Некоторые дополнительные сведения по этому вопросу приведены также в § 4 настоящей главы.

§ 3. Понятие о полных наборах булевых операций

Теорема 3 предыдущего параграфа показывает, что для представления любой булевой функции в виде формулы, построенной из аргументов и булевых констант 0 и 1, достаточно использовать всего три типа булевых операций — отрицание, умножение и дизъюнкцию. Всякий набор булевых операций, обладающих подобным свойством, условимся называть *полным набором*.

Кроме набора, состоящего из операций отрицания, умножения и дизъюнкции, можно построить и другие полные наборы булевых операций. Из выписанных в предыдущем параграфе соотношений де Моргана (21) и (22) вытекает, что операция дизъюнкции может быть выражена операциями отрицания и умножения, а

операция умножения — операциями отрицания и дизъюнкции. Поэтому полный набор булевых операций может быть составлен из операции отрицания и любой из двух остальных операций булевой алгебры (умножения или дизъюнкции).

Из операций любого полного набора булевых операций должны строиться какие угодно булевы операции, в частности операции отрицания, дизъюнкции и умножения. Для того чтобы выполнить требуемое построение, достаточно, очевидно, с помощью операций рассматриваемого полного набора представить булеву функцию, определяющую требуемую операцию. Наоборот, если из операций некоторого набора можно построить операции отрицания и умножения или отрицания и дизъюнкции, то, ввиду сказанного выше, этого достаточно для возможности представления любой булевой функции и, следовательно, для полноты нашего набора. В результате приходим к следующему предложению.

Теорема 1. *Для полноты любого набора булевых операций необходимо и достаточно, чтобы с помощью операций этого набора можно было построить функции* $\bar{x}$ *и одну из функций* xy *или* $x \vee y$.

Используя критерий полноты по теореме 1, можно относительно просто установить полноту многих наборов булевых операций. Одним из таких наборов является, в частности, набор, состоящий из операций *умножения и сложения* (по модулю два). Действительно, легко проверить, что справедливо соотношение

$$\bar{x} = x + 1. \tag{32}$$

Таким образом, отрицание может быть выражено сложением. Поскольку умножение само входит в рассматриваемый набор, то на основании теоремы 1 приходим к заключению о полноте этого набора.

С помощью операций сложения и умножения строится еще одна интересная алгебра булевых функций, называемая алгеброй Жегалкина. Эта алгебра по своим общим свойствам (выражаемым тождественными соотношениями) наиболее близко подходит к алгебре с обычными действиями сложения и умножения, изучающейся в средней школе. Подобно обычному сложению, сложение по модулю два удовлетворяет соотношениям ассоциативности и коммутативности (для булева умножения эти свойства были установлены в предыдущем параграфе). Выполняется также дистрибутивный закон $x(y + z) = xy + xz$, позволяющий раскрывать скобки в выражениях точно так же, как в обычной алгебре.

После раскрытия скобок любая формула в алгебре Жегалкина представляется в виде суммы произведений переменных, включая, возможно, произведения, состоящие из одного сомножителя (одиночные буквы) и из нуля сомножителей (константа 1). На основании соотношения $xx = x$ и коммутативности умножения можно считать, что в любое из полученных произведений никакая буква не будет входить более одного раза.

Одинаковые произведения, как и в обычной алгебре, считаются подобными членами и подвергаются операции приведения подобных. Правила такого приведения отличны от соответствующих правил в обычной алгебре, сводясь, в конечном счете, к легко проверяемому тождественному соотношению

$$x + x = 0. \tag{33}$$

Таким образом, любое четное число одинаковых слагаемых взаимно уничтожается, а любое нечетное их число оказывается эквивалентным всего лишь одному слагаемому, поскольку нулевые слагаемые не изменяют величины суммы и могут быть безболезненно вычеркнуты из нее.

Приведением подобных заканчивается описанный нами процесс приведения, который будем называть процессом приведения формул в алгебре Жегалкина к *каноническому виду*. Продемонстрируем этот процесс на примере. Пусть дана некоторая формула $f = (x + y)(x + z) + y(z + x)$ алгебры Жегалкина. После раскрытия скобок эта формула примет вид $f_1 = x + xy + xz + yz + yz + xy$. После приведения подобных члены xy и yz, встречающиеся в формуле дважды, взаимно уничтожаются, а сама формула приводится к окончательному (каноническому) виду $f_2 = x + xz$.

Ввиду полноты набора операций алгебры Жегалкина и возможности приведения к каноническому виду любой ее формулы, всякая булева функция f может быть представлена в алгебре Жегалкина формулой канонического вида. Нетрудно показать, что последняя формула с точностью до возможной перестановки слагаемых и сомножителей определяется функцией f однозначно. Будем называть эту формулу *каноническим полиномом* заданной булевой функции f.

Однозначность определения канонического полинома может быть установлена простым рассуждением. Пусть f_1 и f_2 — два различных канонических полинома булевой функции f. Будучи равными этой функции, полиномы f_1 и f_2 равны между собой как функции (для всех значений переменных). В равенстве $f_1 = f_2$ одинаковые члены в правой и левой частях могут быть взаимно уничтожены. В правой и левой частях возникающего после этого тождественного соотношения $f_1' = f_2'$ нет ни одной пары одинаковых членов (слагаемых).

Зафиксируем одно из слагаемых p, составленное из наименьшего по сравнению с остальными слагаемыми числа букв. Тогда все остальные слагаемые будут отличаться от выбранного слагаемого p хотя бы одной буквой. Фиксируем набор значений переменных так, что все буквы, входящие в p, получают значение 1, а все остальные буквы — значение 0. В силу этого замечания только одно слагаемое, а именно слагаемое p, обратится при выбранном наборе значений переменных в единицу, все же остальные

слагаемые будут равны нулю. Но тогда соотношение $f'_1 = f'_2$ приведется к виду $1 = 0$ (или $0 = 1$), что невозможно, если исходное соотношение $f_1 = f_2$ было тождественным. Тем самым опровергнуто сделанное в начале наших рассуждений предположение о существовании двух различных (хотя и равных друг другу как функций) канонических полиномов f_1 и f_2 у одной и той же булевой функции f.

Канонические полиномы, не содержащие произведений двух и более переменных (т. е. полиномы, представляющиеся суммой отдельных букв и, возможно, константы 1, а также полином, тождественно равный нулю), являются так называемыми *линейными булевыми функциями*. Все остальные булевы функции относятся к числу *нелинейных*. Соответственно указанному делению функций все определяемые ими булевы операции также разделяются на линейные и нелинейные операции.

Легко видеть, что при любых суперпозициях (подстановках друг в друга) линейных булевых функций возникающие в результате суперпозиции функции будут снова линейными. Это означает, очевидно, что *с помощью линейных (булевых) операций не может быть построена никакая нелинейная операция. Отсюда следует, что всякий полный набор булевых операций должен включать в себя хотя бы одну нелинейную операцию.*

Операции отрицания и сложения (по модулю два) являются линейными операциями, поскольку каноническими полиномами представляющих их булевых функций будут линейные формулы $x + 1$ и $x + y$. В то же время функции xy и $x \vee y$, а значит, и определяемые ими операции умножения и дизъюнкции нелинейны. Первая из них имеет в качестве своего канонического полинома формулу xy, а вторая — формулу $x + y + xy$. Обе эти формулы содержат нелинейный член xy.

Введем еще одно деление булевых функций и соответствующих им булевых операций на два класса: *класс монотонных функций* (*операций*) и класс *немонотонных функций* (*операций*). С этой целью определим для наборов значений булевых переменных отношение порядка $\leqslant$, полагая, что $0 \leqslant 0$, $0 \leqslant 1$, $1 \leqslant 1$ и что для любых двух наборов одинаковой длины $(\alpha_1\alpha_2 \ldots \alpha_n)$ и $(\beta_1\beta_2 \ldots \beta_n)$ отношение $(\alpha_1\alpha_2 \ldots \alpha_n) \leqslant (\beta_1\beta_2 \ldots \beta_n)$ справедливо тогда и только тогда, когда для *всех* $i = 1, 2, \ldots, n$ $\alpha_i \leqslant \beta_i$. Если эти наборы различны, то будем говорить, что первый из них *меньше*, а второй — *больше*. Заметим, что некоторые наборы, например наборы (01) и (10), будут при этом *несравнимыми* между собой, поскольку приведенное определение не позволяет считать, что один из них больше или меньше другого.

Булева функция f называется *монотонной*, если при переходе от любого меньшего набора A значений ее переменных к любому большему (по сравнению с A) набору B значение функции не может уменьшаться, т. е. переходить от значения 1 на наборе A к значению 0 на наборе B. Если же хотя бы для одной пары наборов A, B,

таких, что $A \leqslant B$, $f(A) = 1$, а $f(B) = 0$, то функция f называется *немонотонной* булевой функцией. Соответственно делятся на монотонные и немонотонные определяемые этими функциями булевы операции.

При любых суперпозициях (подстановках функции в функцию) монотонных булевых функций получаются снова монотонные функции. Действительно, если функции $f(y_1, y_2, \ldots, y_m)$ и $\varphi(x_1, x_2, \ldots, x_n)$ монотонны, а функция φ подставляется, скажем, на место переменной y_1, то, в силу монотонности функции φ при любом увеличении набора значений переменных $y_2, y_3, \ldots, y_m, x_1, x_2, \ldots, x_n$, набор значений переменных $\varphi, y_2, y_3, \ldots, y_m$ либо увеличится, либо останется неизменным. В обоих случаях значение сложной функции $f(\varphi, y_2, y_3, \ldots, y_m)$, в силу монотонности функции $f(y_1, y_2, \ldots, y_m)$, не может уменьшаться, чем и доказывается ее монотонность.

При переходе к операциям установленный факт означает, что *с помощью одних лишь монотонных булевых операций не может быть построена никакая немонотонная булева операция. Ввиду существования немонотонных булевых операций (например, операции отрицания) отсюда следует, что всякий полный набор булевых операций должен обязательно включать в себя хотя бы одну немонотонную операцию.*

Наиболее простым примером немонотонной операции является операция отрицания. Оказывается также, что в известном смысле всякая немонотонная операция содержит в себе операцию отрицания. Более точно, справедливо следующее предложение.

Теорема 2. *С помощью любой немонотонной булевой операции может быть построена операция отрицания.*

Рассмотрим произвольную немонотонную булеву операцию. Она определяется некоторой немонотонной булевой функцией $f(x_1, x_2, \ldots, x_n)$. Ввиду немонотонности функции f найдутся два набора A и B значений ее переменных таких, что A меньше, чем B, а функция f принимает на наборе A значение 1 и на наборе B значение 0. Набор A отличается от набора B тем, что на некоторых из мест, где у набора B стоят единицы, у набора A стоят нули. Заменяя последовательно, один за другим, эти нули единицами, рано или поздно придем от набора A, где $f(A) = 1$, к набору B, где $f(B) = 0$. Следовательно, при одной из очередных замен нуля единицей значение функции f должно измениться с 1 на 0. Это означает, что для какого-то i $(1 \leqslant i \leqslant n)$ $f(\alpha_1, \alpha_2, \ldots, \alpha_{i-1}, 0, \alpha_{i+1}, \ldots, \alpha_n) = 1$, а $f(\alpha_1, \alpha_2, \ldots, \alpha_{i-1}, 1, \alpha_{i+1}, \ldots, \alpha_n) = 0$, где $\alpha_1, \alpha_2, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_n$—некоторые булевы константы (0 или 1). Но тогда, как легко понять, булева функция $f(\alpha_1, \alpha_2, \ldots, \alpha_{i-1}, x, \alpha_{i+1}, \ldots, \alpha_n)$ от одной переменной x не может быть ничем иным, как только отрицанием этой переменной, т. е. $\bar{x}$. Интерпретируя функцию f как булеву операцию, получим требуемое представление с помощью этой операции отрицания.

При классификации булевых операций будем исключать из рассмотрения нульместные операции (константы 0 и 1), а также тривиальную одноместную операцию, повторяющую значения своего аргумента. Естественно также не различать между собой операции, возникающие из одной и той же булевой функции при различных перестановках ее аргументов. Учитывая это, *будем иметь единственную одноместную операцию — отрицание* $\bar{x}$ *и восемь двухместных операций — умножение* xy, *дизъюнкцию* $x \vee y$, *сложение* $x + y$, *операцию равнозначности* $x \sim y$, *импликацию* $x \supset y$, *операцию запрета* $\bar{x}y$, *операцию Шеффера* $\bar{x} \vee \bar{y}$ *и операцию Пирса* $\bar{x}\bar{y}$.

Легко проверить, что монотонными из всех перечисленных девяти булевых операций являются только две — операции умножения и дизъюнкции. К линейным относятся лишь три операции — операции отрицания, сложения и равнозначности. Таким образом, приходим к следующему результату.

Теорема 3. *Среди девяти одноместных и двухместных булевых операций операции умножения и дизъюнкции являются нелинейными (но монотонными), операции отрицания, сложения и равнозначности являются немонотонными (но линейными). Остальные же четыре операции — операции запрета, импликации, Шеффера и Пирса — являются одновременно и нелинейными, и немонотонными.*

Из теорем 2 и 3 нетрудно вывести следующий важный результат.

Теорема 4. *С помощью любой нелинейной операции может быть получена либо операция умножения, либо операция дизъюнкции.*

Рассмотрим произвольную нелинейную операцию, задаваемую некоторой нелинейной булевой функцией $f(x_1, x_2, \ldots, x_n)$. Канонический полином этой функции содержит хотя бы одно произведение с двумя или более сомножителями. Выделим среди всех таких произведений одно из тех, которые имеют наименьшую длину l. Это произведение содержит не менее двух сомножителей и, следовательно, имеет вид $x_i x_j p$, где p — произведение некоторого множества букв (быть может, пустого или содержащего лишь одну букву). Сохраняя буквы x_i и x_j неизменными, заменим все буквы, входящие в p, единицами, а все остальные (из числа букв $x_1, x_2, \ldots, x_n$) — нулями. После такой подстановки выделенное нами произведение обратится в $x_i x_j$, а все остальные произведения длины, большей единицы, обратятся в нули, поскольку каждое из них содержит хотя бы одну букву, отличную от x_i, x_j и букв, входящих в p.

После указанной подстановки получим булеву функцию двух переменных $\varphi(x_i, x_j)$, канонический полином которой имеет вид $x_i x_j + \alpha x_i + \beta x_j + \gamma$, где α, β, γ — булевы константы (0 или 1). Если эта функция равна $x_i x_j$ или $x_i \vee x_j = x_i x_j + x_i + x_j$, то теорема доказана. В противном случае, будучи нелинейной, эта функция, в силу теоремы 3, будет обязательно немонотонной. Но тогда, по теореме 2, с помощью определяемой функцией φ булевой

операции можно выразить отрицание $\bar{x} = x + 1$. Располагая функциями x_i, x_j, $x_i + 1$, $x_j + 1$, можем в функции φ произвести замену x_i на $x_i + \beta$, а x_j — на $x_j + \alpha$, после чего получим функцию $\psi(x_i, x_j)$ с каноническим полиномом $(x_i + \beta)(x_j + \alpha) + \alpha(x_i + \beta) + \beta(x_j + \alpha) + \gamma = x_i x_j + \alpha x_i + \beta x_j + \alpha\beta + \alpha x_i + \alpha\beta + \beta x_j + \alpha\beta + \gamma = x_i x_j + \delta$, где буквой δ обозначена булева константа $\alpha\beta + \gamma$. Если $\delta = 0$, то $\psi(x_i, x_j) = x_i x_j$, и теорема доказана. Если же $\delta = 1$, то $\psi(x_i, x_j) = x_i x_j + 1 = \overline{x_i x_j}$. Поскольку отрицание нами уже построено, из последней функции снова легко получается произведение $x_i x_j$.

Таким образом, во всех случаях удается с помощью заданной операции построить выражения для функции xy или $x \vee y$, а следовательно, и для определяемых ими операций, что и требовалось доказать. Теперь легко вывести следующее условие полноты для наборов булевых операций.

Теорема 5. *Для того чтобы набор булевых операций был полным, необходимо и достаточно, чтобы в состав этого набора входила хотя бы одна нелинейная и хотя бы одна немонотонная операция.*

Необходимость сформулированного условия была установлена выше, а достаточность является непосредственным следствием теорем 2 и 4.

Условимся называть полный набор булевых операций *неприводимым*, если из него нельзя исключить ни одной операции без того, чтобы набор не потерял свойство полноты. Теорема 5 позволяет легко перечислить все неприводимые полные наборы, составленные из одноместных и двухместных булевых операций. Это — четыре набора, каждый из которых состоит из одной-единственной операции (импликации, запрета, операции Шеффера и операции Пирса), и шесть полных неприводимых наборов, состоящих из двух операций: комбинации операции умножения с каждой из операций отрицания, сложения или равнозначности и комбинации операции дизъюнкции с каждой из тех же самых трех операций.

Использованное нами понятие полноты разрешало при построении булевых функций пользоваться не только аргументами этих функций и операциями из соответствующего полного набора, но и булевыми константами 0 и 1. Если исключить возможность пользования константами, то возникает новое понятие полноты, которое будем называть *усиленной полнотой*, или *полнотой в сильном смысле*.

Далеко не все полные наборы булевых операций удовлетворяют условию усиленной полноты. Например, набор, состоящий из операций сложения и умножения, являясь полным, не является, тем не менее, полным в сильном смысле. Легко видеть, что без пользования константой 1 все строящиеся с помощью этого набора операций булевы функции непременно обращаются в нуль в точке, в которой все их аргументы принимают нулевые значения.

Таким образом, с помощью одних лишь операций сложения и умножения (без константы 1) не может быть представлен целый ряд булевых функций, например функция $\bar{x}$ или функция, тождественно равная единице.

В то же время наборы, составленные из операций отрицания и умножения или отрицания и дизъюнкции, оказываются полными не только в обычном, но и в сильном смысле. Для того чтобы убедиться в этом, достаточно, очевидно, доказать возможность представления с помощью указанных операций констант 0 и 1. Последнее же дается формулами $0 = x\bar{x}$, $1 = \overline{x\bar{x}}$, $1 = x \vee \bar{x}$, $0 = \overline{x \vee \bar{x}}$.

Необходимые и достаточные условия усиленной полноты для наборов булевых операций найдены Постом [62]. Чтобы сформулировать эти условия, необходимо познакомиться с тремя новыми замечательными классами булевых функций и определяемых ими операций.

Булева функция (операция) $f(x_1, x_2, \ldots, x_n)$ называется *функцией* (операцией), *сохраняющей нуль*, *если* $f(0, 0, \ldots, 0) = 0$, *функцией* (*операцией*), *сохраняющей единицу*, если $f(1, 1, \ldots, 1) = 1$, и *самодвойственной функцией* (операцией), если $f(x_1, x_2, \ldots, x_n) = \overline{f(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)}$. Упомянутый выше результат Поста можно сформулировать теперь так.

Теорема 6. *Для того чтобы набор булевых операций был полным в усиленном смысле, необходимо и достаточно, чтобы этот набор включал в себя хотя бы одну нелинейную операцию, хотя бы одну немонотонную операцию, хотя бы одну операцию, не сохраняющую нуль, хотя бы одну операцию, не сохраняющую единицу, и хотя бы одну операцию, не являющуюся самодвойственной.*

Необходимость условий, сформулированных в теореме 6, доказывается точно таким же способом, как и в случае обычной полноты: необходимо убедиться лишь (а это сделать нетрудно), что, не используя констант, с помощью операций, сохраняющих нуль, можно построить только такие булевы функции (а значит, и булевы операции), которые также сохраняют нуль. Аналогичное положение будет также с операциями, сохраняющими единицу, и с самодвойственными операциями. Доказательство достаточности сводится к установлению возможности построения констант 0 и 1 и последующему применению теоремы 5. С подробностями этого доказательства можно познакомиться в статье С. В. Яблонского [83] (см. также В. М. Глушков [26]).

Из перечисленных выше девяти одноместных и двухместных булевых операций не сохраняют нуль шесть операций: отрицание, операция равнозначности, импликация, а также операции Шеффера и Пирса.

К числу операций, не сохраняющих единицу, также относятся шесть операций: отрицание, сложение, операции запрета, Шеффера и Пирса.

Наконец, несамодвойственными являются все операции, кроме

отрицания: умножение, дизъюнкция, сложение, импликация, а также операции равнозначности, запрета, Шеффера и Пирса.

Наиболее замечательным свойством обладают операции Шеффера и Пирса: каждая из них, рассматриваемая в отдельности, представляет собой полный в сильном смысле набор булевых операций. Эти наборы, разумеется, неприводимы в том смысле, что из них нельзя удалить ни одной операции без потери набором свойства усиленной полноты.

Легко показать, что всякая операция, не сохраняющая нуль, либо не сохраняет и единицу, либо является несамодвойственной операцией. Отсюда следует, что в любом неприводимом усиленно полном наборе булевых операций не может быть более четырех (а не пяти, как могло бы показаться априори) различных операций, причем неприводимые усиленно полные наборы, состоящие из четырех различных булевых операций, действительно существуют.

§ 4. Применения булевой алгебры в теории комбинационных схем

Комбинационные схемы представляют собой наиболее простые технические устройства для преобразования дискретной информации. Предположим, что в нашем распоряжении конечное число типов сигналов той или иной природы (механической, электрической, оптической и т. п.), составляющих так *называемый алфавит сигналов* S. *Комбинационной схемой* будем называть любое устройство P, реализующее некоторый алфавитный оператор $A = A(S)$ в алфавите S и удовлетворяющее следующим условиям.

1. Областью определения оператора A является множество слов в алфавите S, имеющих фиксированную длину $m \geqslant 1$ (зависящую от выбора устройства P).

2. Все входные слова из области определения оператора A преобразуются схемой P в выходные слова одной и той же длины $n \geqslant 1$ (также зависящей от выбора P).

3. Все буквы (сигналы), составляющие входное слово, подаются *одновременно* на m точек схемы P, называемых ее *входными полюсами*, и *в то же время*, также одновременно, все буквы (сигналы) соответствующего выходного слова возникают в других n точках схемы, называемых ее *выходными полюсами*. Входные и выходные полюсы строго фиксированным способом нумеруются и сопоставляются соответствующим местам входного и выходного слов. Так, что i-я буква входного слова всегда подается на i-й входной полюс ($i = 1, 2, \ldots, m$), а j-я буква выходного слова возникает на j-ом выходном полюсе $j = 1, 2, \ldots, n$.

Разумеется, всякое реальное техническое устройство обладает некоторым внутренним запаздыванием, так что условие одновременности возникновения входных и выходных сигналов в комбинационной схеме не следует понимать слишком буквально. Речь идет о некоторой абстракции того реально встречающегося случая, когда указанным запаздыванием можно пренебречь по сравне-

нию с интервалом дискретности работы схемы, определяемым временем замены одного входного слова другим.

На практике комбинационные схемы характеризуются обычно *отсутствием в них памяти*. Это означает, что выходное слово возникает на выходных полюсах схемы лишь на то время, пока на входные полюсы подается соответствующее входное слово. После того как подача входных сигналов прекращается, схема «забывает» эти сигналы, так что они не могут влиять на процесс формирования ответа схемы на следующую комбинацию сигналов, подаваемую на ее входные полюсы.

Условия 1 и 2 накладывают, на первый взгляд, весьма сильные ограничения на алфавитные операторы, которые можно реализовать комбинационными схемами. В действительности же словами одной и той же длины (подбираемой каждый раз в соответствии с конкретными условиями) можно закодировать любую конечную совокупность слов.

Таким образом, при подходящем кодировании комбинационные схемы могут реализовать любые алфавитные операторы с конечными областями определения.

Наиболее простой прием выравнивания длин любого фиксированного множества слов за счет кодирования состоит в приписывании (вообще говоря, многократном) к словам меньшей длины специально вводимой в алфавит пустой буквы с целью доведения числа составляющих их букв до числа букв, составляющих самое длинное слово рассматриваемого множества. Возможны, разумеется, и другие приемы решения этой задачи.

Заметим еще, что при надлежащей трактовке работы комбинационных схем можно считать, что одна и та же комбинационная схема способна реализовать не один, а любое конечное множество алфавитных операторов. С этой целью достаточно разделить все входные полюсы схемы на так называемые *информационные* и *управляющие* полюсы. Если считать преобразуемым входным словом лишь ту комбинацию входных сигналов, которая подается на информационные полюсы, то, фиксируя различные *управляющие слова* (т. е. слова, подаваемые на управляющие полюсы), будем получать различные алфавитные операторы, сопоставляющие *информационным входным словам* выходные слова схемы.

Прием вариации реализуемых комбинационной схемой алфавитных операторов с помощью управляющих слов совершенно аналогичен описанному в первой главе приему организации работы универсального алгоритма: на вход универсального алгоритма подается не только подлежащее переработке информационное слово, но и управляющее слово, в качестве которого выбирается изображение подлежащего реализации конкретного алгоритма.

По техническим соображениям оказывается более простым и удобным в качестве алфавита сигналов выбирать *двоичный* алфавит. В этом случае два типа сигналов отождествляют обычно с булевыми константами 0 и 1. Комбинационные схемы с таким алфавитом

сигналов будем называть *двоичными*, или *булевыми*, *комбинационными схемами*.

В двоичной комбинационной схеме каждый выходной сигнал представляет собой некоторую булеву функцию от входных сигналов схемы. Если схема имеет m входных и n выходных полюсов, то реализуемый ею алфавитный оператор полностью характеризуется системой n булевых функций от m переменных, задающих выходные сигналы на каждом из n выходных полюсов как функции сигналов на m входных полюсах. Эту систему функций будем называть *выходными функциями* рассматриваемой схемы, а саму схему — *булевым* (m, n)*-полюсником*.

Результаты предыдущего параграфа подводят теоретическую базу под одну из основных задач теории булевых (m, n)-полюсников — задачу их *синтеза*. Суть задачи синтеза комбинационных схем вообще и булевых (m, n)-полюсников в частности заключается в разработке методов, позволяющих строить сколь угодно сложные схемы из фиксированного (обычно весьма небольшого) числа типов *элементарных комбинационных схем*, называемых в случае двоичных схем *логическими элементами*.

В качестве логического элемента может быть выбран любой булев $(m, 1)$-полюсник. Ввиду сказанного выше его работа может быть охарактеризована выходной функцией $f(x_1, x_2, \ldots, x_m)$, представляющей собой булеву функцию от m переменных, которая задает зависимость единственного выходного сигнала выбранного нами элемента от совокупности всех его входных сигналов. Говорят, что выбранный логический элемент реализует эту булеву функцию или, соответственно, определяемую этой функцией булеву операцию.

Предположим теперь, что зафиксирован какой-либо набор логических элементов. Синтез комбинационной схемы из элементов выбранного набора заключается в последовательном *подсоединении* выходных полюсов одних элементов ко входным полюсам других элементов так, чтобы не подсоединять к одному и тому же входному полюсу нескольких выходных полюсов и не образовывать замкнутых цепей, по которым сигнал, выйдя из какого-либо элемента Q и проходя, возможно, через другие элементы, снова мог бы попасть на один из входных полюсов того же самого элемента Q. При этом будем предполагать, что в нашем распоряжении для любого элемента выбранного набора неограниченное число копий этого элемента, так что недостатка в *количестве* (но не числе типов!) логических элементов никогда не будет.

После окончания описанного процесса подсоединения выходных полюсов одних элементов ко входным полюсам других некоторое множество M входных полюсов и некоторое множество N выходных полюсов окажутся свободными от каких-либо соединений с другими полюсами. Естественно теперь принять множество M за множество входных, а множество N за множество выходных

полюсов построенной в результате описанного процесса сложной схемы.

Если в процессе соединения полюсов соблюдались перечисленные выше ограничения, то построенная схема задает выходной сигнал на каждом из n полюсов множества N как вполне определенную булеву функцию от сигналов на всех m полюсах множества M. Ее можно рассматривать поэтому как комбинационную схему в двоичном алфавите или, более точно, как булев (m, n)-полюсник.

Легко понять, что множество N выходных полюсов схемы может быть дополнено за счет полюсов, подвергшихся соединениям, которые будем называть *внутренними узлами* схемы. При использовании некоторых типов конкретной физической реализации двоичных сигналов можно подсоединять к одному и тому же входному полюсу несколько выходных полюсов. Неоднозначности за счет прихода в один и тот же полюс нескольких сигналов не возникает, ввиду наличия так называемого *естественного разделения сигналов*. Естественное разделение заключается в том, что нулевой сигнал на том или ином полюсе образуется тогда и только тогда, когда все одновременно приходящие на этот полюс сигналы равны нулю. Если же хотя бы один из приходящих сигналов был равен единице, то и совокупный сигнал равен единице. В этом случае входные сигналы схемы могут, очевидно, подаваться и на некоторые ее внутренние узлы, в силу чего они включаются в множество M входных полюсов схемы.

Если синтезированная схема имеет единственный выходной полюс и характеризуется, таким образом, единственной выходной булевой функцией $f(x_1, x_2, \ldots, x_m)$, то описанный процесс построения схемы методом последовательного соединения узлов по существу повторяет процесс последовательного построения представляющей функцию $f(x_1, x_2, \ldots, x_m)$ формулы с помощью операций, реализуемых использованными нами логическими элементами. Синтез произвольного булева $(m, 1)$-полюсника оказывается возможным, если набор указанных операций был усиленно полным. Поскольку всякий (m, n)-полюсник может быть скомпанован из n штук $(m, 1)$-полюсников, то в случае выполнения условия усиленной полноты, получим возможность строить произвольные двоичные комбинационные схемы.

На практике оказывается, однако, как правило, нетрудным подавать на синтезируемую схему по специально выделенным для этой цели каналам сигналы, равные тождественно (во все моменты времени) нулю и единице. Для нулевого сигнала часто, впрочем, не требуется никакого специального канала, поскольку он при ряде физических реализаций сигналов возникает на каждом изолированном, т. е. на неподсоединенном никуда входном полюсе. В этом случае условием возможности синтеза произвольной двоичной комбинационной схемы является уже не усиленная, а обыч-

ная полнота набора операций, реализуемых выбранными логическими элементами. При этом для краткости говорят о полноте либо неполноте набора самих логических элементов, а не реализуемых ими булевых операций.

К логическим элементам, наиболее часто употребляемым на практике, относятся так называемые *совпадения* и *разделения*, реализующие соответственно булевы операции умножения и дизъюнкции. Как правило, наряду с *двухвходовыми* совпадениями и разделениями, реализующими функции xy и $x \vee y$, широко употребляются *многовходовые* вариации этих элементов, реализующие булевы функции $x_1 x_2 \ldots x_n$ и $x_1 \vee x_2 \vee \ldots \vee x_n$.

Булев (1,1)-полюсник, выполняющий операцию отрицания, также часто фигурирует в числе логических элементов под названием *инвертора*. При реализации сигналов 0 и 1 в так называемых *потенциальных* схемах с помощью двух различных уровней электрического потенциала схемы совпадений и разделений могут быть построены с помощью сопротивлений и полупроводниковых диодов, а инвертор — с помощью сопротивлений и полупроводникового триода (транзистора).

При использовании в качестве набора логических элементов двухвходовых совпадений и разделений и инвертора задача синтеза булевых (m ,1)-полюсников сводится к задаче построения формул булевой алгебры, представляющих выходные функции этих (m, 1)-полюсников. Интерес представляет построение не какой-нибудь схемы с заданной выходной функцией (ввиду только что сказанного, это сделать нетрудно), а построение достаточно *экономной* схемы, расходующей по возможности меньшее число логических элементов. Задача построения экономных схем сводится в рассматриваемом случае к задаче минимизации формул булевой алгебры.

Очень часто на практике, строя ту или иную комбинационную схему, имеют возможность подать на ее входные полюсы не только интересующие нас входные сигналы $x_1, x_2, \ldots, x_n$, но и их отрицания $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n$. В этом случае для синтеза схемы оказывается, очевидно, достаточным иметь только совпадения и разделения, а задачу построения достаточно экономных схем решают обычно лишь в классе так называемых *двухступенчатых* схем, т. е. схем, в которых все совпадения предшествуют разделениям либо, наоборот, все разделения предшествуют совпадениям. Подобные схемы описываются, очевидно, дизъюнктивными либо конъюнктивными нормальными формами, методы минимизации которых разобраны в § 2 настоящей главы.

В качестве примера синтеза двухступенчатой комбинационной схемы рассмотрим синтез булева (6,1)-полюсника с выходной функцией $f = \bar{x}yz \vee xy \vee x\bar{y}\bar{z} \vee yz \vee x\bar{z}$, предполагая, что на шесть входных полюсов нашей схемы подаются сигналы x, y, z, $\bar{x}$, $\bar{y}$, $\bar{z}$, а в качестве логических элементов выбираются двухвходовые совпадения и разделения.

Если строить схему в строгом соответствии с первоначально заданной представляющей функцию f формулой, то схема будет содержать 7 совпадений и 4 разделения. Если же минимизировать эту формулу по методу Блейка так, как это сделано (именно для этой формулы) в конце § 2, то окажется, что заданную функцию f можно представить гораздо более простой формулой: $f = yz \vee x\bar{z}$.

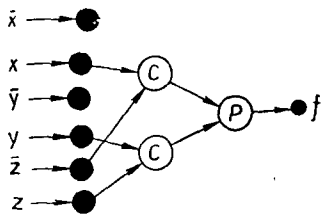

Рис. 5.

Соответствующая этой формуле схема содержит всего два совпадения и одно разделение. Изображая совпадения и разделения кружочками с буквами C и P внутри, можем представить построенную схему наглядно (рис. 5). В построенной схеме входные полюсы, на которые подаются сигналы $\bar{x}$ и $\bar{y}$, фактически не используются. Поэтому заданная выходная функция может быть реализована булевым (4,1)-полюсником, а не (6,1)-полюсником, как это предполагалось вначале.

В рассмотренном примере выходная функция синтезируемой схемы была задана в виде некоторой формулы булевой алгебры, так что процесс синтеза сводился по существу лишь к упрощению этой формулы. На практике чаще всего сталкиваются со случаем, когда выходные функции синтезируемой схемы задаются таблицами своих значений. В этом случае первым этапом процесса синтеза схемы служит нахождение каких-нибудь (не обязательно самых простых) формул, представляющих заданные функции. Универсальным способом для такого построения является способ, основанный на использовании совершенных дизъюнктивных нормальных форм (см. § 2): любая булева функция может быть представлена в виде дизъюнкции конституэнт единицы, соответствующих тем наборам значений переменных, на которых эта функция обращается в единицу. Полученное представление подвергается затем минимизации.

В случае, когда число переменных относительно невелико, удобно искать тупиковые и даже минимальные дизъюнктивные нормальные формы, представляющие заданные функции, непосредственно по таблицам значений этих функций. Для облегчения такого поиска употребляются специальные формы записи указанных таблиц в виде так называемых *карт Карнау* (диаграмм Вейча).

Карта Карнау представляет собой таблицу с 4 строками, обозначенными различными наборами значений первых двух переменных x, y, и с 4 столбцами, обозначенными различными наборами значений последних двух переменных z, u. Поле карты (для случая четырех переменных) разбивается, таким образом, на 16 квадратов, которые занумерованы последовательными числами от 0 до 15 включительно. Карта Карнау для четырех переменных:

xy \ zu	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

При задании с помощью указанной карты той или иной булевой функции $f(x, y, z, u)$ в каждом квадрате записывается значение этой функции (0 или 1) на том наборе значений переменных, номер которого совпадает с номером данного квадрата (первые два элемента набора, о котором здесь идет речь, обозначают строку, а вторые два его элемента — столбец, на пересечении которых расположен рассматриваемый квадрат).

При таком задании в случае частных булевых функций на карте Карнау возникают квадраты трех типов: обозначенные нулем, обозначенные единицей и никак не обозначенные. Последние квадраты соответствуют тем наборам, на которых значения рассматриваемой функции не определены. В случае всюду заданных булевых функций во все квадраты будут вписаны либо нули либо единицы, поэтому такие функции могут быть заданы указанием лишь тех квадратов, в которых будут записаны единицы, или, как будем говорить, указанием *конфигурации единиц* рассматриваемой функции.

Карта Карнау построена так, что конфигурации единиц, задающие различные элементарные произведения, распознаются весьма просто. Для рассматриваемого случая четырех переменных конфигурации единиц элементарных произведений длины 4 (конституэнт единицы) сводятся к отдельным, или, как будем говорить теперь, *элементарным*, квадратикам карты Карнау.

Соответствующие конфигурации, задающие все $C_4^3 \cdot 2^3 = 32$ элементарных произведения длины 3, представляют собой всевозможные пары элементарных квадратиков, стоящих рядом и составляющих таким образом прямоугольник с размерами 2×1. Необходимо лишь мысленно отождествить противоположные края карты Карнау — верхний с нижним и левый с правым. В результате такого отождествления нужно считать, например, что элементарные квадратики с номерами 4 и 6 или с номерами 0 и 8 стоят рядом, в то же время элементарные квадраты 5 и 9 или 7 и 2 не должны рассматриваться как рядом стоящие.

Аналогично конфигурации единиц, задающие все $C_4^2 \cdot 2^2 = 24$ элементарных произведения длины 2, представляются всевозможными объединениями четверок элементарных квадратиков, образующих (4×1)-прямоугольники и (2×2)-квадраты, а для всех

$C_4^1 \cdot 2 = 8$ элементарных произведений длины 1 соответствующие представления даются всевозможными объединениями элементарных квадратиков в (4 × 2)-прямоугольники. При этом не следует забывать об отождествлении противоположных краев карты Карнау.

Элементарное произведение, соответствующее любой из перечисленных выше конфигураций единиц, легко найти, так как такое произведение составляется из всех тех и только тех сомножителей ($x, \bar{x}, y, \bar{y}, z, \bar{z}, u, \bar{u}$), которые обращаются в единицу на всех охватывающихся данной конфигурации наборах значений переменных.

Используя это правило, легко найти, например, что конфигурации, состоящей из элементарных квадратиков 5 и 4, соответствует элементарное произведение $\bar{x}y\bar{z}$, а конфигурации ((2 × 2) — квадрату!), состоящей из элементарных квадратиков 0, 2, 8, 10, соответствует элементарное произведение $\bar{y}\bar{u}$.

При задании булевой функции f картой Карнау нахождение представляющих эту функцию тупиковых и минимальных дизъюнктивных нормальных форм сводится к нахождению наиболее экономных накрытий конфигурации единиц, задающей функцию f, описанными выше конфигурациями единиц, соответствующими элементарным произведениям различной длины (см. § 2).

Рассмотрим в качестве примера задачу нахождения такого минимального накрытия для булевой функции f, заданной картой Карнау

xy \ zu	00	01	11	10
00	1	0	0	1
01	—	0	0	0
11	0	—	1	—
10	1	0	1	1

Предполагается, что в квадратиках, в которых проставлены черточки, значения функции f могут быть любыми, так что, если для образования той или иной желательной конфигурации необходимо в том или ином из этих квадратиков поставить единицу, всегда можно это сделать.

Легко видеть, что все (4 × 2)-прямоугольники, какие только можно построить на заданной карте, захватывают хотя бы один нуль функции f. Это означает, что среди элементарных произведений длины 1 нет импликант рассматриваемой функции. Имеется два (2 × 2)-квадрата, не захватывающих нулей функции f: «квадрат», составленный из четырех угловых элементарных квадратиков, и «квадрат», стоящий в правом нижнем углу карты (он охва-

тывает три единицы и одну черточку). Вместе указанные квадраты накрывают все единицы функции f, и потому эта функция (с точностью до безразличных значений, обозначенных черточками) может быть представлена в виде дизъюнкции соответствующих им элементарных произведений $f = \bar{y}\,\bar{u} \vee xz$.

Найденная дизъюнктивная нормальная форма является, как легко видеть, минимальной для функции f при любых возможных расшифровках безразличных значений, обозначенных черточками.

Описанный прием непосредственного нахождения минимальных дизъюнктивных форм применим не только для булевых функций от четырех переменных, но и для функций от меньшего числа переменных. Карты Карнау общего вида для функций от трех и двух переменных:

xy \ z	0	1
00	0	1
01	2	3
11	6	7
10	4	5

x \ y	0	1
0	0	1
1	2	3

Пользуясь первой таблицей, необходимо мысленно отождествить верхний и нижний края, так что элементарные квадратики с номерами 0; 4 и 1; 5 следует считать соседними.

С помощью некоторых дополнительных ухищрений можно построить карты Карнау для 5 и 6 переменных. Для большего числа переменных в общем случае задача нахождения минимальных представлений непосредственно по таблицам булевых функций становится столь громоздкой, что соответствующие карты Карнау уже мало помогают делу. В этих случаях приходится прибегать к аналитическим методам минимизации формул типа метода Блейка и других аналогичных методов.

Нахождение минимальных дизъюнктивных нормальных форм для выходных функций булевых многополюсников чрезвычайно полезно не только для синтеза описанных выше двухступенчатых схем на элементах совпадения и разделения, но и для синтеза схем на *вентильных элементах*, называемых обычно просто *вентилями*.

Вентилем называется двоичная комбинационная схема с двумя входными и одним выходным полюсами. Работа вентиля состоит в том, что он пропускает или не пропускает на свой выходной полюс сигнал, подаваемый на один из его входных полюсов (называемый вентильным), в зависимости от того, подан ли на второй из входных полюсов (называемый управляющим) сигнал, равный единице, или, соответственно, сигнал, равный нулю.

В вентильных схемах (т. е. в схемах, составленных из вентилей) на управляющие входные полюсы всех вентилей подаются сигналы, равные некоторым исходным переменным x, y, z, ... и их отрицаниям $\bar{x}$, $\bar{y}$, $\bar{z}$, ... Кроме того, есть еще один входной полюс схемы, на который подается вентильный входной сигнал, тождественно равный единице. Для вентильных сигналов выполняется свойство *естественного разделения* (см. выше), обеспечивающее при подаче нескольких вентильных сигналов на один и тот же полюс возникновение на этом полюсе сигнала, равного дизъюнкции всех этих сигналов. Выходные сигналы вентильных схем также являются сигналами вентильного типа.

Вентильную схему для случая одного выходного полюса можно полностью построить по всякой формуле, представляющей выходную функцию схемы, с помощью операций умножения и дизъюнкции, применяемых ко входным переменным и их отрицаниям (примером подобной формулы может служить любая дизъюнктивная нормальная форма). При таком построении всякому умножению соответствует последовательное, а всякой дизъюнкции — параллельное соединение вентилей либо вентильных цепей, составленных из нескольких вентилей.

Если обозначать вентиль кружочком с буквой B внутри, то вентильная схема, составленная в соответствии с формулой $f = (x \vee \bar{y})z \vee \bar{x}y$, будет иметь вид, изображенный на рис. 6. Во внутреннем узле схемы, обозначенном буквой A, генерируется вентильный сигнал $x \vee \bar{y}$ (результат параллельного соединения вентилей с управляющими сигналами x и $\bar{y}$). В узле B генерируется вентильный сигнал $\bar{x}$, а в узле C — вентильный сигнал $\bar{x}y$ (результат последовательного соединения вентилей с управляющими сигналами $\bar{x}$ и y). Наконец, выходной сигнал всей схемы в целом (в полюсе D) является результатом параллельного соединения двух вентильных цепей с выходными (вентильными) сигналами $(x \vee \bar{y})z$ и $\bar{x}y$.

К вентильным схемам относятся так называемые *релейно-контактные схемы*, построенные на электромагнитных реле. Вентили этого типа (контакты реле) обладают *двусторонней проводимостью*, передавая вентильные сигналы не только в прямом направлении (от вентильного входного полюса к выходному), но и в прямо противоположном. Это обстоятельство порождает дополнительные трудности при построении теории релейно-контактных схем (связанные с существованием так называемых мостиковых схем и появлением не запланированных вначале путей передачи сигналов). Подобные трудности не возникают обычно в случае электронных вентилей, не обладающих двусторонней проводимостью.

При построении вентильных схем на вентилях всех типов существенную помощь может оказать так называемый *метод кас-*

кадов (см. [65]), основанный на использовании соотношения, справедливого для любой булевой функции f,

$$f(x_1, x_2, \ldots, x_{n-1}, x_n) = f(x_1, x_2, \ldots, x_{n-1}, 1)\, x_n \vee f(x_1, x_2, \ldots \ldots, x_{n-1}, 0)\, \bar{x}_n. \qquad (34)$$

Справедливость этой формулы легко видеть, полагая в ней $x_n = 1$ и $x_n = 0$.

Применительно к вентильным схемам, а также к схемам, построенным из совпадений и разделений, формула (34) сводит задачу синтеза схемы с n-местной выходной функцией $f(x_1, x_2, \ldots, x_n)$

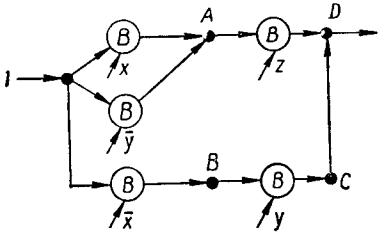

Рис. 6.

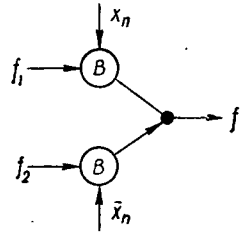

Рис. 7.

к задаче синтеза схемы с двумя $(n-1)$-местными выходными функциями $f_1(x_1, x_2, \ldots, x_{n-1}) = f(x_1, x_2, x_{n-1}, 1)$ и $f_2(x_1, x_2, \ldots \ldots, x_{n-1}) = f(x_1, x_2, \ldots, x_{n-1}, 0)$.

Каскад вентильной схемы, реализующей указанное сведение, изображен на рис. 7. При нескольких выходных функциях схема построенного нами (n-го) каскада усложняется, однако сам процесс сведения остается по существу тем же самым. Продолжая процесс сведения, мы в конце концов построим требуемую вентильную схему, составляющуюся в общем случае (для n переменных) из n каскадов.

Применение метода синтеза, аналогичного описанному методу каскадов, позволило К. Шеннону [80] установить следующую оценку числа вентилей (любого типа), необходимых для реализации (в виде выходной функции некоторой вентильной схемы) произвольной булевой функции от n аргументов.

Теорема 1. *Каково бы ни было вещественное положительное число* ε, *существует такое целое число* $N = N(\varepsilon)$, *что любую булеву функцию от* $n \geqslant N$ *переменных можно реализовать в виде выходной функции вентильной схемы, содержащей не более* $(1+\varepsilon)\frac{2^{n+2}}{n}$ *вентилей. При любом* n *для подобной реализации требуется не более* $\frac{2^{n+3}}{n}$ *вентилей.*

Аналогичные *оценки сложности* схем, но для общих предположений относительно употребляемых наборов логических элементов, были установлены О. Б. Лупановым (см., например, [51]).

Было показано также, что существуют такие булевы функции, которые нельзя реализовать менее чем $(1-\varepsilon_n)\frac{2^{n+2}}{n}$ вентилями, причем величина ε_n в этом случае стремится к нулю при неограниченном росте n.

Представляет интерес обобщение приведенных результатов на случай произвольных булевых (m, n)-полюсников, строящихся с помощью любых двухвходовых логических элементов. Поскольку всякий булев (m, n)-полюсник реализует некоторый алфавитный оператор A с конечной областью определения, минимальная возможная сложность $L(A)$ булевых (m, n)-полюсников, реализующих данный оператор A, может быть принята в качестве естественной количественной *оценки сложности самого оператора* A. При этом ввиду отсутствия достаточно обоснованных причин для того, чтобы отдать предпочтение тому или иному двухвходовому логическому элементу, наиболее естественно, по-видимому, при построении указанных булевых (m, n)-полюсников, пользоваться всеми типами двухвходовых логических элементов, считая *сложностью схемы* общее число составляющих ее логических элементов.

Для оценки сложности алфавитных операторов с бесконечными областями определения описанный метод непосредственно не подходит. Если, однако, требуется получить не абсолютную, а лишь относительную практическую оценку сложности нескольких алфавитных операторов, можно предварительно финитизировать (сделать конечными) их области определения, отбрасывая все входные слова, длины которых превосходят некоторое число N. Число это должно быть выбрано таким, чтобы *вероятность* встретить в практике применения рассматриваемых операторов входные слова большей чем N длины была достаточно малой.

Если для всех $n = 1, 2, \ldots$ заданы вероятности $\varrho(n)$ появления входных слов длины n, то можно также поступать следующим образом: заданный алфавитный оператор A разбить на операторы $A_1, A_2, \ldots$ так, что оператор A_n имеет в качестве своей области определения множество всех слов из области определения оператора A, длина которых равна n, и действует на эти слова точно так же, как и оператор $A (n = 1, 2, \ldots)$. Пусть $L(n)$ — сложность оператора A_n, подсчитанная описанным выше способом. Тогда сложностью исходного алфавитного оператора естественно назвать бесконечную сумму $\sum_{n=1}^{\infty} L(n)\varrho(n)$.

Более рациональные оценки сложности алфавитных операторов могут быть получены при использовании для представления операторов вместо комбинационных схем дискретных автоматов, обладающих памятью. Основы теории таких автоматов рассматриваются в следующей (третьей) главе настоящей книги.

§ 5. Понятие об исчислении высказываний

Исчисление высказываний представляет собой первую и наиболее простую часть математической логики. Основной задачей, которую ставит перед собой математичеекая логика, является формализация сложных мыслительных процессов, из которых складывается так называемое логическое мышление. Подобная формализация достигается с помощью построения *логических исчислений*.

Всякое логическое исчисление включает в себя прежде всего те или иные средства для *формализации записи* различного рода утверждений, о которых есть смысл говорить, *истинны* они или *ложны*. Подобного рода утверждения в математической логике принято называть *высказываниями*. Формализация, о которой здесь идет речь, состоит в том, что для обозначения высказываний или их составных частей, а также для обозначения различного рода *операций*, позволяющих строить более сложные высказывания из более простых, вводится строго определенная система символов. В результате формализации получаем возможность записывать высказывания в виде *формул*, строящихся из введенных символов по определенным правилам.

Несмотря на всю важность формализации записи высказываний, сама по себе формализация еще не составляет исчисления. Для построения того или иного логического исчисления необходимо определить еще некоторые формулы и операции над формулами, называемые *аксиомами* и *правилами вывода* соответствующего исчисления, которые позволяли бы выводить формальным путем всевозможные логические *следствия* из любой заданной системы утверждений и давали бы возможность охарактеризовать формально все так называемые *тождественно истинные* высказывания (формулы) рассматриваемого исчисления.

Для того чтобы понять, что такое тождественно истинные высказывания, рассмотрим некоторые примеры. Высказывания типа «кислород является газом» или «дважды два — одиннадцать» представляют собой примеры так называемых *элементарных постоянных* высказываний. Элементарность этих высказываний состоит в том, что их нельзя расчленить на более простые составные части, которые сами являлись бы высказываниями. Действительно, выражения «является газом» или «дважды два» не представляют собой законченных высказываний, поскольку применительно к ним бессмысленно ставить вопрос истинны они или ложны. Термин «постоянные» применительно к приведенным высказываниям должен подчеркнуть, что речь идет о вполне определенных высказываниях, относящихся к вполне определенным областям знания.

Заметим, что истинность или ложность этих высказываний зависит от условий, в которых они рассматриваются, и устанавливается, как правило, за пределами математической логики. Применительно к первому высказыванию это понятно само собой (кислород при определенных условиях может быть не только газом,

но и жидкостью и даже твердым телом). Второе же высказывание на первый взгляд представляется заведомо ложным. В действительности, однако, стоит нам предположить, что вместо десятичной системы счисления употребляется троичная при условии сохранения привычных для нас названий многозначных чисел, как высказывание «дважды два — одиннадцать» ($2 \times 2 = 3 \cdot 1 + 1 = 11$) из ложного превратится в истинное.

Поэтому при применениях математической логики бывает необходимым предполагать условия, в которых делается то или иное *постоянное* высказывание, заданными столь точно и определенно, что *значение истинности* этого высказывания не может претерпевать изменений в процессе получения различного рода выводов и следствий из него в рамках используемого логического исчисления. Таким образом, любое постоянное высказывание на всем протяжении того или иного логического вывода должно считаться либо все время истинным, либо все время ложным.

В исчислении высказываний не интересуются внутренней структурой элементарных высказываний, рассматривая их как единые целые. Естественно поэтому для их обозначения использовать отдельные буквы того или иного алфавита (обычно латинского). Отдельными буквами могут обозначаться и так называемые *переменные высказывания*. Термин «переменное высказывание» применительно к тому или иному символу означает, что вместо этого символа всегда может быть подставлено *любое* конкретное постоянное высказывание, как истинное, так и ложное.

Высказывания, как постоянные, так и переменные, можно объединять в *сложные высказывания*, используя в качестве связок слова «и», «или», «если — то», «не» и т. п. Если в состав сложных высказываний входят переменные высказывания, то при замене их одними высказываниями сложное высказывание может оказаться истинным, а при замене другими — ложным. Например, сложное высказывание «А и В», где А и В — переменные высказывания, будет, очевидно, истинным в том и только в том случае, когда оба высказывания А и В будут истинными.

Существуют, однако, и такие сложные высказывания, содержащие в своем составе переменные высказывания, которые остаются истинными при любых значениях, какие только можно придать указанным переменным высказываниям. Например, сложное высказывание «если неверно то, что высказывание A ложно, то высказывание A истинно» остается истинным, какое бы высказывание ни подставили на место переменного высказывания A. Подобные высказывания принято называть *тождественно истинными высказываниями*. Задача выделения тождественно истинных высказываний во множестве всех возможных высказываний является важнейшей задачей любого логического исчисления.

После всех наших предварительных замечаний перейдем к построению собственно *исчисления высказываний*, или, как его еще иногда называют, *пропозиционального исчисления*.

Исчисление высказывания строится из формальных объектов трех типов. Объектами первого типа являются переменные и постоянные высказывания, не расчленяемые на отдельные составные части. Для их обозначения будем пользоваться прописными латинскими буквами (с индексами или без индексов), называя их пропозициональными буквами. Объектами второго типа являются *пропозициональные связки* — формальные эквиваленты приводившихся выше слов-связок «не», «или», «и», «если — то». Для их обозначения используются соответственно символы *отрицания* ($\neg$), *дизъюнкции* ($\vee$), *конъюнкции* ($\wedge$) и *импликации* ($\supset$). Заметим, что при чтении формул символ импликации удобнее заменять словом «влечет», а не словами «если — то». Объектами третьего типа являются скобки, служащие для выражения порядка, в котором действуют перечисленные нами пропозициональные связки.

Подобно тому как в начале § 2 строились формулы булевой алгебры, из введенных нами формальных объектов строятся формулы исчисления высказываний. Различие заключается в употреблении дополнительного символа $\supset$ (формального аналога булевой операции импликации), а также в замене точки в обозначении конъюнкции (умножения) символом $\wedge$ и в употреблении в качестве знака отрицания вместо черты над отрицаемым выражением символа $\neg$, ставящегося *перед* отрицаемым выражением.

Формулами исчисления высказываний являются отдельные буквы, а также все выражения, строящиеся рекуррентно из уже построенных формул $\mathfrak{A}$ и $\mathfrak{B}$ по правилам: $\neg(\mathfrak{A})$, $(\mathfrak{A}) \wedge (\mathfrak{B})$, $(\mathfrak{A}) \vee (\mathfrak{B})$, $(\mathfrak{A}) \supset (\mathfrak{B})$. Как и в случае формул булевой алгебры, с целью упрощения записи часть скобок можно опустить, если это не вызывает недоразумений в порядке применения пропозициональных связок. Принимается, что при отсутствии скобок порядок применения связок определяется последовательностью $\neg$, $\wedge$, $\vee$, $\supset$, а для одноименных связок — порядком их появления в формуле, читаемой слева направо. В ряде случаев при исчислении высказываний вводится дополнительный символ $\approx$ (или $\sim$; читается как «эквивалентно») для сокращенного обозначения в виде $(\mathfrak{A}) \approx (\mathfrak{B})$ выражения $((\mathfrak{A}) \supset (\mathfrak{B})) \wedge ((\mathfrak{B}) \supset (\mathfrak{A}))$. При употреблении этого символа предполагается, что он занимает последнее место в выписанной только что последовательности (после символа $\supset$).

В качестве иллюстрации принятого способа сокращения числа скобок заметим, что формула $\neg A \vee B \wedge C \supset B \vee C$ понимается как $((\neg A) \vee (B \wedge C)) \supset (B \vee C)$, а не каким-либо иным образом, формула $A \approx B \supset C$ должна пониматься как $(A) \approx ((B) \supset (C))$, а не как $((A) \approx (B)) \supset (C)$ и т. д.

Введенные выше определения решают лишь первую часть задачи построения пропозиционального исчисления — проблему формализации записи сложных высказываний. Вторая часть этой

задачи — нахождение способа для определения тождественно истинных высказываний — может быть решена двумя путями: содержательным и формальным.

При *содержательном подходе*, более простом для понимания, ни на минуту нельзя забывать о содержательном смысле понятий пропозициональных букв и пропозициональных связок. При этом используется в максимально возможной степени обедненное понятие содержательного смысла. Так, если пропозициональная буква A означает то или иное постоянное высказывание, то нет необходимости помнить само это высказывание, нужно знать лишь значение так называемой *функции истинности* этого высказывания: «истина», если высказывание A истинно, и «ложь», если высказывание A ложно.

Функцию истинности любой пропозициональной буквы, обозначающей переменное высказывание, отождествляем с самой этой буквой, рассматривая ее как *булеву переменную*. Таким образом, содержательный смысл пропозициональных букв в нашем построении исчерпывается их возможностью принимать два значения: «истина» и «ложь».

Содержательное значение пропозициональных связок будем связывать лишь с *функциями истинности* строящихся с их помощью сложных высказываний. Каждую формулу $\mathfrak{A}$ исчисления высказываний можно интерпретировать как формулу в булевой алгебре с включенной в нее дополнительно операцией импликации. Входящие в формулу $\mathfrak{A}$ постоянные высказывания должны быть заменены соответствующими булевыми константами (значениями их функций истинности). Соответствующие же переменным высказываниям символы рассматриваются как аргументы представляемой формулой $\mathfrak{A}$ булевой функции. Эта функция и называется *функцией истинности сложного высказывания, выражаемого формулой* $\mathfrak{A}$.

На содержательном уровне построения исчисления высказываний тождественно истинными считаются те и только те формулы этого исчисления (сложные высказывания), функции истинности которых принимают значение «истина» при всех значениях переменных.

Напомним, что, в силу принятого в § 1 соглашения, значение «истина» соответствует единице, а значение «ложь» — нулю. Используя приведенные в § 1 таблицы значений для конъюнкции $x \wedge y$ (таблица выражается кортежем (0001)), для дизъюнкции $x \vee y$ (кортеж (0111)), для импликации $x \supset y$ (кортеж (1101)), и помня, что отрицание переводит 1 в 0, а 0 в 1, легко найти таблицу значений функции истинности для любой формулы исчисления высказываний. Эту таблицу называют обычно *таблицей истинности* рассматриваемой формулы (или определяемого ею сложного высказывания). При ее заполнении употребляют сокращенные обозначения: **И**—для «истины» и **Л**—для «лжи». В качестве примера приведена таблица истинности формулы $A \supset B$, определяющая содержатель-

ный смысл импликации (рассматриваемый как пропозициональная связка):

A	B	$A \supset B$
Л	Л	И
Л	И	И
И	Л	Л
И	И	И

Из приведенной таблицы видно, что смысл термина «влечет» (соответствующего пропорциональной связке $\supset$) в исчислении высказываний несколько иной, чем в обычной речи. Действительно, обычно, когда говорят, что некоторое высказывание A влечет за собой другое высказывание $-B$, имеют в виду, что высказывания A и B связаны между собой причинно. Так, сложное высказывание, утверждающее, что высказывание «это вещество представляет собой кислород» влечет за собой высказывание «это вещество является газом», представляется нам (со сделанными выше оговорками относительно газообразности кислорода) истинным и разумным. В то же время сложное высказывание, которое утверждает, что высказывание «зимой холодно» влечет за собой высказывание «дважды два — четыре», представляется нам сплошной бессмыслицей. Между тем, в силу построенной выше таблицы истинности для формулы $A \supset B$, в исчислении высказываний второе из приведенных нами сложных высказываний должно считаться истинным никак не в меньшей мере, чем первое.

Причину указанного положения понять нетрудно. В самом деле, ограничив себя условием рассматривать элементарные высказывания только с позиций того, истинны они или ложны, мы тем самым сделали фактически *неразличимыми между собой* все истинные (как и все ложные) элементарные высказывания. Поэтому, в частности, при определении содержания, вкладываемого в связку $\supset$, мы вынуждены оперировать лишь понятиями истинности и ложности, а на таком пути заведомо невозможно проникнуть во внутреннюю структуру элементарных высказываний всех классификаций, что, разумеется, необходимо для установления причинных связей между ними.

Раскрытие внутренней структуры элементарных высказываний и связанное с ними увеличение возможностей для логического анализа достигаются путем более сложных логических исчислений, в частности так называемым *исчислением предикатов* (см. гл. VI). Что же касается исчисления высказываний, то мы должны мириться с относительной бедностью его выразительных средств, представляющей собой как бы своеобразную плату за простоту и прозрачность этого исчисления.

Пропозициональная связка $\supset$ используется в дальнейшем как инструмент для получения логических следствий из тех или иных формул пропозиционального исчисления и других, более высоких логических исчислений. Такие следствия должны быть истинными при истинности исходных формул. Поэтому конструкция вывода обязательно должна исключить возможность (указанием на свою ложность в этом случае) получения *ложных* следствий при *истинности* исходных формул. В то же время *при ложности* исходных данных получение *любых* следствий (как истинных, так и ложных) не свидетельствует, разумеется, *о ложности самой конструкции вывода*. Это обстоятельство и находит свое конкретное выражение в таблице истинности для формулы $A \supset B$. Все сказанное здесь станет более понятным после ознакомления с формальным аспектом исчисления высказываний.

Описанный нами содержательный аспект исчисления высказываний позволяет относительно просто решить вопрос о тождественной истинности любого сложного высказывания (заданного той или иной формулой исчисления): достаточно перебрать все возможные наборы значений истинности составляющих его переменных высказываний и проверить, на всех ли этих наборах функция истинности рассматриваемого сложного высказывания принимает значение «истина». Например, формула $A \wedge B \supset A$ будет тождественно истинной формулой исчисления высказываний на основании следующей проверки: если $A =$ **Л** и $B =$ **Л**, то формула $A \wedge B \supset A$ сводится к **Л**$\supset$**Л**, что, в силу таблицы истинности, дает значение **И**; то же самое будет при $A =$**Л** и $B =$**И**; при $A =$**И**, в зависимости от значений B(**Л** или **И**), сводим нашу формулу либо к **Л**$\supset$**И**, либо к **И**$\supset$**И**, что, на основании таблицы истинности, в обоих случаях приводит к значению **И**.

Можно использовать для доказательства тождественной истинности формул исчисления высказываний технику преобразований в булевой алгебре. Нужно лишь предварительно заменить все импликации согласно формуле $(\mathfrak{A} \supset \mathfrak{B}) = (\neg \mathfrak{A} \vee \mathfrak{B})$ (см. § 1). Применительно к рассмотренному только что примеру получаем следующую цепочку преобразований: $A \wedge B \supset A = \neg (A \wedge B) \vee \vee A = (\neg A \vee \neg B) \vee A = \neg A \vee A \vee \neg B = 1 \vee \neg B = 1$. Эта цепочка доказывает тождественную истинность заданной нам формулы.

Аналогично могут быть рассмотрены также *тождественно ложные* высказывания, т. е. такие (сложные) высказывания, функции истинности которых принимают значение «ложь» при всех значениях составляющих данное высказывание переменных высказываний. Легко понять, что класс всех тождественно ложных высказываний совпадает с отрицаниями всевозможных тождественно истинных высказываний.

Несмотря на простоту и прозрачность, содержательный аспект исчисления высказываний имеет и ряд недостатков. Во-первых, метод доказательства истинности формул, основанный на пере-

боре всех наборов значений аргументов, не допускает непосредственного перенесения этого метода на более сложные исчисления, в которых множество таких наборов может оказаться бесконечным. Во-вторых, развитые нами выше методы позволяют непосредственно определять не тождественно истинные высказывания, а высказывания, истинные при тех или иных дополнительных предположениях (например, формула $A \supset B$, не являющаяся тождественно истинной, становится истинной при условии, что ложна формула $A \wedge \neg B$). Задачи же такого рода постоянно возникают при различных приложениях логических исчислений. Можно, правда, развить соответствующий метод в рамках булевой алгебры, однако при этом усугубится еще один существенный недостаток, связанный с содержательным аспектом исчисления высказываний, — *недостаточная формализация процесса доказательства и самого понятия доказательства* истинности тех или иных формул.

Перечисленные недостатки устраняются при полностью *формальном* подходе к построению исчисления высказываний, формализующем не только способ записи формул (для этого достаточен уже описанный способ), но также понятие тождественно истинных формул и процесс вывода логических следствий из тех или иных высказываний.

Формальный аспект исчисления высказываний характеризуется тем, что в этом случае *полностью отвлекаются* от содержательного смысла формул, рассматривая их просто как конечные последовательности индивидуально различимых символов.

Для характеристики множества всех тождественно истинных формул строится *система аксиом* рассматриваемого исчисления. Такие системы могут выбираться различными способами. Остановимся на одной из наиболее употребительных систем аксиом исчисления высказываний (см. С. К. Клини [42]). Эта система включает в себя следующие аксиомы:

1. $A \supset (B \supset A)$.
2. $(A \supset B) \supset ((A \supset (B \supset C)) \supset (A \supset C))$.
3. $A \supset (B \supset A \wedge B)$.
4. $A \wedge B \supset A$.
5. $A \wedge B \supset B$.
6. $A \supset A \vee B$.
7. $B \supset A \vee B$.
8. $(A \supset C) \supset ((B \supset C) \supset (A \vee B \supset C))$.
9. $(A \supset B) \supset ((A \supset \neg B) \supset \neg A)$.

$$10.\ \neg\neg A \supset A. \qquad 11.\ \frac{\mathfrak{A},\ \mathfrak{A} \supset \mathfrak{B}}{\mathfrak{B}}.$$

Первые десять аксиом представляют собой просто десять формул исчисления высказываний, объявляемых *тождественно истинными по определению*. Тождественная истинность аксиом предполагает возможность *подстановки* вместо входящих в них пропозициональных букв A, B, C *любых формул* исчисления высказываний (не обязательно истинных). Такая подстановка, по определению, не нарушает тождественной истинности формулы (аксиомы), подвергшейся этой подстановке.

Одиннадцатая аксиома имеет свою специфику. Это так называемое *правило вывода*, позволяющее, по определению, считать доказанной истинность формулы $\mathfrak{B}$, если истинность формул $\mathfrak{A}$ и $\mathfrak{A} \supset \mathfrak{B}$ уже была установлена ранее. Если формулы $\mathfrak{A}$ и $\mathfrak{A} \supset \mathfrak{B}$ были при этом *тождественно истинными*, то тождественно истинной будет и формула $\mathfrak{B}$. Предполагается по определению, что все тождественно истинные формулы (и только такие формулы) исчисления высказываний могут быть получены из аксиом в результате описанных подстановок и применений (вообще говоря, многократных) правила вывода 11.

Априори ниоткуда не следует, что охарактеризованное подобным *формальным путем* множество всех тождественно истинных формул пропозиционального исчисления будет совпадать с множеством всех тождественно истинных формул, определенных выше *содержательно*. Поскольку формальная тождественная истинность формул устанавливается некоторой процедурой вывода или доказательства, их называют также (формально) *доказуемыми* формулами, или (формальными) теоремами.

Формулы, тождественно истинные в содержательном смысле, для краткости будем называть просто *содержательно истинными*, противопоставляя их *формально истинным* (т. е. формально доказуемым) формулам.

Понятие формального вывода (доказательства) может быть распространено на тот случай, когда, кроме аксиом, дано еще некоторое количество формул $\mathfrak{A}_1$, $\mathfrak{A}_2$, ..., $\mathfrak{A}_n$ исчисления высказываний в качестве *условно истинных* формул. Эти формулы уже не выводимы из аксиом (формально не доказуемы) и не относятся поэтому к формально истинным формулам. Предположение же об их истинности носит условный характер и сохраняется лишь на протяжении рассматриваемого вывода. В отличие от аксиом исчисления высказываний, в этих формулах нельзя вообще говоря, заменять входящие в них пропозициональные буквы произвольными формулами. Иначе говоря, условная истинность, в отличие от формальной истинности, не носит *тождественного* характера.

Сами же правила вывода по существу сохраняются прежними. Основную роль, как и прежде, играет понятие *непосредственного*

следствия (аксиома 11): *формула* $\mathfrak{B}$ *называется непосредственным следствием формул* $\mathfrak{A}$ *и* $\mathfrak{A} \supset \mathfrak{B}$. Говорят, что формула $\mathfrak{C}$ исчисления высказывания *выводится из формул* $\mathfrak{A}_1, \mathfrak{A}_2, \ldots, \mathfrak{A}_n$, если она может быть получена из этих формул и аксиом 1—10 исчисления высказываний в результате применения (конечного числа раз) правила непосредственного следствия. Более точно, выводимыми будут в рассматриваемом случае все те и только те формулы, которые получаются в результате последовательного применения трех правил. 1. Любая из формул $\mathfrak{A}_1, \mathfrak{A}_2, \ldots, \mathfrak{A}_n$ выводима. 2. Любая из аксиом 1—10 (с учетом возможности подстановки вместо входящих в них букв любых формул) выводима. 3. Если формулы $\mathfrak{A}$ и $\mathfrak{A} \supset \mathfrak{B}$ выводимы, то выводимой будет также формула $\mathfrak{B}$. Цепочка формул, получающихся в результате последовательного применения этих трех правил, которая кончается некоторой формулой $\mathfrak{C}$, называется *формальным выводом* этой формулы.

Для обозначения выводимости употребляется специальный символ $\vdash$ (читается как «дает»), слева от которого выписываются условно истинные формулы, а справа — их следствия: $\mathfrak{A}_1, \mathfrak{A}_2, \ldots, \mathfrak{A}_n \vdash \mathfrak{C}$. Аксиомы исчисления высказываний здесь явно не выписываются (возможность их употребления при выводе как бы включается в символ $\vdash$), так что для любой формально истинной формулы $\mathfrak{B}$ можно писать $\vdash \mathfrak{B}$. Иными словами, формально истинные формулы считаются выводимыми из пустого множества (условно истинных) формул. Поэтому аксиомы 1—10 можно также рассматривать как своеобразные правила вывода, которые выводят представляющие их формулы из пустого множества формул.

Приведем простейшие примеры формального вывода, нумеруя последовательные шаги.

1. $A \supset (A \supset A)$ (аксиома 1, в которой буква B заменена буквой A).

2. $(A \supset (A \supset A)) \supset ((A \supset ((A \supset A) \supset A)) \supset (A \supset A))$ (аксиома 2, в которой буква B заменена формулой $A \supset A$, а буква C — буквой A).

3. $(A \supset ((A \supset A) \supset A)) \supset (A \supset A)$ (применение правила вывода 11 к формулам, полученным на шагах 1 и 2).

4. $A \supset ((A \supset A) \supset A$ (аксиома 1, в которой буква B заменена формулой $(A \supset A)$).

5. $A \supset A$ (применение правила вывода 11 к формулам, полученным на предыдущих двух шагах).

Приведенная цепочка формул представляет собой, на основании определения, *формальное доказательство* формулы $A \supset A$, т. е. вывод ее из пустого множества (условно истинных) формул. Таким образом, формула $A \supset A$ принадлежит к числу формально истинных формул, и ее можно записать $\vdash A \supset A$.

Другим примером является получение следствий из трех условно истинных формул A, B, $A \supset (B \supset C)$. За 5 шагов из этих формул может быть получена формула C.

1. A (первая данная нам (условно истинная) формула).

2. B (вторая данная нам формула).

3. $A \supset (B \supset C)$ (третья данная нам формула).

4. $B \supset C$ (непосредственное следствие (по правилу вывода 11) из формул 1 и 2).

5. C (непосредственное следствие формул 2 и 4). Таким образом, формула C выводима из формул A, B, $A \supset (B \supset C)$, и мы можем записать $A, B, A \supset (B \supset C) \vdash C$.

Хотя условно истинные формулы и не обладают тождественной истинностью, однако, как легко видеть, в окончательной записи (условной) выводимости с использованием символа $\vdash$ любая буква может быть заменена произвольной формулой исчисления высказываний, если только такая замена производится *одновременно* как слева, так и справа от знака выводимости. Замена же лишь в одной части ведет, вообще говоря, к ошибке.

Подобным же способом можно доказать соотношения

цепное заключение $A \supset B, B \supset C \vdash A \supset C$; (35)

перестановка посылок $A \supset (B \supset C) \vdash B \supset (A \supset C)$; (36)

импортация $A \supset (B \supset C) \vdash A \wedge B \supset C$; (37)

экспортация $A \wedge B \supset C \vdash A \supset (B \supset C)$; (38)

контрпозиция $A \supset B \vdash \neg B \supset \neg A$; (39)

$\wedge$ - введение $A, B \vdash A \wedge B$; (40)

слабое $\neg$ - удаление $A_1 \neg A \vdash B$. (41)

Если обозначить буквой Γ — произвольную конечную совокупность формул исчисления высказываний, то, применяя несколько более сложную технику доказательства (индукцию по длине вывода), можно получить следующий результат (так называемую *теорему о дедукции*).

Теорема 1. *Если в исчислении высказываний формула B выводима из совокупности формул Γ и A, то из Γ выводима формула $A \supset B$.*

В теории доказательств имеют значение также две общие схемы доказательств.

1. Доказательство путем разбора случаев: если $\Gamma, A \vdash C$ и $\Gamma, B \vdash C$, то

$$\Gamma, A \vee B \vdash C. \tag{42}$$

2. Приведение к абсурду: если $\Gamma, A \vdash B$ и $\Gamma, A \vdash \neg B$, то

$$\Gamma \vdash \neg A. \tag{43}$$

Легко проверить, что все аксиомы исчисления высказываний 1—10 представляют собой содержательно истинные формулы. Иначе говоря, соответствующие им функции истинности принимают при всех значениях переменных значение «истина». Это свойство,

очевидно, сохраняется при подстановках вместо входящих в аксиомы букв любых формул исчисления высказываний. Из таблицы истинности для импликации непосредственно следует, что из содержательной истинности формул $\mathfrak{A}$ и $\mathfrak{A} \supset \mathfrak{B}$ вытекает содержательная истинность формулы $\mathfrak{B}$. Но тогда, очевидно, все доказуемые (формально истинные) формулы будут непременно содержательно истинными. Верно (хотя и гораздо сложнее доказуемо) также и обратное, так что может быть сформулирован следующий важный результат.

Теорема 2. *При формальном построении исчисления высказываний с помощью системы аксиом* 1—11 *формально доказуемыми (формально истинными) будут все те и только те формулы этого исчисления, которые являются тождественно истинными в содержательном смысле.*

Теорема 2 содержит в себе фактически два результата относительно выбранной системы S аксиом исчисления высказываний. Первый результат состоит в том, что система S *содержательно непротиворечива*, или, иными словами, что с помощью системы S нельзя доказать ни одной формулы, которая не была бы содержательно истинной формулой.

Второй результат утверждает *содержательную полноту* системы аксиом S: нет ни одной содержательно истинной формулы высказываний, которую нельзя было бы доказать формально с помощью этой системы аксиом.

Возникает вопрос, нельзя ли определить свойства непротиворечивости и полноты чисто формально, не прибегая к содержательным построениям? Оказывается, можно.

Естественно называть систему аксиом исчисления высказываний формально непротиворечивой, если с ее помощью нельзя вывести какую-нибудь формулу $\mathfrak{A}$ вместе с ее отрицанием $\neg\mathfrak{A}$, и формально противоречивой — в противном случае.

Из свойства слабого $\neg$-удаления непосредственно следует, что в случае формальной противоречивости системы аксиом любая формула исчисления высказываний стала бы формально доказуемой. Поскольку, в силу теоремы 2, для системы S последнее обстоятельство не имеет места, то эта система оказывается не только содержательно непротиворечивой, но и *формально непротиворечивой*, или, как часто говорят, — просто *непротиворечивой* системой аксиом.

Свойство формальной полноты для системы аксиом может быть определено следующим образом: *система аксиом называется формально полной (или полной в узком смысле), если добавление к этой системе любой недоказуемой в ней формулы в качестве новой аксиомы приводит к тому, что расширенная подобным образом система аксиом оказывается формально противоречивой.* При этом предполагается обычно, что исходная система аксиом была формально непротиворечивой.

Можно показать, что введенная нами система аксиом 1—11

исчисления высказываний является *не только содержательно, но и формально полной* системой аксиом. При условии выполнения свойства содержательной непротиворечивости и при использовании в качестве правил вывода лишь правила 11 из свойства формальной полноты непосредственно вытекает свойство содержательной полноты, поскольку в противном случае любая недоказуемая содержательно истинная формула могла бы быть использована для непротиворечивого расширения исходной системы аксиом.

В выбранной нами системе аксиом нет ни одной лишней аксиомы. Более точно, ни одна из формул 1—10 не может быть формально доказана с помощью совокупности всех остальных аксиом. Это свойство называется свойством *независимости* аксиом выбранной системы. Свойство независимости доказывается отдельно для каждой аксиомы с помощью построения такой содержательной интерпретации, при которой эта аксиома не выполняется, а все остальные аксиомы выполняются.

Заметим, наконец, что, хотя присоединение недоказуемых формул в качестве новых аксиом к выбранной нами системе аксиом S исчисления высказываний, в силу свойства формальной полноты этой системы, иарушает свойство ее формальной непротиворечивости, ничто не мешает нам присоединить к системе S недоказуемые (в S) формулы $\mathfrak{A}_1, \ldots, \mathfrak{A}_m$ в качестве не тождественно, а лишь условно истинных формул. Можно показать, что противоречие (возможность вывода некоторой формулы вместе с ее отрицанием) в этом случае возникает тогда и только тогда, когда конъюнкция $\mathfrak{A}_1 \wedge \mathfrak{A}_2 \wedge \ldots \wedge \mathfrak{A}_m$ является *тождественно ложной формулой*.

В результате такого присоединения возникает та или иная формальная теория, выходящая за рамки собственно математической логики, поскольку присоединяемые формулы $\mathfrak{A}_1, \mathfrak{A}_2, \ldots, \mathfrak{A}_m$ не являются истинными в строго логическом смысле. Если в наших построениях есть тот или иной содержательный смысл, то содержательная истинность формул $\mathfrak{A}_1, \mathfrak{A}_2, \ldots, \mathfrak{A}_m$ должна постулироваться либо иметь какое-то заведомо внелогическое обоснование. В таком случае эти формулы естественно рассматривать как *аксиомы* строящейся на их основе формальной теории. Чтобы не путать их с собственно логическими аксиомами 1—11, последние называют в этом случае не аксиомами, а *схемами аксиом*, подчеркивая тем самым, что каждая из аксиом 1—11 представляет собой фактически целое *множество* аксиом, получаемых из соответствующей этой аксиоме формулы в результате замены входящих в нее букв произвольными формулами исчисления высказываний.

Глава III

ТЕОРИЯ АВТОМАТОВ

§ 1. Абстрактные автоматы и автоматные отображения

Рассмотрим алфавитные преобразования, реализуемые дискретными преобразователями информации, выдающими некоторый выходной сигнал (букву выходного алфавита) в ответ на каждый входной сигнал (букву входного алфавита). Такие преобразователи, рассматриваемые безотносительно к их внутренней структуре, принято называть *абстрактными автоматами.*

Для задания абстрактного автомата должны быть заданы три множества: *входной алфавит* $\mathfrak{X}$, *выходной алфавит* $\mathfrak{Y}$ *и множество внутренних состояний* автомата, которые будем обозначать буквой $\mathfrak{A}$. Автомат работает в дискретном времени, последовательные моменты которого удобно отождествлять с последовательными натуральными числами $t = 0, 1, 2, \ldots$ (что всегда можно сделать, выбирая должным образом единицу измерения времени).

В каждый данный момент дискретного *автоматного* времени $t = 0,1,\ldots$ автомат A находится в некотором определенном состоянии $a = a(t)$ из множества $\mathfrak{A}$ его внутренних состояний, которое для краткости будем называть *множеством состояний* автомата A. Состояние $a_0 = a(0)$ в начальный момент времени $t = 0$ называется *начальным состоянием* автомата A. Если начальное состояние остается неизменным при любых экспериментах с автоматом, то этот автомат называется *инициальным* автоматом. Поскольку, однако, практически мы не рассматриваем никаких других автоматов, кроме инициальных, термин «инициальный» часто опускается.

В каждый момент t автоматного времени, начиная с $t = 1$, на вход автомата поступает в качестве *входного сигнала* одна из букв входного алфавита $\mathfrak{X}$ $x = x(t)$. Конечные упорядоченные последовательности входных сигналов $x(1)x(2) \ldots x(k)$ автомата называются *входными словами* этого автомата. На вход автомата может подаваться любое входное слово из некоторого фиксированного заранее множества допустимых входных слов.

Любое допустимое слово $p = x(1)\, x\,(2) \ldots x\,(k)$, поданное на вход данного инициального автомата A, вызывает появление на выходе автомата *выходного слова* $q = y(1)y(2) \ldots y(k)$, представляющего собой некоторую упорядоченную конечную последовательность *выходных сигналов* автомата A (букв его выходного алфавита $\mathfrak{Y}$), имеющего ту же самую длину, что и соответствующее ему и однозначно им определяемое входное слово p. Получаемое соответствие φ между допустимыми входными словами p и соответствующими им выходными словами q называется (алфавитным) *отображением, индуцируемым рассматриваемым инициальным автоматом* A.

Указанное отображение φ однозначно определяется заданием двух функций δ и λ, называемых соответственно *функцией переходов* и *функцией выходов* рассматриваемого автомата A.

Функция переходов определяет состояние $a(t)$ автомата в любой момент дискретного автоматного времени t по входному сигналу $x(t)$ в тот же самый момент и состоянию $a(t-1)$ в предыдущий момент автоматного времени

$$a(t) = \delta(a(t-1),\ x(t)). \tag{44}$$

Функция выходов определяет зависимость выходного сигнала $y(t)$ автомата от тех же самых переменных

$$y(t) = \lambda(a(t-1),\ x(t)). \tag{45}$$

Задавая любое входное слово $p = x(1)\, x\,(2) \ldots x(k)$ и начальное состояние $a(0)$ автомата, с помощью соотношений (44) и (45) можно последовательно определить все буквы соответствующего выходного слова

$$q = \varphi(p) = y(1)\, y(2) \ldots y(k).$$

Таким образом, соотношения (44) и (45) действительно определяют индуцируемое автоматом отображение φ.

Функции переходов и выходов представляются обычно абстрактными частичными функциями $\delta(a, x)$ и $\lambda(a, x)$, задающими однозначные отображения некоторого множества пар (a, x) $(a \in \mathfrak{A},\ x \in \mathfrak{X})$ в множества $\mathfrak{A}$ и $\mathfrak{Y}$ соответственно. Допустимыми входными словами называются все те и только те входные слова p, на которых с помощью функций δ и λ описанным выше способом определяются соответствующие им выходные слова $\varphi(p)$.

Автомат называется *конечным*, если конечны все три определяющих его множества $\mathfrak{A}$, $\mathfrak{X}$, $\mathfrak{Y}$. Поскольку мы ограничиваемся рассмотрением почти исключительно конечных автоматов, слово «конечный» часто опускается. Автомат называется *вполне определенным*, если его функции переходов и выходов заданы на всех парах (a, x), и *частичным* — в противном случае.

Конечные автоматы принято задавать двумя таблицами, называемыми соответственно таблицей переходов и таблицей выхо-

дов автомата. Строки обеих таблиц обозначаются различными буквами входного алфавита $\mathfrak{X}$ автомата, а столбцы — различными его состояниями. На пересечении x-ой строки и a-го столбца таблицы переходов стоит элемент $\delta(a, x)$, т. е. некоторое состояние автомата из множества его внутренних состояний, а на пересечении x-ой строки и a-го столбца таблицы выходов — элемент $\lambda(a, x)$, т. е. некоторая буква выходного алфавита $\mathfrak{Y}$ автомата. Тем самым задание таблиц переходов и выходов определяет как множества $\mathfrak{X}$, $\mathfrak{Y}$, $\mathfrak{A}$, так и функции переходов и выходов автомата. Для фиксирования начального состояния принято обычно обозначать этим состоянием первый слева столбец обеих упомянутых таблиц. Таким образом, удается задавать с помощью двух таблиц произвольные конечные автоматы, в том числе и инициальные.

Другим способом задания конечных автоматов, обеспечивающим большую наглядность, является задание автоматов с помощью *направленных графов*. Вершины графа (изображаемые на рисунках в виде кружочков) отождествляются с различными состояниями автомата. Стрелка, соединяющая вершину i с вершиной j, означает, что существует входной сигнал x, переводящий автомат из состояния i в состояние j, т. е. удовлетворяющий соотношению

$$j = \delta(i, x).$$

Чтобы отличать, какими именно входными сигналами вызывается данный переход автомата из состояния i в состояние j, стрелка, соединяющая соответствующие этим состояниям вершины графа, отмечается символами этих входных сигналов. Выходной сигнал y, определяемый парой (i, x), обычно ставится на графе рядом с входным сигналом x и для отличия от входных сигналов заключается в скобки.

Рассмотрим пример задания конечного автомата с помощью таблиц переходов и выходов направленного графа. Выберем для этой цели относительно простой автомат с тремя внутренними состояниями 1, 2, 3, двумя входными сигналами x, y и двумя выходными сигналами u, v. Предположим, что указанный автомат задается таблицами переходов и выходов

	1	2	3
x	2	3	3
y	3	2	2

;

	1	2	3
x	u	u	v
y	v	u	u

.

Этим таблицам соответствует направленный граф, изображенный на рис. 8

Автоматы, рассмотренные нами выше, принято обычно называть *автоматами Мили* (по имени ученого, впервые рассмотревшего некоторые вопросы, связанные с функционированием таких автоматов; см. [55]). На практике часто приходится иметь дело

также с несколько иначе определяемыми автоматами, называемыми *автоматами Мура* (см. [57]).

Автоматы Мура отличаются от автоматов Мили только способом определения их функций выходов. Вместо соотношения

$$y(t) = \lambda(a(t-1),\ x(t)),$$

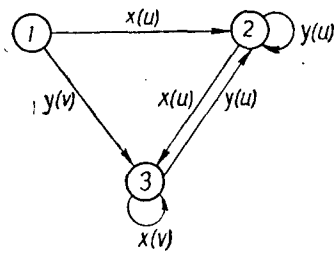

Рис. 8.

определяющего выходной сигнал для автоматов Мили, в случае автоматов Мура используется несколько иное соотношение

$$y(t) = \mu(a(t)). \qquad (46)$$

С помощью соотношения (46) и выписанного соотношения (44), как и в предыдущем случае, определяется отображение, индуцируемое любым данным автоматом Мура.

В силу причин, которые будут рассмотрены ниже, функцию $y = \mu(a)$ назовем *сдвинутой функцией выходов автомата Мура*. Значение этой функции на любом состоянии a принято называть *отметкой* этого состояния. Конечные автоматы Мура удобно задавать с помощью так называемых *отмеченных таблиц переходов*. Отмеченная таблица переходов представляет собой не что иное, как обычную таблицу переходов автомата, у которой над символами состояний, обозначающих различные столбцы таблицы, проставлены отметки этих состояний. Например, отмеченная таблица переходов

	u	u	v
	1	2	3
x	2	3	3
y	3	2	2

задает автомат Мура, имеющий ту же самую таблицу переходов, что и автомат Мили, у которого состояниям 1 и 2 соответствует выходной сигнал u, а состоянию 3 — выходной сигнал v. При изображении автоматов Мура с помощью графов символы выходных сигналов отмечают соответствующие им вершины графа, а не ребра, как в случае автоматов Мили.

Условимся считать, что выдача выходных сигналов у автомата Мура начинается с момента времени $t = 1$ (а не с момента времени $t = 0$). При этом условии для любого автомата Мура A_1 нетрудно построить автомат Мили A_2, имеющий ту же самую таблицу переходов и индуцирующий то же самое отображение, что и автомат A_1.

Действительно, если $\delta(a, x)$ — функция переходов и $\mu(a)$ — сдвинутая функция выходов автомата Мура A_1, то можно опреде-

лить автомат Мили A_2, задав его функцией переходов $\delta(a, x)$ и функцией выходов $\lambda(a, x) = \mu(\delta(a, x))$. Тогда

$$y(t) = \lambda(a(t-1), x(t)) = \mu(\delta(a(t-1), x(t))) = \mu(a(t)),$$

чем и доказывается, что автоматы A_1 и A_2 совершенно одинаково реагируют на любую последовательность входных сигналов. Описанное построение назовем интерпретацией заданного автомата Мура *как автомата Мили*. Физический смысл подобной интерпретации (в реальных автоматах) состоит в сдвиге автоматного времени на один элементарный промежуток времени, в силу чего в построенном автомате Мили A_2 выходные сигналы опережают на одну единицу автоматного времени соответствующие им выходные сигналы автомата Мура A_1. Именно поэтому функции выходов автоматов Мура называются сдвинутыми функциями выходов.

Описанный временной сдвиг дает возможность рассматривать автоматы Мура как частный случай автоматов Мили всякий раз, когда нас интересует не истинное время появления того или иного выходного сигнала, а лишь порядок следования выходных сигналов во времени. Именно с таким положением мы сталкиваемся в *абстрактной теории автоматов*, интересующейся лишь отображениями, индуцируемыми автоматами и переходами в их памяти, а не способом составления данного автомата из имеющихся в нашем распоряжении *элементарных* автоматов.

При решении последнего вопроса, составляющего предмет так называемой *структурной теории автоматов*, автоматы Мура приходится рассматривать как отдельный класс автоматов, не являющийся собственным подклассом класса всех автоматов Мили. Различие между этими двумя классами автоматов в структурной теории обусловливается тем, что в автоматах Мили выходной сигнал возникает одновременно с индуцирующим его входным сигналом, а в автоматах Мура — с *задержкой* на одну единицу автоматного времени.

Возможность интерпретации всякого автомата Мура как автомата Мили в абстрактной теории автоматов не означает, разумеется, наличия обратной возможности. Тем не менее для любого автомата Мили A можно построить автомат Мура B, который будет индуцировать то же самое отображение, что и автомат A. При этом, в отличие от предыдущего, множество состояний автомата B не будет, вообще говоря, совпадать с множеством состояний автомата A, хотя и будет конечным всякий раз, когда конечно последнее множество.

Действительно, пусть задан произвольный автомат Мили A с множеством состояний $\mathfrak{A}$, входным алфавитом $\mathfrak{X}$, выходным алфавитом $\mathfrak{Y}$, функцией переходов $\delta(a, x)$, функцией выходов $\lambda(a, x)$ и начальным состоянием a_0. Условимся для простоты обозначений здесь и далее вместо функции переходов употреблять символ умножения, обозначая значение функции $\delta(a, x)$ произведением ax.

Построим автомат Мура B, выбрав в качестве множества $\mathfrak{B}$ его состояний множество, состоящее из начального состояния a_0 и множества всевозможных пар (a, x), где $a \in \mathfrak{A}$, $x \in \mathfrak{X}$. Входной и выходной алфавиты автомата B совпадают соответственно с входным и выходным алфавитами автомата A. Функцию переходов автомата B определим, полагая

$$a_0 x = (a_0, x) \text{ и } (a, x_i) x_j = (a x_i, x_j).$$

Сдвинутую функцию выходов $\mu(b)$ автомата B определим на каждом состоянии $b = (a, x)$, отличном от начального состояния a_0, с помощью соотношения $\mu(b) = \lambda(a, x)$. На начальном состоянии значение функции μ может быть выбрано произвольно. В результате построен некоторый автомат Мура B.

Нетрудно понять, что автомат B индуцирует то же самое отображение, что и автомат A. В самом деле, обозначим буквой φ отображение, индуцируемое автоматом A, а буквой ψ — отображение, индуцируемое автоматом B. Пусть для любого входного слова $p = p_1 x_i$ длины $n \geqslant 1$ уже доказано, что $\varphi(p) = \psi(p) = q$ (для входного слова длины 1, т. е. для любого однобуквенного слова x, очевидно, $\varphi(x) = \psi(x) = \lambda(a_0, x)$).

Рассмотрим реакции обоих автоматов на произвольное слово px_j длины $n + 1$. Условимся здесь и далее словом al обозначать состояние, в которое перейдет автомат, находящийся первоначально в состоянии a, если на его вход последовательно, буква за буквой, подать произвольное входное слово l. В силу определения функции переходов в автомате B, $a_0 p = (a_0 p_1, x_i)$. После подачи на автоматы A и B входного сигнала x_j автомат A выдает выходной сигнал $y = \lambda(a_0 p, x_j)$. Автомат B, очевидно, перейдет в состояние

$$b = (a_0 p_1 x_i, x_j) = (a_0 p, x_j)$$

и выдаст выходной сигнал $\mu(b)$, равный, в силу определения функции μ, сигналу $\lambda(a_0 p, x_j)$.

Тем самым показано, что автоматы A и B одинаково реагируют на любые входные слова длины $n + 1$. Производя индукцию по n, приходим к заключению, что отображения, индуцируемые автоматами A и B, одинаковы. Этот вывод справедлив не только для обычных (вполне определенных) автоматов, но и для частичных автоматов.

Охарактеризуем более подробно отображения, индуцируемые автоматами. Заметим, что требование прихода входного сигнала и выдача выходного сигнала в *каждый* момент автоматного времени, не выполняющееся, на первый взгляд, во многих конкретных автоматах, в действительности легко удовлетворяется, если ввести специальные буквы для обозначения *пустых* входных и выходных сигналов (т. е. отсутствия каких-либо реальных физических сигналов) и рассматривать эти буквы наравне с другими буквами входного и выходного алфавитов.

Легко видеть, что отображение φ, индуцируемое произвольным автоматом Мура или Мили, удовлетворяет двум условиям:

1) *любому слову l во входном алфавите $\mathfrak{X}$ отображение φ сопоставляет слово $\varphi(l)$ в выходном алфавите $\mathfrak{Y}$, имеющее одинаковую со словом l длину;*

2) *если слово l_1 совпадает с начальным отрезком слова l, то слово $\varphi(l_1)$ является начальным отрезком слова $\varphi(l)$.*

Назовем сформулированные условия *условиями автоматности* отображения φ, а всякое соответствие между словами в алфавитах $\mathfrak{X}$ и $\mathfrak{Y}$, удовлетворяющее этим условиям, — *автоматным отображением*, или *автоматным оператором*.

Нетрудно показать, что *всякое автоматное отображение может быть индуцировано с помощью некоторого абстрактного автомата (не обязательно конечного)*.

Пусть автоматное соответствие φ отображает множество слов в алфавите $\mathfrak{X} = (x_1, x_2, \ldots, x_n)$ в множество слов в алфавите $\mathfrak{Y} = (y_1, y_2, \ldots, y_m)$. Построим автомат A, внутренними состояниями которого будут всевозможные слова в алфавите $\mathfrak{X}$, причем начальным состоянием будет служить пустое слово e (слово нулевой длины, состоящее из пустого множества букв). Функция переходов δ определяется тривиально: если l — любое состояние автомата (слово в алфавите $\mathfrak{X}$), а x_i — любой входной сигнал, то значение функции $\delta(l, x_i)$ полагается равным слову lx_i. Определив функцию выходов λ соотношением $\lambda(l, x_i) = y_j$, где y_j — последняя буква слова $\varphi(lx_i)$, получим автомат, который реализует исходное отображение φ.

Если отображение φ множества слов в алфавите $\mathfrak{X}$ в множество слов в алфавите $\mathfrak{Y}$ задается частичным автоматом, то оно будет, разумеется, лишь частичным отображением, определенным не на всех словах. Однако для него по-прежнему будут выполняться оба условия автоматности при дополнительном предположении, что $\varphi(l)$ существует. Второе условие автоматности при этом приобретает усиленную форму: *если $\varphi(l)$ существует, а l_1 — начальный отрезок слова l, то $\varphi(l_1)$ существует и совпадает с некоторым начальным отрезком слова $\varphi(l)$.*

Назовем перефразированные условия условиями *автоматности частичного отображения* φ, а всякое частичное отображение, удовлетворяющее этим условиям, — *частичным автоматным отображением*.

Легко установить справедливость следующего предложения.

Теорема 1. *Всякое частичное автоматное отображение может быть индуцировано с помощью некоторого частичного автомата (не обязательно конечного)*.

Это предложение доказывается точно таким же способом, как и в случае полного отображения. Различие заключается в том, что состояниями частичного автомата считаются не все слова входного алфавита, а лишь те из них, на которых определено отображение φ.

Условия автоматности на первый взгляд очень суживают класс отображений, которые можно задавать с помощью абстрактных автоматов. Хорошо известно, в частности, что требование равенства длин входных и выходных слов не удовлетворяется для большей части алгоритмов, которые должны выполняться теми или иными конкретными автоматами. Это затруднение, представляющееся на первый взгляд весьма серьезным, в действительности легко устраняется с помощью перекодирования входной и выходной информаций на основе весьма простого приема.

Стандартный прием превращения любого частичного соответствия φ между словами в алфавитах $\mathfrak{X}$ и $\mathfrak{Y}$ в частичное автоматное соответствие основан на введении в алфавиты $\mathfrak{X}$ и $\mathfrak{Y}$ не содержащейся в них ранее буквы α, которую назовем *пустой буквой*. Появление пустой буквы на входе автомата соответствует случаю, когда на вход автомата в действительности ничего не подается. Аналогично появление пустой буквы в качестве выходного сигнала означает отсутствие какого-либо сигнала на выходе автомата.

Рассмотрим произвольное слово l длины n в алфавите $\mathfrak{X}$, которому первоначально заданное частичное отображение φ сопоставляет слово $q = \varphi(l)$ в алфавите $\mathfrak{Y}$, имеющее длину m. Обозначим буквой l_1 слово в алфавите $\mathfrak{X}_1 = \mathfrak{X} \cup (\alpha)$, получающееся в результате приписывания к слову l *справа* m экземпляров буквы α. Аналогично буквой q_1 обозначим слово в алфавите $\mathfrak{Y}_1 = \mathfrak{Y} \cup (\alpha)$, получающееся в результате приписывания *слева* к слову q n экземпляров буквы α. Назовем этот прием *стандартным приемом выравнивания длин слов.*

Определим новое частичное отображение φ_1 между словами в алфавитах $\mathfrak{X}_1$ и $\mathfrak{Y}_1$, полагая $q_1 = \varphi_1(l_1)$ и повторяя этот прием для любого слова l в алфавите $\mathfrak{X}$, на котором отображение φ определено. Доопределим это соответствие на всех начальных отрезках $l_1^{(i)}$ слова l_1, полагая, что $\varphi_1(l_1^{(i)})$ совпадает с начальным отрезком слова $\varphi_1(l_1)$, имеющим равную с $l_1^{(i)}$ длину.

При таком доопределении возникает угроза потери однозначности отображения φ, поскольку слово $l_1^{(i)}$ может входить в качестве начального отрезка не только в исходное слово l_1, но и в другие слова, например в слово s_1, полученное в результате применения стандартного приема выравнивания длин слов из некоторого слова s в алфавите $\mathfrak{X}$.

Так как слово s_1 имеет вид $s_1 = s\alpha\alpha \ldots \alpha$, а слово l_1 — вид $l_1 = l\alpha\alpha \ldots \alpha$, где слова s и l не содержат буквы α, то $p = s = l$, если слово $l_1^{(i)}$ имеет справа хотя бы одну букву α : $l_1^{(i)} = p\alpha \ldots$. В этом случае, следовательно, слова s_1 и l_1 должны совпадать между собой и опасности возникновения неоднозначности нет.

Остается, таким образом, рассмотреть случай, когда слово $l_1^{(i)} = p$ состоит исключительно из букв алфавита $\mathfrak{X}$. В этом случае длина слова p не превосходит, очевидно, длин слов l и s. Но тогда, в силу стандартного приема выравнивания длин слов, начальные

отрезки слов $\varphi(l)$ и $\varphi_1(s_1)$, имеющие равную со словом $l_1^{(i)} = p$ длину, состоят сплошь из буквы a и, следовательно, совпадают между собой. Таким образом, возникновение неоднозначности исключено и в этом случае.

Построенное нами частичное отображение φ_1 между словами в алфавитах $\mathfrak{X}_1$ и $\mathfrak{Y}_1$ по самому способу построения удовлетворяет обоим условиям автоматности для частичных отображений и представляет собой, следовательно, искомое частичное автоматное отображение.

Описанный прием преобразования любого частичного отображения в автоматное является общим, однако, именно в силу своей общности, он не всегда приводит к самому экономному (с точки зрения расходования дополнительных букв) решению. Это обстоятельство особенно легко уяснить на случае, когда само исходное частичное отображение φ удовлетворяло обоим условиям автоматности. Ясно, что самым экономным решением в рассматриваемом случае будет $\varphi_1 = \varphi$. Между тем описанный стандартный прием (который применим, разумеется, и в этом случае) приведет к ненужному увеличению длин исходных слов, участвующих в соответствии.

Таким образом, найденный общий прием не избавляет от необходимости поиска более экономных решений. Такие экономные решения обычно находят, добавляя пустые буквы к словам не сразу в том количестве, которое предусматривается стандартным приемом выравнивания длин слов, а постепенно, шаг за шагом, проверяя на каждом шаге выполнение условий автоматности и останавливаясь, как только они окажутся выполненными в первый раз. Подобный *усовершенствованный* прием выравнивания длин слов рано или поздно приведет к возникновению автоматного отображения.

Большой интерес представляет проблема нахождения экономного перекодирования отображения, заданного на том или ином алгоритмическом языке (например, на языке нормальных алгоритмов), с целью превращения его в автоматное соответствие, а также задача построения теории алгоритмов, которые удовлетворяют условиям автоматности и поэтому сокращенно называются *автоматными алгоритмами*. Один из возможных подходов к теории автоматных алгоритмов развивается в следующем параграфе.

§ 2. События и представление событий в автоматах

Пусть A — произвольный (вообще говоря, частичный) инициальный автомат, φ — индуцируемое им отображение. Для каждой буквы y_i выходного алфавита $\mathfrak{Y} = (y_1, y_2, \ldots, y_m)$ автомата A рассмотрим множество R_i всех слов l во входном алфавите $\mathfrak{X} = (x_1, x_2, \ldots, x_n)$ этого автомата, для которых слово $\varphi(l)$ определено и оканчивается буквой y_i.

Назовем определенное таким образом множество R_i событием, представленным в (частичном) автомате A выходным сигналом y_i ($i = 1, 2, \ldots, m$) Если M — любое множество выходных сигналов,

то событием, представленным в частичном автомате A множеством M, будем называть объединение событий, представляемых всеми элементами этого множества.

Легко видеть, что множества R_i попарно не пересекаются, а множество S всех слов в алфавите $\mathfrak{X}$, не вошедших ни в одно из множеств R_i ($i = 1, 2, \ldots, m$), состоит из всех запрещенных для данного частичного автомата слов. Запрещенными здесь и далее будем называть все слова во входном алфавите, которые при подаче их на вход данного частичного автомата приведут хотя бы для одного составляющего их входного сигнала к не определенному в автомате выходному сигналу. Совокупность всех запрещенных слов S условимся называть *областью запрета* данного частичного автомата A. Условимся еще событием в алфавите $\mathfrak{X}$ называть любое множество слов в этом алфавите.

Исходя из введенных определений, можно сформулировать полученный выше результат в виде следующего предложения.

Теорема 1. *Задание частичного автоматного отображения* φ, *реализуемого частичным автоматом A с входным алфавитом* $\mathfrak{X} = (x_1, x_2, \ldots, x_n)$ *и с выходным алфавитом* $\mathfrak{Y} = (y_1, y_2, \ldots, y_m)$, *однозначно определяет разбиение множества F всех слов в алфавите* $\mathfrak{X}$ *на* $m + 1$ *попарно не пересекающихся событий в алфавите* $\mathfrak{X}$, *а именно событий* $R_1, R_2, \ldots, R_m$, *представленных в автомате A выходными сигналами* $y_1, y_2, \ldots, y_m$, *и область запрета S данного (частичного) автомата A.*

И наоборот: зная события $R_1, R_2, \ldots, R_m$, представленные в некотором частичном автомате A выходными сигналами $y_1, y_2, \ldots, y_m$, можно однозначно восстановить реализуемое этим автоматом частичное отображение φ между словами входного алфавита $\mathfrak{X}$ и выходного алфавита $\mathfrak{Y}$, *не пользуясь* функциями переходов и выходов автомата.

Пусть дано произвольное слово $l = x_{i_1}x_{i_2} \ldots x_{i_n}$ в алфавите $\mathfrak{X}$. Для каждого $k (1 \leqslant k \leqslant n)$ найдем выходной сигнал y_{j_k} по правилу: y_{j_k} есть выходной сигнал, представляющий в автомате A событие R_{j_k}, которое содержит начальный отрезок $x_{i_1}x_{i_2} \ldots x_{i_k}$ длины k слова l. Если для всех $k = 1, 2, \ldots, n$ существуют соответствующие им y_{j_k}, то полагаем $\varphi(l) = \varphi(x_{i_1}x_{i_2} \ldots x_{i_n}) = y_{j_1}y_{j_2} \ldots y_{j_n}$. В случае же несуществования хотя бы для одного $k = 1, 2, \ldots, n$ выходного сигнала y_{j_k} с требуемыми свойствами полагаем, что частичное отображение φ на слове l не определено.

Нетрудно видеть, что в силу определения событий, представленных в автомате, введенное таким образом частичное отображение φ как раз и будет тем частичным отображением, которое индуцируется заданным частичным автоматом A.

На основании изложенного можно сформулировать следующее предложение.

Теорема 2. *Задание частичного автоматного отображения* φ *между словами в алфавитах* $\mathfrak{X}$ *и* $\mathfrak{Y} = (y_1, y_2, \ldots, y_m)$ *эквивалент-*

но заданию событий R_1, R_2, ..., R_m, представленных в индуцирующем отображение φ частичном автомате A выходными сигналами y_1 y_2, ... , y_m.

Теорема 2 подводит базу под изучение автоматных отображений (в частности, автоматных алгоритмов). Для описания таких отображений достаточно задавать разбиение множества всех слов входного алфавита на конечное число попарно не пересекающихся событий. Для того чтобы соответствующие описания носили конструктивный характер, необходимо ограничиться рассмотрением лишь таких событий, которые допускают эффективное описание.

Естественно, что простое конструктивное описание допускают прежде всего конечные события, т. е. события, состоящие из конечного числа слов. Их можно описывать с помощью простого перечисления входящих в них элементов. Для характеристики некоторых важных классов бесконечных событий целесообразно ввести ряд операций на множестве событий, превратив тем самым это множество в алгебру — алгебру событий.

Для наших целей наиболее удобной является система из трех операций, представляющих собой модификацию операций, введенных впервые Клини [40] (см. также Копи, Элгот, Райт [45] и Глушков [21]).

Первой операцией является операция теоретико-множественного объединения событий. Будем обозначать эту операцию символом $\vee$ и называть *дизъюнкцией событий*.

Второй операцией является операция *умножения событий*, которую не следует путать с операцией теоретико-множественного пересечения. Если событие S состоит из слов l_α ($\alpha \in M$), а событие R — из слов q_β($\beta \in N$), то *произведением событий* S и R называется событие, состоящее из всевозможных слов вида $l_\alpha q_\beta$ ($\alpha \in M$, $\beta \in N$). Операция умножения событий некоммутативна: вообще говоря, события SR и RS различны.

Третьей операцией является так называемая *итерация события*, для обозначения которой будем употреблять фигурные скобки, так что $\{S\}$ обозначает итерацию события S. Итерация любого события S определяется как объединение пустого слова, события $S = S^1$, события $S \cdot S = S^2$, события $S \cdot SS = S^3$ и т. д. до бесконечности. Иначе говоря, если событие S состоит из слов $l_\alpha (\alpha \in M)$, то его итерация $\{S\}$ состоит из всевозможных слов, имеющих вид $l_{\alpha_1} l_{\alpha_2} \ldots l_{\alpha_n}$, где $\alpha_1, \alpha_2, \ldots, \alpha_n \in M$, а $n = 0, 1, 2, 3, \ldots$.

Фигурные скобки, применяемые для обозначения итерации, будем называть *итерационными скобками*. Для обозначения порядка действий будем пользоваться круглыми скобками, которые назовем *обыкновенными*. При отсутствии скобок, изменяющих обычный порядок действий, первыми должны выполняться итерации, затем умножения и, наконец, дизъюнкции.

Условимся одноэлементные события, т. е. события, состоящие из одного слова, обозначать символом этого слова. Если $\mathfrak{X} =$

$= (x_1, x_2, \ldots, x_n)$, то *элементарными событиями в этом алфавите* называются $m + 1$ одноэлементных событий $x_1, x_2, \ldots, x_m, e$.

Здесь и далее буквой e будем обозначать *пустое* слово, состоящее из пустого множества букв и имеющее, следовательно, нулевую длину. Это слово играет лишь служебную, вспомогательную роль. Условимся, в частности, не считать различными события, отличающиеся между собой лишь пустым словом. Таким образом, слово можно по желанию либо присоединить, либо исключить из любого рассматриваемого события. Это связано с тем, что в силу принятых нами определений пустое слово не может быть представлено в автомате.

Введем теперь понятие, являющееся одним из центральных во всех последующих рассмотрениях.

Регулярным событием в конечном алфавите $\mathfrak{X} = (x_1, x_2, \ldots, x_n)$ *называется любое событие, которое может быть получено из элементарных событий* $x_1, x_2, \ldots, x_m$, e *в этом алфавите с помощью применения конечного числа операций дизъюнкции, умножения и итерации.*

Это определение восходит к определению регулярного события, данному еще Клини [40], хотя по форме существенно отличается от него (см. Глушков [21]). Заметим, что одно и то же событие может быть по-разному представлено через элементарные события. Каждое такое представление (формулу алгебры событий) будем в дальнейшем называть *регулярным выражением.*

Одной из основных задач в алгебре событий является установление законов *эквивалентных преобразований* регулярных выражений, т. е. таких преобразований, которые не изменяют представляемых этими выражениями событий (с точностью до пустого слова e).

К законам, особенно часто применяющимся при эквивалентных преобразованиях в алгебре событий, относятся законы ассоциативности для дизъюнкции и для умножения, закон коммутативности для дизъюнкции, левый и правый дистрибутивный законы для умножения по отношению к дизъюнкции ($S(R \vee Q) = SR \vee SQ$, $(R \vee Q)S = (RS \vee QS)$ и др.).

Законы дистрибутивности позволяют, в частности, раскрывать скобки и выносить общие множители за скобки (как в обычной алгебре). Следует лишь помнить при этом, что умножение в алгебре событий, вообще говоря, некоммутативно.

Любое слово можно представить как произведение элементарных событий — отдельных букв, составляющих это слово. Любое же конечное событие представляется в виде дизъюнкции составляющих его слов. Отсюда, в частности, следует, что все конечные события регулярны.

Использование итерации приводит к построению бесконечных регулярных событий. Вместе с тем нетрудно построить простые примеры нерегулярных бесконечных событий. Для этого достаточно выбрать такую возрастающую последовательность целых

чисел $n_1, n_2, \ldots, n_i, \ldots$, что разности $n_{i+1} - n_i (i = 1, 2, \ldots)$ не ограничены в совокупности (этому условию удовлетворяет, например, последовательность квадратов чисел натурального ряда), и в любом входном алфавите $\mathfrak{X}$ построить событие S, состоящее из всех слов в алфавите $\mathfrak{X}$, имеющих длины, равные n_1, n_2 и т. д.

Построенное таким образом событие S обязательно нерегулярно. Действительно, допуская противное, можно было бы найти для S некоторое регулярное выражение R. Поскольку событие S бесконечно, это выражение содержит хотя бы одни итерационные скобки, заключающие внутри себя выражение, отличное от пустого слова e. Заменим все остальные итерационные скобки в выражении R пустым словом, а выделенные скобки — выражением $\{p\}$, где p — произвольное непустое слово из события, заключенного в выделенные скобки. В результате получим регулярное выражение R_1 для некоторого события, содержащегося в событии S.

Из выражения R_1 непосредственно следует, что в событие S входят слова вида rs, rps, $rpps$, $rppps$, ..., длины которых составляют бесконечную возрастающую арифметическую прогрессию. Но это противоречит способу построения события S. Следовательно, событие не может быть представлено никаким регулярным выражением, т. е. является нерегулярным событием.

Определим еще понятие *циклической глубины* регулярного выражения, подразумевая под ней максимальное число вложенных друг в друга пар итерационных скобок, содержащихся в этом выражении. Например, выражение $\{x\{y\}\{x\}\}$ имеет циклическую глубину 2, а выражение $\{x \vee y\}x\{y\}$ — циклическую глубину 1. Под *циклической глубиной регулярного события* условимся подразумевать минимальную циклическую глубину представляющих его регулярных выражений.

Регулярные сообытия имеют особое значение для абстрактной теории автоматов, поскольку класс *регулярных событий совпадает с классом событий, представимых в конечных автоматах.* В последующих параграфах докажем это важное предложение; здесь же рассмотрим вопрос о соотношении классов событий, представимых в автоматах Мили и Мура.

Общее определение представления событий в автомате, данное в начале настоящего параграфа, относилось к автомату Мили. Поскольку автомат Мура является частным случаем автоматов Мили, то это определение применимо в полной мере и к нему. Однако на практике для автоматов Мура удобно представлять события не свойством выхода в момент подачи последнего входного сигнала слов, составляющих события, а свойством состояния автомата, в котором он окажется после поступления на вход автомата слова того или иного события.

Иначе говоря, принято считать, что в случае автоматов Мура события представляются некоторыми *множествами состояний автомата.* В силу определения автоматов Мура, этот способ представления событий совершенно эквивалентен способу представле-

ния событий множествами выходных сигналов. Разница состоит лишь в том, что при представлении событий множествами состояний автомата оказывается представимым (с помощью начального состояния) пустое слово e, которое не может быть представлено никаким выходным сигналом (если, разумеется, не начинать счет времени с отрицательных моментов времени).

Однако выше мы условились не считать различными события, отличающиеся друг от друга лишь пустым словом e. Поэтому оба способа представления событий (состояниями или выходными сигналами) в случае автоматов Мура являются действительно эквивалентными.

Поскольку автоматы Мура в абстрактной теории можно рассматривать как частный случай автоматов Мили, кажется естественным, что класс событий, представимых в автоматах Мура, беднее класса событий, представимых в автоматах Мили. В действительности это не так.

Предположим, что некоторое событие S представлено в некотором автомате Мили A множеством M его выходных сигналов. Нетрудно понять, что событие S может быть представлено некоторым множеством внутренних состояний автомата Мура B (индуцирующего то же самое отображение φ, что и автомат A), который был построен в предыдущем параграфе.

Напомним, что состояниями автомата B являются всевозможные пары (a, x), составленные из состояний a автомата A и букв x его входного алфавита $\mathfrak{X}$, а также начальное состояние a_0 автомата A. Сдвинутая функция выходов μ автомата B на начальном состоянии a_0 определяется произвольно, а на состоянии $b=(a, x)$ — с помощью соотношения $\mu(b)=\lambda(a, x)$, где $\lambda(a, x)$ — функция выходов автомата A.

Если $h=gx_j$ — произвольное непустое входное слово, то в автомате A последней буквой соответствующего ему выходного слова будет, очевидно, буква $y=\lambda(a_0g, x_j)$. Автомат B, как нетрудно видеть, будет переведен словом h из начального состояния a_0 в состояние (a_0g, x_j).

Таким образом, все непустые слова исходного события будут представлены в автомате B множеством K всевозможных состояний (a_i, x_j), для которых справедливо соотношение $\lambda(a_i, x_j) \in M$.

§ 3. Анализ конечных автоматов

Проблема анализа состоит в определении событий, представляемых в автомате множествами выходных сигналов (в случае автоматов Мили) или множествами внутренних состояний (в случае автоматов Мура). Поскольку всякий автомат Мура можно интерпретировать как автомат Мили, достаточно научиться анализировать лишь автоматы Милн.

Будем решать проблему анализа лишь для случая конечных автоматов Мили. Все представляемые в таких автоматах события

непременно регулярны. Алгоритм анализа применяется к таблицам переходов и выходов анализируемого автомата, а в качестве заключительной информации дает регулярные выражения для событий, представимых каждым из выходных сигналов автомата. Событие, представляемое произвольным множеством выходных сигналов, запишется тогда как дизъюнкция событий, представленных отдельными выходными сигналами, составляющими данное множество.

Рассмотрим произвольный конечный автомат Мили A с множеством внутренних состояний $(a_1, a_2, \ldots, a_p)$ с входным алфавитом $\mathfrak{X} = (x_1, x_2, \ldots, x_n)$ и выходным алфавитом $\mathfrak{Y} = (y_1, y_2, \ldots, y_m)$.

Считая заданными начальное состояние a_1, функцию переходов $\delta(a_i, x_j)$ и функцию выходов $\lambda(a_i, x_j)$ автомата A, будем искать регулярное выражение R для события, представленного каким-либо выходным сигналом, скажем сигналом y_1. Выпишем внутренние состояния автомата $a_1, a_{j_1}, \ldots, a_{j_k}$, в которые автомат A переводится из начального состояния a_1 последовательными начальными отрезками e, x_{i_1}, $x_{i_1} x_{i_2}$, ..., $x_{i_1} x_{i_2} \ldots x_{i_k}$ некоторого входного слова q. Вставляя символы полученных состояний в слово q после соответствующих им начальных отрезков, преобразуем это слово в новое слово $q' = a_1 x_{i_1} a_{j_1} x_{i_2} \ldots a_{j_{k-1}} x_{i_k} a_{j_k}$, которое условимся называть *путем, соответствующим слову* q. Удаляя в заданном пути символы внутренних состояний a_j, получим входное слово, соответствующее данному пути.

Будем еще употреблять так называемые *урезанные пути*, получающиеся из обычных путей выбрасыванием крайнего правого символа внутреннего состояния a_{j_k}. Путь, соответствующий данному входному слову q, обозначим через q', а урезанный путь—q''.

Очевидно, что для принадлежности непустого входного слова q событию R_i, представляемому в автомате A выходным сигналом y_i, необходимо и достаточно, чтобы соответствующий слову q урезанный путь q' оканчивался парой $a_{j_{k-1}} x_{i_k}$, для которой функция выходов принимает значение, равное y_i. Назовем все такие (урезанные) пути *путями типа* y_i, или, обобщая (для любого i), *путями представляющего типа*.

Пути (неурезанные), соответствующие входным словам, переводящим автомат из какого-либо состояния a_j в то же самое состояние a_j, назовем *путями типа* a_j, или *путями циклического типа*. Если в некотором пути q' циклического типа a_j не содержатся символы каких-либо внутренних состояний $a_{k_1}, a_{k_2}, \ldots, a_{k_r}$, то путь q' будем называть также путем типа $a_j[a_{k_1}, a_{k_2}, \ldots, a_{k_r}]$ (символы, стоящие в квадратных скобках при этом называются запрещенными).

Путь q' произвольного типа называется *простым*, если соответствующий ему урезанный путь q'' не содержит двух одинаковых символов внутренних состояний. В конечном автомате существует лишь конечное число различных простых путей. Все простые пути

любого данного типа могут быть найдены непосредственно по таблице переходов или (для путей представляющего типа) по таблице переходов и таблице выходов автомата.

Построим некоторые вспомогательные события в алфавите $\mathfrak{Z} = (x_1, x_2, \ldots, x_n, a_1, a_2, \ldots, a_p)$, элементами которых являются урезанные пути в заданном автомате A. Событие $S(y_1)$ типа y_1 определим как событие, состоящее из всех (урезанных) путей типа y_1, простое событие $P(y_1)$ типа y_1 — как дизъюнкцию всех простых (урезанных) путей типа y_1. Простым событием $P(t)$ любого данного циклического типа $t = a_j[a_{k_1}, a_{k_2}, \ldots, a_{k_r}]$ $(j = 1, 2, \ldots, p, r \leqslant p - 1)$ будем называть итерацию дизъюнкции всех простых путей типа t (дизъюнкция пустого множества путей есть невозможное событие, итерация которого совпадает с пустым словом e), а событием $S(t)$ типа t — событие, состоящее из всех урезанных путей типа t. Наконец, *условно простым событием* $U(t)$ типа $t = a_j[a_{k_1}, a_{k_2}, \ldots, a_{k_r}]$ назовем итерацию части события $S(t)$, содержащей слова лишь с одним вхождением символа a_j.

Пусть дано некоторое множество (урезанных) путей типа y_1, заданное с помощью регулярного выражения Q. Вставляя в это регулярное выражение перед любым входящим в него символом внутреннего состояния a_j регулярное выражение события $S(a_j)$ типа a_j или события типа $a_j[a_{k_1}, a_{k_2}, \ldots, a_{k_r}] = t$, получим новое регулярное выражение, представляющее по-прежнему лишь пути типа y_1. Назовем такую операцию *вставкой* события $S(a_j)$ в событие Q.

Пусть теперь q'' — произвольный (урезанный) путь типа y_1. Первым (слева) символом внутреннего состояния, входящим в этот путь, будет символ a_1. Выделим также последнее (крайнее правое) вхождение символа a_1 в путь q'': $q'' = a_1 \ldots a_1 s$, где слово s уже не содержит символа a_1. Тогда путь q'' можно представить произведением некоторого числа слов условно простого события $U(a_1)$ типа a_1 и слова $a_1 s$.

В слове s выделим первый (слева) символ внутреннего состояния: $s = x_{i_k} a_{i_k} \ldots$; найдя также последнее вхождение этого символа в слово s, получим возможность представить слово s произведением буквы x_{i_k}, некоторого числа слов условно простого события типа $a_{i_k}[a_1]$ и некоторого слова $a_{i_k} r$, где r не содержит уже двух символов внутренних состояний a_1 и a_{i_k}.

Далее придем к выводу, что путь q'' содержится в событии, которое получается в результате вставки в некоторый простой путь типа y_1 перед единственной входящей в него буквой a_1 условно простого события типа a_1, а перед следующими по порядку символами внутренних состояний $a_{i_k}, a_{i_e}, \ldots$, входящими в этот путь, — условно простых событий типа $a_{i_k}[a_1]$, $a_{i_e}[a_1, a_{i_k}]$ и т. д.

Но точно такой же процесс, очевидно, может быть повторен со словами условно простых событий, которые были выделены из исходного пути q''. После этого придем к выводу, что путь q'' типа y_1

входит в событие, которое получается в результате вставки в простое событие $P(y_1)$ типа y_1 уже не условно простых, а обычных простых событий типов a_1, $a_{j_k}[a_1]$, ... и последующей вставки в пути, из которых состоят вставленные простые события, условно простых событий: для простого события типа a_1 — некоторых типов $a_x[a_1]$, $a_y[a_1, a_x]$, ..., для простого события типа $a_{j_k}[a_1]$ — некоторых типов $a_u[a_1, a_{jk},]$, $a_v\ [a_1 a_{jk} a_u]$,... и т. д.

Далее придем к выводу, что и на втором этапе можно вставлять не условно простые, а обычные простые события, вставляя, в свою очередь (уже на третьем этапе), в составляющие их слова условно простые события еще более высоких (в смысле числа запрещенных букв) циклических типов.

Увеличивая число этапов последовательных вставок, придем, в конце концов, к вставке событий циклических типов, у которых все буквы, кроме одной, являются запрещенными. Поскольку для такого рода типов разницы между условно простыми и обычными простыми событиями одинакового типа уже не существует, то процесс увеличения числа этапов и новых вставок тем самым завершается.

В результате приходим к выводу, что путь q'' входит в событие S_1, получающееся в результате конечного числа последовательных вставок (разбитых на ряд этапов) в простое событие типа y_1 простых событий все более и более высоких циклических типов. Ввиду произвольности выбора пути q'', событие S_1 содержит в себе событие $S(y_1)$.

Вместе с тем, как отмечалось выше, процесс вставок, подобный описанному процессу, не может привести к событию, содержащему пути отличного от y_1 типа. Следовательно, $S_1 = S(y_1)$, а описанный нами процесс вставок дает регулярное выражение R_1 для события $S(y_1)$, состоящее из всех путей типа y_1.

Опуская теперь в регулярном выражении R_1 все символы внутренних состояний (заменяя их пустым словом), получим регулярное выражение R, которое, как легко видеть, является не чем иным, как регулярным выражением для искомого события, представленного в автомате A выходным сигналом y_1.

Нами доказано следующее предложение.

Теорема 1. *Событие, представленное в произвольном конечном автомате Мили (а следовательно, и в произвольном конечном автомате Мура) любым множеством выходных сигналов, обязательно регулярно. Существует общий конструктивный прием (алгоритм анализа конечных автоматов), позволяющий находить регулярные выражения для событий, представленных множествами выходных сигналов в произвольном конечном автомате.*

Описанному алгоритму анализа конечных автоматов можно придать более удобную для практических приложений форму [22]. Для этого будем оперировать не событиями в множестве путей, а формальными выражениями, называемыми *комплексами*.

Для любого множества M выходных сигналов заданного конечного автомата Мили A *комплексом типа* M (или *комплексом*

выходного типа) называется дизъюнкция всех простых урезанных путей, кончающихся парами $a_j x_i$, которым соответствуют выходные сигналы, содержащиеся в множестве M. *Комплексом типа* $a_i[a_{i_1}, a_{i_2}, \ldots, a_{i_r}]$ (*комплексом циклического типа*) называется формальное выражение, получаемое в результате объединения знаком дизъюнкции всех простых путей типа $a_i[a_{i_1}, a_{i_2}, \ldots, a_{i_r}]$ с вычеркнутой буквой a_i и заключения полученного формального многочлена в итерационные скобки ($a_i, a_{i_1}, \ldots, a_{i_r}$ — любые попарно различные внутренние состояния автомата, а $0 \leqslant r \leqslant p-1$, где p — число внутренних состояний автомата A).

Первый шаг алгоритма анализа. По таблицам переходов и выходов автомата и данному (представляющему) множеству выходных сигналов путем перебора всех возможных вариантов простых путей находим комплекс $K(M)$ типа M и комплексы $K(a_i)$ для всех внутренних состояний a_i автомата A.

Второй шаг. По комплексам $K(a_i)$ исключением ненужных термов в итерационных скобках находим нужные для дальнейших построений комплексы высших циклических типов $a_i[a_{i_1}, a_{i_2}, \ldots, a_{i_r}]$ ($r \geqslant 1$).

Третий шаг. Исходя из комплекса типа M, последовательно заменяем все символы внутренних состояний a_i комплексами циклических типов до получения выражения R, не содержащего ни одного из символов внутренних состояний. Правило замены можно сформулировать так:

если путь $a_1 x_{i_1} a_{1_1} x_{i_2} a_{j_2} \ldots$ *входит в комплекс выходного типа* M, *то буква* a_1 *заменяется комплексом типа* a_1, *буква* a_{j_1} — *комплексом типа* $a_{j_1}[a_1]$, *буква* a_{j_2} — *комплексом типа* $a_{j_2}[a_1, a_{1_1}]$ *и т. д. Если член* $x_{i_1} a_{j_1} x_{i_2} a_{i_2} \ldots$ *входит в комплекс типа* $a_j[N]$, *где* N — *множество* (*быть может, пустое*) *внутренних состояний, отличных от* a_i, *то буква* a_{1_1} *заменяется комплексом типа* $a_{1_1}[a_1, N]$, *буква* a_{1_2} — *комплексом типа* $a_{1_2}[a_1, a_{1_1}, N]$ *и т. д.* —

На третьем шаге, в результате применения конечного числа раз правила замены, получаем искомое регулярное выражение R для события, представленного в автомате A множеством M выходных сигналов.

Из описанного алгоритма непосредственно вытекает следующее предложение.

Теорема 2. *Всякое событие, представленное в конечном автомате Мили* (*или Мура*), *имеющем* n *внутренних состояний, допускает регулярное выражение, циклическая глубина которого не превосходит* n.

В качестве примера найдем регулярное выражение для события S, представленного выходным сигналом v в автомате, таблицы переходов и выходов которого были выписаны в § 1 настоящей главы (его граф изображен на рис. 7).

Непосредственно по таблицам находим комплекс $K(v)$ типа v

$$K(v) = {}_1y \vee {}_1y_3x \vee {}_1x_2x_3x$$

и комплексы типов 1, 2 и 3

$$K(1) = e,\ K(2) = \{y \vee x_3y\},\ K(3) = \{x \vee y_2x\}.$$

Выпишем некоторые комплексы высших циклических типов:

$$K(2[1]) = K(2);\ K(3[1, 2]) = \{x\};$$

$$K(2[3]) = K(2[1, 3]) = \{y\};\ K(3[1]) = K(3).$$

Обозначая операцию вставки комплексов стрелкой, получим следующую последовательность вставок:

$$K(v) \to y \vee yK(3[1])x \vee xK(2[1])xK(3[1, 2])x =$$
$$= y \vee y\{x \vee y_2x\}x \vee x\{y \vee x_3y\}x\{x\}x \to y \vee y\{x \vee$$
$$\vee yK(2[1, 3])x\}x \vee x\{y \vee xK(3[1, 2])y\}x\{x\}x =$$
$$= y \vee y\{x \vee y\{y\}x\}x \vee x\ y \vee x\{x\}y\ x\{x\}x.$$

Последнее из полученных регулярных выражений и является искомым регулярным выражением для события S. Оно допускает преобразования и упрощения с использованием соотношений, существующих в алгебре событий.

§ 4. Абстрактный синтез конечных автоматов

Задача абстрактного синтеза противоположна задаче анализа конечных автоматов: необходимо найти эффективный метод, позволяющий по регулярным выражениям событий найти таблицы переходов и выходов какого-нибудь конечного автомата, представляющего эти события.

Задача синтеза автоматов Мура является более общей, чем задача синтеза автоматов Мили: поскольку всякий автомат Мура можно интерпретировать как автомат Мили, то, научившись синтезировать автоматы Мура, тем самым научимся синтезировать и автоматы Мили. Будем поэтому решать задачу синтеза автоматов Мура.

Пусть в некотором конечном алфавите $\mathfrak{X} = (x_1, x_2, \ldots, x_n)$ задано p регулярных выражений $R_1, R_2, \ldots, R_p$. Занумеруем все вхождения букв алфавита $\mathfrak{X}$ в выражения $R_1, R_2, \ldots, R_p$ последовательными натуральными числами, которые в дальнейшем будем называть *индексами соответствующих мест* этих выражений. Особо подчеркнем, что различные вхождения одной и той же буквы алфавита $\mathfrak{X}$ получают при этом различные индексы.

При разворачивании регулярного выражения в слово каждую из последовательно выписываемых букв этого слова отождествляем с тем или иным вхождением соответствующей буквы в разворачиваемое выражение. Условимся считать, что при таком отождествлении мы попадаем в те или иные места регулярного выражения, а именно: при отождествлении последней выписанной буквы с вхождением, занумерованным индексом j, будем счи-

тать, что находимся в j-ом месте соответствующего регулярного выражения. Говорят, что j-е место регулярного выражения x_k-следует за i-ым местом, если после отождествления последней буквы некоторого слова q с вхождением, имеющим индекс i, можно отождествить последнюю букву слова qx_k с вхождением, имеющим индекс j. В каждом регулярном выражении выделяется еще *начальное место*, которому приписывается индекс 0 (одинаковый для всех данных регулярных выражений). Если в процессе отождествления *первая* буква x_k некоторого слова отождествляется с каким-либо ее вхождением в регулярное выражение, имеющим индекс j, то считают, что j-е место x_k следует за нулевым (начальным) местом (общим для всех заданных регулярных выражений).

Наконец, если принадлежность некоторого слова p событию с регулярным выражением R устанавливается в результате отождествления последней буквы слова p с ее вхождением в R, имеющим индекс j, то j-е место выражения R называется *конечным местом* этого выражения.

Для любого конечного множества регулярных выражений $R_1, R_2, \ldots, R_p$ в одном и том же алфавите $\mathfrak{X}$, пользуясь определенным выше порядком действий в алгебре событий, нетрудно составить *таблицу следования* мест. Строки такой таблицы обозначаются буквами алфавита $\mathfrak{X}$, а столбцы — индексами всех мест выражений $R_1, R_2, \ldots, R_p$. На пересечении x_i-ой строки с j-ым столбцом таблицы следования выписываются индексы всех мест, которые x_i-следуют за j-ым местом. Если таких индексов нет, в соответствующем месте таблицы ставится специальный значок, обозначающий пустое множество индексов. Условимся в качестве такого значка употреблять звездочку.

Построим автомат Мура A, внутренними состояниями которого будут всевозможные подмножества индексов мест в заданных регулярных выражениях $R_1, R_2, \ldots, R_p$ (включая пустое подмножество). Функция переходов δ этого автомата строится следующим образом: для любого состояния a_i автомата A (множества индексов мест заданных событий) и любой буквы x_j входного алфавита состояние $a_k = \delta(a_i, x_j)$ определяется как множество индексов всех мест, которые x_j-следуют хотя бы за одним из мест, индексы которых входят в a_i.

Сдвинутая функция выходов μ автомата Мура A строится для выходного алфавита $\mathfrak{Y}$, состоящего из всевозможных подмножеств (включая пустое подмножество) множества всех символов R_1, $R_2, \ldots, R_p$ заданных регулярных выражений. Для любого состояния a_i (множества индексов) автомата A в качестве $\mu(a_i)$ выбирается множество всех тех регулярных выражений $R_1, R_2, \ldots, R_p$, для которых хотя бы один из индексов, входящих в a_i, является индексом конечного места.

Нами построен некий конечный автомат Мура A. Из способа построения его функций переходов и выходов непосредственно сле-

дует, что он представляет (при выборе в качестве начального состояния 0) каждое из заданных событий $R_1, R_2, \ldots, R_p$ и дополнение S их объединения. Событие R_i представлено множеством всех тех выходных сигналов (множеств символов $R_1, R_2, \ldots, R_p$), в состав которых входит символ $R_i (i = 1, 2, \ldots, p)$. Событие S представляется, очевидно, пустым множеством символов $R_1, R_2, \ldots, R_p$.

В результате нами доказано следующее предложение.

Теорема 1. *Любое регулярное событие может быть представлено в конечном автомате. Существует единый конструктивный прием (алгоритм синтеза), позволяющий по любому конечному множеству регулярных событий, заданных регулярными выражениями, построить представляющие эти события конечные автоматы Мура или Мили.*

Объединяя доказанную теорему с результатом, полученным в § 2, получим следующий результат.

Теорема 2. *Регулярные события и только они представимы в конечных автоматах.*

Аналогичный результат для автоматов специального вида (нервных сетей) и для более громоздкого по форме определения регулярного события был получен ранее Клини [40].

Из полученных результатов вытекает также следующее предложение.

Теорема 3. *Пересечение двух (а значит, и любого конечного числа) регулярных событий и дополнение (в множестве всех слов в основном алфавите) любого регулярного события являются также регулярными событиями.*

Описанный выше алгоритм синтеза конечных автоматов допускает также следующую более удобную для практических целей интерпретацию [21].

Пусть дано p регулярных выражений $R_1, R_2, \ldots, R_p$ в произвольном конечном алфавите $\mathfrak{X} = (x_1, x_2, \ldots, x_n)$. Если какое-либо из выражений R_i является дизъюнкцией нескольких термов, то можно, не нарушая общности, считать, что оно заключено в обычные (неитерационные) скобки. *Местами* в выражениях $R_1, R_2, \ldots, R_p$ будем называть специально вводимые знаки раздела (вертикальные черточки), ставящиеся между любыми двумя знаками (буквами, скобками, знаками дизъюнкции) этих выражений, а также слева от выражения (начальное место) и справа от выражения (конечное место).

Места, непосредственно слева от которых стоит буква основного алфавита $\mathfrak{X}$, и начальное место называются *основными местами*; места, непосредственно справа от которых стоит буква алфавита $\mathfrak{X}$, — *предосновными*. Начальные места всех выражений $R_1, R_2, \ldots, R_p$ отождествляем между собой в одно единственное начальное место. Все основные места обозначаем различными неотрицательными целыми числами — *основными индексами* этих мест. Начальное место получает при этом основной индекс 0.

Действие каждого основного индекса распространяется не только на соответствующее место, но и на места (основные и неосновные), ему подчиненные. Правило подчинения мест выражает порядок действий в алгебре событий. Оно определяется следующим *правилом распространения индексов.*

Индексы места перед любыми скобками (итерационными или обыкновенными) распространяются на начальные места всех термов, стоящих внутри этих скобок. Индексы конечного места любого терма, заключенного в скобки, распространяются на место, непосредственно следующее за этими скобками. Индексы места, непосредственно предшествующего итерационным скобкам или символам пустого слова, распространяются на место, непосредственно следующее за этими скобками (соответственно — за данным символом e). Наконец, индексы места, непосредственно следующего за итерационными скобками, распространяются на начальные места всех заключенных в эти скобки термов.

Все индексы, возникшие на основных и неосновных местах в результате применения только что сформулированного правила, называются *неосновными.* Само правило при этом должно применяться до тех пор, пока его применение не будет больше приводить к появлению новых индексов ни на одном месте.

Индексация заданных регулярных выражений, разметка мест и распространение индексов согласно сформулированному правилу составляют *первый шаг* алгоритма синтеза.

Второй шаг состоит в построении таблицы переходов искомого автомата A. При этом входными сигналами служат буквы исходного алфавита $\mathfrak{X}$, а внутренние состояния автомата отождествляются с множествами основных индексов. Условимся для определенности обозначать эти множества дизъюнкцией составляющих индексов, а пустое множество индексов — звездочкой.

Правило построения таблицы переходов автомата состоит в следующем.

Начальным состоянием автомата A служит одноэлементное множество, состоящее из индекса 0. Состояние a_i переводится входным сигналом x_k в состояние a_j, состоящее из основных индексов всех основных мест, отделенных буквой x_k от непосредственно предшествующих им предосновных мест, в числе индексов которых (основных или неосновных) содержится хотя бы один индекс из числа индексов, входящих в состояние a_i.

Применяя на практике сформулированное правило, удобно выделять основные индексы, помещая их над специально проводимой для этой цели горизонтальной чертой. Все индексы (основные и неосновные) предосновных мест также целесообразно выделять, например заключая их в прямоугольную рамку. При построении таблицы переходов достаточно ограничиться лишь состояниями, фактически появляющимися в процессе построения таблицы, отправляясь от начального (нулевого) состояния.

Третий шаг синтеза состоит в построении сдвинутой таблицы выходов, или, что то же самое, в отметке состояний автомата A соответствующими им выходными сигналами. В качестве выходных сигналов выбираются различные множества символов исходных регулярных выражений (включая пустое множество). *Правило отметки состояний* состоит в следующем.

Состояние a_i отмечается множеством тех символов выражений $R_1, R_2, \ldots, R_p$, в состав индексов (основных и неосновных) конечных мест которых входит хотя бы один индекс из a_i.

Состояния, отмеченные пустым множеством символов, называются также *неотмеченными*.

Заметим, что построенный автомат Мура представляет событие R_i множеством всех тех выходных сигналов, в состав которых входит символ R_i ($i = 1, 2, \ldots, p$).

Четвертый шаг алгоритма синтеза состоит в переобозначении внутренних состояний и выходных сигналов с целью более простой записи таблицы переходов и сдвинутой таблицы выходов. При этом внутренние состояния чаще всего нумеруются последовательными натуральными числами $1, 2, \ldots, k$.

Наконец, *пятый шаг* алгоритма синтеза употребляется тогда, когда требуется синтезировать не автомат Мура, а автомат Мили. Он заключается в построении обычной (несдвинутой) таблицы выходов. Как следует из § 1 настоящей главы, для этого достаточно в таблицу переходов подставить вместо внутренних состояний отмечающие их выходные сигналы.

При решении практических задач, возникающих при синтезе автоматов, часто удобно приписывать некоторым основным местам одинаковые основные индексы, тем самым отождествляя эти места. Такое же отождествление возможно, если отождествляемым местам подчинены одинаковые множества предосновных и конечных мест (места, удовлетворяющие этому условию, называются *подобными*).

Другой случай, когда оказывается возможным отождествление мест, относится к так называемым *соответственным* местам. Соответственными называются все те места в разных регулярных выражениях $R_1, R_2, \ldots, R_p$ или в различных термах, заключенных в одни и те же скобки, к которым ведут одинаковые пути (множества слов) от начального места или соответственно от места, непосредственно предшествующего скобкам.

При использовании описанного выше алгоритма синтеза основные индексы соответственных мест входят в состояния синтезируемого автомата всегда вместе. Именно этим и обусловливается возможность одинаковой их индексации. Обоснование возможности отождествления подобных мест вытекает из алгоритма минимизации, описываемого в § 5 настоящей главы.

Заметим, что места следует отождествлять лишь по одному из признаков (подобия или соответственности), так как одновременные отождествления по обоим признакам могут привести к

ошибкам. В частности, поскольку начальные места отождествлены фактически по признаку соответственности, нельзя, вообще говоря, при наличии более одного события отождествлять начальное место в каком-либо событии с другим местом по признаку подобия.

Непосредственно из описанного алгоритма синтеза вытекает справедливость следующего предложения.

Теорема 4. *События, заданные регулярными выражениями* $R_1, R_2, \ldots, R_p$ *в некотором конечном алфавите* $\mathfrak{X}$, *могут быть представлены в конечном автомате* (*Мили или Мура*), *имеющем не более* 2^{n+1} *внутренних состояний, где* n — *суммарное число вхождений букв алфавита* $\mathfrak{X}$ *в выражения* $R_1, R_2, \ldots, R_p$.

Рассмотрим, какие изменения в алгоритм синтеза следует внести в том случае, когда, кроме исходных событий $R_1, R_2, \ldots, R_p$, в алфавите $\mathfrak{X} = (x_1, x_2, \ldots, x_m)$ задана также область запрета S.

Область запрета может быть задана либо с помощью некоторого регулярного выражения, либо как совокупность слов в алфавите $\mathfrak{X}$, не вошедших ни в одно из событий $R_1, R_2, \ldots, R_p$. Оба эти способа по существу эквивалентны между собой, поскольку от первого способа задания можно перейти ко второму и наоборот.

Область запрета S по самому определению допускает умножение справа на совокупность F всех слов (включая пустое слово) в алфавите $\mathfrak{X}$: $SF = S$. Поэтому при задании области запрета регулярным выражением R можно, не нарушая общности, предполагать, что выражение R имеет вид

$$R = R_1\{x_1 \vee x_2 \vee \ldots \vee x_n\}.$$

Описанный выше алгоритм синтеза дает решение задачи при наличии области запрета. Однако при этом многие переходы в синтезированном автомате окажутся лишними в том смысле, что никогда не будут использоваться при реальной работе автомата. Задача состоит в том, чтобы выявить все такие переходы и построить вместо обычного (вполне определенного) автомата частичный автомат, в таблицах переходов и выходов которого на местах запрещенных переходов стоят черточки. Переход к частичному автомату дает дополнительные возможности к последующему упрощению автомата.

Указанная задача в случае задания области запрета как совокупности слов в алфавите $\mathfrak{X}$, не вошедших ни в одно из заданных регулярных выражений $R_1, R_2, \ldots, R_p$, решается совершенно очевидным способом. После выполнения описанного выше алгоритма синтеза выходной сигнал, обозначенный пустым множеством символов $R_1, R_2, \ldots, R_p$, будет соответствовать появлению запрещенного входного слова. Следовательно, достаточно заменить этот выходной сигнал в таблице выходов черточкой и поставить черточки во всех местах таблицы переходов, соответствующих появлению запрещенного выходного сигнала (при наложении таблицы

выходов на таблицу переходов места, отмеченные черточкой, в обеих таблицах должны совпадать).

В случае задания области запрета регулярным выражением S следует применить обычный алгоритм синтеза к выражениям S, R_1, R_2, ..., R_p и считать запрещенными все выходы, обозначенные множествами, в состав которых входит символ S. Запрещенные выходы в таблице выходов заменяются черточками, которые описанным выше способом переносятся на таблицу переходов. Ясно, что такой прием действительно приводит к решению поставленной задачи.

В рассмотренном случае следует считать, что выражение имеет вид

$$S = \hat{S}_1 \{x_1 \vee x_2 \vee \ldots \vee x_n\}.$$

Если первоначально заданное выражение для области запрета не удовлетворяло этому условию, его следует заменить выражением

$$R = S \{x_1 \vee x_2 \vee \ldots \vee x_n\}.$$

В качестве примера рассмотрим синтез частичного автомата Мили, представляющего событие $R = x\{y\}$, при наличии области запрета $S = yx\{x \vee y\}$. На первом шаге проведем разметку мест, индексацию и распространение индексов в выражениях R и S, используя возможность отождествления подобных мест:

$R =$	x	$\{$	y	$\}$
	$\overline{0}$ 1		1	
		$\underline{1}$		1

$S =$	y	x	$\{$	x	$\vee$	y	$\}$
	$\overline{0}$	$\overline{2}$	3	3		3	
			$\underline{3}$		$\underline{3}$		3

Выполняя второй и третий шаги алгоритма, придем к отмеченной таблице переходов автомата Мура

	$-R-$		$-S-$		
	0	1	2	*	3
x	1	*	3	*	3
y	2	1	*	*	3

На четвертом шаге введем переобозначения $0 \to 1$, $1 \to 2$, $2 \to 3$, $* \to 4$, $3 \to 5$, $(\ ) \to u$, $R \to v$, $S \to w$ (скобками $(\ )$ здесь обозначено пустое множество символов R и S). После этого получим отмеченную таблицу переходов

	u	v	u	u	w
	1	2	3	4	5
x	2	4	5	4	5
y	3	2	4	4	5

Совершая на пятом шаге переход к автомату Мили, получим таблицы переходов и выходов

	1	2	3	4	5
x	2	4	5	4	5
y	3	2	4	4	5

;

	1	2	3	4	5
x	v	u	w	u	w
y	u	v	u	u	w

.

Наконец, совершаем переход к частичному автомату. Запрещенным в данном случае следует считать сигнал w. Тогда таблицы переходов и выходов будут иметь вид

	1	2	3	4	5
x	2	4	—	4	—
y	3	2	4	4	—

;

	1	2	3	4	5
x	v	u	—	u	—
y	u	v	u	u	—

.

Однако состояние 5 является лишним, поскольку автомат никогда не перейдет в него, отправляясь из начального состояния 1. Отбрасывая это лишнее состояние, придем к окончательным таблицам переходов и выходов

	1	2	3	4
x	2	4	—	4
y	3	2	4	4

;

	1	2	3	4
x	v	u	—	u
y	u	v	u	u

.

§ 5. Минимизация абстрактных автоматов

Как указывалось выше, абстрактный автомат рассматривается нами как устройство для реализации автоматных отображений. В связи с этим естественно не различать автоматы, эквивалентные между собой, т. е. автоматы, индуцирующие одинаковые отображения.

Основной задачей, решаемой в данном параграфе, является задача минимизации автоматов, т. е. задача нахождения автомата с минимальным числом состояний в классе всех автоматов, эквивалентных данному. Излагаемые методы решения этой задачи представляют собой развитие идей Мили [55], Ауфенкампа и Хона [3,4].

Пусть a и b — два состояния одного и того же или двух различных автоматов Мили, имеющих общий входной и общий выходной алфавиты. Если для любого входного сигнала x_i выходные сигналы, определяемые парами (a, x_i) и (b, x_i), одинаковы, то состояния a и b называются 1-*эквивалентными*.

Если 1-эквивалентные состояния переводятся любым входным сигналом x_i также в 1-эквивалентные между собой состояния, то они называются 2-*эквивалентными*. Если 2-эквивалентные состояния переводятся любым входным сигналом в 2-эквивалентные между собой состояния, то они называются 3-*эквивалентными*, и т. д.

Легко понять, что i-эквивалентные состояния в случае применения к ним любого входного слова длины i порождают одинаковые выходные слова ($i = 1, 2, \ldots$).

Отношение i-эквивалентности для любого $i = 1, 2, \ldots$ обладает свойствами рефлексивности, симметричности и транзитивности. Отсюда следует, что множество всех внутренних состояний данного автомата Мили разбивается этим отношением на попарно не пересекающиеся классы i-эквивалентных между собой состояний. Такие классы будем называть классами *i-эквивалентности*, или *i-классами*.

Состояния, являющиеся i-эквивалентными для всех $i = 1, 2, \ldots$, называются *эквивалентными состояниями*, а определяемые отношением эквивалентности классы — *классами эквивалентности*, или ∞ - *классами*.

Из определения эквивалентных между собой состояний непосредственно вытекает справедливость такого предложения.

Теорема 1. *Два состояния одного и того же или двух различных автоматов Мили тогда и только тогда эквивалентны между собой, когда применение к ним любого входного слова вызывает появление одного и того же выходного слова.*

Данное предложение позволяет сформулировать также следующий результат.

Теорема 2. *Два автомата Мили тогда и только тогда эквивалентны между собой (в смысле совпадения индуцируемых ими автоматных отображений), когда эквивалентны их начальные состояния.*

Применение одного и того же входного слова p к двум эквивалентным состояниям a и b переводит их снова в эквивалентные состояния ap и bp. Поскольку эквивалентные состояния являются вместе с тем и 1-эквивалентными, то для любого входного сигнала x_i пары (a, x_i) и (b, x_i) определяют одинаковые выходные сигналы.

Таким образом, для каждого автомата Мили A можно построить новый автомат Мили B с теми же, что и у автомата A, входным и выходным алфавитами, принимая за множество его внутренних состояний множество всех классов эквивалентности автомата A. Переходы и выходы в автомате B определяются следующим образом: класс эквивалентности K_1 переводится входным сигналом x_i в класс эквивалентности K_2, содержащий состояние $a_j x_i$, где a_j — любое состояние, содержащееся в классе K_1. Паре $K_1 x_i$ сопоставляется при этом выходной сигнал, определяемый парой $a_j x_i$. Построенный таким образом автомат будем называть *канонической минимизацией* автомата Мили A.

Из способа построения автомата B и из предложения 1 вытекает справедливость следующего предложения.

Теорема 3. *В канонической минимизации любого автомата Мили любые два различных внутренних состояния не эквивалентны между собой.*

Для реализации автоматных соответствий достаточно ограничиться рассмотрением лишь так называемых *связных* автоматов,

т. е. таких автоматов, у которых каждое состояние является достижимым, или, иначе говоря, которые в результате воздействия подходящего входного слова могут быть переведены из начального состояния в любое другое внутреннее состояние.

Действительно, если отображение φ индуцируется несвязным автоматом A, то достижимые состояния автомата A образуют новый автомат B, который является связным и индуцирует то же самое отображение φ. Заметим, что в результате применения описанного в предыдущем параграфе алгоритма синтеза всегда получаются связные автоматы.

Рассмотрим какой-либо связный автомат Мили A, индуцирующий заданное автоматное отображение φ. Каноническая минимизация B автомата A также связна и реализует то же самое отображение φ (при выборе в качестве начального состояния класса эквивалентности K_0, содержащего начальное состояние автомата A).

Пусть D — произвольный автомат, реализующий то же самое соответствие, а d_0 — его начальное состояние. В силу связности автомата B для каждого его состояния K_i можно выбрать входное слово p_i так, чтобы $K_0 p_i = K_i$ $(i \in M)$. Построим отображение ψ множества состояний автомата B в множество состояний автомата D, полагая $\psi(K_i) = d_0 p_i$ $(i \in M)$.

Ясно, что начальные состояния K_0 и d_0 автоматов B и D эквивалентны между собой. Но тогда эквивалентными являются также состояния K_i и $d_i = \psi(K_i)$ при любом $i \in M$. Если $\psi(K_i) = \psi(K_j)$, то отсюда вытекает эквивалентность состояний K_i и K_j. В силу же предложения 3 это означает, что $K_i = K_j$. Таким образом, соответствие ψ взаимно однозначно, что влечет за собой справедливость следующего предложения.

Теорема 4. *Каноническая минимизация связного автомата Мили, индуцирующего любое заданное автоматное отображение* φ, *представляет собой автомат, имеющий наименьшее возможное число внутренних состояний среди всех автоматов Мили, индуцирующих то же самое отображение* φ (*т. е. среди всех автоматов, эквивалентных автомату* A).

Это утверждение полностью решает задачу минимизации автоматов Мили при условии, что существует конструктивный прием построения классов эквивалентности для любого заданного (связного) автомата Мили. Такой прием был предложен Ауфенкампом и Хоном [4] для случая конечных автоматов Мили. Он основан на следующем легко доказуемом предложении.

Теорема 5. *Если для некоторого* i *разбиение состояний автомата на* $(i+1)$*-классы совпадает с разбиением на* i*-классы, то оно является разбиением и на* ∞*-классы.*

В самом деле, если любая пара (a_k, a_j) i-эквивалентных состояний является также $(i+1)$-эквивалентной, то состояния a_{i_k} и a_{i_j} любым входным сигналом x_r переводятся в i-эквивалентные между собой состояния. Но тогда они переводятся этим сигналом и в $(i+1)$-эквивалентные между собой состояния. Следовательно,

состояния a_k и a_j не только $(i+1)$-эквивалентны, но и $(i+2)$-эквивалентны между собой.

Далее получим, что состояния a_j и a_k n-эквивалентны при всех $n = i,\ i+1,\ i+2,\ \ldots$ и, следовательно, являются эквивалентными между собой состояниями. Ввиду произвольности выбора состояний a_j и a_k предложение 5 доказано.

Алгоритм Ауфенкампа — Хона построения класса эквивалентности (∞-классов) основан на последовательном построении i-классов для всех $i = 1, 2, \ldots$. Поскольку разбиение на $(i+1)$-классы является подразбиением разбиения на i-классы, то в случае конечности автомата A через конечное число шагов получим на основании теоремы 5 искомое разбиение на ∞-классы.

Разбиение на 1-классы совершается непосредственно по таблице выходов автомата: в один и тот же 1-класс объединяются все состояния, которым соответствуют одинаковые столбцы в таблице выходов. Далее строится так называемая 1-таблица, получающаяся в результате замены в таблице переходов автомата внутренних состояний содержащими их 1-классами.

В один и тот же 2-класс объединяются все состояния, принадлежащие к одному и тому же 1-классу, которым соответствуют одинаковые столбцы в 1-таблице. Далее поступают аналогичным образом: заменяя в таблице переходов состояния автомата содержащими их 2-классами, получают 2-таблицу. По 2-таблице находят 3-классы, объединяя в один 3-класс все состояния одного и того же 2-класса, которым соответствуют одинаковые столбцы в 2-таблице.

Придя через конечное число шагов к разбиению на ∞-классы, строят каноническую минимизацию исходного автомата A непосредственно по его таблицам переходов и выходов.

В результате нами построен алгоритм минимизации произвольных конечных автоматов Мили. Для случая *автоматов Мура* необходимо внести в этот алгоритм некоторые изменения, поскольку, интерпретируя автомат Мура как автомат Мили и минимизируя его в соответствии с описанным алгоритмом, мы построим автомат, хотя и эквивалентный исходному, но, возможно, не являющийся уже автоматом Мура.

Для того чтобы при минимизации автомат Мура оставался автоматом Мура, очевидно, необходимо и достаточно, чтобы одинаково отмеченные состояния автомата не относились к различным классам эквивалентности.

Наиболее просто можно удовлетворить это условие, вводя для автоматов Мура понятие *0-эквивалентности* состояний и разбиения множества состояний на *0-классы*: 0-эквивалентными будем называть любые *одинаково отмеченные* состояния автомата Мура. Если два 0-эквивалентных состояния любым входным сигналом переводятся в два 0-эквивалентных состояния, то они называются 1-эквивалентными.

Все дальнейшие построения (определение i-классов для $i \geqslant 2$, определение классов эквивалентности и построение канонической

минимизации) осуществляются точно так же, как и в случае автоматов Мили. Разумеется, в случае автоматов Мура при построении канонической минимизации B можно задавать для нее не таблицу выходов, а *сдвинутую* таблицу выходов, отмечая состояния автомата B (классы эквивалентности) теми же выходными сигналами, которыми отмечены входящие в них состояния исходного автомата.

Однако теория минимизации автоматов Мура в только что описанной форме не вполне эквивалентна соответствующей теории для случая автоматов Мили. В частности, предложение, аналогичное теореме 1, на автоматы Мура не распространяется.

Для достижения эквивалентности обеих теорий необходимо в качестве реакции автомата Мура на входное слово p рассматривать не то выходное слово q, которое получается в силу общего определения автомата, данного в § 1, а слово $y_i q$, где y_i — выходной сигнал, отмечающий то состояние, в котором находился автомат до начала подачи слова p. При таком определении реакции автомата Мура на входное слово для этого автомата будут, очевидно, справедливы предложения, получающиеся из теорем 1, 2, 3, 4 настоящего параграфа заменой всех встречающихся в их формулировках автоматов Мили на автоматы Мура. Теорема 5 остается, разумеется, справедливой для автоматов Мура и при обычном определении выходных реакций автоматов.

При переходе к обычному пониманию выходных реакций автомата в особом положении оказываются лишь такие состояния, в которые автомат не может перейти, отправляясь из других состояний (для случая связных автоматов таким свойством может обладать лишь начальное состояние). Легко проверить, что для восстановления параллелизма в теориях минимизации автоматов Мура и Мили в этом случае достаточно не отмечать подобные состояния никакими выходными сигналами. Это позволяет при образовании 0-классов относить такие состояния к любому 0-классу и увеличивает тем самым возможности минимизации.

Возникающий здесь параллелизм имеет, однако, место не при минимизации обычных всюду определенных автоматов, а при переходе к более общей задаче минимизации частичных автоматов.

Сущность задачи минимизации частичных автоматов состоит в следующем: дан частичный автомат (Мура или Мили) A, индуцирующий частичное автоматное отображение φ, определенное на некотором множестве M слов входного алфавита. Требуется построить частичный автомат (Мура или соответственно Мили) B, который индуцирует частичное автоматное отображение, совпадающее на множестве M с отображением φ, и имеет наименьшее число внутренних состояний среди всех автоматов (Мура или Мили), удовлетворяющих этому условию.

Поскольку есть лишь конечное число различных частичных автоматов, у которых входной и выходной алфавиты общие с данным конечным частичным автоматом A и число состояний которых не превосходит числа состояний автомата A, то сформулированная

задача алгоритмически разрешима. Однако существующие методы ее точного решения связаны с большим перебором (см. например, С. Гинзбург [19]) и поэтому практически малопригодны.

На практике обычно пользуются следующим приемом минимизации частичных автоматов: мысленно заполняя прочеркнутые места в таблицах переходов и выходов данного частичного автомата A, объединяют состояния в i-классы и минимизируют по тем же самым правилам, что и в случае обычных (всюду определенных) автоматов. Все или некоторые из возникающих вариантов объединения состояний в классы проверяются, и из полученных канонических минимизаций выбирают ту, которая имеет наименьшее число состояний.

Указанный прием действительно решает задачу построения частичного автомата B с меньшим числом состояний, чем у исходного автомата A, причем индуцируемое автоматом B частичное автоматное отображение ψ совпадает с частичным автоматным отображением φ, индуцируемым автоматом A на области определения отображения φ. При этом область определения отображения ψ, вообще говоря, больше области определения отображения φ.

Покажем, как работает описанный прием минимизации на примере частичного автомата Мили A, заданного следующими таблицами переходов и выходов:

	1	2	3	4
x	2	—	2	4
y	—	3	4	4

;

	1	2	3	4
x	u	—	u	u
y	—	u	v	v

.

Минимизируя заданный автомат, получим две исходные возможности объединения в 1-классы, приводящие к наименьшему количеству классов,

$$a_1 = (1, 3, 4,), \ b_1 = (2) \text{ и } a_1 = (1.2), \ b_1 = (3.4).$$

Рассмотрим сначала первую возможность: 1-таблица запишется

	1	2	3	4
x	b_1	—	b_1	a_1
y	—	a_1	a_1	a_1

.

Из 1-таблицы видно, что класс a_1 нужно **разделить на два** 2-класса: $a_2 = (1, 3)$ и $c_2 = (4)$; третий 2-класс b_2 **совпадает с** 1-классом $b_1 = (2)$; 2-таблица будет иметь вид

	1	2	3	4
x	b_2	—	b_2	c_2
y	—	a_2c_2		c_2

.

Из 2-таблицы видно, что 3-классы совпадают с 2-классами, которые являются, следовательно, искомыми ∞-классами. Каноническая минимизация в разобранном варианте представится таб-

лицами

	a_2	b_2	c_2
x	b_2	—	c_2;
y	c_2	a_2	c_2

	$a_2b_2c_2$
x	u — u.
y	v u v

Во втором варианте минимизации разбиение **на 1-классы** $a_1 = (1, 2)$ и $b_1 = (3, 4)$ приводит к 1-таблице

	1	2	3	4
x	a_1	—	a_1	b_1
y	—	b_1	b_1	b_1

и к разбиению на **2-классы** $a_2 = (1, 2)$; $b_2 = (3)$; $c_2 = (4)$. **2-таблица** запишется

	1	2	3	4
x	a_2	—	a_2	c_2,
y	—	b_2c_2		c_2

откуда следует, что полученное разбиение на 2-классы дальше не измельчается и, следовательно, совпадает с разбиением на ∞-классы. Каноническая минимизация в этом варианте будет задаваться таблицами

	a_2	b_2	c_2
x	a_2	a_2	c_2;
y	b_2	c_2	c_2

	a_2	b_2	c_2
x	u	u	u.
y	u	v	v

Полученные канонические минимизации существенно различны: первая реагирует на входное слово y выходным словом v, а вторая — выходным словом u.

§ 6. Структурный синтез конечных автоматов

На этапе структурного синтеза выбираются элементарные автоматы, из которых осуществляется синтез структурной схемы заданного абстрактного автомата Мили или Мура. Будем предполагать, что элементарные автоматы бывают двух родов: элементарные *автоматы с памятью*, или запоминающие *элементы* (триггеры, линии задержки и т. д.), и элементарные *автоматы без памяти*, называемые также *логическими элементами*. Для простоты ограничимся случаем, когда в нашем распоряжении лишь один тип запоминающих элементов.

Входные и выходные сигналы как элементарных автоматов, так и всего рассматриваемого автомата в целом обозначаются (кодируются) конечными последовательностями букв некоего фиксированного конечного алфавита, называемого *структурным алфавитом*. Обычно в качестве структурного алфавита выбирается *двоичный алфавит*, состоящий из двух букв: 0 и 1. Вторым алфавитом, играющим важнейшую роль на этапе структурного синтеза, является алфавит, состоящий из символов внутренних состояний вы-

бранных элементов памяти. Назовем его алфавитом состояний. Алфавит состояний может не совпадать со структурным алфавитом, однако на практике в качестве алфавита состояний выбирается обычно также двоичный алфавит.

Одной из главных задач, решаемых в процессе структурного синтеза автоматов, является выписывание так называемых *канонических уравнений*, устанавливающих зависимость сигналов, подаваемых на входы запоминающих элементов, от выходных сигналов этих элементов и сигналов, подаваемых на вход всего автомата в целом. Чтобы обеспечить правильное функционирование схем, нельзя допускать участия входных сигналов, подаваемых на вход запоминающих элементов, непосредственно в образовании выходных сигналов, которые по цепям обратной связи подавались бы в тот же самый момент времени на эти входы. Иначе говоря, запоминающие элементы должны представлять собой автоматы Мура, а не автоматы Мили. Сложный же автомат, образованный этими элементами, может, разумеется, быть как автоматом Мура, так и автоматом Мили.

Предположим, что используемые при структурном синтезе элементарные автоматы с памятью являются автоматами Мура. Сдвинув соответственно начало отсчета временных интервалов для выходных сигналов, будем считать, что выходной сигнал в любой момент времени t каждого элемента памяти определяется внутренним состоянием этого элемента *в тот же самый* момент времени.

Чтобы иметь возможность синтезировать произвольные автоматы при минимальной затрате элементов памяти, целесообразно в качестве таких элементов выбирать автоматы Мура, имеющие *полную систему переходов и полную систему выходов*, которые для краткости будем называть *полными автоматами*. Полнота системы переходов означает, что для любой пары состояний автомата найдется входной сигнал, переводящий первый элемент этой пары во второй. Это требование эквивалентно требованию: в каждом столбце таблицы переходов должны встречаться все состояния данного автомата. Полнота системы выходов в случае автоматов Мура означает, что каждому состоянию автомата поставлен в соответствие свой особый выходной сигнал, отличный от выходных сигналов других состояний. Поэтому для полных автоматов Мура естественно просто отождествлять выходные сигналы с соответствующими внутренними состояниями автомата. В дальнейшем будем придерживаться такого способа.

Выберем в качестве элемента памяти некоторый полный автомат Мура B. Внутренние состояния этого автомата обозначим буквами $z_1, z_2, \ldots, z_r$ $(r \geqslant 2)$. Они, согласно принятому условию, будут также выходными сигналами. Для обозначения входных сигналов элемента памяти будем употреблять буквы $s_1, s_2, \ldots, s_q$. Алфавит $(z_1, z_2, \ldots, z_r)$ есть не что иное, как алфавит состояний. Структурный алфавит выберем несколько позднее, а пока покажем, как находить так называемые *канонические уравнения* авто-

мата, память которого составляется из элементов выбранного типа.

Предположим, что задан абстрактный конечный автомат Мили или Мура A с входным алфавитом $\mathfrak{X} = (x_1, x_2, \ldots, x_n)$, выходным алфавитом $\mathfrak{Y} = (y_1, y_2, \ldots, y_m)$ и множеством внутренних состояний $\mathfrak{A} = (a_1, a_2, \ldots, a_p)$, задаваемый функцией переходов $\delta(a, x)$ и функцией выходов $\lambda(a, x)$. Пусть выбранный элемент памяти B задается функцией переходов $\nu(z, s)$. Поставим задачу нахождения канонических уравнений автомата A при условии, что его память строится из нескольких экземпляров $B^{(1)}, B^{(2)}, \ldots, B^{(k)}$ элементарного автомата B.

Для построения автомата A число k элементов памяти должно удовлетворять условию $r^k \geqslant p$. В этом случае различные внутренние состояния a_i можно отождествлять с различными наборами состояний автоматов $B^{(1)}, B^{(2)}, \ldots, B^{(k)}$. Процесс такого отождествления будем называть *кодированием* состояний автомата A в принятом алфавите состояний. Разумеется, процесс кодирования по самому своему существу не однозначен. Однако не будем входить в подробности вопроса, связанного с выбором того или иного варианта кодирования, а будем считать этот вариант уже заданным.

После кодирования состояния автомата A будут обозначены k-мерными векторами $(z^{(1)}, z^{(2)}, \ldots, z^{(k)})$, компонентами которых служат различные буквы $z_1\ z_2, \ldots, z_r$ алфавита состояний; двухместная функция выходов $\lambda(a, x)$ автомата A превратится в $(k+1)$-местную функцию $\lambda^0(z^{(1)}, \ldots, z^{(k)}, x)$, которую по-прежнему будем называть функцией выходов. Эквивалентом функции переходов $\delta(a, x)$ после кодирования будет система из k $(k+1)$-местных функций $\varphi^{(1)}(z^{(1)}, \ldots, z^{(k)}, x), \ldots, \varphi^{(k)}(z^{(1)}, \ldots, z^{(k)}, x)$ *переходов в элементах памяти.* Функция $\varphi^{(i)}$ определяет состояние, в которое должен перейти i-й элемент памяти в момент времени $t+1$, если в момент времени t автомат A находился в состоянии $(z^{(1)}, z^{(2)}, \ldots, z^{(k)})$, а на его вход был подан сигнал $x(i = 1, 2, \ldots, k;\ t = 0, 1, 2, \ldots)$.

Следующим важным этапом является построение *функций возбуждения* $\sigma^i(z^{(1)}, \ldots, z^{(k)}, x)$ элементов памяти $(i = 1, 2, \ldots, k)$. Значение каждой такой функции $\sigma^{(i)}$ при выбранном состоянии $(z^{(1)}, z^{(2)}, \ldots, z^{(k)})$ автомата A и входном сигнале x определяется как входной сигнал $s^{(i)}$ i-го элемента памяти, вызывающий переход в i-ом элементе памяти, обусловленный i-ой функцией переходов, т. е. переход $z^{(i)} \to \varphi^{(i)}(z^{(i)}, \ldots, z^{(k)}, x)(i = 1, \ldots, k)$. Входной сигнал $s^{(i)}$ может быть выбран не одним способом. Поэтому построение функций возбуждения элементов памяти по функциям переходов в них осуществляется неоднозначно.

Выбор наилучшего способа построения функций возбуждения связан с задачами последующего этапа — этапа комбинационного синтеза. Впрочем, для многих типов элементов памяти (линии задержки, триггеры со счетным входом и др.) соответствующий переход выполняется однозначно. Для ряда типов элементов можно

указать частичные комбинаторно-вычислительные приемы, позволяющие более простым, по сравнению с изложенным общим приемом, способом находить функции возбуждения [23].

Функции возбуждения $\sigma^{(i)}$, приравненные определяемым ими входным сигналам $s^{(i)}$, и дают искомые канонические уравнения для обратных связей в автомате A. Однако эти функции имеют еще не вполне удовлетворительный вид: состояния автомата A закодированы в универсальном (для выбранного типа элементов памяти) алфавите состояний, *не зависящем от выбора автомата* A; для обозначения входных и выходных сигналов используются различные алфавиты, в том числе и такие, которые зависят от выбора автомата A. Необходимо поэтому фиксировать структурный алфавит $\mathfrak{B}$, определяемый обычно фактически принятым кодированием входных сигналов элемента памяти, и закодировать конечными последовательностями букв этого алфавита не только входные сигналы $\sigma^{(i)}$ элементов памяти, но также входные и выходные сигналы x и y всего автомата в целом. Такое кодирование преобразует найденную выше систему функций возбуждения $\sigma^{(i)}$ в новую систему функций

$$\sigma_j^{(i)}(z^{(1)},\ldots,z^{(k)}u_1, u_2, \ldots, u_g)\,(i = 1,2,\ldots,k;\ j = 1,2,\ldots,l),$$

где каждая из функций $\sigma^{(i)}$ представляет собой входной сигнал (букву структурного алфавита), который нужно подать на j-й входной канал i-го элемента памяти в то время, когда автомат A находится в состоянии $(z^{(1)},z^{(2)},\ldots, z^{(k)})$, а на его входные каналы подаются сигналы (буквы структурного алфавита) $u_1, u_2, \ldots, u_g$.

Аналогично найденная выше функция выходов $\lambda^0(z^{(1)}, \ldots, z^{(k)}, x)$ заменится системой функций $\lambda_j(z^{(1)}, z^{(2)}, \ldots, z^{(k)}, u_1,u_2, \ldots, u_g)(j = 1,2,\ldots, h)$, где функция λ_j определяет выходной сигнал (букву структурного алфавита), появляющийся на j-ом выходном канале автомата A в то время, когда автомат A находится в состоянии $(z^{(1)}, z^{(2)}, \ldots, z^{(k)})$, а на его входные каналы подаются сигналы $u_1, u_2, \ldots, u_g$.

Полученные функции $\sigma_j^{(i)}$ и λ_j назовем соответственно *структурными функциями возбуждений и структурными функциями выходов автомата* A.

В случае (обычно встречающемся на практике), когда как структурный алфавит, так и алфавит состояний являются двоичными алфавитами, структурные функции возбуждений и выходов можно рассматривать как обычные булевы функции. Задача дальнейшего этапа (этапа комбинационного синтеза) заключается в фактическом построении найденных функций из элементарных логических функций, реализуемых *выбранными логическими элементами*. Методы решения этой задачи были развернуты в § 4 предыдущей главы.

В качестве элементов памяти в большинстве современных цифровых автоматов употребляются полные автоматы Мура с двумя внутренними состояниями. Интересно проанализировать вопрос

о том, сколько элементарных автоматов и какие удовлетворяют этим свойствам. Рассмотрим случай, когда полный автомат Мура с двумя состояниями 0 и 1 имеет только два входных сигнала — x и y. Из условий полноты следует, что в каждом столбце таблицы переходов автомата должны встречаться оба состояния — 0 и 1. Это ограничение приводит к тому, что существует всего 4 возможные таблицы переходов

	0	1
x	0	0;
y	1	1

	0	1
x	0	1;
y	1	0

	0	1
x	1	1;
y	0	0

	0	1
x	1	0.
y	0	1

После переобозначения входных сигналов третья таблица совпадает с первой, вторая — с четвертой. Таким образом, имеется лишь два существенно различных автомата требуемого вида, задаваемых таблицами переходов

	0	1
x	0	0
y	1	1

и

	0	1
x	0	1.
y	1	0

Если положить $x = 0$, $y = 1$, то первая таблица задает хорошо известный элемент памяти, называемый элементом *задержки* (на один такт), а вторая — не менее известный элемент, называемый *триггером со счетным входом*. Заметим, что обычное электромагнитное реле с замыкающим контактом можно рассматривать как элемент задержки.

При увеличении числа входных сигналов появляются новые типы элементов памяти: *триггер с раздельными входами*, задаваемый таблицей переходов

	0	1
x	0	1
y	0	0
z	1	1

смешанный триггер, задаваемый таблицей

	0	1
x	0	1
y	0	0
z	1	1
u	1	0

и другие.

При наличии лишь двух внутренних состояний бесполезно увеличивать число входных сигналов более чем до четырех, поскольку при большем числе входных сигналов некоторые из них начнут дублировать друг друга (вызывать одинаковые переходы в автомате). Нетрудно поэтому составить каталог всех полных существенно различных автоматов Мура с двумя состояниями (суще-

ственно различными будем считать автоматы, таблицы переходов которых не переходят одна в другую при переобозначениях входных сигналов). Кроме перечисленных четырех типов автоматов, в этот каталог войдут еще три автомата, задаваемых таблицами переходов

	0	1
x	0	1
y	0	0
z	1	0

;

	0	1
x	0	1
y	1	1
z	1	0

;

	0	1
x	0	0
y	1	1
z	1	0

.

Разумеется, каждый из перечисленных типов автоматов допускает различные модификации за счет различного кодирования входных сигналов в двоичном алфавите. Рассмотрим в качестве примера полный синтез (абстрактный и структурный до получения канонических уравнений) автомата Мура A, представляющего собой последовательный двоичный квадратор. Автомат A работает следующим образом: на его вход разряд за разрядом младшими разрядами вперед подается двухразрядное двоичное целое число. На выходе автомата также последовательно, начиная с младших разрядов, должен появиться квадрат этого числа. Иначе говоря, автомат A должен реализовать следующее частичное отображение φ:

$$0000 \to 0000$$
$$1000 \to 1000$$
$$0100 \to 0010$$
$$1100 \to 1001.$$

Нетрудно проверить, что отображение φ, продолженное на начальные отрезки слов, удовлетворяет условию автоматности. Обозначая нулевой сигнал на входе буквой x, а на выходе—u и соответственно единичный сигнал на входе — y, а на выходе — v, запишем это соответствие в виде

$$xxxx \to uuuu$$
$$yxxx \to vuuu$$
$$xyxx \to uuvu$$
$$yyxx \to vuuv.$$

В полученном соответствии выходным сигналом u представлено событие $R = x \vee xx \vee xxx \vee xxxx \vee yx \vee yxx \vee yxxx \vee xy \vee xyxx \vee yy \vee \vee yyx$, а выходным сигналом v — событие $Q = xyx \vee y \vee yyxx$. Слова, не вошедшие в события R и Q, являются запрещенными. За счет запрещенных слов можно увеличивать эти события без опасений нарушить в синтезируемом автомате его реакцию на заданные слова.

Поскольку события R и Q не пересекаются, можно на основании сказанного заменить событие R дополнением Q' события Q. В таком случае автомат можно синтезировать по одному лишь событию Q: событие Q' окажется автоматически представленным

множеством всех неотмеченных состояний этого автомата. Имея в виду, что для отметки состояний так или иначе будут употребляться потом символы u и v, обозначим событие Q буквой v, а его дополнение $Q' - u$.

Процесс абстрактного синтеза приводит к следующей разметке регулярного выражения для события $Q=v$:

$$v = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline (& y & \vee & x & y & x & \vee & y & y & x & x &) \\ \hline 0 & 1 & & \underline{2} & \underline{3} & 4 & & \underline{1} & \underline{5} & \underline{6} & 4 & \\ \hline \multicolumn{2}{c}{\underline{0}} & \multicolumn{2}{c}{\underline{0}} & \multicolumn{3}{c}{\underline{0}} & \multicolumn{4}{c}{} & \multicolumn{2}{c}{1} \\ \multicolumn{13}{c}{} & \multicolumn{2}{c}{4} \end{array}.$$

Здесь одни и те же основные индексы присвоены паре соответственных мест (индекс 1) и паре подобных мест (индекс 4). Отмеченная таблица переходов автомата, соответствующая проведенной разметке, запишется

	u	v	u	u	v	u	u	u
	0	1	2	3	4	5	6	*
x	2	*	*	4	*	6	4	*
y	1	5	3	*	*	*	*	*

Поскольку начальное состояние 0 представляет лишь пустое слово, его отметку можно считать неопределенной.

После переобозначения состояния (*) получим таблицу:

	—	v	u	u	v	u	u	u
	0	1	2	3	4	5	6	7
x	2	7	7	4	7	6	4	7
y	1	5	3	7	7	7	7	7

Множество состояний автомата Мура, задаваемого последней таблицей, можно разбить на два 0-класса: $a_0 = (0, 2, 3, 5, 6, 7)$ и $b_0 = (1,4)$. Строим 0-таблицу и по ней определяем 1-классы:

	0	1	2	3	4	5	6	7
	a_0	b_0	a_0	a_0	b_0	a_0	a_0	a_0
x	a_0	a_0	a_0	b_0	a_0	a_0	b_0	a_0
y	b_0	a_0	a_0	a_0	a_0	a_0	a_0	a_0

$$a_1 = (0), \quad b_1 = (1,4), \quad c_1 = (2, 5, 7), \quad d_1 = (3,6).$$

Построим 1-таблицу и определим 2-классы:

	0	1	2	3	4	5	6	7
	a_1	b_1	c_1	d_1	b_1	c_1	d_1	c_1
x	c_1	c_1	c_1	b_1	c_1	d_1	b_1	c_1
y	b_1	c_1	d_1	c_1	c_1	c_1	c_1	c_1

$a_2 = (0)$, $b_2 = (1,4)$, $c_2 = (2)$, $d_2 = (3,6)$; $f_2 = (5)$, $g_2 = (7)$.

2-таблица и 3-классы будут иметь вид

	0	1	2	3	4	5	6	7
	a_2	b_2	c_2	d_2	b_2	f_2	d_2	g_2
x	c_2	g_2	g_2	b_2	g_2	d_2	b_2	g_2
y	b_2	f_2	d_2	g_2	g_2	g_2	g_2	g_2

; $a_3 = (0)$, $b_3 = (1)$, $c_3 = (2)$, $d_3 = (3,6)$; $e_3 = (4)$, $f_3 = (5)$, $g_3 = (7)$.

Поскольку 3-классы совпадают, как легко проверить, с 4-классами, они будут и ∞-классами, так что автомат можно упростить за счет объединения состояний 3 и 6. После соответствующей перенумерации состояний отмеченная таблица переходов искомого автомата Мура A запишется

	—	v	u	v	u	u	u
	1	2	3	4	5	6	7
x	3	7	7	7	6	4	7
y	2	5	6	7	7	7	7

.

Переходя к структурному синтезу, выберем в качестве элементов памяти линии задержки с таблицей переходов

	01
0	00
1	11

.

Так как синтезируемый автомат имеет 7 состояний, а элемент памяти — 2 состояния, то следует выбрать 3 элемента памяти ($2^3 \geqslant 7$). Обозначим их внутренние состояния буквами z_1, z_2, z_3, а входные сигналы — s_1, s_2, s_3 (все эти величины могут принимать значения 0 и 1). Состояния автомата A обозначим значениями вектора (z_1, z_2, z_3). Выберем так называемую естественную систему кодирования состояний, при которой каждое состояние кодируется записью своего номера в двоичной системе счисления

$1=001$; $2=010$; $3=011$; $4=100$; $5=101$; $6=110$; $7=111$.

Обозначим физический входной сигнал всего автомата A буквой c, физический выходной сигнал — d и перепишем отмеченную таблицу переходов этого автомата в соответствии с принятой системой кодирования (такую таблицу называют обычно *физической* таблицей переходов автомата, в противоположность рассматривавшейся ранее *абстрактной* таблице переходов, в которой не были учтены особенности кодирования)

d	—	1	0	1	0	0	0
c \ z	001	010	011	100	101	110	111
0	011	111	111	111	110	100	111
1	010	101	110	111	111	111	111

Полученная физическая таблица переходов автомата задает состояния $z_1' = z_1(t+1)$, $z_2' = z_2(t+1)$, $z_3' = z_3(t+1)$ элементов памяти в каждый последующий момент времени $t+1$ как булевы функции их состояний $z_1 = z_1(t)$, $z_2 = z_2(t)$, $z_3 = z_3(t)$ и входного сигнала $c = c(t)$ автомата A в настоящий момент времени t. Из таблицы переходов элемента задержки видно, что его состояние в любой последующий момент времени $t+1$ совпадает с сигналом на его входе в настоящий момент времени. Можно поэтому считать, что выписанная таблица задает структурные функции возбуждения искомого автомата A

$$s_i = \sigma_i(z_1, z_2, z_3, c) \quad (i = 1, 2, 3).$$

Таблицы значений, определяющие соответственно функции s_1, s_2, s_3, выпишем сразу в виде карт Карнау (см. § 4 гл. II)

z_1z_2 \ z_3c	00	01	11	10
00	—	—	0	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

z_1z_2 \ z_3c	00	01	11	10
00	—	—	1	1
01	1	0	1	1
11	0	1	1	1
10	1	1	1	1

z_1z_2 \ z_3c	00	01	11	10
00	—	—	0	1
01	1	1	0	1
11	0	1	1	1
10	1	1	1	0

Методами, развитыми в предыдущей главе, находим минимальные дизъюнктивные нормальные формы для функций возбуждения (входных сигналов элементов памяти) автомата A

$$s_1 = z_1 \vee z_2;\ s_2 = z_3 \vee \bar{z}_2 \vee \bar{z}_1\bar{c} \vee z_1c;\ s_3 = z_1c \vee z_1z_2z_3 \vee \vee \bar{z}_2\bar{z}_3 \vee c\bar{z}_3 \vee \bar{z}_1\bar{c}.$$

Структурную функцию выходов (определяющую выходной сигнал d автомата A как функцию состояний элементов его памяти) также определяем непосредственно по отмеченной физической

таблице переходов автомата A. Карта Карнау для этой функции запишется

z_1z_2 \ z_3'	0	1
00	—	—
01	1	0
11	0	0
10	1	0

По карте Карнау легко найдем минимальную дизъюнктивную форму, задающую требуемую функцию выходов d,

$$d = \bar{z}_1\bar{z}_3 \vee z_2\bar{z}_3.$$

Заметим, что все наши функции оказались определенными не на всех наборах и мы доопределяли их таким образом, чтобы получить по возможности более простые представления для них.

Введем в качестве логических элементов инверторы, а также двухвходовые совпадения и разделения. Обозначая их кружками

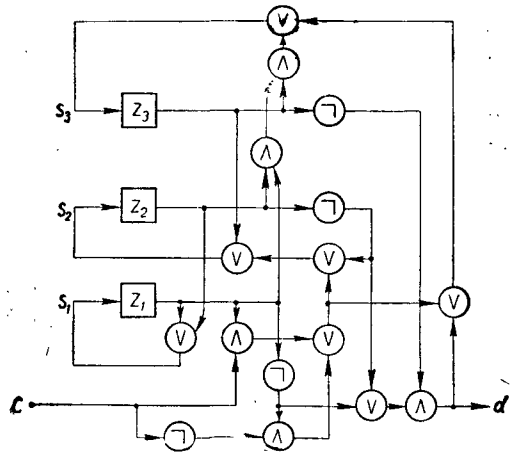

Рис. 9.

с символами соответствующих им операций $\neg$, $\wedge$, $\vee$, а элементы памяти (задержки) квадратиками с буквами z_1, z_2 и z_3 внутри, получим схему автомата A, которая изображена на рис. 9. Эта схема кроме трех элементов задержки включает в себя 4 инвертора, 5 совпадений и 7 разделений (всего 16 логических элементов и 3 элемента памяти).

Глава IV

САМООРГАНИЗУЮЩИЕСЯ СИСТЕМЫ

§ 1. Понятие о самоизменении и самоорганизации в автоматах

С понятием алгоритма и дискретного автомата связывается обычно представление об их неизменности во времени. Соответствующие им алфавитные отображения представляются чем-то жестким, раз навсегда заданным. Применительно к классическому понятию алгоритма и к обычному пониманию способа функционирования дискретного автомата подобное представление является в какой-то мере обоснованным. Вместе с тем ни для кого не секрет, что самые совершенные из моделируемых в настоящее время самоорганизующихся систем моделируются на универсальных электронных цифровых машинах, представляющих собой не что иное, как дискретные автоматы с «жесткой» структурой, с помощью программ, которые являются по существу записями алгоритмов в некоторой специальной форме.

Возникающее здесь противоречие является в значительной мере кажущимся. Истина состоит в том, что различие между «жесткими» и «самоизменяющимися» преобразователями информации в большинстве случаев достаточно условно и определяется не столько конструкцией самого преобразователя, сколько организацией эксперимента с ним. Один и тот же преобразователь информации в одних условиях следует рассматривать как жесткий, неизменный, а в других — как самоизменяющийся и самоорганизующийся.

Для уточнения сказанного рассмотрим произвольный дискретный автомат A с входным алфавитом $\mathfrak{X}$, выходным алфавитом $\mathfrak{Y}$, множеством внутренних состояний $\mathfrak{A}$ и начальным состоянием a_0. Обычное представление о характере функционирования автомата неявно предполагает, что после подачи на его вход какого-либо слова p в алфавите $\mathfrak{X}$ и получения соответствующего выходного слова $q = \zeta(p)$ в алфавите $\mathfrak{Y}$ автомат снова возвращается в начальное состояние. Таким образом, в момент начала подачи каждого нового входного слова автомат оказывается всегда в одном и том же состоянии a_0. Благодаря этому индуцируемое автоматом

отображение ζ является жестким, неизменным в том смысле, что результат преобразования отображением ζ любого входного слова p зависит лишь от самого слова p, а не от того, в какой момент оно было подано на вход автомата.

Назовем каждое из возможных входных слов автомата *вопросом*, а соответствующее ему выходное слово — *ответом*. В таком случае «жесткость» автомата состоит в том что на один и тот же вопрос он всегда и при всех условиях дает один и тот же ответ. Автомат тем самым лишен какой бы то ни было способности к обучению и совершенствованию своих ответов.

Однако описанный выше перевод автомата в начальное состояние перед каждым новым вопросом вовсе не обязателен. Более того, он не предусмотрен непосредственно в том определении функционирования автомата которое было дано в § 6 гл. III. Представляется естественным определять функционирование автомата таким образом, чтобы начало каждого следующего вопроса заставало автомат в том состоянии, в котором он оказался после окончания ответа на предыдущий вопрос При таком определении автомат, который мы ранее считали жестким, неизменным, будет, вообще говоря менять с течением времени свои ответы и может, в частности, оказаться обучающимся самосовершенствующимся и т. п.

Определим теперь более точно способ функционирования дискретного автомата применительно к теории самоорганизующихся систем. Важнейшими понятиями, определяющими способ функционирования автомата, являются понятия *цикла* и *циклирования информации*.

Пусть на вход автомата подана некоторая (конечная или бесконечная) последовательность букв: $x_{i_1} x_{i_2} \ldots x_{i_n}$. Ей соответствует некоторая последовательность букв $y_{j_1} y_{j_2} \ldots y_{j_n}$ на выходе автомата. Предположим, что нами выделена некоторая возрастающая последовательность $k, l, \ldots$ моментов дискретного времени ($1 < k < l < \ldots$). Тогда каждая пара слов $(x_{i_1} x_{i_2} \ldots x_{i_k}, y_{j_1} y_{j_2} \ldots y_{j_k})$, $(x_{i_{k+1}} x_{i_{k+2}} \ldots x_{i_l}, y_{i_{k+1}} y_{i_{k+2}} \ldots y_{j_l}, \ldots$ будет называться *циклом*, а сама операция выделения циклов — *операцией циклирования* (входной и выходной) информации для рассматриваемого автомата.

В дальнейшем будем предполагать, что для каждого рассматриваемого автомата выделен тот или иной *класс допустимых последовательностей* входных букв и что каждая такая последовательность (вместе с соответствующей ей последовательностью выходных букв) разбита на циклы. Операция циклирования оказывается таким образом определенной, вообще говоря, не на одной какой-либо паре последовательностей, а на всех парах допустимых последовательностей.

При абстрактном подходе в понятия цикла и циклирования не вкладывается никакого дополнительного смысла сверх того, что было уже определено. Однако на практике циклирование всегда предполагает, что пара слов (входное и выходное), составляющая

каждый такой абстрактный *цикл*, представляет собой в том или ином смысле законченный реальный цикл функционирования автомата, который можно рассматривать отдельно от остальных циклов. Наиболее часто встречаются два случая: случай, когда первое слово пары (входное слово) представляет собой задаваемый автомату вопрос, а второе слово пары является ответом на этот вопрос, и случай, когда второе слово пары по-прежнему является ответом, а первое, кроме вопроса, включает еще и оценку данного ответа. Разумеется, и в том и в другом случае предполагается, что входящие, быть может, в состав как первого, так и второго слова пустые буквы не должны приниматься во внимание.

Возникающая в указанных двух случаях ситуация изображена соответственно на рис. 10 и 11.

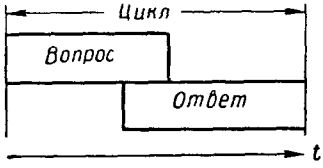

Рис. 10.

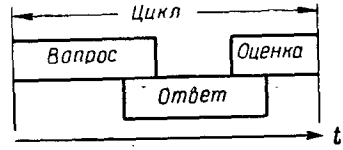

Рис. 11.

Заметим, что в общем случае часто целесообразно при конструировании автоматов осуществлять частичное (а иногда и полное) перекрытие ответа и вопроса (начинать ответ раньше, чем окончился вопрос). Это обстоятельство отражено на рис. 10 и 11. Границы циклов при выполнении операции циклирования определяются также двумя основными способами. Можно, во-первых, просто фиксировать некоторое натуральное число k и потребовать, чтобы входное и выходное слово в каждом цикле содержали ровно по k букв (включая и пустые буквы). Подобное циклирование будем называть *k-циклированием*. Во-вторых, можно определять границу циклов, фиксируя для этой цели специальную букву или слово, называемое *меткой*. Наиболее удобно для разделения циклов ставить такую метку в начале каждого очередного вопроса (будем считать при этом, что метка является частью вопроса). Подобный способ циклирования условимся называть *циклированием по метке*. Само собой разумеется, что комбинация букв, фиксированная в качестве метки, должна использоваться исключительно для этой цели. Внутри цикла также может быть использована метка (например, для указания начала оценки), но эта метка должна отличаться от метки, обозначающей границу цикла. Часто оказывается удобным при конструировании автомата предусматривать выдачу им специальной метки в конце каждого ответа. Будем считать, что при работе автомата встречаются лишь допустимые последовательности входных сигналов и что для каждой такой последовательности произведено соответствующее разбиение на циклы.

Упорядоченная последовательность циклов, предшествующих данному циклу С в той или иной фиксированной допустимой последовательности какого-либо автомата А, называется историей обучения автомата А применительно к циклу С.

Автомат естественно называть *самосовершенствующимся*, или *самообучающимся*, если по мере удлинения истории обучения он улучшает свои ответы. Это определение, разумеется, никоим образом не претендует на точность и должно рассматриваться как предварительное. Определения для понятия совершенствования (самообучения) автоматов будут уточнены в одном из следующих параграфов после предварительного рассмотрения необходимых для подобных определений теоретико-вероятностных понятий. Полезно, однако, уже здесь наметить пути такого уточнения. Прежде всего необходимо уточнить понятие качества ответа с помощью введения некоторой числовой его оценки. При этом условии можно вложить точный смысл в понятие улучшения качества ответа, использованное выше при определении самосовершенствования автоматов.

Далее, необходимо иметь в виду, что даже автоматы с наиболее ярко выраженной тенденцией к самосовершенствованию не обязательно улучшают свои ответы абсолютно на все вопросы. Речь должна идти здесь об улучшении качества ответов в среднем. То же самое касается и истории обучения. Некоторые, относительно редко встречающиеся истории обучения, могут, очевидно, приводить к ухудшению среднего качества ответов, однако, если остальные истории обучения приводят к резкому улучшению качества ответов, автомат вполне может быть признан самосовершенствующимся.

Наконец, следует, очевидно, различать самосовершенствование, заранее предопределенное конструктором автомата (независимое от вида истории обучения), и действительно самопроизвольное самосовершенствование, определяемое фактически имевшей место историей обучения и потому не планируемое заранее. Ясно, что лишь второй тип самосовершенствования действительно заслуживает это название. Что же касается первого типа, то в этом случае конструктор фактически заранее вкладывает в автомат правильные ответы, но с целью имитирования процесса самосовершенствования заставляет автомат до поры до времени держать эти знания под спудом. В результате автомат вначале выдает ответы плохого качества и лишь по истечении некоторого времени (некоторого числа циклов) начинает, используя вложенные в него знания, давать правильные ответы. Вряд ли, однако, подобное улучшение качества ответов автомата с течением времени следует называть самосовершенствованием.

Все сказанное дает известное понятие о тех трудностях, которые необходимо преодолеть при точном определении понятия самосовершенствования. В аналогичном положении находится и понятие *самоорганизации*, которое в нашем представлении является

несколько более общим, чем понятие самосовершенствования. При самосовершенствовании обязательно должно улучшаться качество ответов. При самоорганизации качество ответов может вовсе не определяться. Необходимо лишь, чтобы автомат по мере обучения увеличивал в среднем определенность своих ответов. Соответствующее уточненное определение будет дано после введения необходимых теоретико-вероятностных понятий.

При уточнении понятий самоорганизации и самосовершенствования удобно пользоваться так называемым *циклическим приведением* автоматов. Циклическое приведение определяется для автоматов, у которых фиксировано множество допустимых входных последовательностей и произведено циклирование входной и выходной информации. Для любого автомата A при выполнении этих условий входной и выходной алфавиты могут быть заменены следующим образом: буквами нового входного алфавита $\mathfrak{X}'$ считаются все различные входные слова всех циклов во всех допустимых последовательностях, буквами нового выходного алфавита $\mathfrak{Y}'$ аналогично считаются все различные выходные слова указанных циклов.

Для любого состояния a автомата A и любой буквы x' алфавита $\mathfrak{X}'$ (входного слова какого-нибудь цикла) через $\delta'(a, x')$ обозначим то состояние, в которое переходит автомат A из состояния a под действием входного слова x'. $\lambda'(a, x')$ обозначим выходное слово, выдаваемое автоматом A под действием входного слова x', в случае, если состояние a принято за начальное состояние. Любая допустимая входная последовательность автомата A может рассматриваться как последовательность $x'(1)x'(2)$ ... букв нового входного алфавита $\mathfrak{X}'$. Рассмотрим множество $\mathfrak{A}'$ всех тех состояний автомата A, в которые он может быть переведен из начального состояния a_0 входными словами вида $x'(1)\,x'(2)\ldots\,x'(k)$ $(k \geqslant 0)$, т. е. всевозможными начальными отрезками различных допустимых входных последовательностей. В это множество входит обязательно само начальное состояние a_0.

Теперь нетрудно построить автомат A', у которого множеством внутренних состояний является множество $\mathfrak{A}'$, входной алфавит совпадает с множеством $\mathfrak{X}'$, а выходной алфавит — с множеством $\mathfrak{Y}'$. Функциями переходов и выходов этого автомата будут служить определенные выше функции δ' и λ', а начальным состоянием — состояние a_0. Предполагается, что на вход построенного автомата будут подаваться лишь допустимые входные последовательности (перезаписанные в алфавите $\mathfrak{X}'$). В таком случае, как легко проверить, определение автомата A вполне корректно: как состояний, так и выходных букв вполне достаточно для описания функционирования автомата при воздействии на него произвольной допустимой входной последовательностью.

Построенный таким образом автомат A' условимся называть *циклическим приведением* исходного автомата A. Циклирование информации, соответствующее исходному циклированию, будет,

очевидно, в автомате A' 1-циклированием. Иными словами, в циклическом приведении произвольного автомата как вопросы, так и ответы являются однобуквенными.

При циклическом приведении автоматов число их внутренних состояний может только уменьшаться либо, в крайнем случае, сохраняться неизменным. Число букв входного и выходного алфавитов, вообще говоря, будет при этом возрастать. Ясно, что при k-циклировании исходной информации циклическое приведение не может вызвать переход от конечных алфавитов к бесконечным. Однако в случае циклирования по метке подобный переход является уже вполне возможным — конечный входной или выходной алфавит после циклического приведения может превратиться в бесконечный.

Рассмотрим в виде примера циклическое приведение автомата A с тремя состояниями 1, 2, 3, двумя входными буквами x, y и двумя выходными буквами u, v, функции переходов и выходов которого заданы соответственно таблицами

	1	2	3
x	2	2	2
y	3	2	1

;

	1	2	3
x	u	v	u
y	v	u	v

.

Предполагая все входные последовательности допустимыми и принимая состояние 1 за начальное состояние, в результате циклического приведения придем к автомату A' с двумя состояниями, четырьмя входными буквами $x_1 = xx$, $x_2 = xy$, $x_3 = yx$, $x_4 = yy$, четырьмя выходными буквами $v_1 = uu$, $v_2 = uv$, $v_3 = vu$, $v_4 = vv$. Функции переходов и выходов этого автомата задаются соответственно таблицами

	1	2
x_1	2	2
x_2	2	2
x_3	2	2
x_4	1	2

;

	1	2
x_1	v_2	v_4
x_2	v_1	v_3
x_3	v_3	v_2
x_4	v_4	v_1

.

С помощью циклического приведения весьма наглядно решается вопрос о том, является рассматриваемый автомат применительно к данному циклированию жестким или самоизменяющимся. Действительно, можно сформулировать следующее предложение.

Для того чтобы дискретный автомат A с заданным циклированием был жестким (т. е. *не изменял бы свои ответы на одни и те же вопросы с течением времени*), *необходимо и достаточно, чтобы после циклического приведения функция выходов $\lambda'(a, x)$ не зависела от состояний приведенного автомата A'.*

Независимость функции выходов от состояний автомата означает, очевидно, равенство между собой всех элементов каждой строки таблицы выходов этого автомата (элементы, стоящие в разных строках, могут быть, разумеется, различными).

Применительно к рассмотренному выше примеру только что сформулированное правило сразу обнаруживает самоизменяемость автомата A для случая 2-циклирования его входной информации.

Легко понять, что автомат, у которого функция выходов не зависит от состояний, может быть заменен эквивалентным ему (т. е. индуцирующим то же самое алфавитное отображение) автоматом, имеющим одно-единственное внутреннее состояние. Автомат с одним внутренним состоянием является по существу автоматом без памяти. В абстрактной теории автоматов показывается, что всякий дискретный автомат A может быть минимизирован, т. е., иными словами, заменен эквивалентным ему автоматом B (абсолютной минимизацией автомата A), имеющим наименьшее число состояний среди всех автоматов, которые индуцируют то же самое алфавитное отображение, что и автомат A.

Если после циклического приведения любого дискретного автомата A (с заданным циклированием информации) осуществить еще абсолютную минимизацию, то получим операцию, которую будем называть *полным циклическим приведением* рассматриваемого автомата A (применительно к заданному циклированию). Из приведенных выше рассуждений вытекает справедливость следующего предложения.

Для того чтобы дискретный автомат A с заданным циклированием информации обладал свойством неизменности своих ответов во времени, необходимо и достаточно, чтобы в результате полного циклического приведения автомата A получался автомат, не имеющий памяти.

Верно и обратное: наличие нетривиальной памяти у автомата, получающегося в результате полного циклического приведения рассматриваемого автомата A, означает, что автомат A относится (применительно к заданному циклированию) к самоизменяемым.

Относительность свойства самоизменяемости автоматов (зависимость его от способа циклирования входной информации) легко проиллюстрировать на примере автомата C с двумя состояниями, заданного таблицами переходов и выходов

	1	2
x	2	1
y	1	2

;

	1	2
x	u	v
y	v	u

.

При произвольных допустимых входных последовательностях в случае 2-циклирования входной информации результатом циклического приведения автомата C будет автомат с одним состоянием. Таким образом, даже без минимизации получим автомат без памяти и, в силу сформулированного выше критерия, приходим к выводу о жесткости, неизменности автомата C. В то же самое время при 1-циклировании входной информации автомат C должен быть, очевидно, признан самоизменяющимся. В практических

задачах удается достаточно четко разграничить жесткие и самоизменяющиеся автоматы лишь потому, что циклирование входной информации бывает заранее заданным.

Заметим, что в рассмотренных критериях и разобранных на их основе примерах речь шла не о самоорганизации или самосовершенствовании, а лишь о самоизменяемости автоматов. Разбор примеров самоорганизации и самосовершенствования будет проведен в последующих параграфах после создания соответствующей математической базы и введения точных определений.

В заключение настоящего параграфа остановимся на одном терминологическом вопросе. Речь идет об употреблении терминов «система» или «автомат» в сочетании с понятиями самоизменения, самоорганизации, самосовершенствования и самообучения. Как явствует из уже проведенных рассмотрений, все указанные понятия можно развивать применительно к дискретным автоматам. Однако при подобном подходе к делу по существу теряется возможность проникнуть в структуру соответствующего процесса (самоизменения, самоорганизации и т. п.).

Изучение структуры процессов самоизменения и самоорганизации облегчается при представлении таких процессов не в виде отдельных автоматов (алгоритмов), а в виде систем автоматов (алгоритмов). В простейшем случае такая система состоит из двух автоматов (алгоритмов). Первый из них, называемый *рабочим* автоматом (алгоритмом) непосредственно перерабатывает информацию, подаваемую на вход системы. Второй автомат (алгоритм), называемый *контролирующим*, или *обучающим*, оценивает результаты функционирования рабочего автомата (алгоритма) и вносит в него соответствующие изменения (в случае, когда рассматриваются автоматы, эти изменения вносятся в функции переходов и выходов рабочего автомата).

Над контролирующим автоматом (алгоритмом) первой ступени может быть помещен контролирующий автомат (алгоритм) второй ступени, в задачу которого входит оценивать действия автомата (алгоритма) первой ступени и вносить в него необходимые изменения. По аналогии можно вводить контролирующие автоматы (алгоритмы) третьей, четвертой и какой угодно более высокой ступени. Возникающие подобным образом иерархии автоматов (алгоритмов) и будем называть *системами*, развивая применительно к ним понятия самоизменения, самоорганизации и самосовершенствования.

Разумеется, в абстрактном плане любая сколь угодно сложная система автоматов эквивалентна одному автомату, однако подобное сведение систем к отдельным автоматам приводит к потере возможности изучения некоторых интересных для практики свойств таких систем и прежде всего закономерностей циркуляции информации внутри самой системы. Поэтому в дальнейшем будем иметь дело не только с автоматами (алгоритмами), рассматриваемыми абстрактно, но и с системами автоматов (алгоритмов), для изуче-

ния которых особый интерес представляет их внутренняя структура, т. е. связи между отдельными автоматами (алгоритмами), составляющими систему.

§ 2. Некоторые вспомогательные сведения из теории вероятностей

В настоящем параграфе излагаются некоторые сведения из теории вероятностей, необходимые для дальнейших наших построений. Ввиду того что это изложение носит вспомогательный характер, доказательства большинства формулируемых предложений не приводятся. В случае необходимости читатель может ознакомиться с соответствующими доказательствами по монографиям В. Феллера [81] и Г. Крамера [48]. Предполагается, что читатель знаком с такими элементарными понятиями теории вероятностей как понятие события, вероятности события и т. п.

Для построения теории самоорганизующихся систем весьма существенное значение имеет понятие *случайной величины*. Ограничимся рассмотрением случайных величин, принимающих лишь вещественные значения. При этом, кроме обычных, так называемых одномерных случайных, величин, нам придется рассматривать и *многомерные* случайные величины, значениями которых будут конечные упорядоченные наборы вещественных чисел или, что то же самое, — вещественные векторы той или иной (конечной) размерности.

Важно также различать *непрерывные* и *дискретные* случайные величины. Непрерывная случайная величина может принимать любые значения в той или иной области (открытом множестве) соответствующего векторного пространства, например на каком-нибудь интервале вещественной оси (в том числе и на всей оси) в случае одномерных случайных величин. Совокупностями же возможных значений дискретной случайной величины могут служить лишь дискретные множества точек, т. е. такие множества, каждую точку которых можно заключить в сферу (быть может, очень малого радиуса), не содержащую других точек того же самого множества. Примером дискретного множества может служить множество всех точек того или иного эвклидова пространства, имеющих целочисленные координаты.

Свойство случайности рассматриваемых нами величин проявляется в так называемых *испытаниях*. В каждом испытании рассматриваемая случайная величина принимает то или иное значение из области ее определения. Вероятность, принятая случайной величиной того или иного значения, определяется законом распределения этой случайной величины. Закон распределения дискретной случайной величины x (одномерной или многомерной) задается с помощью вещественной функции $f(x)$, определенной для всех значений, которые может принимать данная случайная величина, так что для любого такого значения x_i величина $f(x_i)$

равняется вероятности принятия случайной величиной x значения x_i в данном испытании.

Области определения рассматриваемых нами дискретных случайных величин могут быть либо конечными, либо счетными бесконечными. В обоих случаях выполняется, очевидно, условие нормировки

$$\sum_i f(x_i) = 1, \tag{47}$$

где суммирование предполагается распространенным на всю область определения данной случайной величины.

Когда случайная величина x непрерывна, закон ее распределения задается с помощью так называемой *функции плотности вероятности* $\varphi(x)$. Эта функция предполагается определенной в области M определения рассматриваемой случайной величины x и интегрируемой в этой области. При каждом очередном испытании вероятность $p(N)$ того, что случайная величина x примет значение из некоторой подобласти N ее области определения, равняется интегралу от плотности вероятности, взятому по этой подобласти:

$$p(N) = \int_N \varphi(x)\, dx. \tag{48}$$

Отсюда непосредственно следует выполнение условия нормировки

$$\int_M \varphi(x)\, dx = 1. \tag{49}$$

Две случайные величины x и y называются *независимыми между собой*, если принятие величиной x того или иного значения не меняет закона распределения величины y и наоборот. Аналогично независимость для любого множества случайных величин означает, что принятие всеми входящими в это множество величинами, кроме одной величины x, каких-либо значений не меняет закона распределения этой последней величины при любом выборе величины x из указанного множества.

Испытания, проводимые с той или иной случайной величиной x, называются *независимыми* испытаниями, если закон распределения величины x остается неизменным на каждом испытании и не зависит, следовательно, от значений, которые величина x принимала в предыдущих испытаниях.

Область определения непрерывной случайной величины можно всегда, в случае необходимости, распространить на все пространство, полагая, что везде, кроме первоначальной области определения, плотность вероятности равняется нулю.

Можно также приближать непрерывные законы распределения дискретными законами и наоборот. В первом случае достаточно разбить область определения M соответствующей непрерывной случайной величины x на конечное число достаточно малых (не

только по объему, но и по диаметру) подобластей M_1, M_2, ..., M_n, выбрать внутри каждой из таких подобластей M_i точку x_i и ввести дискретный закон распределения $f(x)$ на выбранных точках, полагая $f(x_i) = \int\limits_{M_i} \varphi(x)\,dx$ $(i = 1, 2, \ldots, n)$, где $\varphi(x)$ — плотность вероятности исходной непрерывной случайной величины. При этом вероятности, относившиеся ранее к соответствующим подобластям, концентрируются в отдельных точках.

При обратном переходе от дискретного закона распределения к непрерывному, наоборот, производится «размазывание» вероятностей, сосредоточенных первоначально в отдельных точках x_i, на соответствующие подобласти M_i так, чтобы для возникающей таким образом функции плотности вероятности $\varphi(x)$ оказывались справедливыми соотношения

$$\int\limits_{M_i} \varphi(x)\,dx = f(x_i) \qquad (i = 1, 2, \ldots, n).$$

Часто приходится рассматривать бесконечные последовательности дискретных законов распределения $f_i(x)$ $(i = 1, 2, \ldots)$, имеющие некоторый непрерывный закон распределения с функцией плотности вероятности $\varphi(x)$ в качестве своего так называемого *предельного закона распределения*. В понятие предельного распределения вкладывается следующий точный смысл. Во-первых, области значений M_i дискретных случайных величин с законами распределения $f_i(x)$ сходятся к области значений M непрерывной случайной величины x. В рассматриваемых нами случаях такая сходимость будет означать, что все M_i содержатся в M, и для любого сколь угодно малого положительного числа ε, любого сколь угодно большого числа N и для любой точки x из M в каждом из множеств M_i с $i > N$ найдется хотя бы по одной точке, удаленной от точки x меньше чем на ε. Во-вторых, для любой подобласти P области M и для любого сколь угодно малого положительного числа δ для всех номеров i, начиная с некоторого номера, должно выполняться неравенство

$$\left| \int\limits_P \varphi(x)dx - \sum_{x \in P \cap M_i} f_i(x) \right| < \delta. \tag{50}$$

Суммирование в левой части этой формулы берется по всем точкам из M_i, содержащимся в подобласти P.

Для дальнейших построений важное значение имеют понятия *среднего значения* (математического ожидания) случайной величины и ее *центральных моментов* второго порядка.

Пусть $\vec{x} = (x_1, x_2, \ldots, x_n)$—$n$-мерная непрерывная случайная величина с функцией плотности вероятности $\varphi(x_1, x_2, \ldots, x_n)$. Как отмечалось выше, не нарушая общности, функцию φ можно считать определенной на всем бесконечном пространстве. Тогда среднее значение случайной величины x определяется как вектор $\vec{m} = (m_1, m_2, \ldots, m_n)$, вычисляемый по формуле

$$\vec{m} = (m_1, m_2, \ldots, m_n) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \ldots \int_{-\infty}^{\infty} \vec{x}\, \varphi(x_1, x_2, \ldots$$

$$\ldots, x_n)\, dx_1\, dx_2 \ldots dx_n. \qquad (51)$$

Центральные моменты второго порядка λ_{ik} определяются по формулам

$$\lambda_{ik} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \ldots \int_{-\infty}^{\infty} (x_i - m_i)(x_k - m_k)\, \varphi(x_1, x_2, \ldots$$

$$\ldots, x_n)\, dx_1 dx_2 \ldots dx_n \qquad (52)$$

$$(i, k = 1, 2, \ldots, n).$$

Для одномерной случайной величины x имеется единственный центральный момент второго порядка, определяемый по формуле

$$d = \int_{-\infty}^{\infty} (x - m)^2 \varphi(x)\, dx. \qquad (53)$$

Этот момент называют обычно *дисперсией* соответствующего распределения.

Все перечисленные понятия и их определения естественно переносятся и на дискретные случайные величины.

Необходимо лишь в формулах (51)—(53) заменить интегрирование суммированием, распространенным на всю область определения M соответствующей дискретной (векторной) величины $\vec{x}$, а вместо функции плотности вероятности $\varphi(x_1, x_2, \ldots, x_n)$ писать функцию распределения вероятности этой величины $f(\vec{x})$. В результате формула (51), например, перепишется

$$\vec{m} = \sum_{\vec{x} \in M} \vec{x} f(\vec{x}). \qquad (54)$$

Аналогично изменятся и все остальные формулы.

Для многомерных случайных величин центральные моменты второго порядка λ_{ik} удобно объединять в матрицу $\|\lambda_{ik}\|$ и строить из них (в случае их конечности) квадратичную форму $Q(t_1, t_2, \ldots, t_n) = \sum_{i,k} \lambda_{ik} t_i t_k$. Эту форму с помощью ортогонального преобразования (поворота) системы координат всегда можно привести к виду $\sum_{i=1}^{m} \alpha_i (t_i')^2$, где t_i' — новые координаты, а α_i — положительные коэффициенты ($i = 1, 2, \ldots, m$). Формы такого вида называются положительно полуопределенными. Если $m = n$, т. е., если число квадратов после приведения формы Q в точности равно размерности пространства, то форма Q называется *положительно определенной*. В этом случае ее определитель $|Q| = |\lambda_{ik}|$ обязательно отличен от нуля и (строго) положителен.

Для положительно определенной формы $Q = \sum_{i,k} \lambda_{ik} t_i t_k$ можно определить обратную форму $Q^{-1} = Q^{-1}(t_1, t_2, \ldots, t_n) = \sum_{i,k} \mu_{ik} t_i t_k$, коэффициенты которой задаются формулами $\mu_{ik} = \frac{Q_{ik}}{|Q|}$, где Q_{ik} — алгебраическое дополнение элемента λ_{ik} определителя $|Q| = |\lambda_{ik}|$ ($i, k = 1, 2, \ldots, n$). Эта форма снова будет положительно определенной. Полученная матрица $\|\mu_{ik}\|$ будет, очевидно, обратна матрице $\|\lambda_{ik}\|$, поскольку последняя — симметрична (симметричной будет, разумеется, также и матрица $\|\mu_{ik}\|$).

В теории вероятностей имеет большое значение следующий фундаментальный результат [48].

Теорема 1. *Если n-мерные случайные (непрерывные или дискретные) величины $x_1, x_2, \ldots, x_k$ независимы и имеют одно и то же распределение с конечными центральными моментами второго порядка λ_{ik}, для которых форма $Q(t_1, t_2, \ldots, t_n) = \sum_{i,k} \lambda_{ik} t_i t_k$ положительно определена, и со средним значением, равным нулю, то при $k \to \infty$ величина $x = \frac{1}{\sqrt{k}}(x_1 + x_2 + \ldots + x_k)$ имеет предельное непрерывное распределение также с нулевым средним значением и с (многомерной) функцией плотности вероятности*

$$\varphi(t_1, t_2, \ldots, t_n) = \frac{1}{(2\pi)^{\frac{n}{2}} \sqrt{|Q|}} e^{-\frac{1}{2} Q^{-1}(t_1, t_2, \ldots, t_n)}.$$

В случае, когда форма Q вырожденная, переходят к рассмотрению некоторого подпространства L исходного пространства, заменяя случайные величины $x_1, x_2, \ldots, x_k$ их проекциями на это подпространство. Подпространство L выбирается таким образом, чтобы полученная в результате указанного проектирования новая форма Q центральных моментов была уже положительно определенной, а проектирование в любое перпендикулярное к нему подпространство приводило к вырожденной форме (такое пространство всегда существует). Применение теоремы 1 в построенном пространстве L даст в этом случае распределение и в исходном пространстве, поскольку все возможные значения случайных величин $x_1, x_2, \ldots, x_k$ (а значит, и их суммы) с вероятностью 1 лежат в этом подпространстве.

В ряде случаев ограничиваются выбором какого-нибудь подпространства K максимальной возможной размерности из числа всех подпространств с невырожденной формой Q. Покажем, что подпространство L можно получить из подпространства K в результате невырожденного линейного преобразования.

Если в условиях теоремы 1 среднее значение величин $x_1, x_2, \ldots, x_k$ не равно нулю, то, обозначая это среднее значение через $m = (m_1, m_2, \ldots, m_k)$, легко найдем, что средним значением случайной величины $x = \frac{1}{\sqrt{k}}(x_1 + x_2 + \ldots + x_k)$ будет величина $(m_1 \sqrt{k}$

$m_2\sqrt{k}, \ldots, m_n\sqrt{k}$) и при достаточно большом k хорошим приближением для закона распределения случайной величины x будет непрерывный закон с многомерной функцией плотности вероятности вида

$$\varphi(t_1, t_2, \ldots, t_n) = \frac{1}{(2\pi)^{\frac{n}{2}}\sqrt{|Q|}} e^{-\frac{1}{2}Q^{-1}(t_1-m_1\sqrt{k}, t_2-m_2\sqrt{k}, \ldots, t_n-m_n\sqrt{k})}. \quad (55)$$

Применим теперь теорему 1 и формулу (55) к так называемому *биномиальному распределению*. Биномиальное распределение возникает в результате проведения независимых испытаний по так называемой *схеме Бернулли*. Эта схема, помимо свойства независимости испытаний, характеризуется еще и тем, что при каждом испытании возможны только два исхода, возникающие с вероятностями p и $q = 1 - p$ соответственно. Назовем первый исход успехом, а второй — неуспехом испытания и введем случайную величину x_i, принимающую значение 1 в случае успеха i-го испытания и значение 0 — в случае его неуспеха ($i = 1, 2, \ldots, n$).

Случайная величина $k = x_1 + x_2 + \ldots + x_n$ равняется, очевидно, общему числу успехов при n независимых испытаниях. Обозначая C_n^k число сочетаний из n по k (по определению $C_n^0 = 1$), легко найдем, что закон распределения (дискретной) случайной величины k задается функцией

$$f_n(k) = C_n^k p^k q^{n-k} \quad (k = 0, 1, \ldots, n). \quad (56)$$

Этот закон называется биномиальным законом распределения, поскольку величина $C_n^k p^k q^{n-k}$ представляет собой, очевидно, $(n-k+1)$-ый член разложения выражения $(p+q)^n$ по формуле бинома Ньютона.

Случайные величины $x_1, x_2, \ldots, x_n$ имеют один и тот же закон распределения со средним значением $m = 1 \cdot p + 0 \cdot (1 - p) = p$ и дисперсией (центральным моментом второго порядка) $d = (1 - p)^2 \cdot p + (0 - p)^2(1 - p) = p(1 - p)$. Из теоремы 1 и формулы (55) вытекает, что для достаточно больших значений n хорошим приближением закона распределения величины $x = \frac{k}{\sqrt{n}} = \frac{1}{\sqrt{n}}(x_1 + x_2 + \ldots + x_n)$ будет служить закон распределения с функцией плотности вероятности вида

$$\varphi(x) = \frac{1}{\sqrt{2\pi p(1-p)}} e^{-\frac{(x-p\sqrt{n})^2}{2p(1-p)}}. \quad (57)$$

Закон распределения с функцией плотности вероятности вида $\frac{1}{\sqrt{2\pi a}} e^{-\frac{(x-m)^2}{2a}}$ (при $a > 0$) будем называть (обобщенным) *одномер-*

ным нормальным законом распределения. Нетрудно найти, что среднее значение случайной величины, распределенной по этому закону, равняется m, а ее дисперсия равна a.

Легко понять, что при умножении нормально распределенной случайной величины x на постоянный множитель c новая величина $y = cx$ будет также нормально распределенной случайной величиной, причем ее среднее значение будет в c раз больше, а дисперсия — в c^2 раз больше по сравнению соответственно со средним значением и дисперсией исходной величины x.

Сопоставляя полученные результаты с формулой (57), придем к следующему предложению.

Теорема 2. *При достаточно большом числе n испытаний Бернулли с вероятностью успеха p закон распределения для общего числа успехов k может быть приближенно выражен нормальным законом с функцией плотности вероятности вида* $\varphi(k) = \frac{1}{\sqrt{2\pi np(1-p)}} e^{-\frac{(k-pn)^2}{2np(1-p)}}$. *Среднее значение соответствующей этому закону случайной величины равняется pn, а ее дисперсия равна $np(1-p)$, что совпадает с точными значениями средней величины и дисперсии исходной дискретной случайной величины k.*

Благодаря тому, что при выводе утверждения, содержащегося в теореме 2, величина x умножалась на множитель $\sqrt{n}$, непрерывное распределение φ для величины k, полученное в теореме 2, не обладает, вообще говоря, свойством предельного распределения для исходного (дискретного) распределения f величины k при неограниченном росте числа испытаний n.

Нетрудно, однако, заметить, что при достаточно больших значениях n вычисленные в соответствии с распределениями φ и f вероятности нахождения величины k на любом интервале, длина которого имеет порядок величины $\sqrt{n}$ (т. е. имеет вид $c\sqrt{n}$, где c—константа), будут отличаться между собой сколь угодно мало.

На практике приходится обычно вычислять вероятности нахождения величины k на интервалах вида $[pn,\ pn \pm z\sigma]$, где величина $\sigma = \sqrt{np(1-p)}$, равная корню квадратному из дисперсии распределения φ (а значит, и распределения f), носит название *среднеквадратичного разброса* (или среднеквадратичной ошибки) распределений φ и f. Справедлива следующая теорема.

Теорема 3. *Для любого положительного числа z вероятность $\varrho(z)$ того, что общее число успехов в n испытаниях Бернулли с вероятностью успеха, равной p, будет заключено в интервале $[pn,\ pn \pm z\sqrt{np(1-p)}]$, выражается формулой* $\varrho(z) \approx \Phi(z) = \frac{1}{\sqrt{2\pi}} \times \int_0^z e^{-\frac{x^2}{2}}\, dx$. *При любом z, выбирая n достаточно большим, можно сделать ошибку в вычислении $\varrho(z)$ по этой формуле сколь угодно малой.*

Приведем численные значения функции $\Phi(z)$ для некоторых зна-

чений z с четырьмя десятичными знаками: $\Phi(1) = 0{,}3413$; $\Phi(2) = 0{,}4772$; $\Phi(3) = 0{,}4986$; для $z \geqslant 4$ значения $\Phi(z)$ отличаются от 0,5000 менее чем на половину единицы четвертого десятичного знака.

Приближенную формулу в условии теоремы 3 будем называть формулой Муавра—Лапласа. Как указано в теореме 3, точность этой формулы возрастает по мере роста числа n проведенных испытаний.

Рассмотрим серию независимых испытаний, каждое из которых имеет m различных исходов, и пусть p_i ($p_i > 0$) означает вероятность i-го исхода испытания ($i = 1, 2, \ldots, m$). Обозначим общее число проведенных испытаний буквой n, а число тех из них, которые кончились i-ым исходом, — k_i ($i = 1, 2, \ldots, m$). Легко понять, что каждая из величин k_i, рассматриваемая сама по себе, распределена по биномиальному закону. Поставим задачу нахождения совместного (многомерного) закона распределения нескольких величин k_i, например величин $k_1, k_2, \ldots, k_r$ ($1 \leqslant r \leqslant m$). Как нетрудно проверить, решение этой задачи дается формулой

$$f(k_1, k_2, \ldots, k_r) =$$

$$= \frac{n!}{k_1!\, k_2! \ldots k_r!(n - k_1 - k_2 - \ldots - k_r)!} \times$$

$$\times p_1^{k_1} p_2^{k_2} \ldots p_r^{k_r} (1 - p_1 - p_2 - \ldots - p_r)^{n-k_1-k_2-\ldots-k_r}. \qquad (58)$$

Для этого закона распределения, который называется обычно *полиномиальным законом распределения*, можно точно так же, как было сделано выше, применительно к его частному (одномерному) случаю, получить непрерывное приближение. С этой целью рассмотрим многомерные случайные величины $x^j = (x_1^j, x_2^j, \ldots, x_r^j)$, такие, что величина x^j принимает одно из значений (100 ... 0), (010 ... 0), ..., (00 ... 01) или (000 ... 00) в соответствии с тем, какой исход имело j-е испытание рассматриваемой нами серии — первый, второй, ..., r-й или любой отличный от них ($j = 1, 2, \ldots, n$). Все случайные величины x^j имеют один и тот же закон распределения, их средние значения равны, очевидно, $(p_1, p_2, \ldots, p_r)$. Центральный момент второго порядка λ_{ii} равен, очевидно, величине $(1 - p_i)^2 p_i + (0 - p_i)^2 (1 - p_i) = p_i(1 - p_i)$ ($i = 1, 2, \ldots, r$). Центральный момент второго порядка λ_{ik} при $i \neq k$ также легко вычислить: $\lambda_{ik} = (1 - p_i)(0 - p_k) p_i + (0 - p_i)(1 - p_k) p_k + (0 - p_i)(0 - p_k)(1 - p_i - p_k) = -p_i p_k$.

Определитель $|\lambda_{ik}|$ матрицы $\|\lambda_{ik}\|$ центральных моментов будет в этом случае равен, как нетрудно проверить, произведению $p_1 p_2 \ldots p_r(1 - p_1 - p_2 - \ldots - p_r)$. Таким образом, матрица $\|\lambda_{ik}\|$ будет вырожденной лишь в случае, когда $r = m$. Во всех остальных случаях квадратичная форма $Q(t_1, t_2, \ldots, t_r) = \sum_{i,k} \lambda_{ik} t_i t_k =$

$= \sum_{i=1}^{r} p_i\ (1 - p_i) t_i^2 - \sum_{i \neq k} p_i p_k t_i t_k$ будет положительно определенной, так как ее определитель $|Q| = |\lambda_{ik}|$ положителен.

Применяя теорему 1 к случайной величине $y = \frac{1}{\sqrt{n}}(y_1 + y_2 + \ldots + y_n)$, где $y_i = (x_1^i - p_1,\ x_2^i - p_2,\ \ldots,\ x_r^i - p_r)$, приходим к выводу, что при $r < m$ она имеет предельный (при $n \to \infty$) закон распределения с функцией плотности вероятности вида

$$\frac{1}{(2\pi)^{\frac{r}{2}} \sqrt{|Q|}} e^{-\frac{1}{2} Q^{-1}(t_1, t_2, \ldots, t_r)}.$$

Многомерная случайная величина $z = \left(\frac{k_1}{n} - p_1,\ \frac{k_2}{n} - p_2,\ \ldots,\ \frac{k_r}{n} - p_r\right)$ связана с величиной y соотношением $z = \frac{1}{\sqrt{n}} y$ и будет поэтому иметь тот же закон распределения, но с дисперсией, в n раз меньшей дисперсии величины y. Следовательно, функция плотности вероятности предельного закона распределения для z запишется

$$\frac{1}{\left(\frac{2\pi}{n}\right)^{\frac{r}{2}} \sqrt{|Q|}} e^{-\frac{n}{2} Q^{-1}(z_1, z_2, \ldots, z_r)}.$$

Теперь нетрудно установить следующий результат.

Теорема 4. *Пусть дана серия независимых испытаний с m различными исходами, имеющими соответственно вероятности p_1, p_2, ..., p_m (одинаковые для всех испытаний). Если в серии из n испытаний через k_1, k_2, ..., k_m обозначить количества испытаний, закончившихся соответственно 1-ым, 2-ым, ..., m-ым исходом, то для любого наперед заданного положительного числа ε для вероятности ϱ одновременного выполнения всех неравенств $\left|\frac{k_i}{n} - p_i\right| < \varepsilon$ $(i = 1, 2, \ldots, m)$ существует оценка $\varrho > 1 - \frac{a}{n^{\frac{m-1}{2}}} e^{-bn}$ (где a и b — положительные константы, не зависящие от n).*

Для доказательства сформулированной теоремы заметим прежде всего, что все неравенства $\left|\frac{k_i}{n} - p_i\right| < \varepsilon$ $(i = 1, 2, \ldots, m)$ заведомо выполняются, если выполнены $m - 1$ неравенств

$$\left|\frac{k_i}{n} - p_i\right| < \frac{\varepsilon}{m-1} \quad (i = 1, 2, \ldots, m-1). \tag{59}$$

Действительно,

$$\left|\frac{k_m}{n} - p_m\right| = \left|\frac{n - k_1 - k_2 - \ldots - k_{m-1}}{n} - (1 - p_1 - p_2 - \ldots\right.$$

$$\ldots - p_{m-1})\Big| = \left|\left(\frac{k_1}{n} - p_1\right) + \left(\frac{k_2}{n} - p_2\right) + \ldots +$$

$$+ \left(\frac{k_{m-1}}{n} - p_{m-1}\right)\right| < \frac{\varepsilon}{m-1}(m-1) = \varepsilon .$$

Как следует из проведенных перед формулировкой теоремы 4 рассмотрений, величины $z_i = \frac{k_i}{n} - p_i$ $(i = 1, 2, \ldots, m-1)$ имеют предельный (при $n \to \infty$) закон распределения с функцией плотности вероятности вида $\varphi(z_1, z_2, \ldots, z_{m-1}) = \alpha n^{\frac{m-1}{2}} e^{-nP(z_1, z_2, \ldots, z_{m-1})}$, где α — некоторая положительная константа, а P — положительно определенная квадратичная форма с коэффициентами, не зависящими от n. Для достаточно больших n вероятность β того, что хотя бы одно из неравенств (59) не выполняется, имеет, очевидно, оценку сверху вида

$$\beta < \underset{R_1}{\int\int} \ldots \int \varphi(z_1, z_2, \ldots, z_{m-1})\, dz_1 dz_2 \ldots dz_{m-1}, \qquad (60)$$

где область R_1 — внешняя часть гиперкуба, ограниченного гиперплоскостями $z_i = \delta_1 \left(\delta_1 < \frac{\varepsilon}{m-1},\ i = 1, 2, \ldots, m-1\right)$.

После поворота осей координат с целью приведения формы P к сумме квадратов с положительными коэффициентами $b_1, b_2, \ldots, b_{m-1}$ можно в повернутый относительно новых осей гиперкуб вписать новый гиперкуб R, ограниченный гиперплоскостями $z_i' = \delta$ $(\delta < \delta_1,\ i = 1, 2, \ldots, m-1)$. Интегрирование преобразованной функции плотности вероятности по области, внешней по отношению к этому новому гиперкубу, даст снова оценку вида (60), которую можно лишь усилить, заменив все коэффициенты $b_1, b_2, \ldots, b_{m-1}$ наименьшим среди них коэффициентом, обозначенным через g.

В результате получим новую оценку

$$\beta < \alpha n^{\frac{m-1}{2}} \left(2\int_{\delta}^{\infty} e^{-gnx^2} dx\right)^{m-1}. \qquad (61)$$

Так как

$$\int_{\delta}^{\infty} e^{-gnx^2} dx < \frac{1}{\delta}\int_{\delta}^{\infty} e^{-gnx^2} x\, dx = \frac{1}{\delta}\cdot\frac{1}{2gn} e^{-gn\delta^2},$$

то легко получить окончательную оценку

$$\beta < \frac{\alpha}{(g\delta)^{m-1} n^{\frac{m-1}{2}}} e^{-gn\delta^2}. \qquad (62)$$

Обозначая $\frac{\alpha}{(g\delta)^{m-1}}$ буквой a, а $g\delta^2$ буквой b, получим требуемую

в условии оценку, правда, пока для всех n, начиная с некоторого, возможно достаточно большого, значения. Можно, однако, учесть в выведенной оценке также и остающееся конечное множество M значений n, увеличивая, в случае необходимости, константу a. Поскольку вероятность ϱ заведомо больше нуля, достаточно выбрать величину a столь большой, чтобы правая часть оценки, о которой идет речь, стала отрицательной для всех значений n, принадлежащих множеству M.

Тем самым теорема 4 полностью доказана.

В заключение настоящего параграфа опишем еще одно часто встречающееся распределение—так называемое *пуассоново распределение*. Это распределение можно трактовать как приближение для биномиального распределения при условии, что число испытаний n велико, вероятность успеха ϱ в каждом испытании мала, а произведение $\lambda = np$ не мало, но и не велико. В таком случае вероятность α того, что точно k испытаний приведут к успеху, выражается приближенной формулой

$$\alpha \approx e^{-\lambda}\frac{\lambda^k}{k!}. \qquad (63)$$

В частности, для $k = 0$ $\alpha \approx e^{-\lambda}$.

Распределение Пуассона имеет максимум вероятности при максимальном значении k, удовлетворяющем неравенству $k \leqslant \lambda$. В теории дискретных самоорганизующихся систем с пуассоновым распределением приходится встречаться при организации обучения автоматов словами или последовательностями слов различной длины. При случайном выборе слов, на которых производится обучение, часто достаточно хорошее приближение для закона распределения длин слов дает именно пуассоново распределение.

Заметим, что среднее значение величины, распределенной по закону Пуассона (63), равняется λ.

§ 3. Количественная мера самоорганизации и самосовершенствования в автоматах

В первом параграфе мы познакомились с понятием *самоизменения* в автоматах: автомат естественно называть самоизменяющимся, если он меняет с течением времени ответы на задаваемые ему вопросы (применительно к какому-нибудь циклированию входной и выходной информации). Однако не всякое самоизменение следует отождествлять с *самоорганизацией*. Опираясь на интуитивное представление о самоорганизации, мы должны называть самоорганизующимся такой автомат, который улучшает организацию своих ответов при улучшении организации возможных его историй обучения. Для количественной характеристики указанных улучшений естественно воспользоваться таким теоретико-вероятностным понятием, как понятие *энтропии*.

Будем использовать понятие энтропии лишь для дискретных случайных величин. Пусть задана дискретная случайная величи-

на x с областью определения R и с законом распределения $f(x)$. В таком случае *энтропией этой величины*, или, что то же самое, *энтропией распределения* $f(x)$, будем называть распространенную на область определения R данной случайной величины отрицательную сумму произведений вероятностей $f(x)$ на их логарифмы:

$$H = -\sum_{P} f(x) \log f(x). \tag{64}$$

При пользовании этой формулой принимается, что для $f(x) = 0$ произведение $f(x) \log f(x)$ равно нулю. В качестве основания системы логарифмов может быть выбрано любое вещественное число, строго большее единицы. Легко видеть, что при изменении основания системы логарифмов величины энтропий для всех законов распределения умножаются на один и тот же постоянный множитель. Обычно на практике пользуются либо двоичными логарифмами (с основанием, равным двум), либо натуральными, либо, наконец, десятичными.

Как показывается в теории информации (см., например, С. Голдман [32]), энтропия представляет собой естественную меру неопределенности значений случайной величины: чем эта неопределенность больше, тем больше величина энтропии. В частности, если случайная величина может принимать всего лишь два значения с вероятностями p и $q = 1 - p$ соответственно, то наибольшее значение энтропии достигается при равенстве этих вероятностей: $p = q = \frac{1}{2}$, что соответствует интуитивному понятию о максимально возможной в этом случае неопределенности. Если же одна из вероятностей p или q обращается в нуль, то, как легко видеть, обращается в нуль и значение энтропии, что опять хорошо согласуется со здравым смыслом, поскольку в этом случае неопределенности фактически нет.

При объединении двух независимых случайных величин x и y в одну многомерную (с размерностью, равной сумме размерностей величин x и y) случайную величину $z = (x, y)$ энтропия распределения величины z равняется сумме энтропий распределений величин x и y.

Действительно, если законы распределений величин x и y задаются функциями $f_1(x)$ и $f_2(x)$, то закон распределения величины z задается, очевидно, произведением этих функций $f_1(x)f_2(y)$. Энтропия величины z (H_z) вычисляется тогда по формуле

$$H_z = -\sum_{x \in P_1} \sum_{y \in P_2} f_1(x) f_2(y) \log [f_1(x) f_2(y)] =$$

$$= -\sum_{y \in P_2} f_2(y) \sum_{x \in P_1} f_1(x) \log f_1(x) - \sum_{x \in P_1} f_1(x) \sum_{y \in P_2} f_2(y) \log f_2(y) = H_x + H_y,$$

где P_1 и P_2 области определения величин x и y, а H_x и H_y — их энтропии.

Свойство энтропий независимых распределений складываться при объединении этих распределений в одной называется *свойством аддитивности* энтропии.

Применительно к автоматам, работающим по простому циклу вопрос — ответ (без оценки качества ответа), к определению количества самоорганизации можно подойти с помощью рассмотрения двух энтропийных характеристик — *энтропии обучения* и *энтропии экзамена* автомата. В дальнейшем изложении будем следовать в основном работе [25].

Пусть $(p_1, p_2, \ldots, p_n) = P$ — последовательность вопросов (входных слов), заданных автомату в период его обучения. Эту последовательность будем называть *обучающей последовательностью*. Предположим, что в той или иной фиксированной серии экспериментов с автоматом для каждой обучающей последовательности P задана вероятность $\varrho(P)$ появления этой последовательности в экспериментах рассматриваемой серии (предполагается, что в пределах данной серии эта вероятность не меняется от эксперимента к эксперименту). Тем самым задается некоторое распределение R вероятностей $\varrho(P)$ обучающих последовательностей. Энтропия этого распределения, которую будем называть *энтропией обучения с* данным законом распределения для обучения, вычисляется по обычной формуле

$$H^R(\text{обуч}) = -\sum_P \varrho(P) \log \varrho(P). \qquad (65)$$

Для определенности условимся при подсчете энтропий пользоваться натуральными логарифмами.

В случае, когда фиксированы автомат A и его начальное состояние a_0, всякое распределение вероятностей $\varrho(P)$ обучающих последовательностей однозначно определяет некоторое распределение вероятностей $\alpha(a)$ на множестве всех состояний этого автомата. Здесь $\alpha(a)$ обозначена вероятность того, что после окончания процесса обучения автомата он окажется в состоянии a. Если через S_a обозначить событие на входе автомата, представляемое состоянием a (множество входных слов, переводящих автомат из начального состояния в состояние a), то получим

$$\alpha(a) = \sum_{P \in S_a} \varrho(P), \qquad (66)$$

где суммирование распространено на все слова вида $P = p_1 p_2 \ldots p_n$, содержащиеся в S_a (для краткости записи последовательность слов $P = (p_1, p_2, \ldots, p_n)$ отождествлена здесь со словом $p_1 p_2 \ldots p_n$, составленным из элементов этой последовательности).

Фиксируем теперь некоторое распределение вероятностей $\gamma(p)$ вопросов p, задаваемых автомату после окончания процесса его обучения. Распределения $\alpha(a)$ и $\gamma(p)$ вместе с функциями переходов и выходов рассматриваемого автомата A однозначно определяют

распределение вероятностей $\beta(p, q)$ для пар «вопрос (p) — ответ (q)». Энтропию последнего распределения назовем *энтропией экзамена* и обозначим H^Q(экз). Через Q обозначим так называемый *закон экспериментирования* с автоматом, представляющий собой объединение двух законов распределения — распределения обучающих последовательностей и распределения вопросов на экзамене.

Величина H^Q (экз) определяется по формуле

$$H^Q(\text{экз}) = -\sum \beta(p, q) \log \beta(p, q). \qquad (67)$$

Используя в случае необходимости операцию циклического приведения автоматов, можно, не нарушая общности, рассматривать впоследствии лишь однобуквенные вопросы и ответы. При этом обучающие последовательности P превратятся в слова, составленные из отдельных составляющих их букв-вопросов, расположенных в том порядке, в котором они задавались автомату в процессе обучения.

Для дальнейшего построения теории необходимо задаться некоторым *классом* законов экспериментирования с автоматом и присвоить каждому входящему в этот класс закону Q некоторую вероятность или, в случае непрерывных законов распределения, некоторую плотность вероятности $\varphi(Q)$.

Простейшим случаем является *схема независимых испытаний*, когда на каждом шаге, как в режиме обучения, так и в режиме экзамена, вероятность $\gamma(P)$ появления любого данного вопроса постоянна и зависит только от этого вопроса. Ввиду ограничения лишь 1-циклированными автоматами, задание закона Q экспериментирования с автоматом эквивалентно в этом случае присвоению некоторых вероятностей $\nu_i = \nu(x_i)$ появления на входе автомата каждой из букв x_i его входного алфавита. Сумма всех ν_i, разумеется, должна равняться при этом единице.

В случае схемы иезависимых испытаний закон экспериментирования с автоматом естественно отождествлять с вектором $\nu = (\nu_1, \nu_2, \ldots)$, составленным из вероятностей появления различных входных букв (входной алфавит предполагается каким-либо образом упорядоченным). Класс законов наиболее естественно отождествлять с множеством всех векторов $\nu = (\nu_1, \nu_2, \ldots)$, удовлетворяющих естественным ограничениям $0 \leqslant \nu_i \leqslant 1$ и $\Sigma\nu_i = 1$ ($i = 1, 2, \ldots$), с заданным на этом множестве равномерным законом распределения. Схему независимых испытаний с указанным выбором класса законов распределения условимся называть *равномерной схемой независимых испытаний*. При этом ограничимся случаем, когда длина обучающей последовательности фиксирована, либо, в случае необходимости, будем предполагать, что эти длины описываются некоторым законом распределения (чаще всего пуассоновым).

Если дан какой-либо закон экспериментирования Q, то он, как отмечалось выше, включает в себя два закона распределения —

закон распределения обучающих последовательностей и закон распределения вопросов на экзамене. Соответствующие им случайные величины при обычно принятой организации экспериментов по самосовершенствованию автоматов следует считать независимыми. Поэтому энтропия совместного распределения этих двух величин, которую условимся называть *энтропией соответствующего закона экспериментирования* Q и обозначать H^Q, будет равняться сумме двух энтропий — энтропии обучения H^Q (обуч) и энтропии вопросов на экзамене H^Q (вопр). Последнюю энтропию не следует смешивать с энтропией экзамена H^Q (экз), которая относится не к распределению вопросов на экзамене, а к распределению пар вопрос — ответ. Энтропия экзамена зависит не только от распределения вопросов и распределения обучающих последовательностей, но и от самого автомата, в то время как энтропия закона экспериментирования с автоматом от автомата не зависит.

Предположим, что в классе K законов экспериментирования с автоматом зафиксирован какой-нибудь закон Q_0, имеющий максимальную возможную энтропию H^{Q_0}. Вводя приращения энтропий эксперимента и экзамена по формулам

$$\Delta H^Q = H^Q - H^{Q_0}; \quad \Delta H^Q(\text{экз}) = H^Q(\text{экз}) - H^{Q_0}(\text{экз}), \tag{68}$$

получим возможность для любого автомата A и класса K законов экспериментирования с автоматом (с плотностью вероятности $\varphi(Q)$) ввести две усредненные характеристики

$$s(A, K) = -\int_K \Delta H^Q(\text{экз})\,\varphi(Q)\,dQ; \tag{69}$$

$$z(A, K) = \int_K \frac{\Delta H^Q(\text{экз})}{\Delta H^Q}\,\varphi(Q)\,dQ. \tag{70}$$

Интегралы в этих формулах берутся по области, состоящей из всех законов рассматриваемого класса K. Чем больше величина этих интегралов, тем больше у рассматриваемого автомата A средняя способность к самоорганизации. Нулевому значению соответствует отсутствие способностей к самоорганизации, а отрицательные значения означают, что при улучшении организации обучения организация ответов автомата в среднем ухудшается. Иначе говоря, автомат ведет себя не как самоорганизующаяся, а как «самодезорганизующаяся» система.

Поскольку формула (70) приводит к значительно более сложным вычислениям, чем формула (69), в качестве основного количественного критерия для оценки способности автомата к самоорганизации выберем критерий $s(A, K)$, а не критерий $z(A, K)$.

Рассмотрим в качестве примера два автомата A и B, функции переходов и выходов которых задаются таблицами:

для автомата A —

	1	2
x	1	1
y	2	2

;

	u	v
x	u	v
y	u	v

;

для автомата B —

	1	2
x	1	2
y	2	2

;

	u	v
x	u	v
y	u	v

.

В этих таблицах цифрами 1 и 2 обозначены состояния автоматов, буквами x, y — входные сигналы (вопросы), а буквами u, v — выходные сигналы (ответы).

В качестве класса K законов распределения выберем такой класс, в котором вероятности появления вопросов x и y на экзамене равны между собой, а законы распределения обучающих последовательностей возникают из схемы независимых испытаний при вероятностях появления вопросов x и y, равных p и $1 - p$ соответственно (p пробегает в пределах класса K все значения от 0 до 1 с равными вероятностями). Кроме того, фиксируем длину n обучающих последовательностей, а соответствующий выбранному значению n критерий $s(A, K)$ обозначим $s_n(A, K)$.

Автомат A будет находиться в состоянии 1, если последний заданный ему при обучении вопрос был x, и в состоянии 2, если последний заданный ему вопрос был y. Отсюда следует, что вероятности пар вопрос — ответ будут равны: для пары (x, u) — $\frac{1}{2}p$, для пары (y,u) — $\frac{1}{2}p$, для пары (x, v) — $\frac{1}{2}(1-p)$ и для пары (y, v) — также $\frac{1}{2}(1-p)$. Следовательно, энтропия экзамена

$$H^Q(\text{экз}) = -\frac{1}{2}p\ln\frac{1}{2}p - \frac{1}{2}p\ln\frac{1}{2}p - \frac{1}{2}(1-p)\ln\frac{1}{2}(1-p) -$$

$$-\frac{1}{2}(1-p)\ln\frac{1}{2}(1-p) = -p\ln p - (1-p)\ln(1-p) - \ln\frac{1}{2}.$$

Максимальной энтропия экспериментирования будет, очевидно, при $p = 1 - p = \frac{1}{2}$. Энтропия экзамена в этом случае определится выражением

$$H^{Q_0}(\text{экз}) = -\frac{1}{2}\ln\frac{1}{2} - \frac{1}{2}\ln\frac{1}{2} - \ln\frac{1}{2} = -2\ln\frac{1}{2}.$$

Приращение энтропии экзамена

$$\Delta H^Q(\text{экз}) = -p\ln p - (1-p)\ln(1-p) + \ln\frac{1}{2}.$$

Плотность вероятности законов Q в выбранном классе K равна, очевидно, единице. Применение формулы (69) дает

$$s_n(A, K) = \int_0^1 \left(p \ln p + (1-p) \ln (1-p) - \ln \frac{1}{2} \right) dp = \ln 2 - \frac{1}{2} \approx 0{,}19.$$

Автомат B будет находиться в состоянии 1 только тогда, когда обучающая последовательность состоит из одних лишь x. Вероятность этого равна, очевидно, p^n. Отсюда вероятности экзаменационных пар (x, u) и (y, u) равны $\frac{1}{2} p^n$, а вероятности пар (x, v) и (y, v) равны $\frac{1}{2}(1 - p^n)$. Как и в случае нахождения $s_n(A, K)$, найдем ΔH^Q (экз), в результате чего получим цепочку формул

$$s_n(B, K) = \int_0^1 \left[p^n \ln \frac{1}{2} p^n + (1 - p^n) \ln \frac{1}{2} (1 - p^n) - \frac{1}{2^n} \ln \frac{1}{2^{n+1}} - \right.$$

$$\left. - \left(1 - \frac{1}{2^n}\right) \ln \frac{1}{2} \left(1 - \frac{1}{2^n}\right) \right] dp = \frac{n \ln 2}{2^n} - \frac{n}{(n+1)^2} - \left(1 - \frac{1}{2^n}\right) \times$$

$$\times \ln \left(1 - \frac{1}{2^n}\right) + \int_0^1 (1 - p^n) \ln (1 - p^n)\, dp = \frac{n \ln 2 + 1}{2^n} - \frac{n}{(n+1)^2} -$$

$$- \frac{1}{n} \left[\frac{1}{1 \left(1 + \frac{1}{n}\right) \left(2 + \frac{1}{n}\right)} + \frac{1}{2 \left(2 + \frac{1}{n}\right) \left(3 + \frac{1}{n}\right)} + \dots \right] -$$

$$- \left(\frac{1}{1 \cdot 2 \cdot 2^{2n}} + \frac{1}{2 \cdot 3 \cdot 2^{3n}} + \dots \right).$$

Используя последнее соотношение, получим оценку

$$s_n(B, K) < \frac{n \ln 2 + 1}{2^n} - \frac{n}{(n+1)^2} - \frac{1}{4n}.$$

Из этой оценки легко узнаем, что при $n \geqslant 5$ величина $s_n(B, K)$ отрицательна. Иными словами, при обучении последовательностями длины, большей 4, в выбранном классе законов экспериментирования автомат B является в среднем «самодезорганизующимся», в то время как автомат A в тех же условиях обнаруживает способность к самоорганизации.

Заметим, что вывод о наличии или отсутствии у автомата способности к самоорганизации зависит от выбора класса законов экспериментирования с этим автоматом. Если, например, в разобран-

ном примере в качестве K выбрать класс законов экспериментирования, возникающий из равномерной схемы независимых испытаний, то, как нетрудно проверить, автомат B также стал бы в среднем самоорганизующимся, хотя величина этой самоорганизации оставалась бы меньшей, чем у автомата A.

При переходе от понятия самоорганизации к понятию самообучения уже нельзя довольствоваться чисто теоретико-вероятностными понятиями. Необходимо вводить понятия, которые характеризовали бы ту или иную направленность процесса самоорганизации. С этой целью наиболее естественно ввести определенную на множестве всевозможных пар вопрос (p) — ответ (q) вещественную функцию $f(p, q)$, величина которой характеризует качество любого ответа на любой данный вопрос p.

Как отмечалось выше, для любого данного автомата A с фиксированным начальным состоянием a_0 задание закона Q распределения вероятностей $\varrho(P)$ на обучающих последовательностях P однозначно определяет распределение вероятностей $\alpha(a)$ на множестве состояний автомата. Обозначим еще через $q = \lambda(a, p)$ ответ автомата A, приведенного предварительно в состояние a, на вопрос p. Величина $f^Q = \sum_{p,a} f(p, \lambda(a, p)) \gamma(p) \alpha(a)$ представляет собой усредненный критерий качества ответов автомата на «экзамене» при обучении его последовательностями, распределенными по закону Q ($\gamma(p)$ — вероятность появления вопроса p на экзамене).

Количеством самообучения автомата A естественно назвать теперь разность $f^Q - f^{Q_0}$, где Q_0 — априорный закон распределения вероятностей обучающих последовательностей, известный конструктору в момент создания автомата, а Q — апостериорный закон распределения, который фактически имел место для некоторого класса экспериментов по обучению. Как правило, энтропия распределения Q_0 больше энтропии распределения Q.

Если теперь задан класс K апостериорных законов распределения Q с плотностью вероятности $\varphi(Q)$, то интеграл $b(A, K) = \int_{Q \in K} (f^Q - f^{Q_0}) \varphi(Q)\, dQ$ представит собой усредненную количественную характеристику для способности рассматриваемого автомата к самообучению (применительно к выбранному классу K, автомату A и вещественной функции f).

§ 4. Автоматы со случайными переходами

Помимо *детерминированных* автоматов в теории самоорганизующихся систем приходится рассматривать автоматы, имеющие *случайные переходы*. Как известно (см. гл. III) в детерминированном автомате задание предыдущего состояния $a(t-1)$ и текущего входного сигнала $x(t)$ однозначно определяет следующее очередное состояние $a(t)$, в которое переходит автомат под влиянием этого входного сигнала из состояния $a(t-1)$. В автомате со случайными переходами задание пары $a(t-1)$, $x(t)$ определяет лишь вероят-

ности $p_{ij}(x)$ перехода автомата из состояния $a(t-1)$, которое обозначим a_i, в любое другое состояние a_j под влиянием входного сигнала $x(t) = x$.

Легко видеть, что детерминированный автомат можно рассматривать как частный случай автомата со случайными переходами, у которого для каждого x при любом данном i лишь точно одна из вероятностей $p_{ij}(x)$ равна единице, а все остальные вероятности равны нулю.

Всякий автомат со случайными переходами естественно задавать с помощью системы матриц $\|p_{ij}(x)\|$, где x пробегает последовательно все входные сигналы автомата. Разумеется, кроме подобной системы матриц, должны быть заданы также функция выходов и начальное состояние автомата.

Матрицы $\|p_{ij}(x)\|$ обладают тем свойством, что сумма элементов любой из их строк равняется единице. Будем предполагать также, что в автомате нет состояний, вероятности перехода в которые из всех других состояний были бы равными нулю. Это означает, очевидно, что матрицы $\|p_{ij}(x)\|$ не имеют столбцов, составленных из одних лишь нулей. Кроме того, все элементы каждой из матриц $\|p_{ij}(x)\|$ представляют собой неотрицательные вещественные числа, не превосходящие единицу.

Матрицы, удовлетворяющие трем перечисленным свойствам, принято называть *стохастическими матрицами*. Таким образом, в случае автоматов со случайными переходами роль функции переходов выполняет функция $\|p_{ij}(x)\|$, однозначно отображающая множество всех входных сигналов автомата в множество стохастических матриц.

Особый интерес представляют автоматы со случайными переходами, имеющими один-единственный (постоянный) входной сигнал. Такие автоматы изучались в классической теории вероятностей под названием *однородных (дискретных) цепей Маркова*. Выходные сигналы в подобных автоматах игнорируются (или отождествляются с состояниями), что позволяет задавать эти автоматы с помощью единственной стохастической матрицы. Для определенности считается обычно, что первая строка (как и первый столбец) такой матрицы соответствует начальному состоянию автомата (марковской цепи).

Цепи Маркова имеют и другую (не автоматную) интерпретацию — в терминах, обычно применяющихся в теории вероятностей. Эта интерпретация основана на понятии испытаний, рассматривавшихся в предыдущем параграфе. Однако речь должна идти теперь не о независимых испытаниях, а о таких испытаниях, при которых вероятности тех или иных исходов каждого очередного испытания зависят от исхода *непосредственно предшествующего ему испытания* и не зависят непосредственно от исходов всех остальных испытаний (само множество возможных исходов не меняется от испытания к испытанию). Таким образом, возникает матрица $\|p_{ij}\|$ так называемых *переходных вероятностей*. Произвольный

элемент p_{ij} этой матрицы представляет собой вероятность j-го исхода в каждом очередном испытании при условии, что исход непосредственно предшествующего ему испытания был i.

Легко видеть, что подобная трактовка вполне эквивалентна автоматной трактовке: исход испытания представляет собой, по существу, просто другое название для состояния автомата (имеющего единственный входной сигнал) со случайными переходами. Имеется, правда, и одно отличие: в автомате со случайными переходами фиксировалось вполне определенное начальное состояние, в цепях Маркова принято задавать вероятности различных исходов начального испытания $p_1, p_2, \ldots, p_n$, что соответствует случайному выбору начального состояния автомата, так что i-е состояние может быть выбрано в качестве начального состояния с вероятностью p_i ($i = 1, 2, \ldots, n$).

Таким образом, для более полной аналогии с марковскими цепями необходимо рассматривать не просто автоматы со случайными переходами (имеющими единственный входной сигнал), а так называемые *случайные автоматы*, у которых случайна не только функция переходов, но и выбор начального состояния, а если к тому же рассмотрению привлекаются выходные сигналы, то случайной должна быть, вообще говоря, и функция выходов. Иначе говоря, функция выходов должна задавать не просто выходной сигнал, а некоторое распределение вероятностей на множестве всех возможных выходных сигналов.

Цепи Маркова (или, соответственно, случайные автоматы) называются *конечными* или *бесконечными* в зависимости от того, конечно или бесконечно возможное множество исходов (или, соответственно, множество состояний автомата). Будем требовать для случайных конечных автоматов общего вида также конечности множеств его входных и выходных сигналов. Ограничимся изучением лишь однородных цепей Маркова, т. е. таких цепей, у которых матрица переходных вероятностей постоянна. *Неоднородные марковские цепи* (с зависящей от времени матрицей переходных вероятностей) в дальнейшем нам встречаться не будут. Поэтому для краткости будем говорить просто о цепях Маркова, понимая всякий раз, если не оговорено противное, что речь идет об однородных цепях.

Рассмотрим автомат A со случайными переходами (марковскую цепь), матрица переходных вероятностей которого $P = \| p_{ij} \|$. Как отмечалось выше, произвольный элемент p_{ij} этой матрицы представляет собой вероятность перехода автомата A из i-го состояния в j-е. Важно подчеркнуть, что здесь речь идет о переходе *за один такт* (т. е. за промежуток между двумя соседними моментами дискретного автоматного времени). Легко понять, что произведение $p_{ik}p_{kj}$ представляет собой вероятность перехода автомата A из i-го состояния в j-е за два такта при условии, что автомат пройдет через k-е состояние.

Сумма $\sum_k p_{ik}\, p_{kj}$, распространенная на все состояния, даст, оче-

видно, полную вероятность перехода автомата A из i-го состояния в j-е за два такта. Вместе с тем эта сумма представляет собой, очевидно, элемент матрицы $P \cdot P = P^2$, стоящий на пересечении i-ой строки и j-го столбца. Аналогично получим: вероятность перехода автомата A из i-го состояния в j-е за три такта его работы равна (i, j)-му элементу матрицы $P^2 \cdot P = P^3$. Продолжая подобным образом, придем к следующему предложению.

Теорема 1. *Для случайного автомата A с одним входным сигналом (однородной марковской цепи), матрица переходных вероятностей которого — P, вероятность перехода из i-го состояния в j-е точно за n тактов равна элементу матрицы P^n, стоящему на пересечении i-ой строки и j-го столбца ($n = 1, 2, 3, \ldots$).*

Элементы матрицы P^n естественно называть *переходными вероятностями за n шагов*. Для определения этих вероятностей в случае конечных цепей Маркова пользуются обычно так называемой *формулой Перрона*, которая выводится в теории матриц. Напомним сначала некоторые определения и понятия этой теории.

Пусть дана матрица $P = \|p_{ij}\|$ n-го порядка. Определитель

$$P(\lambda) = \begin{vmatrix} \lambda - p_{11} & -p_{12} & \ldots & -p_{1n} \\ -p_{21} & \lambda - p_{22} & \ldots & -p_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ -p_{n1} & -p_{n2} & \ldots & \lambda - p_{nn} \end{vmatrix}$$

представляет собой полином n-ой степени от λ, называемый *характеристическим полиномом матрицы P*. Корни этого полинома называются *характеристическими числами* матрицы P.

Обозначим через E единичную матрицу n-го порядка. Тогда элемент матрицы $(\lambda E - P)^{-1}$, стоящий на пересечении i-ой строки и j-го столбца, будет равен $\frac{1}{P(\lambda)} \cdot P_{ji}(\lambda)$, где $P_{ji}(\lambda)$ — алгебраическое дополнение элемента определителя $P(\lambda)$, стоящего на пересечении его j-ой строки и i-го столбца.

Пусть теперь матрица $P = \|p_{ij}\|$ имеет характеристические числа $\lambda_1, \lambda_2, \ldots, \lambda_r$. Обозначим через m_i кратность i-го числа λ_i, т. е., иными словами, максимальное число s такое, что характеристический полином $P(\lambda)$ делится на $(\lambda - \lambda_i)^s$, и определим полином $\psi_i(\lambda)$ по формуле

$$\psi_i(\lambda) = \frac{P(\lambda)}{(\lambda - \lambda_i)^{m_i}}.$$

Тогда элемент $p_{ij}^{(k)}$ матрицы $P^{(k)}$, стоящий на пересечении i-ой строки и j-го столбца, можно определить по формуле

$$p_{ij}^{(k)} = \sum_{\nu=1}^{r} \frac{1}{(m_\nu - 1)!} D_\lambda^{m_\nu - 1} \left[\frac{\lambda^k P_{ji}(\lambda)}{\psi_\nu(\lambda)} \right] \Bigg|_{\lambda = \lambda_\nu}. \qquad (71)$$

Формула (71) является формулой Перрона. В ней $D_\lambda^{m_\nu - 1}$ обозначает производную по λ порядка $m_\nu - 1$. Подстановка значения $\lambda = \lambda_\nu$

должна быть выполнена после дифференцирования. Вывод формулы Перрона можно найти в любой монографии по теории конечных цепей Маркова (см., например, В. И. Романовский [68]).

Особенно простой вид формула Перрона приобретает в том случае, когда все характеристические числа матрицы P имеют кратность, равную единице, т. е. когда $m_1 = m_2 = \ldots = m_r = 1$. Ясно, что в этом случае $r = n$. Поскольку факториал нуля равен единице, а производная нулевого порядка означает отсутствие какого-либо дифференцирования, то для указанного частного случая получим простую формулу

$$p_{ij}^{(k)} = \sum_{\nu=1}^{n} \frac{\lambda_\nu^k P_{ji}(\lambda_\nu)}{\psi_\nu(\lambda_\nu)} \quad (i, j = 1, 2, \ldots, n). \tag{72}$$

Условимся называть формулу (71) общей, а формулу (72) — *специальной формулой Перрона.* Формулы (71) и (72) позволяют решить одну весьма важную задачу теории конечных марковских цепей — задачу нахождения так называемого *предельного распределения.* Если существует предел $\lim\limits_{k\to\infty} P^k = P^\infty$, то элементы матрицы $P^\infty = \| p_{ij}^\infty \|$ естественно называть *предельными переходными вероятностями.* Имея начальное распределение, т. е. вероятности $p_1, p_2, \ldots, p_n$ различных исходов начального испытания, можно получить вероятности p_i^∞ различных состояний в предельном распределении по формулам

$$p_i^\infty = \sum_{j=1}^{n} p_j p_{ji}^\infty \quad (i = 1, 2, \ldots, n). \tag{73}$$

Кажется естественным предположить, что после достаточно большого числа переходов случайного автомата, характеризующего марковскую цепь, влияние начального распределения вероятностей состояний на распределение состояний, полученное в результате этих переходов, может быть сделано сколь угодно малым. Иными словами, предельное распределение, получаемое по формулам (73) не должно зависеть от начального распределения (p_1, p_2, ..., p_n). Если предельное распределение обладает таким свойством, то соответствующая марковская цепь называется *эргодической.* Свойство эргодичности будет, очевидно, иметь место в том и только в том случае, когда для любого данного i все элементы p_{ji}^∞ ($j = 1, 2, \ldots, n$) одинаковы, т. е., иными словами, когда все строки матрицы предельных переходных вероятностей одинаковы.

Можно показать, что характеристические числа стохастических матриц не могут превосходить по модулю единицу. Нетрудно видеть также, что единица является характеристическим числом для любой стохастической матрицы. Если все остальные (отличные от единицы) характеристические числа стохастической матрицы M по модулю строго меньше единицы, то матрица M и соответствующая ей марковская цепь называются *правильными.* Если у правильной

стохастической матрицы P единица является простым корнем характеристического полинома, то матрица P и соответствующая ей марковская цепь называются *регулярными*.

Справедливо следующее предложение [14].

Теорема 2. *Марковская цепь C с конечным числом состояний тогда и только тогда имеет предельное распределение, когда она правильна. Для того чтобы цепь C удовлетворяла свойству эргодичности, необходимо и достаточно, чтобы она была регулярной цепью.*

Для случая регулярной конечной марковской цепи предельные переходные вероятности определяются формулами

$$p_{ij}^{\infty} = \frac{P_{ji}(\lambda)}{\psi_1(\lambda)} \quad (i, j = 1, 2, \ldots, n). \tag{74}$$

Эти формулы получаются из специальной формулы Перрона (72) в результате предельного перехода при $k \to \infty$.

Описанные выше результаты позволяют построить теорию поведения автоматов (случайных и детерминированных) в случайных средах. Ограничимся рассмотрением одних лишь автоматов Мура, поскольку в случае автоматов Мили возникает необходимость в некоторых усложнениях теории, делающих ее менее прозрачной. Условимся также рассматривать детерминированные автоматы как частный случай случайных автоматов, что, как уже отмечалось выше, всегда возможно.

При указанных предположениях всякий автомат A можно задавать матрицей $L = \| \lambda_{ij} \|$ *выходных вероятностей* и семейством матриц $D^{(m)} = \| \delta_{ik}^{(m)} \|$ *переходных вероятностей*. Произвольный элемент λ_{ij} первой матрицы равняется вероятности появления j-го выходного сигнала в случае, когда автомат A находится в i-ом состоянии. Величина $\delta_{ik}^{(m)}$ есть не что иное, как вероятность перехода автомата из i-го состояния в k-ое под влиянием m-го входного сигнала.

Среда задается для некоторого класса автоматов, имеющих одинаковые множества входных сигналов $(x_1, x_2, \ldots, x_n)$ и одинаковые множества выходных сигналов $(v_1, v_2, \ldots, v_s)$. Задать среду для рассматриваемого класса K означает задать зависимость входного сигнала $x(t)$ любого автомата A из класса K в произвольный момент дискретного автоматного времени t от значения $v(t-1)$ его выходного сигнала в момент времени, непосредственно предшествующий рассматриваемому моменту времени t. Предполагается, что эта зависимость одна и та же для всех автоматов из данного класса K. Иными словами, поведение среды определяется только действиями (выходными сигналами) автоматов и не зависит непосредственно от внутреннего устройства автоматов.

Рассмотрим *случайные среды*, в которых определены так называемые *вероятности реакций* r_{jm}, объединяемые в (прямоугольную) *матрицу вероятностей реакций* $R = \| r_{jm} \|$. Величина r_{jm} при-

нимается равной вероятности возникновения на входе действующего в рассматриваемой среде автомата A (из класса K) m-го входного сигнала, если в непосредственно предшествующий момент времени автоматом A был выдан j-й выходной сигнал.

Если вероятности реакций среды постоянны, то соответствующая среда называется *стационарной случайной средой*. В *нестационарных* случайных средах вероятности реакций могут изменяться с течением времени. Как и в случае автоматов, детерминированные среды (со строго определенной функциональной зависимостью $x(t) = f(v(t-1))$ могут рассматриваться как частный случай случайных сред.

Легко видеть, что изучение поведения автоматов Мура (как детерминированных, так и случайных) в стационарных случайных средах сводится к однородным цепям Маркова, состояния которых можно отождествлять с состояниями рассматриваемых автоматов. Действительно, состояние $a(t)$ автомата A в любой момент времени однозначно определяет вероятности выходных сигналов $v(t)$, а следовательно, в силу определения стационарной случайной среды, и вероятности входных сигналов автомата в непосредственно следующий момент времени $t+1$. Последние вероятности однозначно определяют вероятности переходов автомата A из состояния $a(t)$ в любое из следующих состояний.

При принятых выше обозначениях переходные вероятности p_{ik} соответствующей марковской цепи определятся формулами

$$p_{ik} = \sum_j \sum_m \lambda_{ij} r_{jm} \delta_{ik}^{(m)}. \tag{75}$$

Опишем, следуя М. Л. Цетлину [82], ряд простейших задач о поведении автоматов в случайных средах. С этой целью рассмотрим класс K детерминированных автоматов Мура, имеющих два входных сигнала $x_0 = 0$ и $x_1 = 1$ и два выходных сигнала $v_0 = 0$ и $v_1 = 1$. Стационарную случайную среду C зададим матрицей R вероятностей реакций

$$R = \left\| \begin{matrix} 1-p_0 & p_0 \\ 1-p_1 & p_1 \end{matrix} \right\|.$$

Условимся входной сигнал x_1 называть *штрафом*, а входной сигнал x_0 — *нештрафом*. Тогда можно сказать, что при выходном сигнале v_0 среда штрафует автомат с вероятностью p_0, а при выходном сигнале v_1 — с вероятностью p_1.

Рассмотрим сначала автомат Мура A с двумя состояниями $a_1 = 1$ и $a_2 = 2$, заданный матрицей выходных вероятностей $L = \left\| \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} \right\|$ и матрицами переходных вероятностей $D^{(0)} = \left\| \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} \right\|$, $D^{(1)} = \left\| \begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix} \right\|$. Иначе говоря, A является детерминированным автоматом, выдающим в первом состоянии выходной сигнал $v_0 = 0$, во втором состоянии — выходной сигнал $v_1 = 1$, сохраняющий свое состояние под влиянием входного сигнала $x_0 = 0$ и изменяющий состояние на противоположное под влиянием входного сигнала $x_1 = 1$.

В соответствии со сказанным выше, функционирование автомата A в среде C описывается однородной марковской цепью M с двумя состояниями $a_1 = 1$ и $a_2 = 2$. По формулам (75) легко найдем матрицу P переходных вероятностей этой цепи

$$P = \begin{Vmatrix} 1 - p_0 & -p_0 \\ -p_1 & 1 - p_1 \end{Vmatrix}.$$

Характеристический полином $P(\lambda) = \begin{vmatrix} \lambda - 1 + p_0 & -p_0 \\ -p_1 & \lambda - 1 + p_1 \end{vmatrix}$ матрицы равен $\lambda^2 - \lambda(2 - p_0 - p_1) + 1 - p_0 - p_1$, а ее характеристическими числами являются соответственно $\lambda_1 = 1$ и $\lambda_2 = 1 - p_0 - p_1$. Если обе вероятности p_0 и p_1 не равны одновременно нулю или единице, то второе характеристическое число λ_2 по модулю меньше единицы, и, по теореме 2, цепь M будет в этом случае эргодической.

Полином $\psi_1(\lambda)$ будет, очевидно, равен $\lambda - 1 + p_0 + p_1$, и после применения формул (74) легко найдем предельные переходные вероятности рассматриваемой цепи

$$p_{11}^{\infty} = p_{21}^{\infty} = \frac{P_{11}^{(1)}}{\psi_1(1)} = \frac{p_1}{p_0 + p_1} = \frac{p_1}{p_0 + p_1}$$

и

$$p_{12}^{\infty} = p_{22}^{\infty} = \frac{1 - 1 + p_0}{p_0 + p_1} = \frac{p_0}{p_0 + p_1}.$$

Таким образом, при достаточно долгом функционировании в среде C автомат A, независимо от выбора начального состояния, будет находиться с вероятностью $\frac{p_1}{p_0 + p_1}$ в первом состоянии, а с вероятностью $\frac{p_0}{p_0 + p_1}$ — во втором состоянии. Так как вероятность штрафа в первом состоянии автомата равна p_0, а во втором состоянии — p_1, то математическое ожидание S штрафа автомата A на каждом шаге (после достаточно длительного предварительного функционирования) выразится формулой

$$S = p_0 \frac{p_1}{p_0 + p_1} + p_1 \frac{p_0}{p_0 + p_1} = \frac{2 p_0 p_1}{p_0 + p_1}.$$

При $p_0 \neq p_1$ величина S строго меньше средней вероятности штрафа $S_0 = \frac{1}{2}(p_0 + p_1)$. Действительно,

$$S_0 - S = \frac{(p_0 + p_1)^2 - 4 p_0 p_1}{2(p_0 + p_1)} = \frac{(p_0 - p_1)^2}{2(p_0 + p_1)} \geqslant 0,$$

причем равенство достигается, очевидно, лишь тогда, когда $p_0 = p_1$. Таким образом, рассмотренный автомат A обладает *целесообразным поведением* в том смысле, что при помещении в любую стационарную случайную среду, различающую две его возможные реакции, он

стремится к такому поведению, при котором величина штрафа оказывается в среднем меньшей, чем у автомата, выдающего с равными вероятностями оба возможных для автомата A выходных сигнала (реакции).

Выберем теперь вместо рассмотренного автомата A автомат A_n с $2n$ состояниями $1, 2, \ldots, n, n+1, \ldots, 2n-1, 2n$. Положим, что в состояниях $1, 2, \ldots, n$ он выдает выходной сигнал $v_0 = 0$, а во всех остальных состояниях—выходной сигнал $v_1 = 1$. Пусть, далее, таблица переходов автомата A_n запишется

x \ a	1	2	3	$\ldots$	$n-1$	n	$n+1$	$n+2$	$n+3$	$\ldots$	$2n-1$	$2n$
0	1	1	2	$\ldots$	$n-2$	$n-1$	$n+1$	$n+1$	$n+2$	$\ldots$	$2n-2$	$2n-1$
1	2	3	4	$\ldots$	n	$2n$	$n+2$	$n+3$	$n+4$	$\ldots$	$2n$	n

Этой таблице соответствует граф переходов, изображенный на рис. 12.

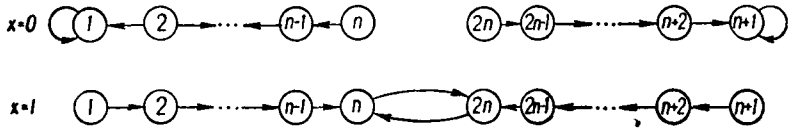

Рис. 12.

Такой автомат по виду его графа естественно называть *автоматом с линейной тактикой*. Разобранный выше автомат A представляет собой, очевидно, частный случай автомата A_n с линейной тактикой, для которого значение n равно единице. Поведение автомата A_n с линейной тактикой в общем случае изучается точно так же, как и в рассмотренном частном случае, хотя, разумеется, соответствующие выкладки значительно усложняются. Эти выкладки приводят к выводу о том, что вычисленное по аналогии с предыдущим случаем значение математического ожидания S_n штрафа автомата A_n за один шаг его работы (после достаточно длительного приспособительного периода) при неограниченном увеличении числа n стремится к естественному минимальному значению $S_{\min}$, равному наименьшему из двух чисел p_0, p_1. Легко понять, что величина $S_{\min}$ является абсолютным минимумом математического ожидания штрафа для всех автоматов, действующих в рассматриваемой случайной среде.

Оказывается, что в ряде случаев аппарат однородных марковских цепей можно с успехом использовать при изучении поведения автоматов не только в стационарных, но и в некоторых нестационарных случайных средах. Пусть, например, имеется несколько стационарных случайных сред $C_1, C_2, \ldots, C_k$, аналогичных описанной выше среде C, но имеющих различные пары вероятностей p_0, p_1. Из этих сред можно построить нестационарную случайную среду N,

вводя матрицу $B = \|b_{ij}\|$ переходных вероятностей некоторой марковской цепи с k состояниями. В любой данный момент t дискретного времени среда N действует как одна из сред $C_1, C_2, \ldots, C_k$. Если образцом для ее действий оказывается среда C_i, то говорим, что среда N находится в i-ом состоянии. Величина b_{ij} представляет собой вероятность перехода среды N из i-го состояния в j-е ($i, j = 1, 2, \ldots, k$). Вероятности b_{ij} предполагаются постоянными и не изменяются с течением времени.

Если теперь в среде N функционирует некоторый автомат A, то пары (C_i, a_j), состоящие из состояния C_i среды N и состояния a_j автомата A, могут быть выбраны в качестве состояний некоторой однородной марковской цепи. Матрица переходных вероятностей этой цепи может быть легко построена по матрицам вероятностей реакций сред $C_1, C_2, \ldots, C_k$, матрице B функции переходов и функции выходов автомата A.

Можно показать [82], что в классе K автоматов A_n с линейной тактикой, действующих в описанной нестационарной среде N, есть (зависящий от выбора среды N) оптимальный автомат A_{n_0}, обладающий минимальным (в классе K) математическим ожиданием предельной величины штрафа на каждом шаге своей работы. Таким образом, в отличие от стационарных сред, для автоматов с линейной тактикой, функционирующих в нестационарных средах, объем памяти (числа состояний) автомата целесообразно увеличивать лишь до известного предела, после чего дальнейшее увеличение памяти приводит не к улучшению, а к ухудшению качества работы автомата.

§ 5. Проблема обучения распознаванию образов

Одной из наиболее значительных областей применения теории самоорганизующихся систем является проблема распознавания зрительных образов. Распознавание зрительных образов и обучение такому распознаванию — яркий пример приспособительных свойств мозга человека. Смысл распознавания образов заключается в том, что человек объединяет некоторые множества наблюдаемых им предметов или явлений в один класс, называемый образом. Образы, которыми оперирует человек,— это не случайные объединения предметов, а такие объединения, которые связаны какими-либо общими свойствами. Рассматривая в основном зрительные образы, будем в дальнейшем называть отдельные составляющие этот образ предметы *изображениями*.

Примерами зрительных образов могут служить множество всех изображений той или иной буквы или цифры, множество изображений всевозможных зданий и т. п. По аналогии со зрительными образами можно рассматривать также звуковые образы (например, множество всех произношений той или иной фонемы, множество всех вальсов и т. п.) и образы любой другой природы. В дальнейшем ограничимся рассмотрением в качестве примеров од-

них лишь зрительных образов, однако все наши теоретические построения будут пригодны не только для зрительных, но и для любых других образов.

Для последующих построений нам прежде всего потребуются определения абстрактных изображений и образов. Будем предполагать, что изображения воспринимаются некоторым множеством чувствительных элементов — *рецепторов*. По аналогии со случаем зрительных образов, воспринимаемых глазом человека, назовем это множество *сетчаткой*. В случае абстрактных изображений и образов нам не потребуется уточнять вопрос о пространственном расположении рецепторов. Однако в случае конкретных зрительных образов такие уточнения полезны. В этом случае будем обычно предполагать, что составляющие сетчатку рецепторы расположены на плоскости в точках, образующих правильную решетку, т. е., иными словами, в точках с координатами $(a + ic,\ b + jc)$, где $c \neq 0$, i пробегает множество значений $0, 1, \ldots, m - 1$, а j — множество значений $0, 1, \ldots, n - 1$. Такую сетчатку в дальнейшем будем называть *правильной прямоугольной* $(n \times m)$*-сетчаткой*.

Задача сетчатки состоит в том, чтобы превратить проектируемое на нее изображение в некоторую совокупность сигналов стандартного вида, выдаваемых составляющими сетчатку рецепторами. Будем различать в дальнейшем два вида рецепторов: так называемые *непрерывные рецепторы*, выходными сигналами которых могут быть произвольные вещественные числа на том или ином фиксированном отрезке числовой оси, и так называемые *дискретные рецепторы*, которые могут выдавать лишь два различных выходных сигнала. Не нарушая общности, можно фиксировать в качестве области значений выходных сигналов непрерывных рецепторов числовой отрезок $[0,1]$, а в качестве возможных значений выходных сигналов дискретных рецепторов — концы этого отрезка, т. е. числа 0 и 1.

В случае зрительных образов будем предполагать всегда, что выходной сигнал непрерывного рецептора равен яркости проектируемой на данный рецептор точки изображения, выраженной в относительных единицах: нуль соответствует абсолютно черным точкам, а единица — абсолютно белым (отражающим 100% падающего на них света) точкам изображения. Для дискретных рецепторов устанавливается некоторый порог яркости. Точкам, яркость которых не превышает величины этого порога, будет соответствовать нулевой выходной сигнал. Более подробно, однако, в случае дискретных рецепторов будем рассматривать лишь двутональные изображения, состоящие либо из точек нулевой яркости (фон), либо из точек единичной яркости (собственно изображение). Примем в дальнейшем именно эту, последнюю, точку зрения.

Абстрактным изображением условимся называть любую фиксированную совокупность выходных сигналов рецепторов, составляющих сетчатку. Если общее число рецепторов в сетчатке равно N, то, в силу принятых выше предложений, абстрактное изо-

бражение может быть естественно отождествлено с некоторой точкой N-мерного единичного куба. В случае непрерывных рецепторов изображениям соответствуют все точки этого куба, а в случае дискретных изображений — только его вершины. В соответствии со сказанным, N-мерный единичный куб (в случае непрерывных рецепторов) или множество его вершин (в случае дискретных рецепторов) будем называть *пространством изображений*. В первом случае это пространство является *непрерывным*, а во втором случае — *дискретным* (состоящим из 2^N различных точек).

В пространстве изображений естественно вводится «*метрика*»: *расстоянием* между двумя точками этого пространства (т. е. между двумя изображениями) называется корень квадратный из суммы квадратов разностей соответствующих координат этих точек

$$d = \sqrt{(x'_1 - x''_1)^2 + (x'_2 - x''_2)^2 + \ldots + (x'_N - x''_N)^2}.$$

R-окрестностью какой-либо точки M пространства изображений будем называть совокупность всех точек этого пространства, удаленных от точки M на расстояние, меньшее или равное R.

Заметим, что введенные определения пригодны как для непрерывного, так и для дискретного пространства изображений. Во втором случае расстояние между любыми двумя точками представляет собой не что иное, как корень квадратный из суммарного числа n несовпадений координат этих точек. Для нас, однако, удобнее в случае дискретных пространств считать расстоянием само это число n. При этом все расстояния будут выражаться целыми числами. После введения расстояния в пространстве изображений можно говорить о *близости* друг к другу тех или иных точек. Из интуитивных представлений, связанных с понятием образа, следует, что изображения, лежащие достаточно близко к тому или иному изображению из некоторого образа, должны принадлежать тому же самому образу. Это обстоятельство должно быть как-то учтено при определении понятия *абстрактного образа*.

В случае непрерывного пространства изображений образом может служить лишь такое множество точек этого пространства, которое вместе с любой точкой M целиком содержит и некоторую ε-окрестность точки M (величина ε зависит от выбора точки M). Множества, обладающие подобным свойством, называются *открытыми множествами*. Итак, в случае непрерывных рецепторов *абстрактным образом* будем называть любое открытое множество пространства изображений.

Примером открытого множества может служить внутренняя часть сферы, имеющей ту же размерность, что и рассматриваемое (непрерывное) пространство изображений. Важно еще раз подчеркнуть, что степень малости изменений, которые можно вносить в изображение, не меняя его принадлежности данному образу, зависит от выбора самого изображения. Допустимые изменения для изображений, расположенных ближе к границе образа

(в рассмотренном примере границей служит поверхность сферы), уменьшаются, а для изображений, достаточно удаленных от границы, — увеличиваются.

В случае дискретного пространства изображений все его подмножества будут, очевидно, открытыми множествами и могут, следовательно, рассматриваться как абстрактные образы. Для сужения класса образов в дискретном пространстве можно воспользоваться понятием *граничного индекса* множества. Пусть M — произвольное подмножество введенного нами дискретного пространства R изображений, а m— число элементов этого множества. Если множество M не совпадает со всем пространством R, то среди его элементов найдутся такие, что на расстоянии, равном единице, от них будут лежать элементы, не принадлежащие множеству M. Назовем такие элементы *граничными элементами* и обозначим через m_1 число всех граничных элементов множества M. Отношение $\frac{m_1}{m}$ называется *граничным индексом* рассматриваемого множества M.

Легко понять, что, чем меньше граничный индекс множества, тем в большей степени оно по своим свойствам напоминает открытые множества непрерывных пространств: все большая часть точек множества содержится в нем со своими 1-окрестностями. Естественно поэтому высказать предположение, которое Э. М. Браверман [11] назвал *гипотезой компактности: образами в дискретном пространстве изображений могут служить лишь такие множества, граничный индекс которых достаточно мал.*

При более детальном исследовании оказывается, что высказанное предположение необходимо уточнить путем некоторых дополнительных теоретико-вероятностных построений. Пусть R — дискретное пространство изображений, в котором зафиксировано некоторое подмножество. Пусть, далее, для каждого элемента x_i пространства R задана вероятность $f(x_i)$ появления этого элемента (изображения) в какой-либо серии экспериментов типа независимых испытаний; $f_S(x_i)$ — условная вероятность появления элемента x_i при условии, что он принадлежит образу S. Если элемент x_i принадлежит образу S, то для любого натурального числа k $Q_k(x_i)$ — множество всех точек, не содержащихся в образе S и удаленных от элемента x_i на расстояние, меньшее или равное k. Величина $g_S(k) = \sum\limits_{x_i \in S} f_S(x_i) \sum\limits_{x_j \in Q_k(x_i)} f(x_j)$ представляет собой вероятность ошибочного приписывания в очередном испытании образу S не содержащегося в нем элемента за счет того, что в предыдущем испытании, в котором был случайно выбран какой-то элемент x_i из образа S, в образ S были зачислены все элементы, лежащие в k-окрестности элемента x_i.

Назовем операцию включения в тот или иной образ S всех элементов k-окрестности некоторого элемента x из этого образа *операцией k-экстраполяции по элементу x*. Найденная выше величина $g_S(k)$ представляет собой вероятность возникновения ошибки за счет операции k-экстраполяции по случайно выбираемому элементу

образа S. Уточнение гипотезы компактности, о котором шла речь выше, состоит в предположении, что для каждого дискретного пространства изображений R существует такое число $N = N(R)$, что $N \geqslant 1$ и для всех значений $k \leqslant N$ вероятность $g_S(k)$ возникновения ошибки в результате k-экстраполяции не превосходит пренебрежимо малую постоянную неотрицательную величину ε для любого образа S. Условимся называть эту гипотезу *гипотезой N-экстраполируемости образов с точностью до* ε.

Операция k-экстраполяции может быть, очевидно, определена и для образов в непрерывных пространствах изображений. Заменяя суммирование интегрированием, можно получить по аналогии с выражением для $g_S(k)$ выражение для вероятности возникновения ошибки за счет экстраполяции в непрерывном случае. Точно таким же образом, как и для дискретных пространств, можно сформулировать гипотезу N-экстраполируемости образов в непрерывных пространствах изображений.

Задача распознавания образов приводит к необходимости точного описания характеризующих этот образ признаков. Однако далеко не во всех случаях такого рода описание можно дать без преодоления весьма значительных трудностей. Поэтому на практике идут обычно по пути построения алгоритмов, позволяющих осуществлять так называемое *обучение* распознаванию образов. Смысл указанного обучения состоит в том, чтобы получить приближенное описание (или, как принято обычно говорить, аппроксимацию) образа в результате указания некоторого множества (вообще говоря, не всех!) составляющих этот образ изображений.

Исходя из гипотезы об N-экстраполируемости образов, можно построить так называемый *общий аппроксимационный алгоритм*, позволяющий осуществлять обучение распознаванию образов. Как и всякий алгоритм самосовершенствующегося типа, общий аппроксимационный алгоритм A имеет два периода работы — период обучения и период экзамена.

В период обучения на вход алгоритма A подаются различные представители образов $R_1, R_2, \ldots, R_n$, которые предстоит распознавать. При этом соответствующие представители (изображения) выбираются случайно (чаще всего методом независимых испытаний) и сопровождаются указанием: к какому из образов $R_1, R_2, \ldots, R_n$ относится каждое из выбранных изображений. Все показанные в период обучения изображения запоминаются и используются в режиме экзамена для определения принадлежности очередного (также случайно выбираемого) изображения r тому или иному из образов $R_1, R_2, \ldots, R_n$.

С этой целью определяются расстояния в пространстве изображений от изображения r сначала до выбранных в период обучения представителей образа R_1, затем до представителей образа R_2 и т. д. до тех пор, пока очередное определяемое расстояние до какого-либо представителя некоторого образа $R_i (i = 1, 2, \ldots, n)$ не окажется меньшим или равным коэффициенту экстраполируемости

N. В этом случае изображение r относится к образу R_i. Если же все расстояния окажутся большими N, изображение r останется неопознанным, т. е. не будет отнесено ни к какому из образов $R_1, R_2, \ldots, R_n$.

Легко понять, что если все образы $R_1, R_2, \ldots, R_n$ экстраполируемы с точностью до ε и могут быть покрыты с помощью такого числа N-окрестностей (сфер радиуса N), которое существенно меньше $\frac{1}{\varepsilon}$, то описанный алгоритм дает хорошую аппроксимацию выбранных образов (с малой вероятностью ошибки на экзамене).

Практически обычно различные образы в пространстве изображений непосредственно не соприкасаются. Если выбрать в качестве N минимальное расстояние между образами, то в дискретном пространстве отсутствие соприкосновения образов означает, что $N \geqslant 2$. Если принять еще, что вероятность появления изображений, не принадлежащих ни к одному из выбранных образов $R_1, R_2, \ldots, R_n$, равна нулю, то все эти образы окажутся, очевидно, $(N-1)$-экстраполируемыми с точностью до $\varepsilon = 0$. При этих предположениях общий алгоритм аппроксимации с помощью $(N-1)$-окрестностей приведет, очевидно, в результате достаточно длительного периода обучения к сколь угодно хорошей аппроксимации (со сколь угодно малой вероятностью ошибки).

Описанный общий алгоритм аппроксимации допускает ряд дальнейших усовершенствований в нескольких различных направлениях. Во-первых, кроме описанного выше режима экзамена можно ввести второй тип такого режима. В этом случае появляющееся в режиме экзамена изображение относится к тому из образов $R_1, R_2, \ldots, R_n$, в котором содержится представитель (запомненный в период обучения), расположенный ближе всех к изображению r. Для определенности полагаем, что, если таких образов окажется несколько, предпочтение отдается тому из них, который имеет наименьший номер.

Второе усовершенствование состоит в экономии памяти: если N — окрестность какого-либо изображения S, появившегося в период обучения, будет полностью покрыта N-окрестностями других изображений, также показанных в период обучения, то изображение S можно, очевидно, безболезненно исключить из памяти и не использовать при экзамене. Это усовершенствование на практике целесообразно применять в несколько иной модификации, при которой заранее ограничивается максимальное число представителей каждого образа, способное запоминаться в период обучения. Для каждого запоминаемого представителя ведется счет его относительной полезности. В качестве критерия относительной полезности какого-либо изображения r можно использовать, например, отношение числа случаев, когда это изображение было использовано для правильного опознавания последующих изображений, к общему числу изображений, появившихся после запоминания изображения r. Запоминанию подлежат только те изобра-

жения, которые не могут быть правильно опознаны с помощью уже имеющихся в памяти изображений. Если, кроме того, память, отведенная для запоминания представителей того или иного образа, оказывается полностью заполненной, то вновь запоминаемый представитель вытесняет из памяти изображение данного образа, имеющее наименьшую относительную полезность.

Третье усовершенствование состоит в том, что, наряду с «естественной» сетчаткой (на которую непосредственно проектируются распознаваемые изображения), употребляется новая сетчатка, выходные сигналы которой являются подходящим способом выбранными функциями выходных сигналов первой сетчатки. Пространство изображений, в котором задаются образы, строится из выходных сигналов второй сетчатки (впрочем, это пространство более естественно называть не пространством изображений, а пространством признаков). С помощью разумно выбранного преобразования исходного пространства можно значительно упростить задачу обучения распознаванию образов. Употребительными, в частности, являются такие преобразования, которые вырабатывают признаки, не зависящие от параллельного переноса и изменения размеров изображений.

Укажем, наконец, еще одну модификацию общего аппроксимационного алгоритма A. Режим, при котором этот алгоритм получает в период своего самосовершенствования некоторые изображения с указанием того, каким образом они принадлежат, носит название *режима обучения*. Помимо этого режима в ряде случаев целесообразно рассматривать так называемый *режим самообучения*.

В режиме самообучения, рассчитанном на образование n различных образов, первоначально задается n случайно выбранных изображений $r_1, r_2, \ldots, r_n$, каждое из которых принимается за представителя некоторого образа. Вновь показываемое изображение s_1 присоединяется к тому образу, представитель которого окажется наиболее близко расположенным к изображению s_1, и число представителей этого образа увеличивается. На следующем шаге вновь появившееся изображение s_2 сравнивают с увеличенным количеством представителей и опять относят к тому образу, представитель (какой-нибудь) которого расположен к s_2 ближе, чем представители всех других образов. В дальнейшем накопление представителей может происходить либо по обычной схеме, либо по описанной выше схеме с вытеснением (с ограниченной памятью). Опознавание изображений в режиме экзамена может осуществляться описанными выше двумя способами: либо способом N-экстраполяции образов, либо способом, основанным на определении кратчайшего расстояния.

Описанный алгоритм может применяться как в случае дискретных, так и в случае непрерывных пространств изображений. Использованный в нем метод аппроксимации не является, разумеется, единственным. Так, вместо аппроксимации сферическими окрестностями может быть использована аппроксимация окрестностями

любой другой формы. Успешные результаты были получены при аппроксимации образов областями, ограниченными гиперплоскостями (см., например, Э. М. Браверман [11]). Возможны также различные модификации описанной выше метрики в пространстве изображений, что приводит, очевидно, к изменению понятия сферических окрестностей: окрестности, сферические в одной метрике, могут не оказаться таковыми в другой метрике, и наоборот.

Опишем еще несколько более специальных алгоритмов для обучения распознаванию образов, успешно применявшихся различными авторами. Одним из наиболее простых, хотя и недостаточно эффективных, алгоритмов такого рода является так называемый *персептрон* Розенблатта [69]. Как и всякое устройство для распознавания образов, персептрон содержит некоторое множество рецепторов — сетчатку. В дальнейшем, не оговаривая этого каждый раз особо, будем рассматривать лишь правильные прямоугольные сетчатки. В зависимости от характера рецепторов персептроны делятся на *непрерывные персептроны* (с непрерывными рецепторами) и *дискретные персептроны* (с дискретными двоичными рецепторами).

Кроме рецепторов, каждый персептрон имеет в своем составе еще два вида элементов, называемых A-элементами и R-элементами.

A-элементы представляют собой упрощенные модели нейронов. В связи с этим в дальнейшем будем называть их просто *нейронами*. В соответствии с характером употребляющихся в персептроне рецепторов различают непрерывные и дискретные нейроны. Как те, так и другие нейроны имеют два вида входов, называемых *возбуждающими* и *запрещающими*. Каждый нейрон имеет конечное число входов и один выход; кроме того, ему сопоставляется некоторое вещественное число, называемое *весом* данного нейрона. В качестве области значений весов нейронов принимается множество всех вещественных чисел, независимо от того, о каких нейронах идет речь — непрерывных или дискретных.

Помимо веса, числа возбуждающих и числа запрещающих входов, нейрон характеризуется еще *законом своего функционирования*, определяющим выходной сигнал нейрона как функцию его входных сигналов и веса. Следует иметь в виду, что входы всех нейронов в персептроне подсоединяются к рецепторам, так что вырабатываемые рецепторами сигналы служат входными сигналами для нейронов.

Непрерывные нейроны, рассматривавшиеся первоначально Розенблаттом [69], имели закон функционирования вида $z = v(\Sigma x - \Sigma y)$, где z — выходной сигнал, v — вес нейрона, Σx — сумма сигналов, поступающих в нейрон по возбуждающим входам, а Σy — сумма сигналов, поступающих по запрещающим входам (x, y, v и z — произвольные числа).

Закон функционирования дискретных нейронов задается обычно указанием некоторого целого рационального числа p, называемого порогом срабатывания нейрона, или просто *порогом*. Если

алгебраическая сумма $\Sigma x - \Sigma y$ возбуждающих и запрещающих входных сигналов меньше порога, то нейрон считается невозбужденным и выдает выходной сигнал, равный нулю. При достижении суммой $\Sigma x - \Sigma y$ порога и при превышении его нейрон возбуждается и выдает выходной сигнал, равный своему весу v (независимо от величины превышения суммы входных сигналов над порогом).

Дискретные нейроны с описанным законом функционирования удобно характеризовать тройкой целых чисел (k, l, p), первое из которых равно числу возбуждающих входов, второе — числу запрещающих входов, а третье — величине порога. Вес нейрона в дальнейших рассмотрениях будет всегда величиной переменной и поэтому не будем вводить его в характеристику нейрона. Дискретные нейроны указанного вида, имеющие одну и ту же характеристическую тройку чисел (k, l, p), условимся относить к одному и тому же типу, независимо от возможного различия их весов.

В дальнейшем предполагается, что все нейроны любого данного персептрона принадлежат к одному и тому же типу. В персептроне, рассчитанном на различение k различных образов, множество всех нейронов разбивается на k попарно непересекающихся групп (подмножеств), поставленных во взаимно однозначное соответствие различаемым образом. Нейроны, принадлежащие группе, поставленной в соответствие i-му образу, будем для краткости называть *нейронами i-го образа* $(i = 1, 2, \ldots, k)$.

Входы каждого нейрона в персептроне подсоединяются к рецепторам сетчатки. Предполагается при этом, что различные входы одного и того же нейрона подсоединяются к различным рецепторам. Выходы же нейронов подсоединяются к специальным сумматорам, называемым R-элементами, причем выходы нейронов одного и того же образа подсоединяются к одному и тому же сумматору, называемому *сумматором этого образа*.

Выходной сигнал сумматора любого данного образа равняется сумме весов всех возбужденных нейронов этого образа. Если ни один из нейронов рассматриваемого образа не возбужден, то выходной сигнал соответствующего сумматора принимается равным нулю. Окончательным выходным сигналом всего персептрона считается тот образ, сумматор которого имеет наибольший выходной сигнал. В случае, когда максимальное значение выходного сигнала достигается одновременно сумматорами нескольких образов, выходной сигнал персептрона считается неопределенным.

Принимая в качестве входного сигнала всего персептрона проектируемое на его сетчатку изображение, получаем в качестве реакции персептрона на этот сигнал тот образ, к которому персептрон относит данное изображение. Ниоткуда, разумеется, не следует, что рассматриваемый персептрон осуществит правильную классификацию изображений в соответствии с *заданным заранее* разбиением множества изображений на различные образы. Это первоначальное разбиение задается человеком. Будем называть

его *исходной* (или *априорной*) классификацией изображений в отличие от *фактической* классификации, осуществляемой выбранным персептроном.

Необходимо поэтому еще задать некоторый процесс изменения характеристик персептрона, позволяющий по мере показа персептрону различных изображений приближать фактически производимую им классификацию к исходной классификации. Этот процесс задается с помощью указания так называемого *закона поощрения*.

Для дискретных персептронов в качестве основного закона поощрения выберем несколько обобщенный закон поощрения в так называемых α-*системах*, рассматривавшихся Джозефом [34]. Этот закон, который будем называть (*обобщенным*) α-*законом*, вполне характеризуется заданием двух неотрицательных констант a и b, не равных нулю одновременно. Смысл данного закона поощрения состоит в том, что после каждого показа персептрону очередного изображения веса одних нейронов увеличиваются на величину, равную a, а веса других — уменьшаются на величину, равную b, (закон поощрения в α-системах Джозефа получается из обобщенного α-закона в случае, когда $a = 1$, $b = 0$).

Различают два режима функционирования персептрона с обобщенным α-законом поощрения. Первый режим, называемый режимом *обучения*, состоит в поощрении (увеличении веса на величину a) всех возбужденных нейронов того образа, которому принадлежит рассматриваемое на данном шаге изображение, и в оштрафовании (уменьшении веса на величину b) всех возбужденных нейронов остальных образов. Ясно, что указание правильного образа, которому принадлежит данное изображение, должно осуществляться человеком-учителем, ибо только ему известна исходная априорная классификация изображений.

Второй режим, называемый режимом *самообучения*, отличается от режима обучения только тем, что определение образа, которому принадлежит рассматриваемое изображение, производится самим персептроном — в качестве такого образа выбирается тот образ, который фактически был выдан персептроном в ответ на показ данного изображения. Разумеется, при этом нет никакой гарантии, что выданный персептроном ответ будет правильным (в смысле исходной классификации изображений). Однако при соблюдении некоторых условий в случае неограниченного увеличения числа шагов в процессе самообучения персептрон может иногда восстановить исходную классификацию изображений.

Кроме (обобщенного) α-закона поощрения, в ряде случаев целесообразно рассматривать еще два закона, которые будем называть соответственно (*обобщенным*) β-*законом и* (*обобщенным*) γ-*законом*. В обоих этих законах сохраняется принцип поощрения и штрафования, принятый в (обобщенном) α-законе. В дополнение к этому в β-законе на каждом шаге (как в режиме обучения, так и в режиме самообучения) происходит уменьшение веса всех нейронов

(как возбужденных, так и невозбужденных) на величины, прямо пропорциональные их весам, с общим для всех нейронов коэффициентом пропорциональности β. В γ-законе осуществляется дополнительное (к операциям α-закона) изменение весов всех нейронов (как возбужденных, так и невозбужденных) на одну и ту же величину, выбираемую на каждом шаге так, чтобы сумма весов всех нейронов всегда была равной нулю.

В случае непрерывных нейронов обобщенный α-закон поощрения состоит в том, что любой нейрон правильного (априорно или с точки зрения персептрона) образа увеличивает свой вес на величину произведения q константы a на суммарный входной сигнал нейрона: $q = a(\Sigma x - \Sigma y)$. Аналогично веса нейронов всех остальных образов уменьшаются на величину $b(\Sigma x - \Sigma y)$ (свою для каждого отдельного нейрона). Дополнения, отличающие β- и γ-законы, остаются теми же, что и в дискретном случае.

При построении теории обучения и самообучения персептронов часто целесообразно рассматривать не отдельные персептроны, а некоторые классы персептронов. *Классом персептронов* будем называть множество персептронов, которые могут отличаться друг от друга лишь способом соединения нейронов с рецепторами и начальными весами нейронов. Все остальные характеристики персептронов, входящих в один и тот же класс, предполагаются одинаковыми. К этим характеристикам относится вид рецепторов и нейронов, общее число рецепторов и структура сетчатки, множество изображений и множество образов, исходная классификация изображений (распределение их по образам), число нейронов каждого образа и, наконец, закон поощрения.

Способ же соединения нейронов с рецепторами и начальные веса нейронов считаются случайными и характеризуются (в пределах выбранного класса) некоторыми законами распределения. Иными словами, класс персептронов рассматривается не как абстрактное множество персептронов, а как множество с заданным на нем полем вероятностей, определяющим вероятность выбора того или иного конкретного представителя рассматриваемого класса. Можно, таким образом, считать, что задание класса определяет некоторый *случайный персептрон*.

Начальные веса нейронов обычно считаются независимыми случайными величинами, имеющими один и тот же закон распределения. Точно так же способ соединения каждого нейтрона с сетчаткой предполагается не зависимым от соединений остальных нейронов. Каждому же возможному способу соединения отдельного нейрона с сетчаткой сопоставляется вероятность этого способа, общая для всех нейронов. При этом подсоединение всех нейронов персептрона (из рассматриваемого класса персептронов) к сетчатке трактуется как серия независимых опытов, характеризующихся указанными вероятностями.

Сочетая вероятностную характеристику для способа соединения нейронов с сетчаткой с законом распределения начальных ве-

сов нейронов, приходим к искомому закону распределения в классе персептронов. Один из наиболее часто встречающихся законов распределения получается, когда все начальные веса детерминированы и равны одному и тому же числу (чаще всего нулю), а подсоединение всех входов любого данного нейрона производится независимо друг от друга на основании того или иного закона распределения (чаще всего равномерного), заданного непосредственно на сетчатке.

При построении теории обучения персептронов приходится рассматривать так называемые *обучающие последовательности* и *классы обучающих последовательностей*. Обучающая последовательность — это просто конечная последовательность изображений, показанных персептрону одно за другим в процессе его обучения или самообучения. Общее число показанных изображений (включая повторения) называется *длиной* обучающей последовательности. Классом обучающих последовательностей называется множество всех последовательностей одной и той же длины, в котором задан закон распределения, определяющий вероятность выбора любой данной последовательности рассматриваемого класса.

Чаще всего такой закон распределения получается с помощью приписывания определенного значения вероятности появления любого изображения из рассматриваемого множества изображений на каждом шаге обучения, причем обычно рассматривается случай, когда эти вероятности одинаковы на всех шагах, т. е., когда обучающая последовательность представляет собой серию независимых экспериментов по выбору изображений с приписанными каждому изображению неизменными вероятностями. В дальнейшем ограничимся именно этим случаем.

Эффективность обучения в данном классе A персептронов с помощью данного класса B обучающих последовательностей определяется как вероятность правильного опознавания очередного изображения p, подаваемого на случайно выбираемый из класса A персептрон после предварительной подачи на него обучающей последовательности, случайно выбранной из класса B. Различают два вида эффективностей. Так называемая *полная* эффективность обучения получается тогда, когда изображение p выбирается случайно (с фиксированными заранее вероятностями появления различных изображений, использованными при установлении закона распределения в классе обучающих последовательностей). Эффективность обучения *по одиночному изображению* q получается, когда в качестве очередного изображения персептрону предлагается распознать именно изображение q. Если вероятности ошибок опознавания одинаковы для всех изображений, то полная эффективность обучения, очевидно, совпадает с одиночной эффективностью обучения по любому изображению.

В следующем параграфе предпримем теоретическое изучение эффективности обучения для дискретных персептронов с α-законом поощрения. А пока заметим, что, как показывают эксперименты, эффективность обучения во всех типах описанных выше персептро-

нов оказывается сравнительно невысокой. Поэтому в алгоритмах для обучения распознаванию образов, действительно применяющихся на практике, вводится обычно ряд дополнительных усовершенствований по сравнению со схемой персептрона. Например, в схеме *адапта Робертса* [67], в целом весьма похожей на схему персептрона с α-законом поощрения, значительное повышение эффективности обучения достигается за счет предварительной нормализации изображения (т. е., иными словами, за счет автоматического сдвига изображения в центр сетчатки и приведения его к стандартному размеру). Применяются также методы предварительной выработки признаков, схемы многоступенчатых персептронов и т. п.

Недостатки персептрона и пути их устранения станут более ясными после ознакомления со следующими двумя параграфами, в которых рассматриваются некоторые вопросы, связанные с его поведением в режимах обучения и самообучения.

§ 6. Теория обучения дискретных α-персептронов

В настоящем параграфе наметим основные контуры теории обучения персептронов. При этом ограничимся рассмотрением лишь дискретных персептронов с обобщенным α-законом поощрения, работающих в режиме обучения (а не самообучения!), не оговаривая этого обстоятельства каждый раз особо. Теория обучения в этом случае получается наиболее простой и прозрачной, поскольку оказывается возможным не следить за функционированием каждого отдельного нейрона, а ограничиться рассмотрением лишь некоторых интегральных характеристик.

В предлагаемом нами варианте теории такой интегральной характеристикой служит так называемый *характеристический тензор* персептрона. Подчеркнем сразу, что применение термина «тензор» в этом случае не связано ни с какими закономерностями преобразования его компонент при изменении системы координат, а служит лишь наименованием некоторой целочисленной таблицы с тремя входами. Для описания такой таблицы введем определенную нумерацию всех изображений, предлагаемых рассматриваемому персептрону, числами от 1 до m и нумерацию всех образов, на которые подразделяются эти изображения, числами от 1 до q. Тогда характеристический тензор персептрона будет представлять собой совокупность компонент T_{ij}^{k}, где индексы i и j пробегают значения от 1 до m, а индекс k — значения от 1 до q. T_{ij}^{k} *обозначается число нейронов k-го образа, которые возбуждаются как i-тым, так и j-ым изображением.*

Характеристический тензор класса персептронов определяется по существу точно так же. Различие состоит лишь в том, что его компоненты T_{ij}^{k} будут на этот раз не детерминированными, а *случайными* величинами, законы распределения которых очевид-

ным образом определяются законом распределения, характеризующим способ подсоединения нейронов к сетчатке.

Из приведенных определений непосредственно следует справедливость соотношения

$$T_{ij}^k = T_{ji}^k \quad (i, j = 1, 2, \ldots, m;\ k = 1, 2, \ldots, q). \tag{76}$$

Ясно также, что любой «диагональный» элемент тензора, например T_{ii}^k, представляет собой число нейронов того или иного (в данном случае k-го) образа, возбуждающихся под действием i-го изображения. Отсюда непосредственно вытекает справедливость неравенства

$$T_{ii}^{k_1} \geqslant T_{ij}^k \quad (i, j = 1, 2, \ldots, m;\ k = 1, 2, \ldots\ q). \tag{77}$$

Персептрон или класс персептронов будем называть *симметричными*, если компоненты характеристического тензора не зависят от верхнего индекса, т. е. если справедливо соотношение

$$T_{ij}^{k_1} = T_{ij}^{k_2} \ (k_1, k_2 = 1, 2, \ldots, q;\ i, j = 1, 2 \ldots. m). \tag{78}$$

Верхний индекс в этом случае лишний, так что симметричные персептроны и классы персептронов естественно характеризовать не трехвходовой (T_{ij}^k), а двухвходовой таблицей (T_{ij}), где $T_{ij} = T_{ij}^1 = T_{ij}^2 = \ldots = T_{ij}^q$, которую условимся называть *характеристической матрицей* персептрона (или класса персептронов).

Введем еще одно обозначение. Для произвольной (конечной) последовательности изображений l через $U_i^k(l)$ обозначим выходной сигнал сумматора k-го образа, индуцируемый в рассматриваемом персептроне i-ым изображением, показанным после обучения персептрона обучающей последовательностью l. Через U_i^k обозначим соответствующий сигнал до начала процесса обучения, т е., иными словами, сигнал $U_i^k(l)$ для случая, когда обучающая последовательность l пуста (имеет длину, равную нулю).

Величины $U_i^k(l)$ будут, очевидно, детерминированными в случае выбора определенного персептрона и случайными в случае рассмотрения класса персептронов. Иногда целесообразно рассматривать также последовательность l как случайную последовательность, пробегающую класс обучающих последовательностей.

Особенностью α-закона обучения персептронов является своеобразное *свойство коммутативности* процесса обучения, выражаемое следующим предложением.

Теорема 1. *В персептроне (или в классе персептронов) с (обобщенным) α-законом поощрения выходной сигнал $U_i^k(l)$ сумматора k-го образа под действием i-го изображения после обучения любой последовательностью l не изменится, если в последовательности l будет произведена произвольная перестановка составляющих ее изображений. Это справедливо для любого изображения i и любого образа k.*

В самом деле, входные сигналы нейронов k-го образа, индуцируемые i-ым изображением, очевидно, не изменяются в процессе обучения, так что они остаются одинаковыми после показа обучающей последовательности l и любой другой последовательности l'. Таким образом, изменение выходного сигнала сумматора в процессе обучения обусловлено исключительно изменением весов нейронов. В силу определения обобщенного α-закона поощрения, изменения весов нейронов при показе любого изображения в процессе обучения (но, вообще говоря, не в процессе самообучения) не зависят от того места, которое данное изображение занимает в обучающей последовательности. Поскольку же суммарное приращение веса любого нейрона в процессе обучения равно просто сумме приращений на каждом шаге процесса, то справедливость теоремы 1 тем самым полностью доказана.

Используя теорему 1, получаем возможность характеризовать любую обучающую последовательность l целочисленным вектором $\vec{v} = (v_1, v_2, \ldots, v_m)$, i-ая компонента которого (для любого $i = 1, 2, \ldots, m$) равна числу вхождений i-го изображения в последовательность l. Условимся называть этот вектор *характеристическим вектором последовательности* l. Класс обучающих последовательностей также можно задавать с помощью характеристического вектора. Однако компоненты вектора будут в этом случае, вообще говоря, уже не детерминированными, а случайными величинами..

Для описания процесса обучения персептронов с (обобщенным) α-законом поощрения исходный персептрон (или класс персептронов) достаточно задавать лишь его характеристическим тензором (T_{ij}^k) и матрицей начальных сигналов сумматоров образов (U_i^k) $(i, j = 1, 2, \ldots, m;\ k = 1, 2, \ldots, m)$. Обучающая же последовательность l (или класс обучающих последовательностей) задается своим характеристическим вектором $(v_1, v_2, \ldots, v_n)$. В общем случае все величины T_{ij}^k, U_i^k, v_i будут случайными. Однако чаще всего рассматриваются различные частные случаи, когда те или иные из указанных величин детерминированы. Заметим, что будем обычно включать указание о выборе множества изображений и обучающих последовательностей в определение персептрона.

Для того чтобы не путать изображения и образы, условимся образы обозначать латинскими буквами, а изображения по-прежнему отождествлять с их номерами. Рассмотрим вопрос об определении выходных сигналов сумматоров образов $U_i^P(l)$. Легко понять, что при использовании (обобщенного) α-закона поощрения величина $U_i^P(l)$ представится в виде суммы начального сигнала U_i^P и приращений весов всех нейронов P-го образа, возбуждающихся i-ым изображением, на всех шагах процесса обучения.

Характеризуя класс обучающих последовательностей характеристическим вектором $(v_1, v_2, \ldots, v_m)$, нетрудно найти выражение

для суммарного приращения величины $U_i^P(l)$, полученной за счет v_j показов j-го изображения. Как следует из определения (обобщенного) α-закона с константами a и b, при каждом показе j-го изображения любой возбуждающийся этим изображением нейрон P-го образа увеличивает свой вес на величину $-a$, если $j \in P$, и уменьшает вес на величину b, если $j \bar{\in} P$. Общее же число нейронов P-го образа, участвующих в образовании выходного сигнала $U_i^P(l)$ и возбуждающихся j-ым изображением, равно, очевидно, T_{ij}^P. Таким образом, суммарное приращение величины за счет v_j показов j-го изображения выразится формулой $aT_{ij}^P v_j$, если $j \in P$, и формулой $-bT_{ij}^P v_j$, если $j \bar{\in} P$. Справедливо поэтому следующее предложение.

Теорема 2. *Пусть задан дискретный персептрон (или класс дискретных персептронов) с характеристическим тензором* T_{ij}^P *и матрицей начальных выходных сигналов сумматоров образов* U_i^P $(i, j = 1, 2, \ldots, m;\ P \in R)$. *Если в рассматриваемом персептроне (классе персептронов) действует (обобщенный)* α*-закон поощрения с константами* a, b, *то после обучения последовательностью (или классом последовательностей)* l *с характеристическим вектором* $(v_1, v_2, \ldots, v_m)$ *для любого образа* P *и любого изображения* i *выходной сигнал* $U_i^P(l)$ *сумматора* P*-го образа под действием* i*-го изображения выражается формулой*

$$U_i^P(l) = U_i^P + a \sum_{j \in P} T_{ij}^P v_j - b \sum_{j \bar{\in} P} T_{ij}^P v_j. \qquad (79)$$

Для любого изображения i через P_i условимся обозначать тот образ, которому принадлежит изображение i в исходной классификации изображений. Используя это обозначение, нетрудно выписать необходимое и достаточное условие для того, чтобы после обучения персептрон правильно классифицировал i-е изображение. Таким условием будет, очевидно, выполнение неравенств

$$U_i^{P_i}(l) > U_i^P(l) \quad \text{для всех } P \neq P_i. \qquad (80)$$

С помощью соотношений (79) и (80) нетрудно рассчитать эффективность обучения персептрона в любом конкретном случае. Особенно простой вид эти соотношения принимают в случае симметричных персептронов. В самом деле, поскольку в этом случае $T_{ij}^{P_i} = T_{ij}^P = T_{ij}$, то соотношения (79) и (80) можно записать в виде системы неравенств

$$U_i^{P_i} + a \sum_{j \in P_i} T_{ij} v_j - b \sum_{j \bar{\in} P_i} T_{ij} v_j > U_i^P + a \sum_{j \in P} T_{ij} v_j - b \sum_{j \bar{\in} P} T_{ij} v_j. \qquad (81)$$

В неравенстве (81) слагаемые вида $-b\Sigma T_{ij} v_j$, для которых j не содержится ни в P_i, ни в P, входят как в левую, так и в правую часть и потому взаимно уничтожаются. После их исключения

получаем более простые соотношения, эквивалентные соотношению (81):

$$U_i^{P_i} + (a+b)\sum_{j\in P_i} T_{ij}v_j > U_i^P + (a+b)\sum_{j\in P} T_{ij}v_j \text{ для всех } P \neq P_i. \quad (82)$$

Неравенства (82) дают необходимые и достаточные условия для правильной классификации i-го изображения *симметричным* персептроном с характеристической матрицей $\|T_{ij}\|$ и начальными сигналами сумматоров образов U_i^P после обучения последовательностью с характеристическим вектором $(v_1, v_2, \ldots, v_m)$. Эти неравенства можно упростить еще больше для персептронов с *симметричными начальными условиями*, т.е. таких персептронов (или классов персептронов), для которых выполняются условия

$$U_i^P = U_i^Q \quad \text{для всех } i = 1, 2, \ldots, m \quad (83)$$

и для всех P и Q.

Используя соотношения (83) и вспоминая, что, в силу определения обобщенного α-закона $a + b \neq 0$, приходим к следующему результату.

Теорема 3. *Пусть задан произвольный дискретный симметричный персептрон (или класс дискретных симметричных персептронов) с характеристической матрицей $\|T_{ij}\|$ и с симметричными начальными условиями, в котором действует (обобщенный) α-закон поощрения. Тогда необходимые и достаточные условия правильности распознавания персептроном (классом персептронов) произвольного i-го изображения после обучения последовательностью (классом последовательностей) с характеристическим вектором $(v_1, v_2, \ldots, v_m)$ выразятся соотношениями*

$$\sum_{j\in P_i} T_{ij}v_j > \sum_{j\in P} T_{ij}v_j \quad \text{для всех } P \neq P_i. \quad (84)$$

Следствие: *эффективность обучения в симметричных дискретных персептронах с симметричными начальными условиями при выполнении в них (обобщенного) α-закона поощрения не зависит от выбора (неотрицательных) констант a и b, характеризующих закон.*

Таким образом, при изучении симметричных дискретных персептронов с симметричными начальными условиями можно, не нарушая общности, вместо обобщенного α-закона поощрения с константами (a, b) пользоваться обычным α-законом поощрения с константами (1,0).

При конкретных расчетах эффективности обучения в классах персептронов обычно принимается, что все нейроны подсоединяются к сетчатке независимо один от другого, причем вероятность α_{ij} такого подсоединения нейрона, что он будет возбуждаться как i-ым, так и j-ым изображением одна и та же для всех нейронов при любых фиксированных значениях i и j.

Если обозначить через T^P общее число нейронов P-го образа, то компонента T_{ij}^P характеристического тензора рассматриваемого класса персептронов может трактоваться как число наступлений некоторого события, имеющего вероятность α_{ij} при T^P независимых испытаниях. В силу теоремы 2 из § 2 настоящей главы, математическое ожидание $E(T_{ij}^P)$ и дисперсия $D(T_{ij}^P)$ случайной величины T_{ij}^P выражаются формулами

$$E(T_{ij}^P) = T^P\alpha_{ij};\ D(T_{ij}^P) = T^P\alpha_{ij}(1-\alpha_{ij})$$
$$(i, j = 1, 2, \ldots, m;\ P \in R). \qquad (85)$$

Сама же величина T_{ij}^P при достаточно больших значениях T^P может считаться нормально распределенной. Заметим еще, что в случае симметричных персептронов величины T^P для различных образов P равны между собой. Поэтому обозначим их просто буквой T, опуская индекс P. Матрицу $\|\alpha_{ij}\|$ условимся называть *основной вероятностной матрицей* рассматриваемого класса персептронов.

Аналогично класс обучающих последовательностей K, образуемый с помощью случайного выбора изображения на каждом шаге обучения, независимо от изображений, выбранных на остальных шагах, можно характеризовать *вероятностным вектором* $(\beta_1, \beta_2, \ldots, \beta_m)$ рассматриваемого класса. Для любого $i = 1, 2, \ldots, m$ i-я компонента β_i этого вектора равна вероятности выбора в качестве изображения, показываемого на любом данном шаге обучения, i-го изображения. В таком случае i-я компонента v_i характеристического вектора класса K представляет собой число наступлений события, имеющего вероятность β_i, при N независимых испытаниях, где N — длина обучающих последовательностей класса K (согласно определению класса обучающих последовательностей, все входящие в класс последовательности имеют одну и ту же длину).

При достаточно больших значениях N для любого изображения i величина v_i может считаться нормальной распределенной, а ее математическое ожидание и дисперсия даются формулами

$$E(v_i) = N\beta_i;\quad D(v_i) = N\beta_i(1-\beta_i)\ \ (i = 1, 2, \ldots, m). \qquad (86)$$

Заметим, что случайные величины v_i, как и случайные величины T_{ij}^D, при различных i и j не являются, вообще говоря, независимыми, что создает дополнительные трудности при вычислении вероятности правильного срабатывания персептрона по формулам (81) и (84). Впрочем, в ряде случаев с помощью введения некоторых добавочных предположений удается обойти эти трудности. Продемонстрируем это обстоятельство на ряде примеров.

Пример 1. Рассматривается дискретный персептрон A с нейронами типа (1, 1, 1,), имеющий правильную квадратную $(n \times n)$-сетчатку и $2n$ изображений, в качестве которых выбирается n горизонтальных линий длины n, объединяемых в образ P,

и n вертикальных линий длины n, объединяемых в образ Q. Все изображения имеют одинаковые вероятности $\left(\text{равные } \frac{1}{2n}\right)$ появления в обучающей последовательности. Предположим, что персептрон A *полный*. Это означает, что как в множестве нейронов P-го образа, так и в множестве нейронов Q-го образа для любого способа подсоединения нейрона к сетчатке найдется в точности по одному нейрону, имеющему точно такое же соединение с сетчаткой. В персептроне A действует обобщенный α-закон поощрения с константами a и b, а начальные веса нейронов равны нулю.

Требуется найти эффективность обучения персептрона A в классе случайных обучающих последовательностей длины $2N$, содержащих точно N показов изображений первого образа и N показов изображений второго образа.

Решение. Персептрон A будет, очевидно, симметричным и будет поэтому полностью характеризоваться своей характеристической матрицей $\|T_{ij}\|$. Легко видеть, что нейрон тогда и только тогда возбуждается i-ым изображением (вертикальной или горизонтальной линией), когда его возбуждающий вход подсоединен к рецептору, лежащему на соответствующей линии, а его запрещающий вход — к рецептору, лежащему вне этой линии. Для любого данного i имеется всего $n(n^2 - n) = n^2(n - 1)$ различных подсоединений такого рода. Ввиду полноты персептрона A, справедлива формула (87)

$$T_{ii} = n^2(n - 1) \quad (i = 1, 2, \ldots, 2n). \tag{87}$$

Предположим, что номерами от 1 до n обозначены горизонтальные линии (изображения образа P), а номерами от $n + 1$ до $2n$ — вертикальные линии (изображения образа Q). По аналогии с тем, как было найдено выражение для T_{ii}, найдем еще два выражения

$$T_{ij} = 0, \tag{88}$$

если i и j — изображения одного и того же образа;

$$T_{ij} = (n - 1)^2, \tag{89}$$

если i и j — изображения различных образов.

Обозначая через E единичную матрицу n-го порядка, а через D — квадратную матрицу порядка n, все элементы которой равны единице, представим характеристическую матрицу M рассматриваемого персептрона в виде

$$M = \begin{Vmatrix} n^2(n-1)E & (n-1)^2 D \\ (n-1)^2 D & n^2(n-1)E \end{Vmatrix}. \tag{90}$$

Пусть $(v_1, v_2, \ldots, v_{2n})$—характеристический вектор рассматриваемого класса обучающих последовательностей. В силу принятого условия, компоненты этого вектора удовлетворяют условию

$$v_1 + v_2 + \ldots + v_n = v_{n+1} + v_{n+2} + \ldots + v_{2n} = N. \tag{91}$$

Необходимое и достаточное условие правильного распознавания любого данного изображения i запишем, в силу теоремы 3,

$$\sum_{j \in P_i} T_{ij} v_j > \sum_{j \bar{\in} P_i} T_{ij} v_j \tag{92}$$

или, учитывая соотношения (87) — (89) и (91),

$$n^2 (n-1) v_i > (n-1)^2 N. \tag{93}$$

Соотношение (93) представим в эквивалентном виде

$$v_i > \left(\frac{1}{n} - \frac{1}{n^2} \right) N. \tag{94}$$

Вероятность появления i-го изображения в каждом из N показов представителей образа P_i равна $\frac{1}{n}$. Поэтому для математического ожидания и дисперсии величины v_i получим выражения

$$E(v_i) = \frac{1}{n} N; \qquad D(v_i) = N \frac{1}{n} \left(1 - \frac{1}{n} \right). \tag{95}$$

В силу теоремы 3 из § 3 настоящей главы, при достаточно большом N вероятность q_i выполнения неравенства (94) может быть вычислена по формуле

$$q_i \approx 0{,}5 + \frac{1}{\sqrt{2\pi}} \int_0^k e^{-\frac{z^2}{2}} dz \qquad (i = 1, 2, \ldots, 2n), \tag{96}$$

где k — величина отношения модуля разности правой части неравенства (94) и математического ожидания $E(v_i)$ к среднеквадратичному уклонению величины v_i, равному квадратному корню из дисперсии. Иными словами,

$$k = \frac{1}{n^2} N : \sqrt{N \cdot \frac{1}{n} \left(1 - \frac{1}{n} \right)} = \sqrt{\frac{N}{n^3 \left(1 - \frac{1}{n} \right)}} \approx \sqrt{\frac{N}{n^3}}. \tag{97}$$

Поскольку величина q_i не зависит от i, она совпадет с вероятностью q правильного распознавания персептроном A любого наугад выбираемого изображения после предварительного показа случайно выбираемой обучающей последовательности длины $2N$ из рассматриваемого класса последовательностей.

Величина же вероятности q есть не что иное, как полная эффективность обучения персептрона A в заданных условиях. Приведем таблицу значений вероятности q для нескольких значений k:

N	n^3	$4n^3$	$9n^3$
k	1	2	3
q	0,841	0,977	0,999

Таким образом, для того чтобы уменьшить вероятность ошибки рассматриваемого персептрона до 0,1% при случайном выборе обучающей последовательности, необходимо пользоваться последовательностями весьма большой длины, равной $18\,n^3$. В то же время из неравенства (92) непосредственно видно, что можно свести эту вероятность к нулю (получив *абсолютно точно* распознавание) за счет показа каждого изображения в точности по одному разу, т. е. с помощью последовательности, имеющей длину, равную всего лишь $2\,n$. Этот пример наглядно демонстрирует невыгодность использования случайных обучающих последовательностей. Вместе с тем он свидетельствует о серьезных отличиях описанного механизма обучения от механизма обучения, реализующегося в мозгу человека.

В самом деле, последний механизм обладает ярко выраженной способностью к *экстраполяции опыта*, т. е. к правильному распознаванию изображений, которые ни разу не появлялись в процессе обучения. В то же время описанный в рассмотренном примере персептрон не дает окончательной гарантии правильного распознавания изображений (при случайной организации процесса обучения) даже тогда, когда среднее число показов каждого изображения достигает весьма значительной величины (имеющей порядок n^3).

Указанный вывод связан, разумеется, до некоторой степени со спецификой самого примера. Нетрудно заметить, однако, что при чисто случайном подсоединении (1, 1, 1)-нейронов к сетчатке (исключая подсоединение к одному и тому же рецептору обоих входов нейрона) математические ожидания компонент характеристической матрицы будут отличаться от компонент характеристической матрицы полного персептрона лишь постоянным множителем, не существенным с точки зрения вычисления эффективности обучения. Поэтому при случайном подсоединении нейронов к сетчатке наиболее вероятным поведением получающихся персептронов будет именно описанное выше поведение полного персептрона.

Таким образом, случайная организация связей нейронов с сетчаткой не может, вообще говоря, обеспечить хорошее качество

функционирования персептрона. Из теоремы 3 следует, что способность персептрона к экстраполяции опыта увеличивается при увеличении тех компонент характеристической матрицы, индексы которых принадлежат одному и тому же образу, и при уменьшении тех компонент, индексы которых принадлежат различным образам.

Условимся говорить, что персептрон обладает *абсолютной способностью к экстраполяции*, если для любого образа P и любого изображения i из этого образа обучение любой последовательностью, содержащей не менее одного раза изображение i, приводит к правильному распознаванию всех изображений этого образа. Получим следующий результат.

Теорема 4. *Для того чтобы дискретный симметричный персептрон с симметричными начальными условиями, в котором действует (обобщенный) α-закон поощрения, обладал абсолютной способностью к экстраполяции, необходимо и достаточно, чтобы все компоненты T_{ij} характеристической матрицы персептрона, индексы которых принадлежат одному и тому же образу, были отличны от нуля, а все компоненты T_{ij}, индексы которых принадлежат различным образам, были равны нулю.*

В самом деле, пусть условие теоремы выполнено. Тогда неравенство (84) будет справедливым, если хотя бы для одного изображения j из P_i величина v_j будет отлична от нуля. В силу теоремы 3. это как раз и означает, что рассматриваемый персептрон обладает абсолютной способностью к экстраполяции.

Предположим, что условие теоремы не выполнено. Это приводит к рассмотрению двух случаев: 1) для некоторого образа Q найдется принадлежащая ему пара изображений i, j такая, что $T_{ij} = 0$; 2) найдется пара изображений k, r, принадлежащих различным образам, и такая, что $T_{kr} \neq 0$. В первом случае, в силу теоремы 3, обучающая последовательность, составленная исключительно из изображений j, не приведет к правильному распознаванию изображения i. Во втором случае рассмотрим обучающую последовательность, составленную из одного изображения k и любого числа v_r, большего чем $\frac{T_{kk}}{T_{kr}}$, изображений r. Тогда, применительно к распознаванию изображения k, подстановка указанных значений в неравенство (84) приводит к неравенству $T_{kk} > T_{kr}\, v_r$. Ввиду выбора v_r это неравенство неверно, что, в силу теоремы 3, означает невозможность правильного распознавания изображения k. Следовательно в обоих случаях персептрон не будет обладать абсолютной способностью к экстраполяции, что и требовалось доказать.

Обычно изображения, принадлежащие одному и тому же образу, нумеруются последовательными целыми числами. При этом условии характеристические матрицы симметричных персептронов естественно разбиваются на клетки, соответствующие различным образам. Абсолютная способность к экстраполяции достигается

в том случае, когда эти матрицы клеточно-диагональные, а диагональные клетки не содержат нулевых элементов. Такой вид характеристических матриц не всегда бывает в полной мере достижим, однако и сколько-нибудь удачное приближение к нему требует, как правило, отказа от вполне случайного соединения нейронов с сетчаткой. Получаемый в результате такого отказа эффект лучше всего продемонстрировать на примере.

Пример 2. Найти эффективность обучения персептрона B, отличающегося от персептрона A из примера 1 лишь тем, что в нем оставлены только те нейроны, оба входа которых подсоединены к рецепторам, лежащим либо на одной горизонтальной, либо на одной вертикальной линии. Условия обучения те же, что и в примере 1.

Решение. Персептрон B, как и персептрон A, будет, очевидно, симметричным. Нетрудно подсчитать, что элементы его характеристической матрицы задаются соотношениями $T_{ii} = n(n-1)$; $T_{ij} = 0$ $(i, j = 1, 2, \ldots, 2n;\ i \neq j)$. Условие правильного распознавания i-го изображения выразится условием $T_{ii}\, v_i > 0$, или, что одно и то же, $v_i > 0$. Иначе говоря, для правильного распознавания i-го изображения необходимо и достаточно, чтобы оно было хотя бы раз показано персептрону в процессе его обучения.

При N случайных показах изображений одного образа вероятность непоявления в обучающей последовательности i-го изображения равна, очевидно, $\left(1-\frac{1}{n}\right)^N \approx e^{-\frac{N}{n}}$, а полная эффективность обучения выражается числом $1-e^{-\frac{N}{n}}$. Для того чтобы уменьшить вероятность неправильного срабатывания персептрона до 0,1%, как это было сделано в примере 1, достаточно положить $N = 7\,n$, или, иными словами, оперировать обучающей последовательностью длины $14\,n$. Напомним, что в первом примере такая же эффективность обучения достигалась лишь на обучающих последовательностях длины $18n^3$.

Любопытно, что столь резкое увеличение эффективности обучения достигнуто не за счет усложнения, а за счет упрощения персептрона, поскольку персептрон B получается из персептрона A с помощью выбрасывания большого числа плохо присоединенных к сетчатке нейронов. Легко подсчитать, что общее число нейронов в персептроне A равно $2n^2(n^2-1)$, а в персептроне B — лишь $4n(n-1)$. Это обстоятельство свидетельствует еще раз о несовершенстве обучающего механизма персептрона и его существенном отличии от процессов обучения, протекающих в мозгу человека.

Рассмотрим еще пример расчета эффективности обучения в классе персептронов.

Пример 3. Определить эффективность обучения класса C дискретных симметричных персептронов с симметричными начальными условиями, подчиняющихся обобщенному α-закону поощрения. Сетчатка, образы и изображения такие же, как и в примере 1. Число нейронов каждого из двух имеющихся образов равно N. Вхо-

ды всех нейронов подсоединяются независимо друг от друга с равной вероятностью к любому рецептору сетчатки, исключая лишь случай одновременного подсоединения обоих входов нейрона к одному и тому же рецептору. Обучающая последовательность содержит каждое из $2n$ изображений в точности по одному разу.

Решение. Легко видеть, что компоненты T_{ij} характеристической матрицы класса C, у которых индексы i и j являются различными изображениями одного и того же образа, равны нулю. Условие правильного распознавания i-го изображения, даваемое теоремой 3, запишется в нашем случае

$$T_{ii} > \sum_{j \in P_i} T_{ij}. \tag{98}$$

Легко видеть, что множества M_{ij} нейронов одного и того же образа, возбуждающихся как i-ым, так и j-ым изображением при различных j, отличных от i, попарно не пересекаются. Все эти множества содержатся, разумеется, в множестве M_{ii}. Так как T_{ij} есть не что иное, как число элементов множества M_{ij}, то для выполнения неравенства (98) необходимо и достаточно, чтобы среди нейронов образа P_i нашелся хотя бы один нейрон, возбуждающийся i-ым изображением, но не возбуждающийся никаким изображением противоположного (отличного от P_i) образа.

Из геометрии изображений непосредственно вытекает, что этому условию удовлетворяют либо нейроны, оба входа которых подсоединены к одной и той же вертикали (если i — горизонтальная линия), либо к одной и той же горизонтали (если i — вертикальная линия). Для любого фиксированного i из общего числа $n^2(n^2-1)$ различных подсоединений этому условию удовлетворяет лишь $n(n-1)$ подсоединений. Вероятность желательного подсоединения равна поэтому $\frac{n(n-1)}{n^2(n^2-1)} = \frac{1}{n(n+1)}$, а вероятность, что такое подсоединение не будет иметь места ни для одного из N нейронов, равна $\left(1-\frac{1}{n(n+1)}\right)^N \approx e^{-\frac{N}{n(n+1)}}$. Следовательно, полная эффективность обучения выражается формулой

$$r \approx 1 - e^{-\frac{N}{n(n+1)}}.$$

Если число нейронов каждого образа равно $7n$ $(n+1)$, т. е. примерно в 7 раз превышает общее число рецепторов, то вероятность неправильного срабатывания наугад выбираемого из класса C персептрона после обучения с помощью показа всех изображений по одному разу будет равна e^{-7}, что равняется приблизительно 0,001.

Как отмечалось выше, построенная теория обучения персептронов свидетельствует о коренных отличиях реализуемого ими механизма процесса обучения от реального процесса обучения

в мозгу человека. Переход от дискретных нейронов к непрерывным, как и замена α-закона поощрения β- или γ-законом, этого положения существенно не меняет. Частично положение может быть исправлено за счет добавления к реализуемым в персептроне процессам пересоединения нейронов, мешающих процессу обучения или недостаточно способствующих ему.

Можно предусмотреть, например, периодическую проверку весов нейронов и случайные пересоединения нейронов с малым весом. Механизмы такого рода реализуются в схемах адапта Робертса [67] и пандемониума Селфриджа [72]. Они позволяют увеличивать коэффициент использования оборудования и уменьшать число нейронов, достигающее в схемах персептронов с чисто случайными связями непомерно больших значений.

Вместе с тем указанная мера совершенно недостаточна для объяснения такой особенности приспособительных функций мозга, как использование тех или иных признаков, выделенных на уже изученных образах для ускорения процесса обучения распознаванию новых образов, содержащих все или часть этих признаков. Легко понять, что подобный процесс можно реализовать в многоступенчатых персептронах, т. е. в таких схемах, у которых сумматоры образов персептрона низшей ступени используются в качестве рецепторов для персептрона следующей ступени. При этом персептроны низших ступеней обучаются распознаванию отдельных свойств образов, а персептроны высших ступеней — распознаванию наборов этих свойств. Соответствующие изменения и усложнения законов поощрения можно выполнить многими различными способами. Заметим, что схема, заключающая в себе по существу идею двухступенчатого персептрона, применена в алгоритме для обучения распознаванию геометрических фигур, описанном в работе В. М. Глушкова, В. А. Ковалевского и В. И. Рыбака [29].

Введение перечисленных усовершенствований не позволяет, однако, приблизиться к моделированию еще одной важной особенности, присущей мозгу, а именно к установлению инвариантности всех образов по отношению к их движению и изменению размеров на основе *ограниченного опыта*, использующего только *небольшую часть* всех образов. Для достижения успеха и в этом направлении необходимо изменить не только конструкцию персептрона, но и методику самого процесса обучения. С этой целью для распознающего устройства вводится возможность вмешательства в организацию обучающей последовательности.

Если, например, распознающему устройству A показывается в качестве представителей того или иного образа несколько различных изображений, то устройство A должно обладать возможностью повторить демонстрацию этих изображений столько раз, чтобы обеспечить в дальнейшем их правильное распознавание. Более того, устройству должна быть предоставлена возможность повторения показа тех же изображений, подвергнутых таким из-

менениям, которым обычно подвергается изображение предмета на сетчатке глаза при изменениях взаимного положения глаза и рассматриваемого предмета.

Можно, разумеется, не вводить описанной обратной связи, позволяющей распознающему устройству изменять обучающую последовательность. Вместо этого сами обучающие последовательности нужно строить так, чтобы после показа того или иного изображения увеличивалась вероятность показа на следующем шаге того же самого изображения, рассматриваемого, быть может, лишь под другим ракурсом, либо, по крайней мере, изображений, принадлежащих тому же самому образу. Иными словами, при обучении распознающих устройств необходимо отказаться от построения процесса обучения по схеме независимых испытаний и перейти к более сложным схемам, описываемым марковскими цепями.

Предлагаемые изменения методов построения обучающих последовательностей намного улучшают функционирование распознающих устройств в режиме простого обучения. Для режима же самообучения эти изменения имеют принципиальное значение, поскольку лишь на таком пути можно надеяться, что классификация изображений, производимая самообучающимися устройствами, будет соответствовать исходной классификации, производимой человеком. Ясно, что описание процессов подобного рода требует гораздо более сложного математического аппарата, чем тот, который был использован в настоящем параграфе.

§ 7. Работа дискретных α-персептронов в режиме самообучения

В предыдущем параграфе было исследовано поведение дискретных α-персептронов в режиме *обучения*. Характерной чертой режима обучения является наличие учителя, которому известна правильная классификация изображений. В настоящем параграфе исследуются некоторые вопросы, связанные с поведением дискретных α-персептронов в режиме *самообучения*. В этом случае учитель отсутствует, а процессы самоорганизации, приводящие к изменению производимой персептроном классификации изображений, определяются введенной в схему персептрона положительной обратной связью.

Хорошо известно, что анализ поведения персептронов в режиме самообучения, сделанный Ф. Розенблаттом [69], весьма далек от какой бы то ни было математической строгости. Отсутствие строго доказанных предложений в этой области приводит к тому, что некоторые авторы приписывают (особенно в публикациях научно-популярного характера) самообучению персептронов многие свойства, которыми оно в действительности не обладает и не может обладать. На основании рассмотрений настоящего параграфа нетрудно сделать ряд выводов, очерчивающих границы

действительных возможностей, заложенных в самообучении персептронов.

Рассмотрим дискретный симметричный α-персептрон, рассчитанный на распознавание двух образов P и Q. Условимся в качестве единого выходного сигнала персептрона рассматривать разность выходных сигналов сумматоров P-го и Q-го образов

$$V_i(l) = U_i^P(l) - U_i^Q(l). \tag{99}$$

Здесь, как и в формулах предыдущего параграфа, индекс i пробегает все изображения (как P-го, так и Q-го) образа, l — любая последовательность изображений, показанная персептрону в процессе его самообучения.

Как и в случае обучения, нетрудно показать, что функционирование симметричного персептрона в режиме самообучения определяется суммой $a + b$ констант поощрения и штрафа, а не этими константами, рассматриваемыми отдельно. Имея в виду также возможность произвольного изменения масштабов, допустимо, не нарушая общности, предположить, что $a = 1$, а $b = 0$. В дальнейшем всегда будем исходить из этого допущения.

Обозначая $\|T_{ij}\|$ характеристическую матрицу персептрона и вспоминая определение α-закона поощрения, легко получим формулу

$$V_i(lj) = V_i(l) + T_{ij} \operatorname{sign} V_j(l). \tag{100}$$

Здесь символы lj обозначают последовательность изображений l, к которой присоединено изображение j.

Формула (100) справедлива для любой пары изображений i, j и для любой последовательности изображений l Функция $\operatorname{sign} x$, как обычно, для положительных значений x принимается равной $+1$, а для отрицательных значений x равной -1. Ясно, что в случае нулевого значения величины $V_j(l)$, по точному смыслу закона поощрения (положительной обратной связи), величина $\operatorname{sign} V_j(l)$ в формуле (100) должна быть не определена. Чтобы избежать неопределенности, в дальнейшем будем, по определению, считать нуль положительной величиной, так что $\operatorname{sign} 0 = +1$.

Имея в виду указанное видоизменение в определении функции $\operatorname{sign} x$, будем рассматривать формулу (100) как способ рекуррентного задания вектора $V(l) = (V_1(l), V_2(l), \ldots, V_m(l))$, определяющего выходные сигналы персептрона под действием любого изображения $j = 1, 2, \ldots, m$ после подачи на вход персептрона последовательности изображений l. Начальное значение этого вектора $V(0) = (V_1(0), V_2(0), \ldots, V_m(0))$ предполагается при этом заданным. Изображение причисляется персептроном к образу P или к образу Q в соответствии с тем, положительна или отрицательна соответствующая компонента $V_j(l)$ рассматриваемого вектора (напомним, что нуль, по принятому соглашению, считается положительным числом).

Поскольку все величины T_{ij} являются целыми (и к тому же

неотрицательными) числами, задача расчета персептрона в режиме самообучения сводится по существу к задаче случайного блуждания по дискретной решетке в пространстве с числом измерений $\leqslant m$. Предполагая, что показ изображений в процессе самообучения производится по схеме независимых испытаний, легко заметить, что вероятности переходов из любой точки такой решетки определяются лишь набором знаков координат этой точки.

Набор знаков координат любой точки решетки определяет, как нетрудно понять, также и классификацию изображений, производимую персептроном, который имеет в качестве вектора своих выходных сигналов радиус-вектор этой точки. С точки зрения теории персептрона представляет прежде всего интерес предельное распределение знаков координат вектора $V(l)$ при неограниченном увеличении длины обучающей последовательности l. Проведенные выше рассмотрения показывают, что требуемое распределение получается из предельного распределения для марковской цепи, соответствующей описанному выше блужданию по дискретной решетке.

Поскольку указанная цепь имеет бесконечное число состояний, нахождение предельного распределения в общем случае является достаточно сложным. Можно, однако, указать ряд случаев, когда нахождение предельного распределения легко сводится к исследованию марковской цепи с конечным числом состояний.

Рассмотрим в качестве примера дискретный симметричный α-персептрон A, рассчитанный на распознавание $2n$ изображений, первые n из которых принадлежат образу P, а последние n — образу Q. Пусть далее для элементов характеристической матрицы персептрона A имеют место соотношения $T_{ij} = a > 0$, если i и j принадлежат одному и тому же образу, и $T_{ij} = 0$, если i и j принадлежат различным образам. В виду теоремы 4 из § 6 настоящей главы, рассматриваемый персептрон обладает абсолютной способностью к экстраполяции и, следовательно, наилучшим образом ведет себя в режиме обучения (обучается правильному распознаванию в результате показа хотя бы одного изображения из каждого образа). Предположим, что начальными условиями будут условия $V_i(0) = b$ $(b>0)$ для $i = 1, 2, \ldots, m$ $(m \leqslant n)$ и $V_j(0) = -b$ для $j = m+1, m+2, \ldots, n, \ldots, 2n$.

Из формулы (100) непосредственно следует, что $V_i(l) < 0$ для любой последовательности l при всех $j = n+1, n+2, \ldots, 2n$. Остальные же компоненты будут выражаться формулами $V_i(l) = b + k\,a$ для $i = 1, 2, \ldots, m$ и $V_i(l) = -b + ka$ для $i = m+1, m+2, \ldots, n$, где k — разность между числом появления изображений, которым соответствуют положительные компоненты $V_i(l')$, и числом изображений, которым соответствуют отрицательные компоненты $V_i(l')$ (l' — соответствующая подпоследовательность последовательности l).

Предположим, что процесс самообучения совершается по схеме независимых испытаний с равными вероятностями появления всех изображений. Поскольку показ изображений одного образа в рас-

сматриваемом случае никак не влияет на распознавание изображений второго образа, можно, не нарушая общности, предполагать, что в процессе самообучения участвуют только изображения образа P (изображениям образа Q всегда отвечает отрицательный выходной сигнал независимо от того, включаются они в процесс самообучения или не включаются).

Допустим, что b не делится на a, и обозначим через t целое число $\left[\frac{a}{b}\right]+1$. Нетрудно понять, что для изучения функционирования персептрона A представляют интерес лишь те значения параметра k, которые заключены в замкнутом интервале $[-t,t]$. Действительно, если в процессе самообучения хотя бы раз величина k достигнет значения t, то в дальнейшем, в силу формулы (100), она может только возрастать, причем персептрон, начиная с этого момента, будет давать положительный выходной сигнал для всех изображений образа (что соответствует правильной классификации) Аналогично, если параметр k примет значение $-t$, персептрон будет давать отрицательный выходной сигнал для всех изображений (что фактически означает отсутствие какой-либо классификации изображений, поскольку все изображения причисляются персептроном к одному и тому же образу).

Теперь, как легко видеть, предельное поведение персептрона A определяется марковской цепью с $2t+1$ состояниями $k=-t$, $-t+1$, ..., $-1,0,1$, ..., $t-1$, t. В силу принятого предположения о вероятностях появления изображений в процессе самообучения, для любого k, отличного от t или $-t$, вероятность перехода в состояние $k+1$, равна $\frac{m}{n}$, а вероятность перехода в состояние $k-1$ равна $\frac{n-m}{n}$. Из состояния t (как и из состояния $-t$) возможен переход только в это же самое состояние, поскольку с точки зрения функционирования персептрона все состояния при $k>t$ (соответственно—при $k<-t$) не отличаются от состояния $k=t$ (соответственно —от состояния $k=-t$).

Вводя обозначения $p=\frac{m}{n}$ и $q=\frac{n-m}{n}$, получим для рассматриваемой марковской цепи матрицу переходных вероятностей

$$M=\begin{Vmatrix} 1 & 0 & 0 & 0 & . & . & . & 0 & 0 & 0 \\ p & 0 & q & 0 & . & . & . & 0 & 0 & 0 \\ 0 & p & 0 & q & . & . & . & 0 & 0 & 0 \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . \\ 0 & 0 & 0 & 0 & . & . & . & 0 & q & 0 \\ 0 & 0 & 0 & 0 & . & . & . & p & 0 & q \\ 0 & 0 & 0 & 0 & . & . & . & 0 & 0 & 1 \end{Vmatrix}$$

Эта матрица в качестве своего двукратного характеристического корня имеет единицу. Вероятности перехода цепи в состояния t и $-t$ равны предельным переходным вероятностям $p^{\infty}_{t+1,1}$ и $p^{\infty}_{t+1,2t+1}$. Для вероятности $p^{\infty}_{t+1,i}$ по формуле Перрона получаем выражение

$$p^{\infty}_{t+1,i} = \lim_{s\to\infty} \frac{d}{d\lambda} \frac{\lambda^s M_{i,t+1}(\lambda)}{\psi_1(\lambda)}\Bigg|_{\lambda=1}. \qquad (101)$$

Легко видеть, что $M_{i,t+1}(\lambda)$ для i, отличного от 1 и от $2t+1$, делится на $(\lambda-1)^2$, и потому для этих значений i $p^{\infty}_{t+1,i}=0$. Для $i=1$ $M_{1,t+1}(\lambda)=(\lambda-1)M_1(\lambda)$, а для $i=2t+1$ $M_{2t+1,t+1}(\lambda)=(\lambda-1)M_2(\lambda)$, где $M_1(\lambda)=p^tQ(\lambda)$, $M_2(\lambda)=q^tR(\lambda)$.

Из формулы (101) легко получим $P^{\infty}_{t+1,t}=cp^t$, $p^{\infty}_{t+1,2t+1}=cq^t$.

Поскольку все остальные предельные переходные вероятности в $(t+1)$-ой строке равны нулю, из условия стохастичности матрицы предельных переходных вероятностей находим значение c: $c=\frac{1}{p^t+q^t}$. Тем самым нами доказано следующее предложение.

При неограниченном продолжении процесса самообучения описанный выше персептрон A с вероятностью $\frac{p^t}{p^t+q^t}$ устанавливает правильную классификацию изображений и с вероятностью $\frac{q^t}{p^t+q^t}$ относит все изображения к одному и тому же образу.

Рассмотренный пример, как легко заметит внимательный читатель, строго говоря, не может быть осуществлен в реальном персептроне, исключая тривиальные случаи $m=n$, $p=1$, $q=0$ и $m=0$, $p=0$, $q=1$. Причина заключается в том, что при сделанных предположениях относительно характеристической матрицы все изображения одного и того же образа возбуждают одно и то же множество нейронов. Поэтому выходные сигналы, индуцируемые изображениями одного и того же образа, всегда, в том числе и в начальный момент, должны быть равны между собой.

Нетрудно, однако, заметить, что, полагая $T_{ii}=a+\delta$ для всех $i=1, 2, \ldots, 2n$ $(\delta>0)$, получаем возможность удовлетворить введенным в примере начальным условиям. Вместе с тем, если δ существенно меньше, чем a, а t относительно невелико, то описанное в примере поведение персептрона будет служить хорошим приближением для его истинного поведения.

Рассмотрим полный дискретный α-персептрон B с (1, 1, 1)-нейронами, с квадратной $(n\times n)$-сетчаткой, рассчитанный на распознавание двух образов P и Q. Образ P состоит из n горизонтальных линий, а образ Q— из n вертикальных линий. Каждая из этих линий составляет отдельное изображение. В предыдущем параграфе отмечалось, что персептрон B можно рассматривать как наиболее характерный представитель класса персептронов со случайными

связями нейронов с сетчаткой. Согласно теореме 1 и следующему за ней корollарию из работы Розенблатта [69], такие персептроны, построенные на непрерывных нейронах, с вероятностью, сколь угодно близкой к единице, при самообучении должны стремиться к состоянию, в котором все изображения относятся к одному и тому же образу. Покажем, что для персептрона B подобное утверждение неверно.

Легко видеть, что для персептрона B можно выбрать любые начальные условия. *Нижней границей модулей начальных* условий будем называть наименьшее из чисел $|V(0)|$ $(i = 1, 2, \ldots, 2n)$. При принятых предположениях справедлива следующая теорема.

Теорема 1. *Для любого сколь угодно малого положительного числа ε найдется такое число S, что в случае, когда нижняя граница модулей начальных условий превосходит S, персептрон B в режиме самообучения (с равновероятным появлением всех изображений) с вероятностью $p>1-\varepsilon$ сохраняет начальную классификацию изображений.*

Доказательство. Обозначим буквой N длину обучающей последовательности l а ν_i — число появлений i-го изображения $(i = 1, 2, \ldots, 2n)$ в этой последовательности. Пусть $V_i(0) = x_i$ $(i = 1, 2, \ldots, 2n)$; k_i — множество всех индексов j (изображений), относящихся к противоположному по сравнению с i образу и таких, что знак x_j совпадает со знаком x_i; z_i — множество всех индексов j, относящихся к противоположному по сравнению с i образу и таких, что знак x_j противоположен знаку x_i (нуль, как и раньше, считается при этом положительным числом).

Как было показано в предыдущем параграфе, произвольный элемент T_{ij} характеристической матрицы персептрона B равняется $n^2(n-1)$, 0 или $(n-1)^2$ в зависимости от того, индексы i и j совпадают или не совпадают, но относятся к одному и тому же образу, или не совпадают и относятся к различным образам. Используя это обстоятельство, с помощью формулы (100) легко узнаем, что исходная классификация изображений сохранится в процессе самообучения, если для всех $N = 1, 2, \ldots$ будут выполняться неравенства

$$n^2(n-1)\nu_i + (n-1)^2\left(\sum_{j\in k_i}\nu_j - \sum_{j\in z_i}\nu_j\right) + |x_i| > 0 \quad (i = 1, 2, \ldots, 2n),$$

и тем более, если будут выполнены неравенства

$$n^2(n-1)\nu_i - (n-1)^2\sum_{j\in k_j\cup z_j}\nu_j + x > 0 \quad (i = 1, 2, \ldots, 2n), \quad (102)$$

где x — минимальное из чисел $|x_i|$ $(i = 1, 2, \ldots, 2n)$. В свою очередь, нетрудно проверить, что неравенства (102) выполняются, если выполнены неравенства

$$\left|\frac{\nu_i}{N} - \frac{1}{2n}\right| < \frac{1}{4n^2}\left(1 + \frac{2x}{N(n-1)}\right) \quad (i = 1, 2, \ldots, 2n). \quad (103)$$

При обозначении величины $\frac{1}{4n^2}$ буквой δ, очевидно, неравенства (103) будут заведомо выполнены, если выполняются неравенства

$$\left|\frac{\nu_i}{N}-\frac{1}{2n}\right|<\delta \qquad (i=1,2,\ldots,2n). \tag{104}$$

В силу теоремы 4 из § 3 настоящей главы, существуют положительные константы a и b, не зависящие от N и такие, что вероятность P_N невыполнения хотя бы одного из неравенств (104) при любом фиксированном значении N имеет оценку $P_N<\frac{a}{N^{n-\frac{1}{2}}}e^{-bN}$. Вероятность невыполнения хотя бы одного из неравенств (104) для значений N от M до ∞ не превосходит суммы ряда $\sum_{N=M}^{\infty}\frac{a}{N^{n-\frac{1}{2}}}e^{-bN}$, а эта сумма, очевидно, меньше чем $R(M)=\frac{a}{M^{n-\frac{1}{2}}}\cdot\frac{e^{-bM}}{1-e^{-b}}$. При $M\to\infty$ величина $R(M)$ стремится к нулю. Выберем M так, чтобы $R(M)<\varepsilon$.

Выбирая теперь $S=2(M-1)n^2(n-1)$, получим, что при $x>S$ неравенства(103) выполнены для всех значений $N=1, 2, \ldots, M-1$. В силу же выбора M, для всех остальных значений N неравенства (103) выполняются с вероятностью, большей $1-\varepsilon$. Поскольку выполнение неравенств (104) для всех значений N от 1 до $<\infty$ означает сохранение исходной классификации изображений, то теорема доказана.

Теорема 1 показывает, что при достаточно больших начальных значениях выходных сигналов для всех изображений рассмотренный персептрон фактически почти лишен способности не только самообучаться, но даже просто изменять изначально задаваемую ему классификацию изображений. Из доказательства теоремы легко видеть, что остающаяся при этом слабая способность к самоизменению имеет наибольшую величину в случае правильной исходной классификации. Иными словами, персептрон наименее склонен к сохранению именно правильного способа функционирования.

Рассмотрим β-закон поощрения. С этой целью фиксируем произвольное число β, заключенное между нулем и единицей, и рассмотрим произвольный симметричный персептрон C с β-законом поощрения, характеристическая матрица которого диагональна, т. е., иными словами, имеет отличные от нуля элементы только на главной диагонали. Как показано в предыдущем параграфе, таким свойством обладает симметричный персептрон C_1 с (1, 1, 1)-нейронами, который рассчитан на распознавание горизонтальных и вертикальных линий и у которого входы каждого нейрона

подсоединяются к элементам сетчатки, расположенным на одной горизонтали или на одной вертикали.

В случае β-закона поощрения основное рекуррентное соотношение для определения выходных сигналов запишется

$$V_i(lj) = (1 - \beta)(V_i(l) + T_{ij}\operatorname{sign} V_j(l)). \qquad (105)$$

Обозначения здесь точно такие же, как и в формуле (100), причем это соотношение также справедливо для произвольных дискретных симметричных персептронов. В случае персептронов с диагональной характеристической матрицей оба слагаемых в правой части формул (100) и (105) имеют всегда один и тот же знак (случай, когда второе слагаемое равно нулю, исключаем из рассмотрения). Отсюда непосредственно вытекает справедливость следующего предложения.

Теорема 2. *Дискретный симметричный персептрон С с диагональной характеристической матрицей полностью лишен способности к самообучению (т. е. сохраняет неизменной любую задаваемую исходную классификацию изображений) как в случае α-закона поощрения, так и в случае β-закона поощрения.*

Легко видеть также справедливость следующего предложения.

Теорема 3. *Никакой дискретный симметричный персептрон (как с α-, так и с β-законом поощрения), работая в режиме самообучения, не может изменить исходной классификации изображений, если эта классификация относит все изображения к одному и тому же образу.*

Полученные нами результаты можно рассматривать как контрпримеры к результатам Розенблатта [69] в той мере, в какой примененные им рассуждения относятся не только к непрерывным, но и к дискретным нейронам. Во всяком случае эти результаты свидетельствуют о том, что асимптотическое поведение персептронов в режиме самообучения является гораздо более сложным и требует значительно более тонких приемов для своего изучения по сравнению с приемами чисто качественного характера, употребляемыми Розенблаттом [69].

Для наглядного представления особенностей поведения персептронов в режиме самообучения по сравнению с режимом обучения рассмотрим случай, когда число изображений равно двум (каждый образ состоит из одного-единственного изображения). Этот случай допускает простую графическую интерпретацию.

Предварительно заметим, что в случае наличия двух образов (но при произвольном числе изображений) функционирование персептрона в режиме обучения, как и в режиме самообучения, удобно характеризовать вектором с компонентами $V_i(l)$ (см. выше). Основное рекуррентное соотношение для этих компонент будет иметь, очевидно, вид

$$V_i(lj) = V_i(l) \pm T_{ij}. \qquad (106)$$

Это соотношение справедливо для любой пары изображений i, j и для любой обучающей последовательности l. Второе слагаемое в правой части берется со знаком плюс, если изображению j в правильной классификации относится положительный выходной сигнал, и со знаком минус, если соответствующий выходной сигнал должен быть отрицательным.

Рассмотрим дискретный симметричный персептрон с α-законом поощрения, характеристической матрицей которого служит матрица $T=\begin{Vmatrix} a & b \\ b & a \end{Vmatrix}$, где $a>b>0$. Предположим, что при правильной классификации первое изображение должно индуцировать положительный выходной сигнал, а второе изображение — отрицательный выходной сигнал. Откладывая по горизонтальной оси координату $V_1(l)$, а по вертикальной оси координату $V_2(l)$, сопоставим каждому вектору $(V_1(l), V_2(l))$ некоторую точку плоскости. Выбирая по одной точке в каждом квадранте, получим наглядное представление действия формулы (106) (рис. 13).

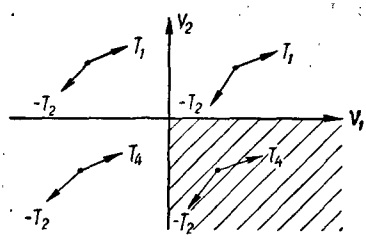

Рис. 13.

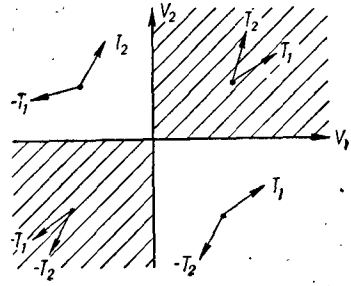

Рис. 14.

На рис. 13 буквой T_1 обозначен вектор (a, b), а T_2—вектор (b, a); характерной особенностью режима обучения является то, что направления векторов (определяющих случайные блуждания точки на решетке) не зависят от расположения точек на плоскости. Равнодействующая этих векторов всегда направлена в сторону того квадранта, в котором знаки координат (выходных сигналов персептрона) соответствуют правильной классификации (в данном случае таким квадрантом является заштрихованный — четвертый квадрант).

Для случая режима самообучения интерпретация соответствующей формулы (100) представлена на рис. 14. В отличие от предыдущего случая, направления векторов, определяющих случайное блуждание, различны в различных квадрантах. Обозначения векторов такие же, как и на рис 13.

Нетрудно заметить качественные отличия ситуации, изображенной на рис. 14, от ситуации, изображенной на рис. 13. Прежде всего первый и третий квадранты (на рис. 14 они заштрихованы)

теперь обладают ловушечным свойством; точка, попавшая в процессе случайного блуждания в один из этих квадрантов, уже не может никогда выбраться из него.

Попадание в эти квадранты означает фактически отсутствие какой-либо классификации (оба изображения причисляются к одному и тому же образу). Вместе с тем из квадранта, соответствующего правильной классификации (четвертый квадрант), как и из квадранта, соответствующего правильной классификации с точностью до знака выходного сигнала (третий квадрант), всегда имеется ненулевая вероятность выхода в соседние квадранты.

Рассматривая возникшую ситуацию в чисто качественном плане, подобно тому, как это делает Розенблатт [69], мы должны были бы прийти к выводу, что изучаемый нами персептрон асимптотически стремится к состоянию, в котором на все изображения даются выходные сигналы одного и того же знака (отсутствие какой-либо классификации). Более точное рассмотрение (повторяющее выкладки, произведенные при доказательстве теоремы 1) приводит, однако, к совершенно другому выводу: как и в случае теоремы 1, при достаточном удалении начальной точки от границ квадранта вероятность продолжения случайного блуждания без выхода из этого квадранта во все последующие моменты времени (вплоть до бесконечности) может быть сделана сколько угодно близкой к единице.

Тем самым еще раз подтверждается опасность, возникающая в том случае, когда общие выводы об асимптотическом поведении персептронов в режиме самообучения основываются на рассуждениях чисто качественного характера, не подтвержденных точными расчетами и оценками.

§ 8. Логические классификационные системы и машины условной вероятности

Системы для распознавания образов, рассмотренные в предыдущих параграфах, представляют собой устройства для классификации некоторых подмножеств в пространстве изображений. Ориентируясь на зрительные, звуковые и другие непрерывные по своей природе образы, мы переносили в той или иной мере свойство непрерывности и на соответствующие классификационные системы. Действительно, даже в случае заведомо дискретных рецепторов гипотеза N-экстраполируемости образов, отдавая дань непрерывности образов, позволяла отбирать для классификации только некоторые, в соответствующем смысле «хорошо устроенные» множества. То же самое неявное использование свойства непрерывности образов имеется и в персептронах (в том числе и дискретных), а также и во всех других упоминавшихся в предыдущих параграфах алгоритмах и устройствах для распознавания образов.

Ограничение числа подлежащих рассмотрению и классификации подмножеств пространства изображений в случае зрительных,

звуковых и других непрерывных по своей природе образов имеет принципиальное значение, ибо без такого ограничения задача распознавания этих образов была бы практически не разрешимой.

Действительно, в случае, когда сетчатка состоит из n двоичных рецепторов, пространство изображений состоит из $I(n) = 2^n$ различных изображений и насчитывает $Q(n) = 2^{2^n}$ различных подмножеств — возможных дискретных образов. Вторая из этих величин уже при $n = 5$ превышает четыре миллиарда, а первая при таком относительно малом числе рецепторов, как 100, изображается единицей с тридцатью нулями!

В силу приведенных соображений становится ясным, что задача построения устройств, хранящих (или вырабатывающих) признаки всех возможных образов, для сколько-нибудь больших значений n практически не разрешима. Вместе с тем в случае, когда число (двоичных) рецепторов не превышает одного-полутора десятков, практически вполне возможно построить машину, способную запоминать и производить различные операции со всеми изображениями (но не со всеми образами), которые можно воспринять с помощью соответствующей сетчатки.

Машины, классифицирующие всевозможные изображения, которые можно получать от двоичных рецепторов, будем называть *логическими классификационными машинами*. Опишем одну из возможных схем подобных машин, предложенную Аттли [2].

Для каждого *свойства* изображения классификационная машина Аттли содержит так называемый распознающий элемент, возбуждающийся под действием этого свойства. Под свойством изображения здесь и далее подразумевается наличие у некоторого множества M рецепторов (зависящего от выбора соответствующего свойства) определенной комбинации выходных сигналов (нулевых и единичных). Рецепторы, не входящие в множество M, могут иметь при этом любые выходные сигналы. Свойство, заключающееся в том, что рецепторы с номерами $i_1, i_2, \ldots, i_m$ имеют единичный выходной сигнал, а рецепторы с номерами $j_1, j_2, \ldots, j_n$ — нулевой сигнал, условимся обозначать $(i_1, i_2, \ldots, i_m; \bar{j}_1, \bar{j}_2, \ldots, \bar{j}_n)$.

Из приведенных определений становится ясным, что задание свойства эквивалентно приписыванию выходному сигналу рецептора сетчатки одного из трех значений: единица, нуль, безразлично. Если общее число составляющих сетчатку рецепторов обозначить через N, то, как легко понять, при общем числе изображений, равном 2^N, число различных их свойств будет равно 3^N. Все свойства любого данного изображения могут быть получены с помощью замещения какого-либо числа составляющих это изображение сигналов (нулевых и единичных) символами безразличия. Число таких замещений (а значит, и число свойств каждого изображения) будет, очевидно, равно сумме $C_N^0 + C_N^1 + \ldots + C_N^N = 2^N$. Таким образом, каждое изображение вызывает возбуждение 2^N элементов классификационной машины.

Назовем i-ым *элементарным свойством* свойство изображения возбуждать i-й рецептор, а *позитивным свойством* — любое объединение элементарных свойств. При N двоичных рецепторах имеется всего лишь N элементарных свойств. Ясно также, что задание любого позитивного свойства эквивалентно заданию некоторого подмножества рецепторов, индуцирующих единичные выходные сигналы. Общее число позитивных свойств равно, таким образом, числу подмножеств множества из N элементов, т. е. 2^N.

Описанную выше классификационную машину, способную различать любые свойства изображений, Аттли называет *бинарной классификационной машиной*, в отличие от так называемой *унитарной классификационной машины*, способной различать лишь позитивные свойства. Легко видеть, что для всякой бинарной машины существует эквивалентная ей (по производимой классификации) унитарная машина, содержащая удвоенное число рецепторов. Достаточно к каждому рецептору, реагирующему на то или иное элементарное свойство, добавить рецептор, реагирующий на отсутствие этого свойства. Хотя, на первый взгляд, после этого понадобится 4^N распознающих элементов, однако в действительности многие из них окажутся лишними, поскольку никогда не будут возбуждаться. После устранения лишних элементов число оставшихся элементов будет в точности то же самое, что и в случае бинарной машины, т. е. 3^N. Простота сведения бинарных машин к унитарным позволяет нам в дальнейшем ограничиться рассмотрением одних лишь унитарных машин.

Распознающие элементы унитарной классификационной машины с N рецепторами удобно представлять себе в виде нейронов, имеющих от одного до N входных каналов и способных возбуждаться лишь в случае одновременного возбуждения всех своих входных каналов. Каждый из входных каналов нейрона подсоединяется к выходному каналу какого-либо рецептора. Нейрон с входами, подсоединенными к рецепторам с номерами $i_1, i_2, \ldots, i_k$, будет соответствовать позитивному свойству $(i_1, i_2, \ldots, i_k)$ и возбуждаться лишь при наличии этого свойства в распознаваемом изображении. Для того чтобы общее число нейронов равнялось в точности 2^N, необходимо предполагать наличие еще одного нейрона без входных каналов, возбуждающегося постоянно вне зависимости от распознаваемого изображения. Этот нейрон соответствует свойству, являющемуся комбинацией пустого множества элементарных свойств.

Рассмотрим произвольный рецептор i и все позитивные свойства, содержащие элементарное свойство i. Среди этих свойств имеется точно одно свойство, содержащее одно элементарное свойство (в данном случае это будет само свойство i), точно $C^1_{N-1} = N-1$ свойств, содержащих по два элементарных свойства (все свойства вида (i, j), где $j \neq i$), точно C^2_{N-1} свойств, содержащих по три элементарных свойства, и т.д. В унитарной машине каждому из пози-

тивных свойств соответствует один нейрон. Общее число нейронов, подсоединенных к рецептору i, выразится суммой $1 + C^1_{N-1} + C^2_{N-1} + \ldots + C^{N-1}_{N-1} = 2^{N-1}$, что составляет в точности половину всех нейронов в унитарной машине.

Производя случайную выборку нейронов при условии, что все нейроны считаются равновероятными, придем к заключению: вероятность того, что выбранный таким образом нейрон будет подсоединен к любому данному рецептору i, равна $\frac{1}{2}$. Таким образом, схема соединений, в какой-то мере близкая к схеме унитарной классификационной машины, может быть получена в результате случайного подсоединения нейронов к рецептору с равной вероятностью подсоединения или неподсоединения хотя бы одного из входных каналов заданного нейрона к любому рецептору.

В описанной схеме классификационной машины (безразлично, бинарной или унитарной) полностью отсутствуют какие-либо элементы самоорганизации или самосовершенствования. Поэтому, следуя Аттли, введем изменения и дополнения в схему унитарной машины, после чего эта машина превратится в так называемую *машину условной вероятности*.

Для упрощения обозначений условимся произвольные позитивные свойства изображений обозначать большими латинскими буквами. Если I и J — позитивные свойства, то через $I \cup J$ будем обозначать объединение этих свойств, т. е. позитивное свойство, состоящее из всех элементарных свойств, входящих либо в свойство I, либо в свойство J, либо в оба эти свойства сразу. Через $I \cap J$ обозначается *пересечение* свойств I и J, т. е., иными словами, позитивное свойство, состоящее из всех тех и только тех элементарных свойств, которые входят одновременно как в свойство I, так и в свойство J.

Пусть теперь на вход некоторой унитарной машины подается какая-либо *обучающая последовательность*, т. е., попросту говоря, какая-либо последовательность изображений. Вообще говоря, не все члены этой последовательности обладают некоторым фиксированным свойством I. Поэтому нейрон, соответствующий свойству I, возбуждается одними членами обучающей последовательности и не возбуждается другими ее членами. Отношение числа членов обучающей последовательности, обладающих свойством I и, следовательно, возбуждающих указанный нейрон, к общему числу членов этой последовательности естественно называть *частотой свойства* применительно к заданной обучающей последовательности (которую будем также называть историей обучения). Для более четкого различия с вводящейся ниже условной частотой только что определенную частоту принято называть *безусловной частотой*.

Обозначим безусловную частоту свойства I через $p(I)$, при этом история обучения предполагается фиксированной.

Возложим на нейроны унитарной классификационной машины дополнительную функцию вычисления безусловной частоты появления соответствующих им свойств. Если изображение, подающееся на вход машины, обладает некоторым свойством I, то соответствующий этому свойству нейрон, вычисляя и запоминая безусловную частоту свойства I, выдает в данный момент выходной сигнал, равный единице. Если же этот нейрон не возбужден (т. е. если очередное изображение не обладает свойством I), то его выходным сигналом будет хранящееся в нем значение безусловной частоты свойства I.

С указанными дополнениями и изменениями унитарная машина приобретает уже некоторые черты, свойственные, если не самоорганизующимся, то, во всяком случае, самоизменяющимся автоматам. Дальнейшее усовершенствование включает в себя подсчет так называемых *условных частот*, классифицируемых унитарной машиной свойств.

Условная частота $p(I/J)$ свойства I по отношению к свойству J представляет собой отношение числа случаев совместного появления свойств I и J (т. е., иначе говоря, появления свойства $I \cup J$) к общему числу случаев появления свойства J

$$p\ (I/J) = \frac{p(I \cup J)}{p\ (J)}. \qquad (107)$$

Будем по-прежнему считать, что нейроны унитарной машины подсчитывают и запоминают безусловные частоты соответствующих им свойств. Как и раньше, в случае *прямого* возбуждения нейрона (т. е. в случае наличия у очередного изображения соответствующего этому нейрону свойства) нейрон выдает сигнал, равный единице. Однако все нейроны, не подвергшиеся прямому возбуждению, должны выдавать теперь не безусловные, а условные частоты соответствующих им свойств. Вопрос состоит лишь в том, по отношению к какому свойству нужно вычислять указанные условные частоты. Легко понять, что в качестве свойства J наиболее естественно выбирать *максимальное позитивное свойство* рассматриваемого изображения, представляющее собой объединение всех элементарных свойств, которыми обладает данное изображение. Действительно, только самое максимальное свойство полностью определяет соответствующее ему изображение, так что условные частоты будут фактически отнесены к частоте появления этого изображения.

Введем понятие *подчинения* для нейронов унитарной машины. *Говорят, что нейрон A подчинен нейрону B, если свойство J, соответствующее нейрону B, включает в себя все элементарные свойства из свойства I, соответствующего нейрону A.*

Нейрон Q, соответствующий максимальному позитивному свойству какого-либо фиксированного изображения, характеризуется, очевидно, тем, что ему подчинены все нейроны, прямо возбу-

ждающиеся этим изображением, а он не подчинен ни одному из этих нейронов, кроме, разумеется, самого себя. Все нейроны, которым подчинен нейрон Q, составляют так называемое *супермножество* $M(Q)$ этого нейрона. В рассматриваемом случае ни один нейрон P из множества $M(Q)$, кроме самого Q, не является прямо возбужденным и должен поэтому выдавать сигнал, равный условной частоте $p(I/J)$ свойства I, соответствующего нейрону P, по отношению к свойству J, соответствующего нейрону Q. Из определения супермножества непосредственно следует, что $I \cup J = I$. Поэтому, в силу формулы (107),

$$p(I/J) = \frac{p(I)}{p(J)}. \tag{108}$$

Таким образом, для всех нейронов из супермножества $M(Q)$ нейрона Q выходные сигналы могут быть определены по формуле (108): для получения выходного сигнала нейрона P из $M(Q)$ хранящееся в нем значение безусловной частоты соответствующего ему свойства нужно поделить на значение безусловной частоты, хранящееся в нейроне Q. Указанную операцию для получения выходных сигналов нейронов из супермножества $M(Q)$ Аттли называет *операцией суперконтроля*. Операция суперконтроля не приводит к противоречию и для самого нейрона Q, поскольку в этом случае выходной сигнал, вычисленный по формуле (108), будет, очевидно, равен единице, что совпадает с истинным значением выходного сигнала нейрона Q, получаемым из условия прямого возбуждения этого нейрона.

Множество всех нейронов, подчиненных любому данному нейрону P, назовем *субмножеством* этого нейрона и обозначим $L(P)$. Используя понятие субмножества, нетрудно определить способ получения выходных сигналов для всех нейронов, которые не подвержены в рассматриваемый момент прямому возбуждению и не входят в супермножество нейрона Q, соответствующего максимальному позитивному свойству рассматриваемого на данном шаге изображения.

Обозначим множество всех таких нейронов через K, и пусть R — произвольный нейрон из K. Если по-прежнему J означает свойство, соответствующее нейрону Q, I — свойство, соответствующее нейрону R, то нейрон P, соответствующий свойству $I \cup J$, будет, очевидно, принадлежать супермножеству $M(Q)$.

В силу формул (107) и (108), выходной сигнал нейрона R равен выходному сигналу нейрона P. Вместе с тем ясно, что нейрон R входит в субмножество $L(P)$ нейрона P. Напрашивается поэтому вывод, что все не определенные до сих пор выходные сигналы (для нейронов из множества K) могут быть получены за счет простой передачи выходных сигналов нейронов из супермножества $M(Q)$ нейронам соответствующих им субмножеств. Такую передачу выходных сигналов на субмножества естественно назвать, по аналогии с суперконтролем, *операцией субконтроля*.

Однако определенная таким образом операция субконтроля не приводит к однозначному определению выходных сигналов, поскольку каждый нейрон R из K входит не в одно, а, вообще говоря, в несколько субмножеств различных нейронов из $M(Q)$. Для устранения возникшей неоднозначности заметим, что среди подмножества H всех нейронов из $M(Q)$, которым подчинен нейрон R, нужный нам нейрон P (соответствующий объединению свойств I и J, как они были определены выше) будет подчинен всем остальным нейронам этого подмножества. Легко видеть, что свойство $I \cup J$ будет иметь в таком случае наибольшую безусловную частоту среди свойств, соответствующих всем нейронам из H. В силу формулы (108), это означает, что выходной сигнал нейрона P является наибольшим среди выходных сигналов всех нейронов из подмножества H.

Таким образом, для обеспечения безошибочного функционирования машины операция субконтроля должна быть дополнена еще одним правилом: *если в силу операции субконтроля на какой-либо нейрон передается несколько различных выходных сигналов, то среди них должен быть выбран наибольший.*

Подчеркнем также еще раз, что операция субконтроля не применяется к нейронам, выходные сигналы которых определяются из условия прямого возбуждения или на основе применения операции суперконтроля.

Унитарная машина со всеми описанными дополнениями и изменениями в законах функционирования нейронов называется *машиной условной вероятности.* Это название оттеняет тот факт, что подсчитываемые машиной условные частоты при достаточно длительных историях обучения, которые будем предполагать обычно проводимыми по схеме независимых испытаний, со сколь угодно высокой степенью достоверности стремятся к соответствующим условным вероятностям одних свойств по отношению к другим. На машине условной вероятности моделируются процессы образования и угасания так называемых *условных рефлексов.* Предположим, что на протяжении всей истории обучения машины свойство J появлялось почти всегда вместе с другим свойством I. В таком случае условная частота $p(J/I)$ свойства J по отношению к свойству I будет близка к единице. Если теперь свойство I появится без свойства J, то реакция (выходной сигнал) нейрона Q, соответствующего свойству J, по своей силе будет мало отличаться от реакции нейрона P, соответствующего свойству I и подвергающегося, следовательно, прямому возбуждению со стороны этого свойства. Будем говорить в таком случае, что *в машине выработался условный рефлекс для свойства J по отношению к свойству I.*

Если после выработки указанного рефлекса он не будет подкрепляться на протяжении достаточно большого числа шагов в последующей истории обучения (т. е. свойство I будет появляться без свойства J), то условная частота $p(J/I)$ будет уменьшаться и может с течением времени стать пренебрежимо малой величиной. При

очередном появлении свойства I без свойства J реакция нейрона Q будет очень малой. Будем говорить в таком случае об *угасании* соответствующего рефлекса.

Описанные процессы возникновения и угасания условных рефлексов, по крайней мере с чисто внешней стороны, весьма напоминают аналогичные процессы, протекающие в живых организмах, в частности в нервной системе человека. Вместе с тем имеется и ряд отличий. Одно из существенных отличий состоит в том, что в описанной нами схеме скорость возникновения и угасания условных рефлексов в самом начале процесса обучения слишком велика, а в конце процесса обучения — слишком мала. Исправить это положение можно, заменив безусловные и условные частоты так называемыми *псевдочастотами*.

Безусловной псевдочастотой любого данного свойства I условимся называть заключенную между нулем и единицей величину r, увеличивающуюся при появлении изображений со свойством I и уменьшающуюся при появлении изображений, не обладающих свойством I. Величина r должна стремиться к единице, если, начиная с некоторого момента, все члены обучающей последовательности обладают свойством I, и — к нулю, если все члены обучающей последовательности, начиная с некоторого момента, не обладают свойством I (предполагаем здесь, что обучающая последовательность может пополняться все новыми и новыми членами в течение сколь угодно большого промежутка времени).

Приведенному определению удовлетворяет, в частности, безусловная частота, которая может поэтому рассматриваться как один из возможных способов задания безусловной псевдочастоты. Более просто и удобно, однако, рассматривать в качестве безусловной псевдочастоты некоторого свойства величину $r_n = r_n(I)$, задаваемую рекуррентными соотношениями

$$r_{n+1} = \begin{cases} r_n + \alpha(1 - r_n) \\ r_n - \beta r_n \end{cases} \qquad (n = 0, 1, 2, \ldots), \tag{109}$$

где величины α и β — положительные константы строго меньше единицы. Если на $(n+1)$-ом шаге обучения появляется свойство I, то пользуются верхней строкой указанных соотношений, в противном случае — нижней строкой. Начальное значение r_0 величины r_n может быть выбрано на интервале $(0,1)$ произвольно.

Если теперь в описанной выше машине условной вероятности заменить безусловные частоты свойств псевдочастотами, вычисляемыми с помощью формул (109), а способ функционирования нейронов оставить прежним, то, выбирая должным образом величины α и β, можно гораздо больше приблизиться к имитации биологических процессов возникновения и угасания условных рефлексов, чем в случае исходной машины условной вероятности. Условные частоты свойств будут по-прежнему вычисляться по формуле (107), однако вместо безусловных частот в правой части этой фор-

мулы будут фигурировать не безусловные частоты, а безусловные псевдочастоты. Поэтому и формула (107) дает теперь фактически не условную частоту, а некоторую величину, которую естественно назвать *условной псевдочастотой* одного свойства по отношению к другому.

Можно, впрочем, определив несколько иным способом понятие условной псевдочастоты, так усовершенствовать машину условной вероятности, что в ней сразу будут определяться условные псевдочастоты свойств без предварительного вычисления и запоминания их безусловных частот или псевдочастот. С этой целью введем в унитарной классификационной машине парные направленные связи между нейронами. Каждой такой связи приписывается некоторый *вес*, могущий принимать любые вещественные значения на интервале (0,1). Веса эти могут изменяться на каждом шаге обучения. Вес связи между нейронами P и Q на n-ом шаге обучения будем обозначать $\lambda_n(P, Q)$. Заметим, что веса $\lambda_n(P, Q)$ и $\lambda_n(Q, P)$ не обязательно равны между собой.

Закон изменения веса связи задается следующими соотношениями, определенными для всех значений $n = 0, 1, 2 \ldots$:

$$\left.\begin{array}{l} \lambda_{n+1}(P,Q) = \lambda_n(P,Q), \\ \text{если на } (n+1)\text{-ом шаге обучения нейрон } P \text{ не возбужден;} \\ \lambda_{n+1}(P,Q) = \lambda_n(P,Q) + \alpha(1 - \lambda_n(P,Q)), \\ \text{если на } (n+1)\text{-ом шаге обучения оба нейрона } P \text{ и } Q \text{ возбуждены;} \\ \lambda_{n+1}(P,Q) = \lambda_n(P,Q) - \beta\lambda_n(P,Q), \\ \text{если на } (n+1)\text{-ом шаге обучения нейрон } P \text{ возбужден, а нейрон } Q \text{ не возбужден.} \end{array}\right\} \quad (110)$$

Здесь, как и в формулах (109), α и β представляют собой положительные константы строго меньше единицы, а начальный вес $\lambda_0(P, Q)$ может быть выбран произвольно на интервале (0,1).

В чисто качественном отношении вес $\lambda_n(P, Q)$ ведет себя точно так же, как условная частота $p_n(J/I)$ свойства J, соответствующего нейрону Q, по отношению к свойству I, соответствующему нейрону P. Действительно, при одновременном появлении свойств I и J происходит увеличение как величины $\lambda_n(P, Q)$, так и величины $p_n(J/I)$. Обе эти величины уменьшаются при появлении свойства I без одновременного появления свойства J. Если первая ситуация (одновременное появление свойств I и J) повторяется много раз подряд, то величины $\lambda_n(P, Q)$ и $p_n(J/I)$ стремятся к единице. При многократном повторении второй ситуации (появление I без J) обе эти величины стремятся к нулю. Естественно поэтому величину $r_n(J/I) = \lambda_n(P, Q)$ называть *условной псевдочастотой* свойства J по отношению к свойству I.

Если условиться, что в машине условной вероятности непосредственно не возбужденные нейроны должны выдавать в качестве своих выходных сигналов не условные частоты, а условные псевдочастоты соответствующих им свойств по отношению к максимальному позитивному свойству I рассматриваемого на данном шаге изображения, то для осуществления этого достаточно от нейрона P, соответствующего свойству I, передать выходные сигналы $\lambda_n(P, R)$ на все прямо не возбужденные нейроны R. Удобно считать при этом, что нейрон P посылает по всем связям вида (P, R) выдаваемый им единичный выходной сигнал, который, проходя по соответствующей связи, ослабляется за счет умножения на величину $\lambda_n(P, R)$, по условию всегда меньшую единицы.

Не трудно видеть, что описанный механизм образования условных рефлексов имеет еще один существенный недостаток. Дело в том, что в силу принятых нами предположений условные рефлексы образуются лишь по отношению к целым изображениям, а не к отдельным их свойствам. Как известно, в случае биологических систем дело обстоит иначе. Более того, рефлексы наиболее часто образуются именно по отношению к частным (не максимальным) свойствам изображений.

Можно, правда, в машине условной вероятности любого из описанных выше типов раз и навсегда зафиксировать свойство I, по отношению к которому вычисляются условные частоты и псевдочастоты. В этом случае свойство может не быть максимальным позитивным свойством, а условный рефлекс будет вырабатываться именно по отношению к свойству изображений, а не к самим изображениям. Все сделанные выше определения закономерностей функционирования машины условной вероятности применимы и к этому случаю, нет необходимости лишь специально отыскивать нейрон, соответствующий максимальному позитивному свойству рассматриваемого изображения: в случае появления свойства I роль этого нейрона будет всегда выполнять нейрон, соответствующий свойству I. В случае же непоявления свойства I все нейроны (за исключением прямо возбужденных нейронов) должны, по определению, выдавать не условные, а безусловные частоты (или псевдочастоты) соответствующих им свойств.

В своем первоначальном определении Аттли рассматривает именно такой способ функционирования машины условной вероятности. Однако, избавившись от одного недостатка, мы приходим к другому и снова получаем существенное отличие от биологических систем, способных вырабатывать условные рефлексы по отношению ко многим свойствам, а не только к одному из них.

Чтобы избавиться от указанных недостатков, достаточно в последней из описанных нами схем снять ограничение, позволяющее формировать выходные сигналы прямо не возбужденных нейронов за счет выходного сигнала одного лишь нейрона P. Вместо этого примем следующее допущение.

На каждый прямо не возбужденный нейрон Q передаются сигналы $\lambda_n(P_i, Q)$ от всех прямо возбужденных нейронов P_i ($i = 1, 2, \ldots$). Выходным сигналом нейрона Q будет наибольший по величине сигнал из числа сигналов $\lambda_n(P_i, Q)$ ($i = 1, 2, \ldots$).

Устройство, реализующее описанный механизм возникновения выходных сигналов нейронов и изменяющее веса связей в соответствии с формулами (110), назовем *машиной условных рефлексов.* Можно надеяться, что в ней нашли свое отражение многие важные свойства реальных сетей нейронов, составляющих нервные системы животных и даже нервную систему человека.

В реальных сетях нейронов, разумеется, осуществляются далеко не все связи, которые требуются полной схемой машины условных рефлексов. Возможно также дальнейшее уточнение законов изменения весов связей. В частности, первая из формул (110), должна быть заменена, по-видимому, формулой $\lambda_{n+1}(P, Q) = (1-\gamma)\lambda_n(P,Q)$, где γ — весьма малая положительная константа. После такого уточнения закон изменения весов связей будет отражать свойство связей уменьшаться с течением времени даже при отсутствии случаев прямого неподтверждения рефлекса, отражаемого этой связью.

Если в машине условных рефлексов отбросить *требование полноты*, т. е. наличия нейронов для всех без исключения позитивных свойств, всех без исключения возможных изображений, то подобные *неполные* машины условных рефлексов могут быть использованы для работы с большим числом рецепторов. На этой основе возможно использование машин условных рефлексов для решения проблем распознавания зрительных образов, рассматривавшихся в предыдущих параграфах. Для получения эффективных результатов в этом направлении необходим целесообразный отбор тех свойств, которым будут соответствовать нейроны в указанной неполной машине.

Представляет интерес еще одна задача, связанная с классификационными системами. В описанных до сих пор классификационных системах все свойства изображений воспринимаются одновременно с самими изображениями. В ряде случаев приходится, однако, иметь дело с такими изображениями, свойства которых выявляются постепенно, по мере обучения. Впрочем, в этих случаях изображения, как правило, столь далеки от своих зрительных прототипов, что будем называть их уже не изображениями, а *понятиями.*

Подобная ситуация возникает при построении самосовершенствующихся систем для распознавания смысла фраз (см. В. М. Глушков, Н. М. Грищенко и А. А. Стогний [30]). В простейшем случае, когда рассматриваются фразы, состоящие лишь из подлежащего и сказуемого, распознаваемыми понятиями можно считать существительные, выбираемые в качестве подлежащих, а их свойствами — возможность или невозможность их осмысленного сочетания с различными глаголами, выступающими в качестве сказуемых.

Если список глаголов фиксирован и включает n различных глаголов, то каждое понятие (существительное) может быть охарактеризовано строкой сочетаемости с этими глаголами, имеющей длину n. На i-ом месте этой строки будет стоять единица или нуль в соответствии с тем, осмысленно или неосмысленно сочетание рассматриваемого существительного с i-ым глаголом данного списка ($i=1, 2, \ldots, n$).

Указанные строки назовем *глагольными строками*. В задачу самосовершенствующейся системы в этом случае будет входить предсказание возможно большего числа свойств рассматриваемых понятий на основе ограниченного опыта. Если опыт (история обучения) состоит в том, что упомянутой системе сообщаются одно за другим осмысленные сочетания случайно выбираемых из заданных списков существительных и глаголов, то по мере поступления таких сообщений будут постепенно заполняться единицами некоторые места глагольных строк выбранных нами понятий (существительных). Задачу обучения в этом случае можно трактовать как задачу воссоздания структуры унитарной машины для классификации свойств выбранных понятий. Для решения этой задачи естественно организовать процесс объединения понятий, сочетаемых с одними и теми же глаголами в классы, соответствующие новым, обобщенным понятиям, которых могло и не быть в исходном списке. Например, за счет объединения ряда понятий (скажем, «отец», «сын», «студент», «профессор» и др.) по признакам сочетаемости с глаголами «жить» и «думать» возникает понятие «человек». Если после образования того или иного класса выяснится, что те или иные его представители обладают каким-либо новым элементарным свойством (например, возможностью сочетания с глаголом «ходить»), то это свойство можно распространить на весь класс (т. е. на все входящие в этот класс понятия). Разумеется, при таком распространении свойств могут возникать ошибки. Для устранения возникших ошибок вводится процесс самостоятельного составления машиной новых фраз за счет сочетания других (случайно выбираемых) понятий из рассматриваемого класса с глаголом, сочетаемость с которым была распространена на весь класс. Осмысленность (или бессмысленность) этих сочетаний должна быть сообщена машине человеком-учителем.

Благодаря наличию в языке связей, подобных связи, которая описывается утверждением типа «почти все думающие являются вместе с тем и говорящими», описанный процесс при разумно выбранных способах образования классов, экстраполяции свойств и составления новых фраз позволяет машине с большой вероятностью осуществить правильное разделение фраз на осмысленные и бессмысленные. При этом существенно, что такое разделение будет произведено для всех фраз, которые можно построить из заданного множества существительных и глаголов. В их числе могут быть, в частности, и такие фразы, которые не строились самой машиной в качестве вопросов учителю и не сообщались ей в процессе обучения.

Опыты по обучению машины распознаванию смысла фраз не только описанной простейшей конструкции, но и фраз, имеющих более сложную конструкцию, были успешно проведены в Институте кибернетики АН УССР [30]. При различных предположениях относительно структуры языка (в данном случае — множества глагольных строек) можно проводить оценки для скорости обучения в реализованных алгоритмах подобно тому, как это было сделано для α-персептронов в предыдущем параграфе. Однако соответствующие выкладки весьма громоздки и значительно менее наглядны, чем в случае персептронов.

§ 9. Самоорганизация и самонастройка. Методы решения сложных вариационных задач

В § 1 настоящей главы было введено понятие системы алгоритмов как естественной формы, в которую облекаются свойства самоорганизации и самосовершенствования. Проведем дальнейшую детализацию этого понятия. Применительно к большинству задач, встречающихся на практике, целесообразно различать собственно самоорганизацию и так называемую *самонастройку*, представляющую собой наиболее простой случай самосовершенствования. Более точно, будем различать простейший тип самосовершенствования на базе самонастройки и более высокий тип самосовершенствования на базе самоорганизации.

Различие, о котором здесь идет речь, заключается в том, что самосовершенствование на базе самонастройки предполагает изменение лишь некоторых числовых параметров в рабочем алгоритме, в то время как самоорганизация связана с изменением структуры самого алгоритма. Разумеется, указанное различие до известной степени условно, поскольку при соответствующей записи алгоритмов изменения в структуре алгоритма могут быть сведены к изменениям числовых параметров. Если, например, осуществлена нумерация всех алгоритмов рассматриваемого класса, то любое изменение алгоритма сводится к изменению соответствующего номера, который можно рассматривать как числовой параметр. Однако, несмотря на относительность различия между самонастройкой и самоорганизацией, на практике, благодаря употреблению какой-либо фиксированной формы записи алгоритмов (алгоритмического языка), это различие удается провести достаточно четко.

Приведем некоторые примеры самонастройки и самосовершенствования структуры схем алгоритмов. В качестве первого примера рассмотрим часто встречающийся на практике случай самонастройки, носящий специальное название *экстремального регулирования*. Суть задачи экстремального регулирования состоит в выдаче в систему регулирования таких значений некоторых параметров $x_1, x_2, \ldots, x_n$, чтобы заданная функция $f = f(x_1, x_2, \ldots, x_n)$ от этих параметров принимала экстремальное (минимальное или максимальное) значение. Функция $f(x_1, x_2, \ldots, x_n)$ зависит при этом

также от некоторых других параметров $y_1, y_2, \ldots, y_m$, которые изменяются независимо от нашего желания и непосредственное управление которыми невозможно. Благодаря их изменению, значения параметров $x_1, x_2, \ldots, x_n$, обеспечивающие желаемый экстремум функции f, не могут быть выбраны раз и навсегда — их необходимо изменять по мере изменения параметров $y_1, y_2, \ldots, y_m$.

На практике чаще всего приходится встречаться со случаем, когда нахождение экстремумов обычными методами (с помощью приравнивания нулю частных производных функции f и решения получившейся системы уравнений) невозможно или нецелесообразно. Причиной этого может служить, например, отсутствие аналитического выражения для функции f. Возникает вопрос о том, какими методами можно решить задачу самонастройки при наличии подобной ситуации. Одним из универсальных методов решения этой задачи в указанном случае является так называемый *метод наискорейшего спуска* (или наискорейшего подъема).

Метод наискорейшего спуска (подъема) служит для нахождения точек минимума (соответственно максимума) функции многих переменных с помощью развертывания специального процесса последовательного приближения к таким точкам. Пусть $f(x_1, x_2, \ldots, x_n)$ — произвольная дифференцируемая функция от n переменных. В пространстве этих переменных выберем произвольную точку $M(a_1, a_2, \ldots, a_n)$ и найдем приближенные значения частных производных $f_i = f'_{x_i}(a_1, a_2, \ldots, a_n)$ в точке M по формулам

$$f_i \approx \frac{1}{\Delta}(f(a_1, a_2, \ldots, a_{i-1}, a_i + \Delta, a_{i+1}, \ldots, a_n) - \\ - f(a_1, a_2, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n)) \qquad (i = 1, 2, \ldots, n),$$

давая всем переменным по очереди одно и то же приращение Δ. Выясним, какие нужно дать приращения одновременно всем аргументам, чтобы в максимально возможной степени приблизиться к точке экстремума, используя лишь значения функции f и ее производных в точке M.

Последнее условие является весьма существенным, поскольку без него вопрос решался бы тривиально; приращения переменных должны быть такими, чтобы они привели нас непосредственно в точку экстремума. Однако точка экстремума нам неизвестна и требуется решить вопрос о продвижении к ней на основании информации и поведения рассматриваемой функции в окрестности выбранной произвольно точки $M(a_1, a_2, \ldots, a_n)$. Обозначая искомые приращения переменных $x_1, x_2, \ldots, x_n$ в точке M через $\Delta_1, \Delta_2, \ldots, \Delta_n$ соответственно и используя формулу полного дифференциала, получим для приращения Δf функции f в точке M приближенную формулу

$$\Delta f \approx f_1\Delta_1 + f_2\Delta_2 + \ldots + f_n\Delta_n.$$

Если условиться делать шаг в сторону точки экстремума (в пространстве переменных $x_1, x_2, \ldots, x_n$) лишь какой-либо одной постоянной длины r, то к уравнению приращения функции f добавится еще одно уравнение

$$\Delta_1^2 + \Delta_2^2 + \ldots + \Delta_n^2 = r^2. \qquad (111)$$

Необходимо выбрать величины $\Delta_1, \Delta_2, \ldots, \Delta_n$ так, чтобы при соблюдении равенства (111) функция Δf достигала минимального (с учетом знака) значения. По методу неопределенных множителей Лагранжа вопрос сводится к нахождению экстремума функции

$$F = \sum_{i=1}^{n} (f_i \Delta_i - \lambda \Delta_i^2) + \lambda r^2$$

от переменных $\Delta_1, \Delta_2, \ldots, \Delta_n$. Дифференцируя по Δ_i и приравнивая полученные частные производные к нулю, получим систему уравнений

$$f_i - 2\lambda \Delta_i = 0 \qquad (i = 1, 2, \ldots, n), \qquad (112)$$

которую необходимо дополнить, разумеется, уравнением (111). Из уравнений (112) находим, что

$$\Delta_i = k f_i, \qquad (113)$$

где k — некоторый коэффициент пропорциональности, общий для всех $i = 1, 2, \ldots, n$.

Уравнения (113) в зависимости от выбора знака у коэффициента пропорциональности k определяют два противоположных направления, при движении по которым из точки M осуществляется (в окрестности точки M) наиболее быстрый рост (при $k > 0$) или наиболее быстрое убывание (при $k < 0$) функции f. Эти направления называются соответственно направлениями *наискорейшего подъема и наискорейшего спуска* в рассматриваемой точке M. Величина продвижения r в любом из указанных направлений называется соответственно *шагом градиентного подъема* или *спуска* в точке M.

В зависимости от того, требуется ли найти точку максимума или точку минимума рассматриваемой функции f, выбирается одно из этих направлений (знак параметра k в уравнениях (113))и осуществляется движение в выбранном направлении (определяемое выбором величины параметра k) до тех пор, пока функция f не изменит характера своего роста в этом направлении, т.е. не перейдет от возрастания к убыванию либо, наоборот, от убывания к возрастанию. Иными словами, осуществляется *максимальное продвижение* в избранном направлении, обеспечивающее изменение функции f в нужном направлении (в направлении убывания при поиске точки минимума функции f и в направлении роста при поиске точки ее максимума).

Обозначив буквой N точку, полученную в результате такого продвижения, с ней производят все те же операции, которые были описаны для точки M. В результате получают новую точку P и т. д. Если функция f достаточно гладкая, то в результате описанного процесса, продолжая его достаточно долго, попадем в сколь угодно малую окрестность некоторой стационарной точки данной функции, т. е. такой точки, в которой все частные производные функции равны нулю (это будет, разумеется, лишь в том случае, когда стационарные точки у данной функции существуют), либо в окрестность точки границы области определения функции f, соответствующей некоторому локальному экстремальному (в данной области) значению функции f.

Искомая точка (абсолютного) экстремума функции f при принятых предположениях находится среди указанных точек. Нет, однако, никакой гарантии, что применение описанной выше методики приведет с первого же раза в точку абсолютного экстремума заданной функции. Необходимо поэтому, варьируя начальную точку M, найти (описанным выше методом) новые стационарные точки функции, чтобы в результате последующего сравнения значений функции в этих точках выбрать среди них искомую точку экстремума.

На практике предпочитают обычно другой путь: выбирают случайный ряд точек $M_1, M_2, \ldots, M_k$ в области определения функции f, а из них выбирают ту, в которой функция имеет максимальное (в случае задачи нахождения точки максимума) или минимальное (в случае нахождения точки минимума) значение. Отправляясь от выбранной таким образом точки, осуществляют наискорейший спуск либо наискорейший подъем по описанной выше методике. При достаточно большом k (зависящем от выбора функции f) с вероятностью, сколь угодно близкой к единице, таким способом будет найдена именно точка абсолютного (а не какого-нибудь локального) экстремума.

Один из возможных вариантов поиска абсолютного экстремума функции двух переменных приведен на рис. 15. Функция на этом рисунке задана своими горизонталями (линиями равного уровня), через M_1, M_2, M_3 обозначены случайно выбираемые начальные точки (точка M_2 — наивысшая среди них), а через N, P, Q — последовательный ряд точек, получаемых из точки M_2 по методу наискорейшего подъема. В этом примере уже после трех шагов наискорейшего подъема практически попадаем в точку абсолютного максимума рассматриваемой функции.

Система алгоритмов, решающая задачу самонастройки, составляется из рабочего алгоритма A, который, воспринимая входное слово (значение функции $f(x_1, x_2, \ldots, x_n)$), определяющее критерий качества регулирования и, возможно, значения некоторых других величин, выдает выходное слово, состоящее из координат некоторой точки $M(a_1, a_2, \ldots, a_n)$, совпадающей в случае стационарного режима (неизменности функции f) с точкой абсолютного эк-

стремума этой функции. В случае нестационарного режима (изменения функции f) вступает в действие алгоритм B, который осуществляет поиск точки абсолютного экстремума измененной функции f тем или иным способом (например, методом наискорейшего спуска или подъема) и заменяет координатами этой точки параметры $a_1, a_2, \ldots, a_n$, выдаваемые рабочим алгоритмом A.

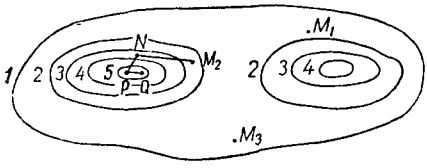

Рис. 15.

Описанная система, состоящая из алгоритмов A и B, может рассматриваться как самонастраивающаяся система алгоритмов.

Рассмотрим теперь пример самосовершенствования с изменением структуры рабочего алгоритма. Предположим, что рабочий алгоритм должен по различным численным значениям коэффициентов p и q приведенного квадратного уравнения $x^2 + px + q = 0$ выдавать корни этого уравнения, однако в начале работы этот алгоритм еще не известен. Поскольку приведенное квадратное уравнение решается по известной формуле $x_{1,2} = -\frac{p}{2} \pm \sqrt{\frac{p^2}{4} - q}$, то искомый алгоритм можно искать в классе формул, построенных с помощью операций сложения, вычитания, умножения, деления и извлечения квадратного корня из букв p и q и целых чисел. Все такие формулы можно занумеровать, расположив их предварительно в порядке возрастания сложности: с ростом номера формулы увеличивается, вообще говоря, число употребляющихся в формуле операций и возрастает максимальная величина содержащихся в ней целочисленных параметров.

Первоначально в качестве рабочего алгоритма выбирается одна из простейших формул, скажем формула $p + q$. Получая очередные значения коэффициентов p и q (например, $p = 3$, $q = 2$), рабочий алгоритм выдает соответствующее значение корня или корней (в данном случае $x = p + q = 5$). Обучающий алгоритм производит проверку полученного решения, подставляя найденные значения корней в исходное уравнение. Если эти значения корней удовлетворяют уравнению, то рабочий алгоритм сохраняется неизменным. Если же решение найдено неверно (как в рассмотренном только что случае), то в качестве рабочего алгоритма выбирается следующая по порядку формула.

Легко понять, что при описанной организации рабочего и обучающего алгоритмов после конечного числа неудачных попыток будет зафиксирована правильная формула для решения квадратного уравнения. Поскольку в процессе поиска изменяется, вообще говоря, структура алгоритма (вид формулы), а не только числовые параметры, то, согласно принятой нами терминологии, построенная система алгоритмов является самосовершенствующейся на базе самоорганизации, т. е. представляет собой более

высокий тип самосовершенствования по сравнению с самонастройкой.

В рассмотренном примере поиск требуемого рабочего алгоритма осуществляется за счет простого перебора всех алгоритмов заданного заранее класса. Можно, разумеется, не использовать предварительную фиксацию какого-либо специального класса алгоритмов, а осуществлять последовательный перебор в классе *всех* алгоритмов, записанных на том или ином фиксированном алгоритмическом языке (например, на языке нормальных алгоритмов), однако в таком случае время поиска, как правило, значительно увеличивается.

Обычно системы подобного поиска осуществляются на быстродействующих электронных вычислительных машинах, выполняющих многие тысячи операций в секунду. При такой скорости решение описанного примера (нахождение формулы для решения приведенных квадратных уравнений) отнимает мало времени. Однако по мере усложнения искомых рабочих алгоритмов время поиска, основанного на переборе всех вариантов, растет столь быстро, что реализация подобного поиска за разумное время, даже при использовании сверхбыстродействующих машин, становится невозможной.

В этом случае прибегают к ступенчатому поиску: сначала ищут какие-либо достаточно простые составные части (блоки) искомого рабочего алгоритма, а затем применяют различные комбинации построенных блоков. Сами блоки могут строиться из еще более мелких блоков, так что процесс дальнейшего дробления поиска на отдельные ступени может быть еще продолжен. Системы ступенчатого поиска позволяют строить весьма сложные самоорганизующиеся системы, аналогичные системам творческого поиска, употребляемым человеком.

Не вдаваясь в дальнейшие подробности самосовершенствования на базе самоорганизации, сосредоточим наше внимание на проблемах, связанных с самонастройкой.

Описанный выше метод наискорейшего спуска (или подъема) также представляет собой некоторый поиск. При этом поиске применяют определенную *стратегию* (или *тактику*), для уменьшения перебора различных вариантов, ведущих к решению задачи. В рассмотренном случае стратегия поиска сводилась к использованию информации о *локальных* свойствах соответствующей функции f, которую назовем *оценочной* функцией (значения этой функции можно рассматривать как оценку качества найденного на том или ином шаге поиска приближения к требуемому решению).

Если число параметров (аргументов оценочной функции) очень велико, возникают различные затруднения при использовании метода наискорейшего спуска (подъема) в описанном выше простейшем виде (зацикливание на второстепенных минимумах или максимумах, слишком медленный темп продвижения к абсолютному экстремуму и т. д.). С целью устранения этих затруднений

вносятся различные изменения и усовершенствования в описанную выше стратегию локального поиска.

Простейшие изменения связаны с выбором того или иного шага градиентного спуска (подъема). Целесообразно, в частности, осуществлять продвижение в заданном направлении не до тех пор, пока приращение Δf оценочной функции изменит знак, а лишь до тех пор, пока относительная величина этого приращения $\frac{\Delta f}{f}$ не будет меньше (по модулю) некоторой фиксированной заранее величины, называемой градиентной пробой.

В ряде случаев удается разбить аргументы оценочной функции f на два класса так, что изменения переменных первого класса приводят к относительно большим изменениям значения функции f, тогда как изменения переменных, входящих во второй класс, изменяют это значение в значительно меньшей степени. Описанные выше методы обеспечивают слишком малую скорость продвижения к экстремуму в направлениях, соответствующих изменениям переменных второго класса. Образно выражаясь, можно осуществить быстрый спуск (в направлении, соответствующем изменениям переменных первого класса) на дно некоторого «оврага», а затем блуждать более или менее случайно по его дну, практически не приближаясь к точке экстремума.

Для устранения описанного недостатка И. М. Гельфанд и М. Л. Цетлин [18] предложили специальный метод, названный ими *методом оврагов*. Суть его состоит в том, что на «склонах оврага» выбирается не одна, а две точки (X_0 и X_1). Из них осуществляется наискорейший спуск на «дно оврага», в результате чего возникают две новые точки A_0 и A_1. Соединяя точки A_0 и A_1 прямой, делают в направлении этой прямой (в сторону убывания оценочной функции) так называемый *овражный шаг*, величина которого обычно значительно больше величины шага градиентного спуска. Этот шаг приводит в новую точку — X_2, из которой снова осуществляется наискорейший спуск в точку A_2. расположенную на «дне» оврага. В направлении, определяемом точками A_1 и A_2, снова делается овражный шаг, приводящий в новую точку — X_3. Из нее снова осуществляется наискорейший спуск на «дно оврага» и т.д.

Мы провели описание метода применительно к спуску (т. е. к нахождению минимума оценочной функции f). Само собой разумеется, что соответствующие построения применимы и к подъему (нахождению максимума функции f). В этом случае вместо спуска в «овраг» осуществляется подъем на «хребет» и дальнейшее продвижение не по «дну оврага», а по «гребню хребта».

Все описанные методы спуска (подъема) относятся к классу методов для отыскания экстремумов функций или функционалов, которые объединим под общим названием *вариационных методов*. В большинстве задач, представляющих интерес для кибернетики, существенное значение приобретают те или иные ограничения воз-

можной величины аргументов оценочной функции f. При этом искомые экстремумы могут достигаться не внутри области определения функции f, а на границах этой области. Рассмотренные выше методы спуска (подъема) пригодны в принципе и для нахождения подобных «граничных» экстремумов. Однако в ряде частных случаев гораздо более эффективными оказываются некоторые специальные вариационные методы.

К числу такого рода методов относятся методы так называемого *линейного программирования* (или линейного планирования). Эти методы применяются в том случае, когда оценочная функция f представляет собой линейную функцию (полином первой степени): $f = a_1x_1 + a_2x_2 + \ldots + a_nx_n + a_0$, а границы области определения переменных составляются из гиперплоскостей, т. е. из поверхностей, задаваемых уравнениями первой степени. В этом случае область определения представляет собой многомерный многогранник (не обязательно конечный), все точки которого удовлетворяют системе линейных неравенств вида $b_{i_1}x_1 + b_{i_2}x_2 + \ldots + b_{i_n}x_n + b_{i_0} \geqslant 0$ $(i = 1, 2, \ldots, m)$. Знаки неравенств могут быть, разумеется, изменены на обратные за счет изменения знаков коэффициентов b_{ij}.

Нетрудно понять, что экстремальное значение оценочная функция f принимает в одной или нескольких вершинах многогранника. В случае нескольких вершин экстремальное значение принимается функцией f на грани (вообще говоря, многомерной), проходящей через все эти вершины. Можно поэтому найти искомое экстремальное значение, перебирая одну за другой все вершины многогранника, однако при большом числе переменных этот метод оказывается чрезвычайно громоздким и практически непригодным. В настоящее время разработаны гораздо более эффективные методы решения задач линейного программирования. Эти методы предусматривают применение не линейных неравенств, а линейных уравнений, к которым можно свести любые неравенства за счет введения новых неизвестных. При таком введении неравенства $\sum_{j=1}^{n} b_{i_j}x_j + b_{i_0} \geqslant 0$ заменяются уравнениями $\sum_{j=1}^{n} b_{i_j}x_j + b_{i_0} = y_i$, где y_i —новые неизвестные, которые должны принимать лишь *неотрицательные значения* $(i = 1, 2, \ldots, m)$.

На практике наиболее часто встречается следующая постановка задачи линейного программирования, которую будем называть *канонической*.

Дана система m линейных алгебраических уравнений с неизвестными $\sum_{j=1}^{n} a_{ij}x_j = b_i$ $(i = 1, 2, \ldots, m)$. *Требуется найти такое неотрицательное* (все $x_j \geqslant 0$) *решение этой системы, для которого некоторая фиксированная линейная форма* (*оценочная функция*) $f = \sum_{k=1}^{n} c_kx_k$ *принимает наименьшее возможное значение.*

Опишем один из возможных эффективных методов решения этой задачи, называемый обычно симплекс-методом. При применении симплекс-метода производятся последовательные преобразования заданной системы уравнений $\sum_{j=1}^{n} a_{ij}x_j = b_i (i = 1, 2, \ldots, m)$ до тех пор, пока она не будет приведена к некоторому специальному виду. Предварительно система трансформируется: все свободные члены b_i делаются неотрицательными (если $b_i < 0$, то достаточно поменять знаки обеих частей i-го уравнения); затем уравнения переписываются в виде $0 = b_i - \sum_{j=1}^{n} a_{ij}x_j$ $(i = 1, 2, \ldots, m)$. Если в полученной системе имеется переменная x_k, входящая лишь в одно уравнение, скажем p-е, и имеющая положительный коэффициент a_{ik}, то такая переменная получает название основной, а соответствующее уравнение решается относительно этой переменной. Выделяя все основные переменные, обозначенные буквами $x_1, x_2, \ldots, x_{k_0}$, приведем нашу систему к виду

$$\begin{aligned} x_k &= b_k - \sum_{j=k_0+1}^{n} a_{kj}x_j \quad (k = 1, 2, \ldots, k_0); \\ 0 &= b_i - \sum_{j=k_0+1}^{n} a_{ij}x_j \quad (i = k_0 + 1,\ k_0 + 2, \ldots, m). \end{aligned} \tag{114}$$

Уравнения второй группы (не разрешенные относительно x_k) называются 0-уравнениями (не исключено, что такими уравнениями окажутся все уравнения системы). Цель дальнейших преобразований состоит в нахождении какого-нибудь неотрицательного решения системы (114). Эти преобразования сводятся к последовательному (вообще говоря, многократному) повторению цикла, состоящего из следующих шагов (см. [6]):

1. *Отыскивается 0-уравнение, у которого свободный член строго больше нуля (если такого 0-уравнения нет, то задача решена, поскольку, полагая* $x_k = b_k$ $(k = 1, 2, \ldots, k_0)$ и $x_i = 0$ $(i = k_0 + 1, k_0 + 2, \ldots, n)$, *получим, очевидно, неотрицательное решение системы* (5)). *Пусть это будет* i*-е уравнение.*

2. *В найденном* (i*-ом*) *уравнении отмечается какой-нибудь положительный коэффициент* a_{ij_1} (*если все коэффициенты* a_{ij} *в* i*-ом уравнении неположительны, то система* (114), *очевидно, не может иметь положительных решений, и, следовательно, поставленная задача линейного программирования неразрешима*).

3. *В том же столбце, в котором содержится выделенный коэффициент* a_{ij_1}(*т. е. в* j_1*-ом столбце*) *находим так называемый разрешающий коэффициент* $a_{i_1j_1}$, *который характеризуется тем, что из всех отношений* $\frac{b_i}{a_{ij_1}}$ *с положительными* a_{ij_1} $(i = 1, 2, \ldots, n)$ *отношение* $\frac{b_{i_1}}{a_{i_1j_1}}$ *имеет минимально возможное значение.*

4. *Уравнение, в которое входит разрешающий коэффициент (т. е. i_1-е уравнение), разрешается относительно переменной x_{j_1}, которая относится после этого к числу основных переменных, и найденное выражение для x_{j_1} подставляется во все остальные уравнения (если i_1-е уравнение не принадлежало к числу 0-уравнений, то переменное x_{i_1}, стоявшее в его левой части, исключается из числа основных переменных).*

5. *После разрешения i_1-го уравнения (относительно x_{i_1}) снова отыскивается 0-уравнение с положительным свободным членом, и с ним производятся все описанные выше действия.*

Описаниый процесс последовательного разрешения продолжается до тех пор, пока не исчезнут все 0-уравнения. Искомое неотрицательное решение получается в результате приравнивания всех основных переменных x_i соответствующим свободным членам b_i ($i = 1, 2, \ldots, n$), а всех неосновных переменных — нулю. Существуют случаи, когда описанный процесс зацикливается и, продолжаясь бесконечно долго, не приводит ни к какому решению. В этих случаях прибегают к изменению выбора 0-уравнения и разрешающего коэффициента на 2-ом и 3-ем шагах процесса последовательного разрешения, что обычно помогает избежать зацикливания.

После окончания процесса последовательного разрешения найденные выражения для основных переменных подставляются в оценочную функцию $f = \sum_{k=1}^{n} c_k x_k$, в результате чего она примет вид $f = d - \sum_{j=r+1}^{n} d_j x'_j$. В последнем выражении суммирование распространяется лишь на неосновные переменные, которым (после соответствующей перенумерации) приписаны номера от $r + 1$ до n.

Далее описанный выше процесс разрешения применяется к системе уравнений для основных переменных, полученной в результате первого применения этого процесса, которая записывается

$$x'_i = b'_i - \sum_{j=r+1}^{n} a'_{ij} x'_j \qquad (i = 1, 2, \ldots, r),$$

и к новому 0-уравнению $0 = d - \sum_{j=r+1}^{n} d_j x'_j$. В качестве разрешающих коэффициентов выбираются лишь коэффициенты a_{ij}. Процесс оканчивается после того, как в 0-уравнении все коэффициенты d_j станут отрицательными. Полагая после этого все неосновные переменные равными нулю, а все основные переменные соответствующими свободным членам, получим требуемое решение исходной задачи линейного программирования. Заметим, что как при первом, так и при втором применении процесса последовательного разрешения все свободные члены (за исключением члена d) остаются все время неотрицательными.

Линейное программирование широко применяется в задачах оптимального планирования транспортных перевозок. Подобные применения линейного программирования были разработаны впервые Л. В. Канторовичем [38]. Подробные обоснования описанного нами симплекс-метода можно найти в специальных монографиях, посвященных линейному программированию.

Опишем еще одну общую схему вариационных задач, к которой сводятся многие задачи так называемого *динамического программирования* (или динамического планирования) [7]. Смысл указанной схемы сводится к следующему: в некотором (вообще говоря, многомерном) эвклидовом пространстве с помощью тех или иных ограничений выделяется некоторый класс кривых, которые будем называть *траекториями*. На множестве всех возможных траекторий задана некоторая оценочная функция (или, как принято обычно говорить, *функционал*). Задача состоит в нахождении траектории, на которой значение этого функционала является наибольшим или наименьшим.

Рассмотрим один из достаточно общих численных (приближенных) методов решения указанной задачи, разработанный В. С. Михалевичем и Н. З. Шором, — *метод последовательного анализа вариантов*. Изложим этот метод применительно к одному простейшему случаю, когда основное пространство двумерно, класс K траекторий состоит из всех кусочно-гладких кривых, соединяющих две фиксированные точки A и B пространства и содержащихся целиком в некоторой фиксированной конечной области Q, а оценочный функционал F обладает свойством аддитивности. Свойство аддитивности состоит в том, что функционал F предполагается определенным не только на целых траекториях, но и на любых их кусках (открытых подмножествах и их замыканиях) и при объединении двух непересекающихся кусков в один соответствующие значения оценочного функционала складываются

$$F(L_1 \cup L_2) = F(L_1) + F(L_2).$$

Сформулированные условия соответствуют задаче динамического программирования в постановке Р. Беллмана.

Первый шаг в методе последовательного анализа вариантов заключается в том, что класс K ограничивается: из всех траекторий, соединяющих точки A и B, выделяются лишь некоторые ломаные линии. Происходит это с помощью проведения нескольких сечений (в данном случае прямых), перпендикулярных отрезку AB и пересекающих его. Вершины рассматриваемых ломаных могут располагаться только на выбранных сечениях. Далее, каждое сечение (в пределах области Q) аппроксимируется конечным множеством точек (возможных вершин ломаных). Густота расположения точек аппроксимирующего множества, как и густота сечений, определяется исходя из требуемой точности решения задачи. Наглядное представление о требуемых построениях дает рис. 16.

На рис. 16 граница области Q обозначена пунктиром, сечения — римскими цифрами, а точки аппроксимирующих сечения множеств — арабскими цифрами. Если количество точек в i-ом сечении обозначить n_i, а общее число сечений — m, то общее количество (N) ломаных, соответствующих выдвинутым условиям, будет определяться, как нетрудно видеть, произведением чисел $n_i : N = n_1 n_2 \ldots n_m$. Эта величина очень быстро растет с ростом числа точек и сечений, в результате чего перебор всех вариантов оказывается практически невозможным.

Можно, однако, сократить перебор, воспользовавшись следую-

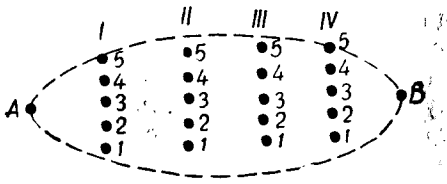

Рис. 16.

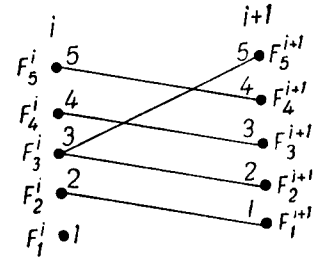

Рис. 17.

щим приемом. Соединим точку A отрезками прямых со всеми точками первого сечения и вычислим значение оценочного функционала F для всех этих отрезков. Каждой точке j первого сечения, сопоставим значение F_j^1 функционала F на отрезке, соединяющем эту точку с точкой A. Для каждой точки k второго сечения находим точку j_k первого сечения так, чтобы значение оценочного функционала F на ломаной Aj_kk было наименьшим по сравнению с его значениями на всех других допустимых ломаных, соединяющих точку A с точкой k. Поскольку функционал F аддитивен, то вопрос сводится к минимизации суммы $F_{j_k} + F(j_k, k)$, где $F(j_k, k)$ — значение функционала F на отрезке $[j_k, k]$. Соответствующее (минимальное среди всех возможных значений) значение функционала F на ломаной Aj_kk обозначим через F_k^2. Для его нахождения используем уже найденные значения функционала F_j^1 в точках предыдущего (первого) сечения, которые по необходимости будут здесь минимальными, а перебор ограничивается лишь всевозможными отрезками, соединяющими точки первого и второго сечений.

Предположим, что для всех точек p i-го сечения уже найдены минимальные значения F_p^i функционала F на всех допустимых ломаных, соединяющих эти точки с точкой A. Рассмотрим участок между i-ым и $(i+1)$-ым сечениями (рис. 17). Для каждой точки q $(i+1)$-го сечения найдем точку p_q i-го сечения так, чтобы сумма $F_{p_q}^i + F(p_q, q)$ была минимальной. Очевидно, что найденное минимальное значение указанной суммы будет представлять собой минимально возможное значение оценочного функционала F на всех допустимых ломаных, соединяющих точку A с точкой q. За-

писывая это значение F_q^{i+1} и заставляя точку q пробежать все точки $(i+1)$-го сечения, получим возможность перейти к рассмотрению (на основе совершенно аналогичных построений) участка между $(i+1)$-ым и $(i+2)$-ым сечениями. Для рассмотренного же участка между i-ым и $(i+1)$-ым сечениями необходимо запомнить лишь функцию $\varphi^i(q)$, сопоставляющую каждой точке q $(i+1)$-го сечения точку $p_q = \varphi^i(q)$ i-го сечения, с которой ее выгоднее всего соединить. В случае, изображенном на рис. 17,

$$\varphi^i(1) = 2; \quad \varphi^i(2) = 3; \quad \varphi^i(3) = 4; \quad \varphi^i(4) = 5; \quad \varphi^i(5) = 3.$$

В результате повторения указанного процесса придем, наконец, к рассмотрению участка между последним (m-ым) сечением и конечной точкой B. Находя для точки B точку $r = \varphi^m(B)$ m-го сечения, с которой ее выгоднее всего соединить, можем по запомненным нами функциям $\varphi^i(x)$ $(i = 1, 2, \ldots, m)$ найти наивыгоднейшую траекторию (допустимую ломаную) $Ak_1k_2 \ldots k_i k_{i+1} \ldots k_m B$, соединяющую точки A и B: точки этой ломаной определяются последовательно (справа налево) с помощью соотношений $k_i = \varphi^i(k_{i+1})$ $(i = m, m-1, \ldots, 1)$, причем в качестве точки k_{m+1} выбирается точка B. Легко видеть, что найденная ломаная, действительно, минимизирует оценочный функционал F. При этом перебор, с помощью которого была найдена ломаная, ограничивается, очевидно, всего лишь $n_1n_2 + n_2n_3 + \ldots + n_{m-1}n_m + n_m$ вариантами вместо $n_1n_2 \ldots n_m$ вариантов при полном переборе (n_i — количество точек в i-ом сечении).

Описанный метод непосредственно обобщается на случай многомерных пространств. Требование аддитивности оценочного функционала F также не является строго обязательным. Достаточно, очевидно, предположить, что для любого начального куска APQ траектории значение $F(APQ)$ функционала может быть представлено в виде $F(APQ) = f(F(A, P), F(P, Q))$, где $f(x, y)$ — вещественная функция, не убывающая по x при любом значении y.

Заметим еще, что в большинстве случаев перебор различных вариантов соединения точек двух соседних сечений может быть значительно уменьшен за счет различного рода ограничений (например, ограничений на максимальный угол наклона отрезков ломаных по отношению к оси x в двумерном случае). В ряде случаев перебор удается сократить за счет использования свойства непрерывности оценочного функционала. Более сложные построения в методике последовательного анализа вариантов связаны со специальной систематизацией ограничений и функционалов, позволяющей строить алгоритмы поиска оптимальной траектории путем последовательного обнаружения и выбрасывания «бесперспективных» отрезков траекторий.

Глава V

ЭЛЕКТРОННЫЕ ЦИФРОВЫЕ МАШИНЫ И ПРОГРАММИРОВАНИЕ

§ 1. Универсальный программный автомат

Одним из наиболее значительных технических достижений нашего времени явилось создание *универсальных программных* автоматов, т. е. автоматических преобразователей информации, позволяющих реализовать любые алгоритмы. Подобными автоматами являются современные универсальные электронные вычислительные (цифровые) машины. Интересно отметить, что, как явствует из самого названия, эти машины были созданы для целей автоматизации вычислений, точнее, — для автоматизации выполнения произвольных *вычислительных* алгоритмов. Термин «универсальные» применительно к этим машинам создатели первых универсальных вычислительных машин понимали (да и теперь еще многие понимают) в смысле универсальности по отношению именно к вычислительным алгоритмам.

Однако, поскольку всякий алгоритм может быть сведен, как отмечалось в гл. I, к вычислению некоторой частично рекурсивной (арифметической) функции, то универсальность по отношению к вычислительным алгоритмам оказывается универсальностью вообще. Это обстоятельство имеет огромное практическое и принципиальное значение, поскольку фактически основой любой области деятельности человека является преобразование информации в соответствии с теми или иными, зачастую весьма сложными наборами алгоритмов.

Наличие в нашем распоряжении универсальных автоматических преобразователей информации, какими являются современные универсальные электронные вычислительные машины, позволяет, по крайней мере в принципе, автоматизировать любую область деятельности человека, основанную на преобразованиях информации. Это может быть и решение сложных задач расчетного характера,

и планирование, и управление производством, и перевод с одного языка на другой, и сочинение музыки, и игра в шахматы, и многое другое. Курьезно, что указанные огромные возможности, заложенные в универсальных электронных машинах, не только не были осознаны первыми их конструкторами, но многими из них даже оспаривались.

В связи с этим необходимо отметить еще одно заблуждение, которое бытует среди лиц, не знакомых с теорией алгоритмов. Существует мнение о том, что удивительные свойства современных электронных цифровых машин основываются на каких-то специфических особенностях применяемых в них элементов — электронных ламп, транзисторов и т. п. В действительности электроника сама по себе не имеет никакого отношения к их принципиальным (качественным) возможностям.

Дело заключается *в особом принципе управления и в наборе операций*, которые могут выполнять эти машины, а элементы, из которых они построены, могут иметь весьма различную физическую природу и могут, в частности, быть чисто механическими. Электронные элементы применяются с целью значительного увеличения *быстродействия* вычислительных машин, а также их *надежности* (в расчете на какое-нибудь фиксированное количество выполняемых машиной операций). Заметим, что первая универсальная вычислительная цифровая машина (Марк-1) была построена не на электронных, а на электромеханических элементах (электромагнитных реле).

Принцип управления, обеспечивающий *алгоритмическую универсальность* (способность реализовать любые алгоритмы) современных универсальных цифровых машин, представляет собой развитие и обобщение принципа, положенного в основу алгоритмической схемы Поста, описанной в гл. I. Как и в схеме Поста, информация в универсальной цифровой машине хранится в памяти, разбитой на *отдельные ячейки* (ячейки памяти). Однако, в отличие от схемы Поста, в каждой ячейке может храниться не одна двоичная цифра (0 и 1), а целое *слово*, составленное из значительного (обычно 30—40) числа двоичных цифр. Можно, если угодно, рассматривать эти слова и как буквы в некотором конечном алфавите, однако этот алфавит будет содержать, как правило, весьма большое количество букв $2^{30} — 2^{40}$.

Обычно поэтому предпочитают рассматривать содержимое каждой ячейки памяти не как отдельную букву, а как *слово* в двоичном алфавите. Составляющие это слово двоичные цифры (0 и 1) принято называть (*двоичными*) *разрядами*, а само слово — *двоичным кодом*, а иногда и просто (*двоичным*) *числом*. Можно, разумеется, считать буквами не содержимое отдельных двоичных разрядов, а некоторые комбинации таких разрядов. Например, любой двоичный код длины, равной трем, можно рассматривать как число в восьмеричной системе счисления, обозначая тройками двоичных

цифр восьмеричные цифры: 000-0, 001-1, 010-2, 011-3, 100-4, 101-5, 110-6, 111-7. Используя не все, а лишь некоторые четверки двоичных цифр для обозначения десятичных цифр, можно представлять состоящие из таких четверок двоичные коды числами в десятичной системе счисления.

Кроме замены двоичных цифр многоразрядными двоичными кодами, имеется и второе существенное отличие в организации памяти (или *запоминающего устройства*) универсальной цифровой машины и памяти для алгоритмов в схеме Поста. Являясь абстрактной алгоритмической схемой, схема Поста предполагала наличие *бесконечного* числа ячеек, или наличие памяти неограниченного объема. В то же время в реальных технических устройствах, которыми являются универсальные цифровые машины, объем памяти *по необходимости ограничен*.

В современных больших универсальных цифровых машинах объем быстродействующей памяти не превышает 100 тыс. ячеек (а обычно даже 4—8 тыс.). Это обстоятельство следует иметь в виду, поскольку оно тесно связано с понятием универсальности машины. Строго говоря, для возможности реализации произвольного алгоритма универсальная цифровая машина должна поместить запись (изображение) этого алгоритма в свою память. Поскольку изображения алгоритмов могут быть сколь угодно длинными, для действительной возможности реализации произвольных алгоритмов память машины должна быть бесконечной.

Имея в виду, однако, что бесконечную память нельзя реализовать ни в каком техническом устройстве, принято называть машину универсальной, если организация ее управления и набор операций таковы, что обеспечили бы возможность реализации любого алгоритма при условии неограниченности объема памяти.

Практически универсальность современных машин обеспечивается обычно тем, что, помимо быстродействующего (так называемого *оперативного*) запоминающего устройства, они снабжаются также относительно медленными (так называемыми *внешними*) запоминающими устройствами, способными обмениваться информацией с оперативным запоминающим устройством. Емкость внешней памяти (выполняемой чаще всего на магнитных лентах) может считаться практически неограниченной, что и определяет (при наличии возможности обмена кодами с оперативной памятью) практическую возможность выполнения на машине произвольного алгоритма.

Последовательность операций, выполняемая универсальной цифровой машиной, определяется закладываемой в ее память *программой*, представляющей собой упорядоченное конечное множество *приказов*, которые можно рассматривать как естественное обобщение приказов, употребляющихся при построении алгоритмической схемы Поста. В отличие от схемы Поста, в которой активная ячейка смещается при выполнении каждого следующего

приказа не более чем на один шаг вправо или влево, в универсальной цифровой машине предусматривается возможность произвольных изменений положения активной ячейки от приказа к приказу. С этой целью в каждый приказ вводятся номера одной или нескольких ячеек памяти, которые являются активными при выполнении данного приказа.

Номера ячеек памяти в универсальных цифровых машинах принято называть *адресами*. Число адресов в приказах современных универсальных цифровых машин (число ячеек памяти, являющихся активными при выполнении любого из этих приказов) обычно варьируется в пределах от 1 до 4. В соответствии с этим различаются *одноадресные, двухадресные, трехадресные и четырехадресные приказы*.

Рассмотрим сначала машины с четырехадресной *системой приказов*, т. е. такие машины, у которых *максимальная* адресность приказов равна четырем. Различные *типы* приказов соответствуют различным операциям, выполняемым машиной. Приказы в машине записываются обычно в виде двоичных кодов, которые могут храниться в памяти машины (как оперативной, так и внешней).

Будем предполагать, что в каждой ячейке памяти может содержаться либо один приказ, называемый также *командой*, или *командным словом*, либо одно информационное слово. Точно так же, как было сделано выше применительно к информационным словам, каждое командное слово (код приказа) можно при желании рассматривать как слово в любом конечном (не обязательно двоичном) алфавите.

Любое командное слово делится на *операционную* и *адресную* части. В операционной части стоит код операции, выполняемой во время действия приказа, который представляется этим командным словом. Все приказы одного и того же типа имеют одинаковые операционные части. В адресной части приказа размещаются адреса ячеек, которые являются активными во время действия приказа.

Операции, выполняемые универсальными цифровыми машинами, делятся обычно на несколько классов. К первому классу относятся *арифметические операции* — сложение, вычитание, умножение и деление. Четырехадресные приказы для выполнения арифметических операций имеют обычно такую структуру: в операционной части приказа стоит условный номер, обозначающий ту или иную операцию (например, один — сложение, два — вычитание, три — умножение, четыре — деление). Первые два адреса в приказе используются для указания адресов ячеек в памяти, хранящих числа, с которыми должна производиться операция, т. е. адреса слагаемых в случае операции сложения, адреса уменьшаемого и вычитаемого в случае операции вычитания и т. д. По третьему адресу приказа производится засылка результата выполнения операции (суммы, разности, произведения или частного). Наконец,

четвертый адрес приказа используется для указания ячейки памяти, в которой хранится приказ, выполняемый вслед за данным приказом.

Совершенно так же, как и в случае арифметических операций, строятся приказы для выполнения *логических операций*. Логические операции являются, как правило, двухместными операциями, выполняемыми *поразрядно*, т. е. отдельно для каждой пары соответствующих друг другу двоичных разрядов, участвующих в операции кодов. К ним относятся, например, *поразрядная конъюнкция* (логическое умножение) и поразрядная дизъюнкция (логическое сложение). Имеются также и одноместные логические операции над кодами. К таким операциям относятся правый и левый (логический) *сдвиги*, превращающие двоичный код $x_1, x_2 \ldots x_n$ в коды $0x_1x_2 \ldots x_{n-1}$ и $x_2x_3 \ldots x_n 0$ соответственно.

Особую роль играют так называемые *операции передачи управления*, служащие для изменения порядка выполнения приказов программы в зависимости от получаемых в процессе ее реализации результатов. Типичной операцией передачи управления (называемой также операцией *условного перехода*) является так называемая *операция условного перехода по точному совпадению слов*. Первые два адреса приказа, реализующего эту операцию, используются для указания ячеек памяти, из которых выбирается два сравниваемых между собой слова. В случае совпадения (равенства) этих слов следующий по порядку приказ извлекается из ячейки памяти, указанной по третьему адресу, а в случае несовпадения — по четвертому адресу приказа условного перехода. Возможны условные переходы и других видов, например условный переход по знаку разности двух слов или по знаку одного какого-нибудь слова (в последнем случае, разумеется, в приказе условного перехода достаточно иметь три, а не четыре адреса).

Память универсальных цифровых машин обычно устраивается таким образом, что при *выборке* (чтении) слова из какой-нибудь ячейки для производства той или иной операции происходит как бы раздвоение этого слова. Один его экземпляр поступает в соответствующее устройство для выполнения операции, а другой остается в ячейке, из которой была произведена выборка. При *записи* нового слова в ту или иную ячейку памяти содержавшаяся ранее в этой ячейке информация автоматически уничтожается (стирается).

Учитывая указанные свойства памяти универсальных цифровых машин, нетрудно заметить, что для выполнения произвольного алгоритма Поста достаточно пользоваться лишь операцией (алгебраического) сложения, одной из операций условного перехода, например операцией условного перехода по точному совпадению слов, и операцией остановки машины.

В самом деле, условимся оперировать лишь какими-либо двумя информационными словами p_0 и p_1, первое из которых отождествим с нулем, а второе — с единицей двоичного алфавита любого

данного алгоритма Поста A. Память рассматриваемой универсальной цифровой машины разделим на три части. Первая часть состоит всего из пяти ячеек: a_{-1}, a_0, a_1, b_0, b_1, в которые помещены числа -1, 0, 1, p_0,p_1; в ячейки второй части будет помещена программа, имитирующая программу (схему) алгоритма A; наконец, третья часть памяти машины M имитирует информационную ленту алгоритма A.

При работе алгоритма A над конкретным входным словом p, на котором этот алгоритм определен, исходная, промежуточная и конечная информации занимают лишь некоторую ограниченную (конечную) часть информационной ленты, поскольку алгоритм работает лишь конечное число шагов и на каждом шаге записывает информацию не более чем в одну новую ячейку. Поэтому, если память машины M достаточно велика, можно поместить в выделенную выше ее третью часть требуемый участок информационной ленты. Затруднения же с возможной недостаточностью объема памяти, в силу принятого выше предположения, не должны приниматься во внимание при решении вопроса о *принципиальной* представимости на машине тех или иных алгоритмов.

Перейдем к непосредственному моделированию приказов алгоритма Поста A приказами машины M.

Как отмечалось в § 4 гл. I, в алгоритмах Поста может встречаться шесть различных типов приказов. Приказ шестого типа (остановка) непосредственно моделируется соответствующим приказом машины M. Приказы первых двух типов (запись в рассматриваемую ячейку нуля и единицы) моделируются приказами машины M, осуществляющими *пересылку информации* из ячеек b_0 или b_1 в активную ячейку, указанную, например, по третьему адресу машинного приказа. Ясно, что такую пересылку можно осуществить приказом сложения, в первых двух адресах которого стоит пара адресов ячеек a_0 и b_0 или ячеек a_0 и b_1, а в третьем адресе — адрес активной ячейки (четвертый адрес, как и в алгоритме Поста, используется для указания адреса приказа, который должен выполняться вслед за действующим приказом).

Постовский приказ пятого типа моделируется, как нетрудно видеть, машинным приказом условного перехода по точному совпадению слов. Достаточно сравнить слово в ячейке b_1 со словом в активной ячейке и перейти к одному из двух приказов в зависимости от результатов этого сравнения.

Наконец, каждый постовский приказ q' третьего или четвертого типа моделируется с помощью группы машинных приказов, число которых равно числу приказов первого, второго и пятого типов в рассматриваемом алгоритме Поста A. Действительно, пусть r' — произвольный приказ первого, второго или пятого типа алгоритма A. В силу сказанного выше, он моделируется одним машинным приказом, который обозначим через r. Пусть постовский приказ q' смещает активную ячейку на одну единицу вправо. В случае машинной программы такое смещение может

осуществиться лишь за счет изменения на $+1$ адресов всех активных ячеек.

Такие ячейки, однако, имеются лишь в приказах машины, моделирующих постовские приказы первого, второго и пятого типов. За счет должной нумерации адресов можно, не нарушая общности, предполагать, что адреса активных постовских ячеек записываются в каком-либо определенном, скажем последнем, адресе приказа. Считая коды целыми двоичными числами, придем к выводу, что для изменения адреса активной ячейки в приказе r на $+1$ достаточно сложить код команды r с кодом $+1$, помещенным в ячейку a_1. Аналогично осуществляется также сдвиг активной ячейки влево.

Из сказанного ясно, что алгоритмическая универсальность любого программно управляемого цифрового автомата будет обеспечена, если с помощью выполняемых им операций можно осуществлять четыре операции:

1) *операцию пересылки содержимого любой ячейки памяти в любую другую ячейку памяти*;

2) *операцию сложения кода приказа, помещенного в любой ячейке памяти, с константами, изменяющими величину заданного (первого, второго, третьего или четвертого) адреса приказа на* $+1$ *или* -1;

3) *операцию условного перехода по точному совпадению слов*;

4) *операцию (безусловного) останова машины.*

В рассмотренном выше случае операции 1) и 2) обеспечивались одной и той же машинной операцией — операцией алгебраического сложения. Обычно же в универсальных цифровых машинах эти операции разделяются, вторая из них носит название *операции переадресации*, или *сложения команд.*

Разумеется, кроме указанных операций, в наборе операций всякой универсальной цифровой машины должны быть операции *ввода* и *вывода* информации из машины, а также (в случае наличия внешней памяти) операции, обеспечивающие двусторонний обмен информацией между оперативным и внешним запоминающими устройствами.

Рассмотрим вопрос о путях уменьшения адресности приказов. Нетрудно понять прежде всего, что четвертый адрес, употребляющийся для указания следующего выполняемого приказа, можно сэкономить за счет такого расположения приказов в памяти машины, чтобы адрес приказа, выполняемого вслед за любым данным приказом p, был всегда на единицу больше адреса самого приказа p. Иными словами, употребление четвертого адреса становится ненужным при условии, что порядок расположения приказов в ячейках памяти соответствует порядку их выполнения машиной.

Нарушение обычного (естественного) порядка следования приказов может произойти лишь в случае, если очередной выполняемой командой была команда условного перехода. Как отмеча-

лось выше, в четырехадресной системе приказов один адрес используется для указания следующего приказа при *невыполнении* условия, по которому совершается условный переход, а второй — для указания следующей команды при *выполнении* этого условия. При замене четырехадресной системы приказов трехадресной системой принимается обычно, что в первом случае (при невыполнении условия) после приказа условного перехода выполняется приказ, записанный в следующей по порядку ячейке памяти, и лишь во втором случае при выполнении условия один из адресов используется для указания адреса приказа, который должен выполняться следующим.

Из сказанного следует, что четвертый адрес может быть сделан излишним не только в обычных приказах, не меняющих дальнейшего порядка выполнения приказов, но и в приказах условного перехода. Возникающая *трехадресная система приказов* называется обычно системой *с естественным следованием приказов*, в отличие от описанной ранее четырехадресной системы *с принудительным следованием приказов*. Преимущество последней системы заключается в большей свободе, предоставляемой ею в вопросе размещения последовательности команд, управляющих работой машины, в запоминающем устройстве. Преимуществом трехадресной системы является упрощение структуры приказа.

Дальнейшее уменьшение числа адресов в приказах может быть достигнуто за счет фиксации некоторой вспомогательной ячейки памяти, обычно отделенной конструктивно от других ячеек запоминающего устройства машины. После фиксации такой ячейки открывается простой путь, позволяющий уменьшить число адресов в приказе до естественного минимума, т. е. до одного адреса. Поясним указанный путь на примере операции сложения, которая требует наличия трех адресов: адреса первого слагаемого a_1, адреса второго слагаемого a_2 и адреса a_3, по которому следует заслать сумму. С помощью фиксированной вспомогательной ячейки b_0 эту трехадресную операцию можно выполнить путем последовательного выполнения трех одноадресных операций — операции пересылки числа из ячейки a_1 в (фиксированную) ячейку b_0, операции сложения числа, содержащегося в ячейке a_2, с числом в ячейке b_0 с последующей записью результата в ячейку b_0 и, наконец, операции пересылки числа из ячейки b_0 в ячейку a_3. Поскольку меняться при этом могут лишь адреса a_1, a_2, a_3, а адрес b_0 раз и навсегда фиксирован, все три указанные операции действительно реализуются с помощью одноадресных приказов.

Описанный способ приводит к *одноадресной системе приказов*, употребляющейся во многих современных универсальных электронных цифровых машинах. Имея одноадресную систему, нетрудно построить также *двухадресную систему приказов*. Второй адрес при этом может употребляться как для указания адреса следующего приказа (двухадресная система с принудительным следованием приказов), так и для указания адресов числовых кодов

(информационных слов), с которыми производятся операции (двухадресная система с естественным следованием команд).

Всякое устройство, обладающее разбитой на отдельные ячейки дискретной памятью, работой которого можно управлять с помощью последовательности размещаемых в некоторых из этих ячеек командных слов — приказов, называется программным автоматом, а сама указанная последовательность приказов — программой работы автомата.

Если набор операций (типов приказов), выполняемых программным автоматом, позволяет составить из них операцию пересылки информационных слов из любой ячейки памяти в любую другую ячейку памяти, операцию переадресации (изменения адресов в приказах) на ± 1, *операцию условного перехода и останов машины и если в качестве программы автомата может быть задана любая конечная последовательность операций из этого набора, то такой автомат называется универсальным программным автоматом.*

С точностью до ограничений, вводимых фиксированным объемом памяти, универсальный программный автомат способен воспроизводить *любой* алгоритм при условии подходящего кодирования букв его входного и выходного алфавитов. Этот вывод относится не только к обычным алгоритмам, но и к случайным и самоизменяемым алгоритмам (см. § 5 настоящей главы).

§ 2. Структура современных универсальных программных автоматов

Современные универсальные программные автоматы состоят из пяти различных основных устройств — *запоминающего устройства* (ЗУ), *арифметического устройства* (АУ), *устройства управления* (УУ), *вводного устройства* (ввод) и *выводного устройства* (вывод). Как отмечалось в предыдущем параграфе, запоминающее устройство (память) служит для запоминания и хранения программы работы автомата, а также начальной, конечной и промежуточной информаций. Устройство ввода служит для ввода в память автомата программы и начальной информации (условия задачи), с помощью же устройства вывода осуществляется вывод из памяти заключительной информации (ответа на задачу, поставленную автомату).

Арифметическое устройство, как показывает само его название, служит для выполнения *арифметических* операций. Однако обычно с помощью арифметического устройства выполняются и другие операции, например логические. В связи с этим точнее было бы называть арифметическое устройство *операционным* устройством. Однако не будем отступать в названии АУ от установившихся традиций.

Наконец, устройство управления объединяет и координирует работу всех остальных устройств универсального программного автомата, осуществляет выборку, расшифровку и организацию

исполнения составляющих программу приказов. Устройства управления в современных универсальных программных автоматах строятся по циклическому принципу. Суть этого принципа состоит в том, что работа автомата во времени разбивается на естественные интервалы, называемые *рабочими циклами*, в течение которых повторяется примерно одна и та же последовательность элементарных операций.

Определение начала и конца рабочего цикла до известной степени условно, поскольку возможен их одновременный сдвиг в любом направлении. Будем предполагать, что рабочий цикл начинается тогда, когда в устройство управления уже передана подлежащая выполнению команда (приказ). В течение цикла осуществляется выполнение этой команды: ее операционная часть используется для настройки на производство определенных операций как самого устройства управления, так и арифметического устройства. Адресная часть команды употребляется для возбуждения соответствующих ячеек ЗУ с целью извлечения или записи в них той или иной информации. В многоадресных системах команд расшифровка различных адресов осуществляется последовательно. Рабочий цикл заканчивается извлечением из ЗУ и передачей в УУ кода следующего приказа, подлежащего выполнению.

Заметим, что устройство управления (УУ) не только передает информацию другим устройствам, но и само воспринимает от них информацию: от ЗУ — код команды, а от АУ — результат проверки условия, определяющего переход к той или иной из двух команд, следующих за командой условного перехода, и некоторые другие сигналы. Как отмечалось в предыдущем параграфе, кроме *оперативного запоминающего устройства* (ОЗУ), современные универсальные программные автоматы снабжаются еще и *внешним запоминающим устройством* (ВЗУ), более медленно действующим, но зато значительно более емким по сравнению с ОЗУ. *Блок-схема* универсального программного автомата, определяющая взаимодействие (обмен информацией) между его основными устройствами, представлена на рис. 18.

Для детализации блок-схемы рассмотрим более подробно структуру составляющих ее отдельных устройств и прежде всего структуру ОЗУ, АУ и УУ. Существенными составными частями всех трех указанных устройств являются так называемые *регистры*. Регистр представляет собой ячейку памяти, предназначенную для хранения одного информационного или командного слова. Однако, в отличие от обычных ячеек памяти, доступ к которым оказывается возможным лишь после осуществления достаточно сложной предварительной коммутации (переключения), регистры являются особо доступными ячейками памяти, входы и выходы которых непосредственно подсоединяются к передающим информацию цепям.

В зависимости от способа функционирования этих цепей универсальные программы-автоматы (универсальные цифровые машины) делятся на два больших класса: *параллельные* и *последо-*

вательные машины. В параллельных машинах (автоматах) при передаче кода из регистра в регистр все разряды этого кода передаются одновременно, а в последовательных машинах — последовательно, один за другим. Ясно, что параллельные машины при прочих равных условиях будут более быстродействующими, чем последовательные, однако они требуют большого числа (равного числу разрядов в машинных кодах слов) параллельных каналов для передачи информации между регистрами, тогда как в последовательных машинах можно ограничиться одним таким каналом.

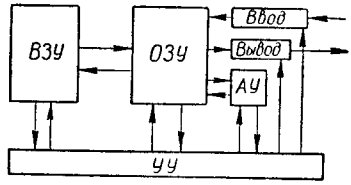

Рис. 18.

Арифметическое устройство современных универсальных электронных цифровых машин обычно состоит из трех регистров, один из которых обладает свойством суммировать передаваемые на него числовые коды и называется поэтому *сумматором*. Числовые коды, которыми оперируют арифметические устройства, представляют собой числа разных знаков, а суммирование, о котором идет речь, понимается как алгебраическое сложение (с учетом знаков). Операция суммирования в параллельных машинах выполняется обычно за два *элементарных такта* машины. Под элементарным тактом здесь подразумевается промежуток времени между двумя последовательными *тактирующими импульсами*, подаваемыми в АУ. Чаще всего источником тактирующих импульсов служит общий для всей машины синхронизирующий генератор, составляющий часть ее устройства управления. Существуют и другие способы организации элементарных тактов, принятые в так называемых *асинхронных* машинах. Подробно эти способы объяснены в руководствах по электронным цифровым машинам.

Операции, выполняемые в различных частях универсального программного автомата в течение одного элементарного такта, принято называть *микрооперациями*. Более сложные операции, выполняемые за несколько элементарных тактов, реализуются с помощью набора микроопераций, называемого *микропрограммой* данной операции. Микропрограмма операции суммирования в арифметических устройствах параллельных машин состоит обычно из двух микроопераций: микрооперации поразрядного сложения и микрооперации, с помощью которой реализуются возникшие в результате поразрядного сложения переносы из одних разрядов в другие. При этом предполагается, что одно из слагаемых было предварительно установлено на *сумматоре*, а другое — на одном из регистров АУ.

Можно, впрочем, построить АУ и таким образом, что после установки слагаемых на регистре и на сумматоре сложение будет осуществляться в результате лишь одной микрооперации — передачи второго слагаемого из регистра в сумматор. Будем считать в дальнейшем, что АУ, о которых будет идти речь, устроены именно

таким образом. Сумматоры подобных АУ принято называть *накапливающими*, поскольку они обладают возможностью накапливать сумму любого числа слагаемых в результате последовательной передачи на сумматор всех слагаемых одного за другим.

Используя то обстоятельство, что сумматор осуществляет алгебраическое сложение (с учетом знаков слагаемых), нетрудно на таком сумматоре организовать также операцию вычитания, осуществляя передачу (из регистра на сумматор) кода, вычитаемого как обычного слагаемого, но с измененным знаком. В набор микроопераций универсальных цифровых машин вводятся поэтому микрооперации не только обычной (прямой) передачи числовых кодов, но и передачи кода с изменением его знака. Необходимо также предусмотреть микрооперации *очистки регистров*, в результате выполнения которых на очищаемых регистрах должны устанавливаться числовые коды, являющиеся изображением числа 0.

Для выполнения операций умножения и деления при естественном способе кодирования чисел перечисленных микроопераций недостаточно. Поэтому, наряду с уже описанными микрооперациями, в набор микроопераций универсальных цифровых машин вводятся еще микрооперации *левого и правого сдвига* на регистрах. При выполнении микрооперации левого сдвига установленный на регистре числовой код $x_1\ x_2 \ldots x_n$ заменяется кодом $x_2\ x_3 \ldots x_n 0$, а при выполнении микрооперации правого сдвига —кодом $0\ x_1 x_2 \ldots \ldots x_{n-1}$. Знак кода (здесь специально не обозначенный) сохраняет свое значение. Разряды кода, расположенные справа, как обычно, представляют здесь младшие разряды числа, а разряды, расположенные слева, — его старшие разряды. Поэтому говорят также, что при правом сдвиге числовой код сдвигается в сторону младших, а при левом сдвиге — в сторону старших разрядов.

В качестве сигналов обратной связи, передавамых из арифметического устройства в устройство управления, выбираются обычно сигналы о знаке числового кода, находящегося в сумматоре, и о цифре самого младшего разряда числового кода, находящегося в одном из регистров АУ, который обозначим буквой P_2. Этот регистр называется также *регистром множителя*. Второй регистр АУ не имеет обратных связей с устройством управления. Он называется обычно *регистром множимого* и обозначается буквой P_1.

В запоминающем устройстве, как правило, имеются лишь два регистра, один из которых называется *регистром числа*, а другой — *регистром адреса*. На регистре адреса запоминается адрес ячейки ЗУ, с которой предстоит работа (запись или чтение кода), а на регистре числа — числовой код, выбираемый из ЗУ или направляемый туда для хранения. Кроме того, имеются обычно еще два канала для приема сигналов от УУ о характере предстоящей работы (запись или чтение). Передача импульса по одному из этих каналов (каналу записи) приводит к тому, что код, установленный в регистре числа, запоминается (записывается) в ячейке ЗУ, адрес ко-

торой совпадает с кодом, установленным в регистре адреса. Передача импульса по второму каналу (каналу чтения) приводит к тому, что код из ячейки ЗУ, адрес которой установлен на регистре адреса, передается на (предварительно очищенный) регистр числа.

Описанные две микрооперации ЗУ называются соответственно *микрооперациями записи и чтения из ЗУ*. Кроме этих микроопераций, в ЗУ предусматриваются микрооперация *очистки регистров ЗУ*, а также микрооперации передачи кодов из регистров АУ и ЗУ в регистры числа и адреса и обратные передачи из ЗУ в АУ и УУ. Говоря о передаче кода из регистра в регистр, всегда будем понимать, если не оговорено противное, обычную передачу, т. е. передачу кода без изменения его знака.

Разберем структуру устройства управления, следуя схеме микропрограммного управления, описанной впервые Уилксом и Стринжером [75]. Микропрограммное устройство управления имеет в своем составе два регистра, называемых соответственно *регистром команд* и *регистром микроопераций*. Регистр команд служит для запоминания текущей выполняемой команды (приказа). В соответствии с принятой структурой приказов регистр команд подразделяется на несколько регистров — *регистр операций*, *регистр первого адреса*, *регистр второго адреса* и т. д. При описании микропрограмм иногда бывает удобно оперировать регистром команд как целым, а иногда—разбивать на указанные составные части.

Регистр микроопераций, называемый также иначе *регистром микрокоманд*, служит для запоминания кода выполняемого в каждый данный момент приказа микропрограммы (микрокоманды), т. е. кода, обозначающего совокупность выполняемых в данный момент микроопераций.

Кроме регистров команд и микроопераций, в универсальных программных автоматах с *естественным порядком следования приказов* (см. § 1 данной главы) имеется еще один регистр, называемый *счетчиком команд*. При посылке импульса на специальный вход этого регистра происходит увеличение на единицу установленного в нем перед этим числового кода. Если счетчик команд был предварительно очищен, то он будет, очевидно, осуществлять счет приходящих на его вход импульсов. Отсюда и происходит само название этого регистра. Счетчик команд служит для хранения адресов команд. В процессе выполнения очередного приказа, отличного от приказа условного перехода, происходит увеличение содержимого счетчика команд на единицу и выборка нового приказа из ЗУ в соответствии с полученным таким образом адресом.

Увеличение содержимого счетчика команд на единицу является одной из микроопераций устройства управления. Из других микроопераций, предусматриваемых в УУ, отметим очистку регистров, передачу кодов из регистра числа ЗУ в регистр команд, передачу кодов из регистров адреса УУ (первого, второго и др.) в регистр адреса ЗУ, передачу кода из регистра команд в сумматор (для осуществления операции переадресации) и передачу

кода из одного из регистров адреса УУ (обычно из третьего) в счетчик команд (при выполнении операции условного перехода). Для выполнения логических операций, о которых упоминалось в предыдущем параграфе, в число микроопераций арифметического устройства вводится ряд новых микроопераций.

Что же касается собственно устройства микропрограммного управления, то в схеме Уилкса и Стринжера оно включает, кроме

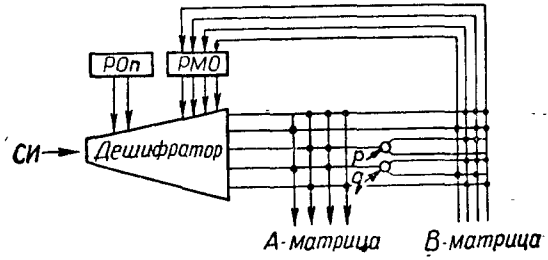

Рис. 19.

упомянутого выше регистра микроопераций, так называемый *дешифратор микроопераций* и две *диодные матрицы**, называемые *А*-матрицей и *В*-матрицей. Упрощенная условная схема устройства микропрограммного управления изображена на рис. 19. На этом рисунке буквами *РОп* обозначен регистр операций (составная часть регистра команд УУ), а *РМО* — регистр микроопераций. Точками обозначены места включения диодов, соединяющих горизонтальные шины матриц с вертикальными шинами. Назначение диодов состоит в том, чтобы пропускать импульсы в прямом направлении (от горизонтальных шин к вертикальным) и препятствовать их прохождению в обратном направлении. При этом условии импульс, поданный на любую горизонтальную шину *D*, перейдет на те и только те вертикальные шины, с которыми данная шина *D* соединена диодами. Если соединение шин осуществляется непосредственно, возможны ложные пути прохождения импульсов, не предусмотренные конструктором. Например, при подаче импульса на вторую сверху горизонтальную шину *А*-матрицы импульс появился бы не только на первой слева вертикальной шине, но и на третьей слева, пройдя на нее через вторую снизу горизонтальную шину. Наличие диодов исключает возможность образования подобных ложных путей, поскольку обратный переход импульсов с вертикальных шин на горизонтальные не возможен.

Назначение изображенного на рис. 19 дешифратора состоит в том, чтобы передавать поступающий на его вход очередной импульс синхронизирующего генератора СИ в точности на одну из гори-

* Диодные матрицы представляют собой две системы проводов, называемых обычно шинами, часть из которых соединена между собой диодами, т. е. элементами, пропускающими ток лишь в одном направлении.

зонтальных шин A-матрицы, однозначно определяемую кодами, установленными в рассматриваемый момент времени на регистрах операций и микроопераций. Некоторые из горизонтальных шин A-матрицы переходят в одну, а некоторые—в две горизонтальные шины B-матрицы. В последнем случае переход импульса с горизонтальной шины A-матрицы на одну из двух горизонтальных шин B-матрицы определяется сигналом обратной связи (на рис. 19—сигналом p или q), поступающим из арифметического устройства.

Переходя на вертикальные шины B-матрицы (определяемые способом включения диодов), импульсы поступают в регистр микроопераций, меняя установленный в нем ранее код. Поэтому следующий импульс СИ, пройдя через дешифратор, перейдет на новую горизонтальную шину. Подсоединяя вертикальные шины A-матрицы к соответствующим устройствам машины так, чтобы передача импульса по каждой из вертикальных шин вызывала выполнение в точности одной микрооперации, получаем возможность с помощью описанного процесса осуществлять любые конечные последовательности микроопераций (микропрограммы), совмещая при этом некоторые из микроопераций в одной микрокоманде.

Записывая микропрограммы для различных операций, которые должна выполнять машина, определяем способ включения диодов в A-матрице и в B-матрице, а следовательно, и конструкцию всего устройства микропрограммного управления При выполнении микропрограммы любой машинной операции содержимое регистра операций остается неизменным, в то время как содержимое регистра микроопераций изменяется с каждым новым элементарным тактом. Лишь в самом конце выполнения операции, после выборки из памяти нового приказа, изменяется и содержимое регистра операций, после чего начинается выполнение микропрограммы следующего рабочего цикла машины.

Поскольку конструкция устройства управления и даже всей машины в значительной мере определяется выбором операций и микропрограмм для них, рассмотрим конкретные примеры микропрограмм для наиболее употребительных машинных операций. При этом для определенности будем считать, что машина, о которой идет речь, является трехадресной параллельной универсальной цифровой машиной с естественным порядком следования команд. Буквой p условимся обозначать функцию знака числа ($\pm$ 1), находящегося в сумматоре, а q — значение младшего разряда числа в регистре множителя P_2 (предполагается, что машина оперирует n-разрядными правильными дробями в двоичной системе счисления).

Микропрограмма сложения

1. Очистка регистров АУ и ЗУ.
2. Передача кода из регистра первого адреса УУ в регистр адреса ЗУ.

3. Чтение в ЗУ.
4. Передача кода из регистра числа ЗУ в сумматор АУ.
5. Очистка регистров ЗУ.
6. Передача кода из регистра второго адреса УУ в регистр адреса ЗУ.
7. Чтение в ЗУ.
8. Передача кода из регистра числа ЗУ в сумматор АУ (чаще всего через регистр АУ P_2).
9. Очистка регистров ЗУ.
10. Передача кода из сумматора АУ в регистр числа ЗУ.
11. Передача кода из регистра третьего адреса УУ в регистр адреса ЗУ.
12. Запись в ЗУ.
13. Очистка регистров ЗУ.
14. Увеличение содержимого счетчика команд на единицу.
15. Передача кода из счетчика команд в регистр адреса ЗУ.
16. Чтение в ЗУ.
17. Очистка регистра команд.
18. Передача кода из регистра числа ЗУ в регистр команд.

Некоторые из микроопераций, составляющих описанную микропрограмму, могут быть совмещены во времени, за счет чего может быть сокращено время рабочего цикла и увеличено быстродействие машины. Примерами таких совмещаемых микроопераций могут служить микрооперации 16 и 17 или микрооперации 10 и 11.

Микропрограмма умножения на положительное число

1. Очистка регистров АУ и ЗУ.
2. Передача кода из регистра первого адреса УУ в регистр адреса ЗУ.
3. Чтение в ЗУ.
4. Передача кода из регистра числа ЗУ в регистр P_1 АУ.
5. Очистка регистров ЗУ.
6. Передача кода из регистра второго адреса УУ в регистр адреса ЗУ.
7. Чтение в ЗУ.
8. Передача кода из регистра числа ЗУ в регистр P_2 АУ.
9. (1) Передача кода из регистра P_1 в сумматор, если $q = 1$, и переход к следующей микрооперации без передачи числа, если $q = 0$.
10. (1) Сдвиг вправо на сумматоре.
11. (1) Сдвиг вправо на регистре P_2.
9. (2) То же самое, что 9(1).
10. (2) То же самое, что 10 (1).
11. (2) То же самое, что 11 (1).

. . . . . .

. . . . . .

9. (n) То же самое, что 9 (1).
10. (n) То же самое, что 10 (1).
11. (n) То же самое, что 11 (1).
12. Очистка регистров ЗУ.
13. Передача кода из регистра третьего адреса УУ в регистр адреса ЗУ.
14. Передача кода из сумматора в регистр числа ЗУ.
15. Запись в ЗУ.
16. Очистка регистров в ЗУ.
17. Прибавление единицы к содержимому счетчика команд.
18. Передача кода из счетчика команд в регистр адреса ЗУ.
19. Чтение в ЗУ, очистка регистра команд.
20. Передача кода из регистра числа ЗУ в регистр команд.

В микропрограмме умножения, в отличие от микропрограммы сложения, существенно используется возможность раздвоения в порядке следования микроопераций в зависимости от сигнала обратной связи q, поступающего из АУ. Нетрудно понять, что последовательное выполнение микроопераций 9, 10, 11 эквивалентно выполнению обычного, хорошо известного алгоритма умножения с округлением применительно к двоичной системе счисления.

В самом деле, при описанной методике сумматор АУ служит для запоминания сумм частичных произведений множимого на отдельные разряды множителя. Такое запоминание осуществляется с точностью до младших разрядов, отбрасываемых при сдвиге вправо содержимого сумматора. Каждый раз множимое добавляется или не добавляется к частичной сумме, хранящейся на сумматоре, в зависимости от того, единице или нулю равен младший разряд сдвинутого вправо множителя. Ясно, что эта процедура вместе с предварительными сдвигами на сумматоре и регистре P_2 означает добавление произведения множимого на очередной (справа) разряд множителя в имевшуюся ранее сумму аналогичных (частичных) произведений, как и требуется в обычном алгоритме умножения. Округление до числа значащих цифр, содержащихся в сомножителях, осуществляется в результате сдвигов на сумматоре, приводящих к уничтожению младших разрядов произведения. В случае умножения не только на положительные, но и на отрицательные числа в микропрограмму вводится добавочная микрооперация формирования знака произведения.

Микропрограмма условного перехода по неравенству

1. Очистка регистров АУ и ЗУ.
2. Передача кода из регистра первого адреса УУ в регистр адреса ЗУ.
3. Чтение в ЗУ.
4. Передача кода из регистра числа ЗУ в сумматор АУ.

5. Очистка регистров ЗУ.

6. Передача кода из регистра второго адреса УУ в регистр адреса ЗУ.

7. Чтение в ЗУ.

8. Передача кода из регистра числа ЗУ в регистр АУ.

9. Передача кода с изменением знака из регистра в сумматор.

10. Если содержимое сумматора $s > 0$, то добавление единицы к содержимому счетчика команд, а если $s \leqslant 0$, то передача кода из регистра третьего адреса УУ в счетчик команд.

11. Очистка регистров в ЗУ.

12. Передача кода из счетчика команд в регистр адреса ЗУ.

13. Чтение в ЗУ.

14. Очистка регистра команд.

15. Передача кода из регистра числа ЗУ в регистр команд.

Описанная микропрограмма осуществляет сравнение числа A_1, записанного по первому адресу команды, с числом A_2, записанным по ее второму адресу. Если окажется, что $A_1 > A_2$, то управление передается следующей по порядку команде. Если же $A_1 \leqslant A_2$, то следующая команда извлекается по третьему адресу текущей команды. Нетрудно понять, что из двух операций условного перехода по неравенству, переставляя местами числа A_1 и A_2, можно образовать операцию условного перехода по точному совпадению слов, описанную в предыдущем параграфе.

Описание основных принципов организации алгоритмического процесса в универсальных электронных цифровых машинах дает общее представление о так называемой *блочной структуре* подобных машин. При реальном проектировании электронных программных автоматов этап блочного синтеза является лишь исходной точкой для разработки тех или иных схемных решений. Выбор этих решений осуществляется на основе теории автоматов и теории комбинационных схем, изложенных в гл. I, III.

§ 3. Понятие о программировании

Программированием называется запись того или иного алгоритма в виде конечной последовательности приказов (команд) универсального программного автомата. Такая последовательность называется программой работы данного автомата. Введенная в автомат вместе с начальными данными (входным словом алгоритма), она заставляет автомат выполнить работу рассматриваемого алгоритма, т. е. преобразовать заданное входное слово (входную информацию) в соответствующее ему выходное слово (выходную информацию). Универсальность набора операций современных программных автоматов обусловливает возможность программирования любого алгоритма при условии, что мы пренебрегаем ограничениями, накладываемыми конечностью объема памяти автомата.

Чтобы понять суть проблем, встающих при программировании, рассмотрим сначала какой-нибудь простой частный пример. Предположим, что нам требуется вычислить сумму вида $\sum_{k=1}^{m} \frac{1}{k^2}$, где m некоторое фиксированное натуральное число. Будем составлять программу решения этой задачи применительно к трехадресной универсальной цифровой машине с естественным порядком следования команд, в набор операций которой включены все четыре арифметические операции (сложение, вычитание, умножение и деление), операция условного перехода по точному совпадению слов и операция останова машины.

Предположим для простоты, что отсутствует необходимость ввода и вывода информации в ЗУ машины. Иначе говоря, начальная информация и программа, которая будет составлена, предполагаются введенными в должные ячейки памяти, а решение задачи получается в выделенных для этой цели ячейках ЗУ (в рассматриваемом случае — в одной ячейке) после остановки машины.

Для решения поставленной задачи введем следующие обозначения:

$$x_k = k;$$
$$y_k = x_k^2 = k^2;$$
$$z_k = \frac{1}{y_k} = \frac{1}{k^2};$$
$$s_k = z_1 + z_2 + \ldots + z_k = s_{k-1} + z_k.$$

Будем считать, что для хранения величин x_k, y_k, z_k и s_k отведены какие-либо четыре ячейки памяти, называемые *рабочими ячейками*. Процесс вычисления искомой суммы s_k естественно разбивается на m совершенно идентичных этапов, на каждом из которых вычисляется соответствующее значение частичной суммы s_k ($k = 1, 2, \ldots, m$). Вычисления начинаются со значения $k = 1$ и $s_0 = 0$ и могут быть записаны в виде следующей схемы:

1) $y_k = x_k^2$;
2) $z_k = \frac{1}{y_k}$;
3) $s_k = s_{k-1} + z_k$;
4) $x_{k+1} = x_k + 1$;
5) если $x_{k+1} = m + 1$, то перейти к следующему приказу; если $x_{k+1} \neq m + 1$, то вернуться к приказу 1;
6) останов.

Выписанная схема представляет собой фактически уже искомую программу, в приказах которой истинные адреса рабочих ячеек заменены символическими обозначениями x_k, y_k, z_k, s_k, называемыми *символическими адресами*. Фиксируя истинные адреса этих ячеек, а также ячеек, содержащих фигурирующие в приведенной программе константы 1 и $m + 1$, нетрудно перейти от программы

с символическими адресами к истинной программе в трехадресных приказах. Будем предполагать при этом, что приказ условного перехода обеспечивает естественное следование приказов при совпадении сравнимых слов (расположенных по первому и второму адресам приказа условного перехода) и передает управление по третьему адресу при несовпадении сравниваемых слов.

Пусть адреса рабочих ячеек x_k, y_k, z_k и s_k равны соответственно 1, 2, 3 и 4; адреса ячеек, содержащих константы 1 и $m+1$, равны 5 и 6, а адрес ячейки, содержащей первый приказ программы, равен 7. В таком случае искомая программа (в истинных адресах) записывается в виде следующей последовательности приказов:

1) умножение 1, 1, 2;
2) деление 5, 2, 3;
3) сложение 4, 3, 4;
4) сложение 1, 5, 1;
5) условный переход 1, 6, 7;
6) останов.

В ячейку с адресом 1 первоначально должно быть помещено число, равное единице, а в ячейку с адресом 4 — число, равное нулю. Первоначальное заполнение рабочих ячеек с адресами 2 и 3, очевидно, безразлично, поскольку нужное заполнение этих ячеек производится первым и вторым приказами программы. Напомним, что память машины предполагается устроенной таким образом, чтобы при записи того или иного слова в любую ячейку автоматически уничтожалось прежнее содержимое этой ячейки. После остановки машины (выполнения шестого приказа) искомое значение суммы s_m получается в четвертой ячейке памяти.

Нетрудно понять, что в приведенной программе ячейку с адресом 2 можно использовать не только для хранения величины y_k, но и величины z_k. Действительно, величина y_k используется исключительно лишь для вычисления значения величины z_k, поэтому после вычисления z_k полученное значение можно послать в ячейку, где ранее хранилась величина y_k, без опасения нарушить правильность последующих вычислений. Существуют общие приемы, позволяющие автоматизировать подобный процесс *экономии рабочих ячеек*.

В качестве второго примера рассмотрим программирование задачи вычисления скалярного произведения двух n-мерных векторов $A=(a_1, a_2, \ldots, a_n)$ и $B=(b_1, b_2, \ldots, b_n)$, т. е. вычисление суммы s-парных произведений их соответствующих компонент: $s=a_1b_1+a_2b_2+\ldots+a_nb_n$. Предположим, что компоненты вектора A расположены в ячейках с адресами $a+1$, $a+2$, ..., $a+n$, а компоненты вектора B — в ячейках с адресами $b+1$, $b+2$, ..., $b+n$, где a и b — некоторые фиксированные натуральные числа, выбранные так, чтобы массивы ячеек, отведенные для хранения компонент векторов A и B, не пересекались между собой.

Для разнообразия составим программу вычисления скалярного произведения применительно к одноадресной машине с естественным порядком следования команд. Приказы, реализующие арифметические операции, при этом понимаются так, что соответствующая операция выполняется с парой чисел, первое из которых находится в сумматоре АУ, а второе — в ячейке, адрес которой указан в приказе. Результат операции остается в сумматоре. Для выполнения арифметических операций с подобными приказами необходимо иметь еще приказы, осуществляющие обмен кодами между сумматором АУ и ячейкой ЗУ, адрес которой указывается в соответствующем приказе.

Команда условного перехода может быть выполнена разнообразными способами. Предположим, что имеет место условный переход по нулю в сумматоре: если в сумматоре АУ при выполнении команды условного перехода окажется записанным число 0, то управление передается следующей по порядку команде, в противном же случае выборка следующей команды осуществляется по адресу, указанному в команде условного перехода. Предполагается для простоты, что i-я команда программы будет храниться в ячейке с адресом i ($i = 1, 2, \ldots$). Отводя для хранения константы переадресации (числа, равного единице) ячейку с адресом c, для хранения числа n (размерности векторов A и B) — ячейку с адресом a, а для хранения частичной суммы $s_k = a_1b_1 + a_2b_2 + \ldots + a_k b_k$ — ячейку с адресом s, придем к следующей программе:

1) засылка в сумматор из ЗУ, $a + 1$;
2) умножение, $b + 1$;
3) сложение, s;
4) засылка из сумматора в ЗУ, s;
5) засылка в сумматор, 1;
6) сложение, c;
7) засылка из сумматора в ЗУ, 1;
8) засылка в сумматор, 2;
9) сложение, c;
10) засылка из сумматора в ЗУ, 2;
11) засылка в сумматор, t;
12) сложение, c;
13) засылка из сумматора в ЗУ, t;
14) засылка в сумматор, a;
15) вычитание, t;
16) условный переход по нулю в сумматоре, 1;
17) останов.

Ячейка с адресом t, фигурирующая в построенной программе, представляет собой так называемый *программный счетчик* числа циклов. Приказами 11, 12 и 13 осуществляется увеличение содержимого этой ячейки на единицу. Если в начале вычислений в ячейке t хранилось число, равное нулю, то после выполнения n циклов (повторений группы приказов 1—13) в нем будет храниться число,

равное n. Тогда в результате вычитания, производимого приказами 14 и 15, в сумматоре получится нуль, и последующая команда условного перехода передаст управление команде 17, которая остановит машину.

Вплоть до этого момента команда условного перехода будет передавать управление первой команде, в результате чего цикл вычислений будет повторяться. Однако это повторение будет не буквальным, поскольку с помощью команд 5, 6, 7 и 8, 9, 10 осуществляется увеличение адресов первой и второй команд программы на единицу. Поэтому команды 1 и 2 приведут к образованию произведения новой пары компонент $a_k b_k$ векторов A и B, команды 3 и 4 — к вычислению и запоминанию в ячейке s нового значения частичной суммы $s_k = s_{k-1} + a_k b_k$.

Для правильного понимания выписанной программы необходимо заметить, что операция засылки в сумматор какого-нибудь кода предполагает предварительную очистку сумматора, а после засылки кода из сумматора в ЗУ сумматор также автоматически очищается. Кроме того, необходимо, чтобы в начале работы в ячейку t было записано число, равное нулю.

Описанный способ выполнения переадресации (изменения адресов команд), когда подвергаются изменению коды самих команд, не удобен. Во-первых, он приводит к удлинению программы (особенно заметному в случае одноадресных машин), а во-вторых, — это самое главное — в результате его применения первоначально заданная программа изменяется и оказывается не пригодной к дальнейшему употреблению без предварительного *восстановления* первоначальных значений адресных частей команд. Такое восстановление вносит дальнейшие усложнения в программу и требует дополнительных ячеек ЗУ для хранения исходных адресов. Поэтому для большинства современных универсальных цифровых машин предпочитают другой способ переадресации, связанный с употреблением так называемых *регистров модификации адресов*, или *индекс-регистров*.

Индекс-регистры входят в состав устройства управления универсального программного автомата и обладают тем свойством, что в процессе выполнения любой команды содержимое того или иного из них автоматически прибавляется к тем адресам команды, которые снабжены специальным *признаком*, соответствующим этому индекс-регистру.

При наличии одного индекс-регистра I в трехадресной системе команд программа вычисления скалярного произведения векторов может быть записана всего лишь пятью командами (обозначения адресов ячеек те же, что и в предыдущей программе):

1) прибавление 1 к содержимому индекс-регистра I;
2) умножение, $a(I)$, $b(I)$, p;
3) сложение, p, s, s;
4) условный переход, I, a, 1;
5) останов.

В этой программе использован условный переход по точному совпадению кодов в индекс-регистре I и в ячейке a (где записана размерность n векторов A и B). В случае совпадения этих кодов управление передается следующей по порядку (пятой) команде, а в случае несовпадения — первой команде программы.

В течение всего времени работы программа сохраняет свой первоначальный вид, изменяется лишь содержимое индекс-регистра I. В случае необходимости в программу может быть включена специальная команда очистки индекс-регистра, устанавливающая его в нуль.

При программировании более сложных алгоритмов, например алгоритма умножения вектора на матрицу или матрицы на матрицу, возникает необходимость в использовании нескольких индекс-регистров для запоминания констант переадресации, изменяющихся различными шагами. Рассмотрим в виде примера умножение n-мерного вектора $B = (b_1, b_2, \ldots, b_n)$ на матрицу A n-го порядка с элементами $a_{ik} (1 \leqslant i \leqslant n,\ 1 \leqslant k \leqslant n)$.

Предположим, что последовательные компоненты вектора B расположены в ячейках памяти с адресами $b+1$, $b+2$, ..., $b+n$, а элементы a_{ik} матрицы A — в ячейках памяти с адресами $a+(k-1)n+i\,(i, k=1, 2, \ldots, n)$. Компоненты вектора $C = BA$ разместим в ячейках с адресами $c+1$, $c+2$, ..., $c+n$ (первоначально в них помещаются числа, равные нулю). Ячейку с адресом t используем как рабочую ячейку для хранения промежуточных результатов (ее первоначальное заполнение для нас безразлично). Наконец, ячейки с адресами 1, 2, ... используем для хранения последовательных приказов, составляющих искомую программу.

Введем три индекс-регистра I_1, I_2, I_3, которые должны быть перед началом работы очищены, и поместим в ячейку с адресом d число n, равное размерности вектора B и порядку матрицы A. С помощью введенных обозначений искомая программа для умножения вектора на матрицу запишется в следующем виде:

1) прибавление 1 к содержимому индекс-регистра I_3;
2) прибавление 1 к содержимому индекс-регистра I_1;
3) прибавление 1 к содержимому индекс-регистра I_2;
4) умножение, $b(I_1)$, $a(I_2)$, t;
5) сложение, $c\,(I_3)$, t, $c\,(I_3)$;
6) условный переход, I_3, d, 2;
7) очистка индекс-регистра, I_1;
8) условный переход, I_3, d, 1;
9) останов.

При дальнейшем усложнении алгоритмов трудности программирования все более и более возрастают. В связи с этим естественно возникает мысль о поисках более экономных способов записи информации, об алгоритме и привлечении самого универсального программного автомата для автоматического перевода таких записей в настоящие рабочие программы. Эта мысль составляет основу

автоматического программирования с помощью так называемых *универсальных программирующих программ* (трансляторов).

Универсальная программирующая программа представляет собой запрограммированный для той или иной универсальной цифровой машины алгоритм перевода записи любого алгоритма в том или ином *формальном алгоритмическом языке* на язык приказов данной машины. В качестве формального алгоритмического языка, о котором здесь идет речь, можно, разумеется, выбрать любой из языков, описанных в гл. I, например язык схем нормальных алгоритмов. Однако подобный выбор не облегчил бы, а, наоборот, усложнил бы решение задачи программирования, поскольку запись алгоритма в любой из абстрактных алгоритмических схем первой главы представляет собой, как правило, значительно более трудную проблему, чем непосредственное программирование на языке приказов современных универсальных цифровых машин. Чтобы убедиться в этом, достаточно рассмотреть случай операции сложения, выполняемой в универсальных цифровых машинах с помощью одного приказа, в то время как, например, при использовании нормальных алгоритмов она реализуется с помощью достаточно сложно записанной схемы, содержащей много элементарных подстановок.

Были поэтому предприняты попытки разработать такие универсальные алгоритмические языки, которые сохраняли бы основные свойства языка современных универсальных цифровых машин, но допускали значительно более простую и легко читаемую запись обычно встречающихся на практике алгоритмов по сравнению с непосредственным программированием в «машинных» языках. Среди языков такого рода отметим, например, язык «Фортран» (США), польский алгоритмический язык САКО, адресный язык (Киев, СССР) и др.

Создание практических алгоритмических языков важно не только потому, что такие языки облегчают программирование, но и потому, что достаточно хорошо разработанный практический алгоритмический язык может стать общепринятым и общепонятным языком для записи различных алгоритмов. Таким языком, получившим в настоящее время широкое международное признание, является разработанный группой европейских и американских ученых язык АЛГОЛ-60. Подробное описание этого языка дано в следующем параграфе, здесь же рассмотрим некоторые приемы, облегчающие непосредственное программирование в машинных языках

Первым приемом, уже рассмотренным выше, является употребление в начальной стадии программирования символических адресов вместо истинных (числовых) адресов. Последующее присвоение истинных значений введенным символическим адресам и экономия рабочих ячеек представляют собой чисто техническую работу и легко поддаются автоматизации. Несмотря на свою простоту, метод символических адресов позволяет существенно упрощать

программирование сложных задач и, самое главное, значительно уменьшает число ошибок.

Вторым методом, с помощью которого может быть существенно упрощено непосредственное программирование, является включение ранее построенных относительно более простых программ в более сложные программы. Программы, специально приспособленные для включения в более сложные программы, называются обычно *подпрограммами*. Накапливая *библиотеку подпрограмм*, программист в ряде случаев может свести непосредственное программирование к комбинации относительно небольшого числа имеющихся в его распоряжении подпрограмм. Для облегчения сопряжения нескольких подпрограмм в одной программе разработаны специальные приемы, позволяющие не вносить изменений в подпрограммы при включении их в самые различные программы. Трудность заключается в том, что последний приказ подпрограммы должен передать управление какому-то приказу основной программы, который изменяется от программы к программе и не известен составителю подпрограммы.

Преодолеть указанную трудность можно, засылая в момент перехода на подпрограмму адрес приказа, к которому надо перейти после ее выполнения, в специальную ячейку памяти, называемую *регистром возврата*. В таком случае подпрограмма должна заканчиваться специальным приказом «переход по регистру возврата», извлекающим очередную команду из ячейки, адрес которой хранится в этом регистре. Заметим, что для использования в качестве регистра возврата любой заданной ячейки оперативной памяти машины целесообразно ввести обращение к памяти по так называемому *адресу второго ранга*. При таком обращении выборка из памяти или запись в память производятся не по адресам, указанным в исполняемой команде, а по адресам, которые хранятся в ячейках памяти и адреса которых указаны в этой команде. При наличии подпрограмм, включаемых в другие подпрограммы (а не в основную программу), приходится использовать несколько регистров возврата либо переходов по адресам второго ранга.

Подобное использование одних подпрограмм внутри других создает основу для ступенчатой организации системы стандартных программ. Рациональность такой организации определяется степенью экономности размещения библиотеки стандартных программ в тех или иных запоминающих устройствах. Этот вопрос становится особо актуальным при схемной реализации различного рода стандартных подпрограмм, обогащающей набор операций, выполняемых машиной. При этом достигается большая экономия труда программистов, получающих возможность пользоваться большими подпрограммами, отводя на каждую из них всего лишь по одному машинному приказу. Подобная ступенчатая организация управления реализована в вычислительной машине «Проминь» Института кибернетики АН УССР (Киев).

Впрочем, даже при отсутствии схемной реализации достаточно богатая библиотека стандартных подпрограмм существенно облегчает программирование, поскольку значительную часть составляемых заново программ будут, как правило, составлять заранее запрограммированные стандартные участки, имеющиеся в библиотеке.

Заметим, что, составляя библиотеку *стандартных подпрограмм*, стремятся обеспечить достаточно большую степень общности решаемых задач. Например, стандартная подпрограмма для умножения матриц составляется для умножения матриц любого (а не только какого-нибудь одного) порядка. Аналогично строятся и другие подпрограммы.

Существенную помощь при непосредственном программировании оказывает также предварительная запись программы в упрощенном виде, называемом обычно *блок-схемой программы*, с последующим программированием каждого отдельного блока. Удобным способом записи блок-схем программ является предложенный А. А. Ляпуновым *операторный способ записи схем программ*.

При использовании этого способа стоящие рядом группы однотипных команд программы (например, команд, реализующих арифметические операции) объединяются в так называемые *операторы*. Наиболее употребительными являются *арифметические операторы* и *операторы переадресации* (изменения содержимого индекс-регистров). Арифметические операторы обозначим буквой A, логические — P, операторы переадресации — I, а оператор останова — F. Кроме того, операторы нумеруются с помощью специальных индексов в порядке вхождения их в программу.

Объединяя группу арифметических операций, всякий арифметический оператор является условным обозначением для операции счета по той или иной, часто достаточно сложной, формуле. Логический оператор осуществляет проверку логических условий, на основании которых производятся те или иные условные переходы (передачи управления в программе, нарушающие естественный порядок следования команд). После логического оператора ставится вертикальная черточка; над и под этой черточкой указываются номера операторов, которым передается управление в том случае, когда логическое условие выполнено, и соответственно в том случае, когда оно не выполняется. Отсутствие номера под или над черточкой означает, что в соответствующем случае управление передается оператору, стоящему непосредственно справа от черточки.

С помощью описанной символики *операторная схема* последней из рассмотренных нами программ может быть записана

$$I_1 I_2 A_3 P_4 \Big|_{2}^{} I_5 P_6 \Big|_{1}^{} F_7.$$

Здесь оператор переадресации I_1 соответствует первому приказу программы, а оператор I_2 — двум следующим приказам. Арифметический оператор A_3 объединяет четвертый и пятый приказы, а остальные операторы включают в себя каждый по одному приказу.

Для облегчения чтения операторных схем черточки, обозначающие условные переходы, могут снабжаться вверху и внизу горизонтальными палочками, направленными влево. В таком случае перед оператором, которому передается управление, также ставится черточка с палочкой, направленной вправо и снабженной тем же номером, что и соответствующая черточка оператора условного перехода. При этом группа операторов, составляющая *цикл*, многократно повторяющийся в результате условных переходов, обрамляется с двух сторон своеобразными «скобками», облегчающими поиск таких циклов.

Используя эти обозначения, выписанную выше операторную схему можно переписать

$$\underset{1}{\lfloor} I_1 \underset{2}{\lfloor} I_2 A_3 P_4 \underset{2}{\rfloor} I_5 P_6 \underset{1}{\rfloor} F_7.$$

Снабжая операторную схему программы описанием каждого входящего в нее оператора (кроме оператора останова), можно после составления такой схемы приступить к раздельному, последовательному программированию этих операторов с последующим объединением составленных подобным путем отдельных кусков программы в единое целое. Эти операции представляют собой в значительной степени техническую работу и могут быть относительно легко автоматизированы с помощью любого универсального программного автомата.

Заметим, что для описания арифметических и логических операторов можно пользоваться обычными арифметическими или логическими формулами. Располагая специальными программами для автоматического перевода таких формул на язык приказов машины и объединяя их со стандартными библиотечными подпрограммами, можно добиться возможности выдачи машине (универсальному программному автомату) задания в такой форме, в какой оно дается квалифицированному человеку-вычислителю. Этот метод объединяет фактически метод стандартных подпрограмм с методом, использующим (в той или иной мере) универсальные программирующие программы. Его естественно называть поэтому методом специализированных программирующих программ, или методом библиотеки программирующих программ [24]. Специализация здесь состоит в том, что соответствующая библиотека ориентируется на некоторый класс типовых задач, позволяя фактически полностью исключить программирование и ограничиться сообщением машине одного лишь условия задачи, которую требуется решить.

§ 4. Универсальный алгоритмический язык АЛГОЛ-60

Международный алгоритмический язык АЛГОЛ-60, который для краткости будем называть просто АЛГОЛом, представляет собой средство для достаточно простой, точной и общепонятной записи вычислительных алгоритмов. Являясь универсальным алгоритмическим языком, он пригоден, разумеется, для записи произвольных (не обязательно вычислительных) алгоритмов, однако в случае преобразования буквенной, а не числовой информации простота и наглядность соответствующей «алгольной» записи в значительной мере теряется. В то время как программирование вычислительных алгоритмов на АЛГОЛе представляет собой гораздо более простую задачу, чем непосредственное программирование для современных универсальных электронных цифровых машин, программирование задач по преобразованию буквенной информации на АЛГОЛе оказывается немногим проще, чем на «машинных» языках.

Основными символами, с помощью которых строится язык АЛГОЛ, служат латинские буквы (26 прописных и 26 строчных букв), арабские цифры (от нуля до девяти включительно), логические значения «истина» и «ложь», а также знаки операций, разделителей и скобок (три последних типа символов называются ограничителями). Имеется еще некоторое число служебных слов, в качестве которых берутся обычно слова английского языка. Эти слова принято писать всегда жирным шрифтом.

Для обозначения чисел используется десятичная система счисления, причем целая часть отделяется от дробной части точкой (а не запятой). Знак плюс перед положительными числами и знак нуля в обозначении целой части для правильной дроби могут опускаться. Для обозначения десятичного порядка (число десять в целой степени) употребляется специальный символ—десятка, опущенная ниже основной строки (обычно ее печатают жирным шрифтом). Числа, употребляемые в АЛГОЛе, разделяются на два типа: **integer** (целый) и **real** (вещественный). К типу **integer** относятся лишь целые числа (со знаком или без знака), не содержащие в своей записи символа десятичного порядка и точки; все остальные числа относятся к типу **real** (число 3.0 относится при этом к типу **real**, а не **integer**).

Примеры чисел типа **integer**: $0, +275, -0634, +0, -2$. Примеры чисел типа **real**: $+5.340_{10}8$ (т. е. число $5{,}34 \cdot 10^8$), $-.063$ (число $-0{,}063$), $-.37_{10}-32$ (число $-0{,}37 \cdot 10^{-32}$), $+_{10}5$ (число 10^5) и др.

Введение тех или иных величин в АЛГОЛе (при записях конкретных программ) сопровождается их предварительным *описанием*. Последующие конкретные представления этих величин должны интерпретироваться в соответствии с указанным описанием. Если, например, какая-либо величина x была описана термином **integer**, а затем было введено ее значение, равное, скажем,

23.4, то это значение должно быть мысленно округлено до ближайшего целого (в данном случае до 23). Значение величины типа **integer**, равное 23.5, округляется при этом до 24, а не до 23 (до ближайшего большего целого)*. Заметим также, что если величина x описана как **real**, то ничто не мешает ей принимать также целые значения, однако при последующих операциях с величиной x поступают, как и со всякой вещественной величиной, не производя округления до ближайшего целого.

Для обозначения различного рода величин (постоянных и переменных) в АЛГОЛе употребляются так называемые *идентификаторы*. Идентификатором может служить любая конечная последовательность букв (латинских) и десятичных цифр, начинающаяся обязательно буквой (а не цифрой). Примерами идентификаторов могут служить *a7LO*, *x*, *ga*, *aPg*, *TOWW* и т. д. В то же время выражения $7x$, $5ab$ или $ab+x$ не могут служить идентификаторами. Использование для обозначения величин не только букв, но и слов, т. е. последовательностей букв (возможно, бессмысленных), делает запас идентификаторов потенциально неограниченным, что имеет большое значение с точки зрения возможности представления любых, сколь угодно сложных алгоритмов. Представляет также известные удобства возможность обозначения величины ее естественным названием, например: force (сила), current (ток) и т. п. Вместе с тем имеется и одно неудобство, с которым необходимо считаться в дальнейшем: при построении арифметических выражений из идентификаторов знак умножения не может опускаться (как это обычно делается в алгебре), ибо выражение $ab + xy$ будет пониматься в АЛГОЛе как сумма двух величин, обозначенных через ab и xy, а не как сумма парных произведений величин a,b и x,y.

Кроме величин, принимающих числовые значения (типа **integer** и **real**), в АЛГОЛе употребляются булевы величины, принимающие лишь два значения — «истина» (**true**) и «ложь» (**false**). Булевы величины обозначаются идентификаторами точно таким же образом, как и числовые величины; в описаниях им присваивается тип **Boolean**.

Однородные величины, например компоненты какого-либо вектора или элементы той или иной матрицы, обычно обозначаются идентификаторами с одним или несколькими индексами. Индексы записываются после идентификатора и заключаются в квадратные скобки. Различные индексы отделяются друг от друга запятыми. В качестве индексов могут фигурировать целые числа (не только положительные), переменные и любые арифметические выражения, которым всегда присваивается тип **integer**.

Примеры записи переменных с индексами: $A[1, -2]$, $ps\ [i]$ $bA8[i,j,1]$.

* Подобных округлений в АЛГОЛе обычно стремятся избегать, поскольку естественное округление величины 23.5 в одних машинах приведет к 23, а в других — к 24.

Переменные с индексами, изменяющимися в некоторых пределах, составляют так называемые *массивы*. Описанию массива в АЛГОЛе предшествует английское слово **array** (массив), перед которым ставится наименование типа (**integer**, **real**, **Boolean**) составляющих массив переменных (если наименование типа переменных в массиве не указывается, то считается, что они имеют тип **real**). При описании массива после идентификатора массива в индексных (квадратных) скобках выписывается так называемый *список граничных пар*. Каждая граничная пара состоит из двух арифметических выражений (или чисел), разделенных между собой двоеточием. Первое из этих выражений представляет собой нижнюю границу (наименьшее возможное значение) соответствующего индекса, а второе — его верхнюю границу (наибольшее возможное значение индекса). Предполагается, что индекс может пробегать все целые значения, заключенные между нижней и верхней границами, причем в случае, когда верхняя граница меньше нижней, соответствующий массив считается неопределенным.

Примеры описания массивов: **real array** $x[1:n, 0:m]$, **Boolean array** $g4[0:5]$, **integer array** $N[-7:1, i:j, 3:3]$.

Количество различных индексов, характеризующих массив, называется *размерностью* этого массива. В только что приведенных примерах массив x имеет размерность 2, массив $g4$ — размерность 1. Что же касается массива N, то формально его размерность равна 3, однако, поскольку последний индекс может принимать лишь одно-единственное значение (равное 3), то фактически величина размерности этого массива сводится к 2.

Заметим, что при описаниях переменных или массивов одного и того же типа название типа может писаться лишь один раз, а соответствующие идентификаторы отделяются друг от друга запятыми. В случае массивов с одинаковыми границами индексные скобки с соответствующим списком граничных пар могут быть выписаны лишь один раз—после последнего идентификатора массива с этими границами. Например, **real** $a,b,x7$ или **integer array** $A,B,Dc[1:2, 1:k]$. Первое описание описывает три переменные, принимающие вещественные значения, а второе — три двумерных массива с одинаковыми границами, составленных из целочисленных величин.

Для разделения описаний переменных или массивов различных типов используется точка с запятой. Например, **real** x,y; **Boolean** A,B,C; **array** $px[1:2, i:k]$; **integer array** $N[-1:0, 5:10], Q[2:4]$.

Границами индексов в массивах могут служить произвольные *арифметические выражения*, играющие большую роль при построении языка АЛГОЛ. Арифметические выражения строятся из чисел и переменных с помощью шести арифметических операций— *сложения* (обозначаемого знаком $+$), *вычитания* (обозначаемого знаком $-$), *умножения* (обозначаемого знаком $\times$), *деления* (обозначаемого знаком косой черты /), *целочисленного деления* (обозначаемого знаком $\div$) и *возведения в степень* (обозначаемого знаком $\uparrow$).

Целочисленное частное $a \div b$ представляет собой целую часть (округление в сторону уменьшения модуля до ближайшего целого) обычного частного a/b. Эта операция применяется лишь к величинам типа **integer**, так что выражения $3.0 \div 5.0$ следует считать неопределенными (в то же время выражение $3 \div 5$ определено и равно нулю). Операция возведения в степень $a \uparrow b$ (a в степени b) при положительном a определена для всех величин b типов **real** и **integer**, а при отрицательном a — только для случая, когда величина b имеет тип **integer**.

При прочих равных условиях в арифметическом выражении сначала должны выполняться операции возведения в степень, затем — операции умножения и деления (обычного и целочисленного), затем операции сложения и вычитания. Одноименные операции (умножение и деление или сложение и вычитание) выполняются в обычном порядке — слева направо. При необходимости выполнения операций в ином порядке употребляются круглые скобки. При возведении в степень $a \uparrow b$ выражения a и b, как правило, должны быть заключены в скобки. Исключения допускаются лишь в том случае, когда соответствующая (не заключаемая в скобки) величина представляет собой число без знака, переменную (с индексами или без индексов), либо функцию (см. ниже).

Примерами арифметических выражений могут служить выражения $x \uparrow 2$ (равное x^2), $3 \uparrow n \uparrow k$ (равное $(3^n)^k$), $ab \times AB + pr \uparrow \uparrow (-q)$, $(x7 + A9) \uparrow (-2)$ и т. д.

Наряду с переменными, представляемыми обычными идентификаторами или идентификаторами массивов, при построении арифметических выражений используются также так называемые *функции*. Всякая функция в АЛГОЛе обозначается присвоенным этой функции идентификатором, после которого в круглых скобках помещается так называемый *список фактических параметров*, т. е., иными словами, аргументов этой функции. Фактическими параметрами могут служить любые выражения (арифметические или булевы), а также идентификаторы массивов и некоторые иные виды идентификаторов, определямые ниже. Параметры отделяются друг от друга либо запятыми, либо строкой вида:)любой комментарий: (. *Комментарием* называется пояснение смысла фактических параметров, которое будем давать обычно на русском языке. При переводе с языка АЛГОЛ на язык той или иной вычислительной машины этот комментарий попросту выбрасывается.

Примерами функций могут служить $f(x)$, $t7(x, y + a)$ AB $(k \uparrow 1.5)$ сила: (p) ускорение: (a). Первая из этих функций является одноместной функцией (т. е. зависит от одного фактического параметра), вторая функция — двухместная, а третья — трехместная (она зависит от фактических параметров $\kappa \uparrow 1.5, p$ и a).

При описаниях функции обычно называются *процедурами* (**procedure**) с указанием типа определяемой ею величины (**integer**, **real** или **Boolean**). За такими функциями, как синус, логарифм и другие, сохраняются обычно применяемые идентификаторы

sin, ln и т. д. Идентификатор *sqrt* будем фиксировать для обозначения квадратного корня, а идентификатор *abs* — для обозначения абсолютной величины.

Описания функций включают в себя заголовки вида: **real procedure** *sin* (*x*), **integer procedure** *abs*(*n*); **Boolean procedure** *A*(*a*,*b*). При описаниях в качестве аргументов функций употребляют так называемые *формальные параметры*, т. е. некоторые идентификаторы, которые при последующем использовании функции могут заменяться любыми фактическими параметрами, т. е. переменными, выражениями (арифметическими, булевыми, именующими), идентификаторами массивов процедур или переключателей, а также так называемыми строками.

Выражения, построенные из чисел, переменных типов **real** и **integer** (с индексами или без индексов) и функций (целых или вещественных) с помощью арифметических операций, называются *простыми арифметическими выражениями*. Для того чтобы строить более сложные арифметические выражения, необходимо познакомиться с так называемыми *булевыми выражениями*.

Булевы выражения строятся из логических значений («истина» и «ложь»), переменных и функций (процедур) типа **Boolean** и так называемых *отношений*, представляющих собой два арифметических выражения $\mathfrak{A}$ и $\mathfrak{B}$, соединенных между собой знаками равенства или неравенства: $\mathfrak{A} = \mathfrak{B}$, $\mathfrak{A} \neq \mathfrak{B}$, $\mathfrak{A} > \mathfrak{B}$, $\mathfrak{A} \geqslant \mathfrak{B}$, $\mathfrak{A} < \mathfrak{B}$, $\mathfrak{A} \leqslant \mathfrak{B}$. В качестве операций при построении булевых выражений употребляются описанные в гл. II логические операции эквивалентности ($\equiv$), импликации ($\supset$), дизъюнкции ($\vee$), конъюнкции ($\wedge$) и отрицания ($\neg$). Старшинство логических операций по отношению друг к другу и способ употребления скобок (в данном случае только круглых) сохраняются такими же, как и в гл. II. Нужно добавить лишь, что арифметические операции (выражения) считаются более старшими по сравнению со всеми операциями отношения, а последние — по сравнению со всеми логическими операциями, так что выражение $a + b > c \times d \supset x \wedge y \vee z$ должно пониматься как $((a + b) > (c \times d)) \supset ((x \wedge y) \vee z)$. Величины a,b,c,d в этом выражении имеют тип **real** или **integer**, а величины x,y,z — тип **Boolean**.

Все определенные до сих пор булевы выражения называются *простыми*. Из булевых выражений $\mathfrak{A}$, $\mathfrak{B}$, $\mathfrak{C}$, из которых первое выражение $\mathfrak{A}$ является простым, можно составить более сложное булево выражение, употребляя служебные слова **if** (если), **then** (тогда) и **else** (иначе). Соответствующая конструкция выглядит так:

if $\mathfrak{C}$ **then** $\mathfrak{A}$ **else** $\mathfrak{B}$.

Принимается, что значение определенного таким образом сложного булева выражения есть $\mathfrak{A}$, если выполнено условие $\mathfrak{C}$ (т. е. если булево выражение $\mathfrak{C}$ принимает значение «истина»), и $\mathfrak{B}$ — в противном случае. Поскольку из трех выражений $\mathfrak{A}$, $\mathfrak{B}$, $\mathfrak{C}$ лишь

выражение $\mathfrak{A}$ должно быть простым, то выражения $\mathfrak{B}$ и $\mathfrak{C}$ могут в свою очередь быть составленными с помощью некоторого условия. Наконец, при построении простых булевых выражений, наряду с логическими значениями, переменными, отношениями и функциями, разрешается использовать *произвольные* булевы выражения (как простые, так и сложные), которые заключены в круглые скобки.

Таким образом, при определении булевых выражений оказываются возможными рекурсивные построения любой глубины. Например, к числу булевых выражений может быть отнесено выражение **if** $a > b$ **then** (**if** $a = b + c$ **then** B **else** C) **else if** D **then** E **else** $F \wedge G(K,L)$, где строчными буквами обозначены переменные типа **real**, а прописными — переменные типа **Boolean**, причем $G(K,L)$ представляет собой функцию (**Boolean procedure**) фактических параметров K и L. Если все это выражение заключить в круглые скобки, то оно превратится в *простое* булево выражение и как таковое может быть использовано в дальнейших построениях.

Совершенно аналогичное положение и в случае арифметических выражений: из двух арифметических выражений $\mathfrak{A}$ и $\mathfrak{B}$, из которых первое является непременно простым, и булева выражения $\mathfrak{C}$ можно построить сложное *арифметическое выражение*

if $\mathfrak{C}$ **then** $\mathfrak{A}$ **else** $\mathfrak{B}$.

Значение этого выражения принимается равным $\mathfrak{A}$, если выполнено условие $\mathfrak{C}$, и равным $\mathfrak{B}$ — в противном случае. Как и в случае булевых выражений, при построении *простых* арифметических выражений разрешается использовать не только числа, переменные и функции, но также и *произвольные* арифметические выражения (простые или сложные), которые заключены в круглые скобки. Так что, например, выражения (**if** $a > b$ **then** $a \uparrow 2$ **else** $a \uparrow 3$) или (**if** $a = b$ **then** $a - b$ **else** $a + b$) $\uparrow$ $(a - b \uparrow 2)$ должны считаться простыми арифметическими выражениями.

Все приведенные до сих пор описания являются по существу вспомогательными. Основным средством для построения алгоритмов в АЛГОЛе служат так называемые *операторы*. АЛГОЛ-60 насчитывает шесть различных типов операторов: *оператор присваивания*, *оператор перехода*, *пустой оператор*, *оператор цикла*, *оператор процедуры* и *условный оператор*. Первые пять типов операторов, в отличие от последнего, условного оператора, называются обычно безусловными операторами.

Оператор присваивания присваивает тем или иным переменным определенные значения, задаваемые каким-либо арифметическим или булевым выражением $\mathfrak{A}$. Переменные, которым присваивается значение, определяемое выражением $\mathfrak{A}$, отделяются друг от друга и от этого выражения специальным разделителем $:=$ (знаком присваивания). Все эти переменные составляют левую часть, а выражение $\mathfrak{A}$ — правую часть оператора присваивания.

Примеры операторов присваивания: $A := k1[0] := v := n + 1 + p$; $m := m + 1$; $B := a > b$; $r[j, zk] := 5 - 3 \times v \uparrow 2$.

При реализации оператора присваивания должен соблюдаться строго определенный порядок выполнения действий. Сначала по порядку (слева направо) вычисляются значения индексов (задаваемых арифметическими выражениями, которые называются в этом случае индексными выражениями) всех переменных левой части. Затем вычисляется значение арифметического выражения в правой части и полученное значение присваивается всем переменным левой части (с уже вычисленными индексами). Таким образом, например, оператор $A := B := p + q$ должен выполняться как $z := p + q$; $B := z$; $A := z$, а не как $B := p + q$; $A := p + q$. Различие заключается в том, что значение арифметического выражения $p + q$ может изменяться при каждом новом вычислении (например, если оно содержит некоторую функцию, значения которой определяются меняющейся в процессе своего выполнения процедурой). Поэтому при выполнении присваивания значение арифметического выражения должно вычисляться лишь один раз, а не применительно к каждому отдельному присваиванию.

Операторы в АЛГОЛе могут снабжаться *метками*, в качестве которых используются произвольные идентификаторы или целые без знака (в последнем случае приписывание нулей впереди числа не изменяет значения метки). Впрочем, с целью облегчения построения *трансляторов* (программ для перевода с языка АЛГОЛ на машинные языки) часто избегают употребления чисел в качестве меток. Метка отделяется от оператора двоеточием. Операторы (помеченные или непомеченные) располагаются последовательно один за другим, отделяясь друг от друга точкой с запятой, например: p: 𝔄; 𝔅; $k1$: 𝔈, где 𝔄, 𝔅, 𝔈 — операторы, а p и $k1$ — метки (оператор 𝔅 является *непомеченным* оператором). Один и тот же оператор может иметь не одну, а какое угодно число меток (отделяемых друг от друга двоеточиями), например, p:A:$r7$: 𝔄 (оператор 𝔄 имеет здесь три метки: p, A и $r7$).

Обычно операторы в АЛГОЛе выполняются последовательно, один за другим, в порядке их записи. Изменение в порядке выполнения операторов осуществляется специальным оператором, называемым *оператором перехода*. В простейшем случае оператор перехода состоит из служебных слов **go to** («перейти к») и какой-либо метки L. Смысл действия этого оператора состоит в том, что, дойдя до него, совершают переход (прыжок) к оператору, имеющему L в качестве своей метки.

В общем случае в операторе перехода после слов **go to** ставится какое-нибудь *именующее выражение*. Метка представляет собой лишь один из простейших примеров именующих выражений. Более сложным примером именующего выражения является выражение, составленное из двух меток, скажем L и M, и некоторого булева выражения 𝔈 : **if** 𝔈 **then** L **else** M. Значение этого именующего выражения равно L, если условие 𝔈 выполнено, и M —

в противном случае. Вместо метки M (но не вместо метки L) в это выражение может быть подставлено какое-нибудь сложное именующее выражение, и подобный процесс подстановки может быть продолжен.

В результате могут возникать сложные рекурсивные построения для оператора перехода, например **go to if** $i = 1$ **then** L **else if** $i = 2$ **then** M **else** P. Для упрощения указанной конструкции употребляется так называемый *переход по переключателю*. Переключатель составляется из некоторого идентификатора и следующего за ним так называемого *индексного выражения*, заключенного в индексные (т. е. в квадратные) скобки. Индексное выражение представляет собой любое арифметическое выражение, которое при вычислении должно всякий раз округляться до ближайшего целого числа.

Если, например, идентификатором переключателя служит s, а индексным выражением — переменная (выражение) i, то оператор перехода по переключателю $s[i]$ запишется **go to** $s[i]$. Само по себе подобное выражение еще не имеет пока смысла. Чтобы придать ему смысл, нужно, кроме выражения $s[i]$, которое будем называть *указателем переключателя* и считать *простым именующим выражением*, ввести еще так называемое *описание переключателя*, помещаемое обычно вместе с описанием типов переменных, массивов и процедур (функций). Описание переключателя начинается служебным словом **switch** (переключатель), после чего идет идентификатор переключателя, затем знак присваивания: $:=$ и, наконец, так называемый *переключательный список*, т. е. список именующих выражений, отделенных друг от друга запятыми. Например: **switch** $s := L,M,P$ (где L,M,P — метки).

Встретив оператор перехода по переключателю, например, **go to** $s[i]$, вычисляют соответствующее индексное выражение, подставляя в него текущие значения переменных (скажем, $i = 2$). После этого обращаются к описанию переключателя с тем же самым идентификатором s и осуществляют переход по тому именующему выражению в этом описании, порядковый номер которого в списке совпадает с найденным значением индексного выражения. В рассматриваемом случае будет осуществлен переход по метке M, т. е. по *второму* элементу переключательного списка.

Если значение индексного выражения в указателе переключателя не может быть вычислено (в результате того, что некоторым переменным еще не присвоены значения) или если это вычисление приводит к числу, не являющемуся номером никакого элемента переключательного списка, то оператор перехода не выполняется и сразу осуществляется выполнение следующего за ним оператора. В рассмотренном выше примере не приводят к цели значения индексного выражения, равные 4,0 или —1. Вместе с тем значения индексного выражения, равные 2.2 или 2.7, приводят (после их округления) к переходам по второму или, соответственно, по третьему элементу переключательного списка (т. е. по меткам M или P).

Метка, указатель переключателя или любое именующее выражение, заключенное в круглые скобки, представляют собой *простые именующие выражения*. Из двух именующих выражений $\mathfrak{A}$ и $\mathfrak{B}$ (из которых первое является непременно простым) и булева выражения $\mathfrak{C}$ можно составить сложное именующее выражение **if** $\mathfrak{C}$ **then** $\mathfrak{A}$ **else** $\mathfrak{B}$, совпадающее с выражением $\mathfrak{A}$ в случае выполнения условия $\mathfrak{C}$ и с выражением $\mathfrak{B}$ — в противном случае. Таким образом, для именующих выражений оказываются возможными точно такие же сложные рекурсивные построения, как и для арифметических (или булевых) выражений.

Третьим типом операторов, употребляемых в АЛГОЛе, является так называемый *пустой оператор*, не выполняющий никакой работы и обозначаемый пустым множеством символов. Обычно пустой оператор снабжается меткой и служит для возвращения по этой метке (в результате применения оператора перехода) в требующийся участок программы. Употребление пустого оператора с меткой совершенно необходимо, например, в том случае, когда из середины программы требуется совершить переход в ее конец (не к последнему непустому оператору программы, а именно в *конец* программы). После метки в пустом операторе, как и в других операторах, должно всегда стоять двоеточие.

Очень важное значение при построении программ в языке АЛГОЛ имеют так называемые *операторы цикла*, смысл которых состоит в том, что некоторый оператор (или группа операторов) выполняется некоторое число раз подряд. Оператор цикла состоит из *заголовка цикла* и собственно оператора (которым может быть любой оператор), выполняющегося многократно в процессе циклирования.

Заголовок цикла начинается служебным словом **for** (для) и кончается служебным словом **do** (выполнить). После слова **for** ставится идентификатор той переменной, которая меняется в процессе выполнения цикла. Эта переменная называется *параметром цикла*. За ней, после символа присваивания $:=$, идет так называемый *список цикла*, перечисляющий те значения, которые должна принять переменная за время прохождения цикла. Список цикла состоит из одного или нескольких *элементов списка цикла*, отделяемых друг от друга запятыми. В простейшем случае в качестве *элементов списка цикла* употребляются арифметические выражения (в частности, просто числа). Например, оператор цикла **for** $i := 1,2,3$ **do** $a[i] := i \uparrow 2$ выполняет последовательные присвоения: $a[1] := 1$; $a[2] := 4$; $a[3] := 9$. Длина цикла равна в этом случае 3.

Если параметр цикла должен принять не три, а, скажем, тысячу различных значений, то перечисление всех этих значений в списке цикла стало бы чересчур громоздким. В этом случае в качестве элементов цикла употребляют специальные конструкции, каждая из которых задает сразу некоторое множество значений параметра цикла.

В АЛГОЛе применяется два типа таких конструкций. Первый тип строится с использованием служебных слов **step** (шаг) и **until** (до) и имеет вид $\mathfrak{A}$ **step** $\mathfrak{B}$ **until** $\mathfrak{C}$, где $\mathfrak{A}$, $\mathfrak{B}$ и $\mathfrak{C}$ — некоторые арифметические выражения. Элемент списка цикла этого типа задает значения параметра цикла следующим образом: на первом шаге параметру цикла присваивается значение арифметического выражения $\mathfrak{A}$, на втором шаге — значение арифметического выражения $\mathfrak{A}_1 = \mathfrak{A} + \mathfrak{B}$, на третьем — значение $\mathfrak{A}_2 = \mathfrak{A}_1 + \mathfrak{B}$ и т. д., пока очередное значение $\mathfrak{A}_n = \mathfrak{A}_{n-1} + \mathfrak{B}$ не превзойдет значение арифметического выражения $\mathfrak{C}$*. Это значение параметру цикла не присваивается и цикл для него не выполняется. Список цикла считается исчерпанным, и, следовательно, должен осуществиться переход к оператору, непосредственно следующему за оператором цикла.

В качестве примера описанной конструкции рассмотрим оператор цикла, имеющий вид **for** $i:=12{,}4$ **step** -1 **until** $0{,}-5$ **do** $a[i] := i+10$. Этот оператор выполняет последовательные присвоения: $a[12]:=22$; $a[4]:=14$; $a[3]:=13$; $a[2]:=12$; $a[1]:=11$; $a[0]:=10$; $a[-5]:=5$. Заметим, что в этом примере элемент списка цикла 4 **step** -1 **until** 0 описывает арифметическую прогрессию (с разностью, равной -1), однако в общем случае шаг, представляемый арифметическим выражением $\mathfrak{B}$ (стоящим после слова **step**), может оказаться переменным, меняясь при каждом новом повторении цикла.

Второй тип элемента списка цикла задается с помощью некоторого арифметического выражения $\mathfrak{A}$, некоторого булева выражения $\mathfrak{B}$ и служебного слова **while** (пока), записываемых в такой последовательности: $\mathfrak{A}$ **while** $\mathfrak{B}$. Этот элемент обеспечивает последовательное присвоение параметру t цикла значений, принимаемых арифметическим выражением $\mathfrak{A}$ до тех пор, пока выполняется условие $\mathfrak{B}$ (т. е. пока выражение $\mathfrak{B}$ имеет значение «истина»). Если при очередном выполнении цикла условие $\mathfrak{B}$ перестает выполняться, то оператор цикла не выполняется, а осуществляется переход к непосредственно следующему за ним оператору. Заметим, что в описанной конструкции запрещается пользоваться словом **step**, так что выражение типа $\mathfrak{A}$ **step** $\mathfrak{B}$ **while** $\mathfrak{C}$ в АЛГОЛе не встречается.

При описанном выше способе построения оператора цикла можно осуществить повторение лишь одного-единственного оператора, следующего непосредственно за служебным словом **do**. Если потребуется повторять в том или ином цикле не один оператор, а некоторую последовательность операторов $A_1; A_2; \ldots; A_k$, то эту последовательность заключают в специальные *операторные* скобки, рассматривая ее после этого как один *составной оператор*.

---

* Если шаг $\mathfrak{B}$ отрицательный, то значение выражения $\mathfrak{C}-\mathfrak{A}_n$ берется с обратным знаком.

В качестве операторных скобок употребляется пара служебных слов **begin** (начать) и **end** (конец), так что составной оператор запишется **begin** A_1; A_2; ...; A_k **end**. Составные операторы, как и обычные, могут снабжаться метками (одной или несколькими).

Наряду с составными операторами, в АЛГОЛе употребляются так называемые *блоки*, отличающиеся от составных операторов тем, что перед входящими в блок операторами, непосредственно после слова **begin**, помещается описание типов некоторых величин (идентификаторов), встречающихся в этом блоке. При этом описанные в блоке величины локализуются лишь в данном блоке и, вообще говоря, теряют свое значение (делаются неопределенными) при выходе из него. Если хотят сохранить значения некоторых из описанных в блоке величин после выхода из него с целью использования их при повторном обращении к блоку, то к описанию их типов добавляется слово **own** (собственный). Пример блока: **begin own real** x; **integer** n; $n := s+i$; $x := a \uparrow n$ **end**.

Заметим, что как составные операторы, так и блоки могут включать в себя снова блоки и составные операторы, допуская любую рекурсивную глубину подобных построений. Идентификаторы, использованные внутри блока для обозначения несобственных величин, могут использоваться вне блока для обозначения любых других величин, не доступных для этого блока (т. е. не фигурирующих в данном блоке и не подвергающихся в нем никаким преобразованиям). Идентификаторы, не описанные в блоке, не будут локализованы в нем и, следовательно, представляют одни и те же объекты как внутри блока, так и вне его.

Метки всегда предполагаются локализованными внутри блока, в котором они встречаются, так что вход в блок может осуществляться лишь через его начало. Никакой оператор перехода, расположенный вне блока, не может осуществлять переход к какому бы то ни было оператору внутри этого блока.

Точно так же нельзя осуществлять переход по метке, расположенной внутри оператора цикла, с помощью оператора перехода, действующего извне цикла. Заметим еще, что при выходе из оператора цикла в результате исчерпания списка цикла значение параметра цикла считается неопределенным. Если же выход из цикла осуществился за счет оператора перехода, содержащегося в составном операторе (или блоке), повторяемом в данном цикле (т. е. стоящем после слова **do**), то значение параметра цикла сохраняется таким, каким оно было непосредственно перед выполнением оператора перехода.

Все описанные выше простые операторы, а также описываемый ниже оператор процедуры и все составные операторы и блоки принадлежат к числу так называемых *безусловных операторов*. В АЛГОЛе вводится еще два типа условных операторов, использующих условие **if** $\mathfrak{B}$ **then** (где $\mathfrak{B}$ — булево выражение), аналогичное условию, использованному при построении сложных арифметических, булевых и именующих выражений.

Оператор «если» строится из описанного условия и следующего за ним *безусловного* оператора, который выполняется в случае, когда условие выполнено, и пропускается (не выполняясь) в противном случае. Пример: **if** $a>в$ **then begin** $A:=n$; **go to** L **end**. Составной оператор **begin** $A:=n$ **go to** L **end** выполняется в том и только в том случае, когда выполнено условие $a>b$.

Собственно *условный оператор* получается добавлением к оператору «если» служебного слова **else** (иначе) и следующего за ним произвольного оператора (быть может, также условного). Этот оператор должен выполняться в том случае, когда не выполнено условие в операторе «если». Общая конструкция условного оператора имеет, таким образом, вид

$$\textbf{if}\ \mathfrak{B}\ \textbf{then}\ A_1\ \textbf{else}\ A_2,$$

где $\mathfrak{B}$ — произвольное булево выражение, A_1 — безусловный оператор, A_2 — произвольный оператор.

Существенное значение при построении АЛГОЛа имеют так называемые *операторы процедуры*. Процедурой называется некоторая совокупность операторов, обозначенная некоторым идентификатором, называемым *идентификатором этой процедуры*. Процедуры в АЛГОЛе играют ту же самую роль, что и подпрограммы при обычном программировании, позволяя ускорять составление сложных программ за счет использования ранее составленных *стандартных* программ. Расшифровка процедуры (фактическая запись составляющих ее операторов) может быть произведена как на языке АЛГОЛ, так и непосредственно на языке соответствующей универсальной цифровой машины.

Оператор процедуры (если только речь не идет об известных стандартных процедурах) должен быть предварительно описан. Такое описание осуществляется с помощью служебного слова **procedure** (процедура), после которого следует так называемый заголовок процедуры, т. е. идентификатор процедуры, а за ним (в круглых скобках) список так называемых *формальных параметров* процедуры, т. е. идентификаторов, отделенных друг от друга специальными ограничителями, а именно запятыми, либо ограничителями вида)строка букв:(.Например: **procedure** sin (x) или **procedure** A (x,y) давление: (p). Первая процедура имеет один формальный параметр (x), а вторая — три формальных параметра x, y, p. Возможны также процедуры без параметров. Их заголовок состоит только из идентификаторов процедур, не сопровождаемых последующими скобками. Сама процедура (так называемое *тело процедуры*) выписывается после заголовка процедуры в виде некоторого оператора.

Собственно *оператор процедуры*, или, более точно, *оператор вызова процедуры*, записывается в том же виде, что и в описании процедуры, но уже без слова **procedure** впереди и при условии, что формальные параметры процедуры заменены ее так называемыми *фактическими параметрами*. Скобки и ограничители —

такие же, как и в описании соответствующей процедуры. Выполнение оператора процедуры состоит в присваивании всем формальным параметрам значений соответствующих фактических параметров или замене формальных параметров фактическими и последующего выполнения процедуры.

В качестве фактических параметров процедур могут употребляться произвольные выражения (арифметические, булевы или именующие), идентификаторы массивов и переключателей, идентификаторы любых процедур и, наконец, так называемые *строки.*

Строки представляют собой любые последовательности символов, заключенных в специальные «строчные» скобки ' '. Скобки эти могут использоваться и внутри строки. Пример: '10 + — [[x ↑ ↑b:'∧∨⊃'≡Ad'. Строки могут использоваться в качестве фактических параметров лишь таких процедур, которые записаны в машинных кодах, а не на языке АЛГОЛ. Чаще всего их употребление ограничивается специальной процедурой *punch*(x), которая осуществляет печать или перфорирование фактических параметров, представляемых вместо формального параметра x. Если, в частности, вместо x будет подставлена некоторая строка, то процедура *punch* осуществит вывод всех символов этой строки из машины на печать или перфорирование. В строке могут поэтому фигурировать не только «алгольные», но и любые другие символы, которые способно выводить рассматриваемое печатающее или перфорирующее устройство.

При описании процедуры указывается также тип ее формальных параметров. При подстановках фактических параметров их типы должны совпадать с типами соответствующих формальных параметров. Во избежание двусмысленностей при такой подстановке приходится осуществлять обычно замену тех локализованных внутри процедуры идентификаторов, которые совпадают с идентификаторами, входящими в подставляемые фактические параметры.

Заметим, что в число параметров процедуры входят, вообще говоря, как входные, так и выходные (получающиеся в результате выполнения процедуры) величины этой процедуры. Если в результате выполнения процедуры получается лишь одна величина (число или логическое значение), то естественно обозначать эту величину идентификатором самой процедуры (вместе со строкой фактических параметров). В этом случае соответствующая процедура называется функцией (см. выше), а при ее описании перед словом **procedure** ставится слово, обозначающее тип выходной величины этой процедуры. Например: **real procedure** sin (x). Часто случается, что в процедуре некоторый формальный параметр x участвует во многих преобразованиях. Например, параметр x в процедуре sin(x) при вычислении синуса с помощью ряда возводится последовательно в степени 3, 5, 7 и т. д. Если при подстановке этот параметр заменить достаточно сложным выражением (фактическим параметром), скажем, x: = (a + b) × (a — b), то при развертывании процедуры можно столкнуться с необходимостью многократного вычисления этого

выражения в каждом случае, когда производится та или иная операция с параметром x. Естественно, что проще вычислить заранее (перед входом в процедуру) значение x и подставить вместо него это значение.

При автоматическом переводе программы с языка АЛГОЛ на машинный язык необходимо каждый раз сообщать транслятору (программирующей программе), значения каких параметров должны быть обязательно вычислены *до подстановки* в процедуру. Все такие параметры в описании отмечаются специальным служебным словом **value** (значение) и помещаются после совокупности формальных параметров заголовка процедуры перед описанием их типов (так называемых спецификаций). Например, вместо описания **real** x; **integer** n может возникнуть описание **value** n; **real** x; **integer** n.

Будем обычно дополнять АЛГОЛ двумя не определяемыми в описаниях процедурами для ввода и вывода информации из машины. Первой процедуре присваивается всегда один и тот же идентификатор *read* (читать), а второй — идентификатор *punch* (перфорировать); фактическими параметрами каждой из этих процедур будут считаться либо некоторые величины типа **real**, **integer** или **Boolean**, либо идентификатор массива любого из этих типов.

Сделаем еще некоторые замечания. Обычно всякая программа в АЛГОЛе представляется в виде блока, т. е. заключается в операторные скобки **begin**—**end**. Для облегчения чтения «алгольных» программ в них могут быть введены так называемые *комментарии*, т. е. пояснения для программиста, не имеющие внутреннего смысла в языке АЛГОЛ и не воспринимаемые поэтому транслятором при автоматическом программировании.

Комментарием считается всякая последовательность символов (не обязательно «алгольных»), начинающаяся служебным словом **comment** (комментарий) после точки с запятой или слова **begin**, кончающаяся точкой с запятой и не содержащая внутри себя других вхождений точки с запятой. Комментарием считается также любая последовательность символов, следующая после слова **end** до точки с запятой или до конца программы, если она не содержит слов **end** и **else**, а также точки с запятой. Например, в выражениях **comment** текст; **begin comment** текст; **end** текст; слово «текст» представляет собой комментарий. С точки зрения «алгольных» программ первое выражение эквивалентно пустому месту, второе — слову **begin**, а третье — слову **end**.

Для электронных цифровых машин малой и средней мощности язык АЛГОЛ-60 является чересчур сложным, чтобы организовать эффективный перевод с него на язык машины. Был предложен поэтому упрощенный вариант АЛГОЛа, получивший название СМОЛГОЛ-61*.

Упрощения состоят в следующем. Во-первых, алфавит ограни-

---

* Описание языка СМОЛГОЛ-61 см.: Communications of the Assoc. for. Comp. Mach., 1961, v. 4, № 11, p. 499—502.

чивается либо одними строчными, либо одними прописными буквами латинского алфавита. Во-вторых, исключаются из рассмотрения логические операции импликации и эквивалентности, а также служебные слова **while**, **Boolean**, **true** и **false**. Таким образом, не разрешается использовать логические значения «истина» и «ложь». Пользование идентификаторами для обозначения логических величин также запрещается. Булевы переменные могут быть введены программистом косвенным образом, с помощью замены логического значения «истина» и «ложь» целыми числами — — 1 и 0. Булевы выражения разрешается использовать лишь в условиях. Если в АЛГОЛе значение какого-нибудь булева выражения $\mathfrak{B}$ присваивалось некоторому идентификатору P—P: $=\mathfrak{B}$, то в СМОЛГОЛе ему должно соответствовать присваивание вида P: $=$ *if* $\mathfrak{B}$ **then** 1 **else** 0, где в правой части стоит уже не булево, а арифметическое выражение.

Далее, длина идентификаторов ограничивается пятью буквами. Более точно, идентификаторы, у которых совпадают первые пять букв, считаются в СМОЛГОЛе одинаковыми. В арифметическом выражениии $a \uparrow b$ в величинах a и b типа **integer** не допускаются отрицательные значения для показателя степени b. Целые числа не употребляются в качестве меток. Шаг в операторе цикла должен оставаться либо все время положительным, либо все время отрицательным, причем в последнем случае знак «минус» должен быть поставлен явно перед выражением, задающим шаг. В списке цикла должен быть лишь один **step—until** элемент.

Во всех процедурах, кроме процедур ввода и вывода, нельзя употреблять строки в качестве фактических параметров. Ни одна процедура не может быть вызвана прежде, чем она будет описана. Исключается возможность использования одной процедуры внутри другой, если они были описаны в одном и том же блоке. Запрещается вторичный вызов одной и той же процедуры до тех пор, пока не будет закончен полностью ее первый вызов. Например, нельзя употреблять рекурсивный вызов процедуры $F(u,v)$ вида $F(x, F(x,y))$. Но повторное применение процедуры после ее окончания не возбраняется, так что выражение $\ln(\ln x)$ вполне допустимо в СМОЛГОЛе. Если процедура P является фактическим параметром другой процедуры, то все параметры процедуры P должны описываться как **value**.

Стандартные процедуры для нахождения знака и абсолютной величины числа нельзя использовать в качестве фактических параметров в любых процедурах. Если необходимо использовать их таким образом, то следует предварительно описать их как функции, т. е. выписать выражение **real procedure** *abs*(x); **value** x; **real** x; **begin** *abs*: $=$ *abs*(x) **end** (и аналогично для **integer procedure** *sign* (x)). Ни процедуры, ни их формальные параметры не могут иметь тип **Boolean**.

Собственные переменные не могут относиться к частям программы вне блока, в котором они определены. Границы массивов должны

быть постоянными. Элементами списков переключателей в описаниях переключателей могут быть только метки, а не произвольные именующие выражения.

Описания процедур, вызываемых в любом блоке, должны осуществляться после описания типов, переключателей и массивов соответствующего блока. Идентификаторы процедур могут появляться внутри процедуры только в том случае, когда они являются левыми частями соответствующих операторов присваивания. Существуют и некоторые другие ограничения.

Заметим, что любая программа, написанная на СМОЛГОЛе, может рассматриваться и как «алгольная» программа. Обратное, вообще говоря, неверно.

§ 5. Примеры программирования на языке АЛГОЛ-60

Рассмотрим сначала простейший пример, который уже фигурировал в §3 данной главы в качестве иллюстрации принципов программирования на машинных языках. Речь идет о вычислении значения суммы $\sum_{k=1}^{m} \frac{1}{k^2}$. Соответствующая «алгольная» программа может быть записана в виде

```
begin real s; integer m, k;
      read (m); s: = 0;
      for k: = 1 step 1 until m do
      s: = s + 1/k↑2;
      punch (s)
end.
```

Процедуры *read*(*m*) и *punch*(*s*) обеспечивают соответственно ввод необходимой информации (верхнего предела суммирования) и вывод результата, т. е. значения искомой суммы. Легко видеть, что программа, записанная на АЛГОЛе, гораздо более наглядна и понятна, чем выписанная в §3 машинная программа, которая решает ту же самую задачу.

Весьма прозрачно и наглядно могут быть записаны также программы других примеров, рассмотренных в §3. Вычисление скалярного произведения двух (вещественных) векторов (*a*1, *a*2, ..., *an*) и (*b*1, *b*2, ..., *bn*) представится в виде

```
begin integer n; read (n);
begin real s; integer i; real array a[1 : n], b [1 : n];
      read (a); read (b); s: = 0:
      for i: = 1 step 1 until n do
```

```
    s: = s + a [i]×b [i];
    punch (s)
end end.
```

Умножение вектора $(b1, b2, \ldots, bn)$ на матрицу $\|A_{ik}\|$ в АЛГОЛе может представиться программой

```
begin integer n; read (n);
begin integer i, k; real array s[1 : n], b[1 : n], A[1 : n, 1 : n];
    read (b); read (A);
    for k: = 1 step 1 until n do
    begin s [k]: = 0;
    for i: = 1 step 1 until n do
    s[k]: = s [k] + b [i] × A [i, k];
    end
    punch (s);
end end.
```

В этой программе один оператор цикла входит в другой. Введены внутренние операторные скобки, поскольку в первом (внешнем) цикле необходимо выполнять последовательность, состоящую из двух операторов.

Две фразы, написанные в конце программы, представляют собой комментарий, который не входит фактически в программу и не воспринимается транслятором (программирующей программой).

Отметим, что язык АЛГОЛ является универсальным алгоритмическим языком и пригоден поэтому для записи любых алгоритмов. Кроме того, как отмечалось выше, все программы, записанные на АЛГОЛе, могут быть реализованы (при условии наличия достаточно большого объема памяти) на любой универсальной электронной цифровой машине.

Используем последнее обстоятельство для иллюстрации того, что универсальные электронные цифровые машины могут выполнять не только обычные алгоритмы, но также алгоритмы со случайными переходами и *любые самоорганизующиеся* системы алгоритмов.

Чтобы получить возможность строить в АЛГОЛе какие угодно случайные алгоритмы, достаточно ввести в него специальную процедуру, которую будем обозначать как *random* (a, b). При каждом обращении к этой процедуре она формирует какое-либо случайное число, принадлежащее отрезку $[a, b]$. При этом предполагается, что выборка осуществляется на основе равномерного закона распределения, по которому все числа указанного отрезка считаются равновероятными. Сами же случайные числа предполагаются

имеющими тип **integer** или **real** в зависимости от того, какой тип приписывается формальным параметрам (границам отрезка) a, b. Разумеется, оба эти параметра должны иметь один и тот же тип.

Способ построения самой процедуры может варьироваться в довольно широких пределах. Можно, например, просто записать в память машины таблицу случайных чисел и построить процедуру их последовательной выборки. В ряде случаев к электронной цифровой машине присоединяется специальный датчик случайных чисел. При этом процедура заключается в выборке чисел, формируемых указанным датчиком, и их последующем преобразовании с целью приведения к заданному интервалу $[a, b]$.

Широко применяются также различные процедуры, формирующие последовательности так называемых *псевдослучайных чисел.* Для образования подобной последовательности можно воспользоваться, например, следующим приемом: выбирается некоторое положительное число a_1 и возводится в квадрат. В полученном числе $a_2 = a_1^2$ выбирается какая-нибудь группа разрядов (обычно не самых старших и не самых младших). Число b_2, образованное этими разрядами, принимается за первое псевдослучайное число. Возводя число b_2 в квадрат, получают новое число $a_3 = b_2^2$, с которым поступают точно так же, как и с числом a_2. Продолжая этот процесс, получаем требуемую последовательность псевдослучайных чисел.

Построенная таким образом последовательность, если длина ее не слишком велика, может считаться практически случайной. При большой же длине последовательности возникают различного рода зацикливания (циклические повторения уже встречавшихся ранее кусков последовательности), что и отличает псевдослучайные последовательности от чисто случайных. Однако для каждого конкретного случая может быть подобрана такая процедура формирования псевдослучайной последовательности, которая даст практически тот же результат, что и процедуры, формирующие чисто случайные последовательности.

С помощью описанных процедур полностью решается задача реализации на универсальных электронных цифровых машинах произвольных случайных алгоритмов. С реализацией на машинах самоорганизующихся систем алгоритмов дело обстоит по существу еще проще, поскольку в этом случае, как правило, не приходится прибегать к каким-либо специальным процедурам. Подобная реализация осуществляется обычными средствами, при помощи программ, записанных на языке АЛГОЛ. Приведем примеры такого рода самоорганизующихся систем, описанных в предыдущей главе.

В качестве первого примера рассмотрим самонастраивающийся алгоритм управления, основанный на методе наискорейшего спуска. Задачей этого алгоритма является выдача таких массивов $A[1:n]$ чисел (управляющих воздействий), чтобы некоторый критерий f имел наименьшее возможное значение. Критерий f представляет собой известную функцию некоторых параметров (показаний)

приборов, контролирующих процесс, значения которых в виде соответствующего массива $B[1:m]$ периодически вводятся в алгоритм.

Параметры, составляющие массив B (результаты управления) изменяются как вследствие изменения управляющих воздействий (массив A), так и в силу других причин, относящихся к управляемому объекту и не зависящих от алгоритма управления. Причины, о которых здесь идет речь, сводятся к изменению некоторых нерегулируемых параметров, причем характер этого изменения заранее не известен управляющему алгоритму. Легко понять, что описанный управляющий алгоритм осуществляет экстремальное регулирование (по критерию f), качество которого будет тем лучше, чем меньше отношение времени нахождения алгоритмом оптимальных управляющих воздействий (массив A) к среднему времени, в течение которого происходит ощутимое изменение нерегулируемых параметров. Нетрудно проверить, что управляющий алгоритм, о котором идет речь, может быть описан на языке АЛГОЛ следующей программой:

```
begin integer i, j, k; real y, s;
      B[1:m], P[1:n];
      L: read (B); y:= f(B); s:= 0;
      for i:= 1 step 1 until n do
      begin A[i]:= A[i] + d;
      punch (A); read (B);
      P[i]:= (f(B) - y)/d;
      s:= s + P[i]↑2;
      A[i]:= A[i] - d
      end;
      r:= d/sqrt (s);
      for j:= 1 step 1 until n do
      A[i]:= r × P[i] + A[i];
      punch (A); read (B);
      if aBs (y - f(B)) ⩾ l then go to L else
      for k:= 1 step 1 until n do
      A[i]:= A[i] - r × P[i];
      punch (A);
      go to L
end.
```

. В построенной программе величина d представляет собой шаг наискорейшего спуска, а величина l — точность достижения минимальной величины критерия f. Функция $sqrt(s)$ равна квадратному корню из s, взятому со знаком «плюс». Массив $P[1:n]$ задает относительные величины оптимальных приращений величин управляющих воздействий $A[1:n]$ на каждом шаге процесса наискорейшего спуска. Предполагается, что величины d, l, вещественные процедуры $f(B)$ и $sqrt(s)$ и начальные значения компонент массива $A[1:n]$ были заранее (до приказа с меткой L) введены в алгоритм.

Рассмотрим теперь алгоритм, выполняющий работу некоторого дискретного α-персептрона P. Предположим, что персептрон P имеет сетчатку, состоящую из N рецепторов, и рассчитан на распознавание двух образов. A-элементами являются (1,1,1)-нейроны, константа поощрения равна единице, а константа штрафа — нулю. Изображение, проектируемое на сетчатку, представляется некоторым булевым массивом $r[1:N]$, который считывается извне при показе персептрону каждого нового изображения. Извне считывается также булева величина a, представляющая собой подаваемый сигнал о правильности (истинности) или неправильности (ложности) выданного персептроном ответа p. Ответ p является просто номером образа, к которому персептрон отнес очередное показанное ему изображение. Через sf и ss обозначим выходные сигналы сумматоров первого и второго образов.

Предположим, далее, что число нейронов как первого, так и второго образа равно n. Через $xf[i]$ и $yf[i]$ обозначим номера рецепторов сетчатки, к которым подсоединены соответственно возбуждающий и запрещающий входы i-го нейрона первого образа, через $xs[i]$ и $ys[i]$ соответствующие номера рецепторов для i-го нейрона второго образа, а $vf[i]$ и $vs[i]$—веса i-ых нейронов первого и второго образов ($i = 1, 2,..., n$). При принятых предположениях алгоритм, выполняющий работу персептрона P (в режиме обучения), может быть записан в виде

```
begin integer p, i, j, k; real sf, ss; Boolean a;
    Boolean array r[1:N];
    L: read (r); sf: = 0; ss: = 0;
    for i: = 1 step 1 until n do
    begin if r[xf[i]] ∧ ¬r[yf[i]] then
    sf: = sf + vf[i];
    if r[xs[i]] ∧ ¬r[ys[i]] then
    ss: = ss + vs[i]
    end;
    if sf > ss then p: = 1;
```

```
if sf < ss then p: = 2;
if sf = ss then go to L;
punch (p); read (a);
if a ∧ (p = 1) then for j: = 1 step 1 until n do
if r[xf[j]] ∧ ¬ r[yf[j]] then vf[j]: = vf[j] + 1;
if a ∧ (p = 2) then for k: = 1 step 1 until n do
if r[xs[k]] ∧ ¬ r[ys[k]] then vs[k]: = vs[k] + 1;
go to L
end.
```

Предполагается, что массивы $xf[1:n]$, $yf[1:n]$, $xs[1:n]$, $ys[1:n]$, $vf[1:n]$ и $vs[1:n]$ заранее введены в программу, а последние два массива (веса нейронов первого и второго образов) состоят не из одних лишь нулей. Иначе сумматоры образов все время выдавали бы равные нулю сигналы (sf и ss) и, поскольку в программе принято, что при равных сигналах сумматоров персептрон не выдает никакого сигнала p, процесс обучения (и, вообще, какого-либо изменения весов) отсутствовал бы.

От последнего ограничения можно избавиться, если учитель не просто подает сигнал одобрения, а сообщает персептрону истинный номер образа, которому принадлежит очередное показываемое персептрону изображение. Именно такой способ функционирования персептрона в режиме обучения рассматривался в предыдущей главе. Укажем на те изменения в описанной выше программе, которые должны быть сделаны применительно к новому типу сигнала a.

В описании величина a должна декларироваться как величина типа **integer**, а не **Boolean**. Изменения в программе могут быть сведены к следующим. После оператора **if** $sf < ss$ **then** $p: = 2$ вместо оператора **if** $xf = ss$ **then go to** L нужно поставить оператор **if** $sf \neq ss$ **then** *punch*(p). Затем в условных операторах, следующих за оператором *read* (a), условия **if** $a \wedge (p = 1)$, **if** $a \wedge (p = 2)$ должны быть заменены условиями **if** $a = 1$ и **if** $a = 2$ соответственно.

Нетрудно описать также и те изменения в исходной персептронной программе, которые должны быть внесены для того, чтобы моделировать не режим обучения, а режим самообучения персептрона. Для этого величина a совсем исключается из программы вместе с соответствующим оператором *read*(a). В последующих условных операторах к условиям, стоящим после служебных слов **do** должны быть добавлены члены $\wedge (sf > ss)$ и $\wedge (sf < ss)$ соответственно.

Таким образом, и режим обучения, и режим самообучения α-персептрона могут быть легко промоделированы на языке АЛГОЛ, а следовательно, и на универсальных электронных циф-

ровых машинах. То же самое относится, как легко понять, к любым видоизменениям и обобщениям схемы персептрона.

Опишем теперь средствами языка АЛГОЛ еще одну самоорганизующуюся алгоритмическую систему, которая моделирует процесс биологической эволюции и образование новых видов. Моделирование подобной (хотя и несколько отличной) системы на универсальной электронной цифровой машине осуществлено А. А. Летичевским [49].

Рассмотрим некоторое дискретное пространство, состоящее из конечного множества точек с номерами от 1 до n включительно. Предположим для простоты это множество циклически упорядоченным. Иными словами, для каждой точки i определим две соседние с ней точки — непосредственно предшествующую ей точку bi и непосредственно следующую за ней точку fi. Если $i \neq 1$, то $bi = i - 1$; для $i = 1$ полагаем $bi = n$. Аналогично, если $i \neq n$, то $fi = i + 1$, а для $i = n$ полагаем $fi = 1$.

Каждой точке i пространства припишем некоторое состояние $s[i]$, которое может принимать любое целое значение от 0 до k включительно. Если $s[i] = 0$, то соответствующая точка считается «безжизненной». Если же $s[i] \neq 0$, то полагаем, что в точке i присутствует некоторое «живое существо», находящееся в состоянии $s[i]$. В качестве подобных «живых существ» в рассматриваемой модели выбираются абстрактные автоматы с одним и тем же числом состояний (равным k), но, вообще говоря, с различными таблицами переходов и выходов.

Кроме того, для каждой точки i нашего пространства задается число $F[i]$, равное 1 или 0 в соответствии с тем, есть в i-ой точке «пища» или нет. В случае наличия «пищи» в той или иной точке ее запасы предполагаются столь большими (или самовосстанавливаемыми), что автомат, находящийся в этой же точке, в процессе своего «питания» практически не меняет их. Массив F изменяется на каждом очередном такте работы алгоритма в соответствии с некоторым «законом природы», который будем предполагать находящимся вне нашего алгоритма.

Помимо собственно состояния $s[i]$ автомата, занимающего точку i, с этим автоматом связываются еще два числа, а именно показания счетчика его «жизни» $L[i]$ и так называемого счетчика «голода» $H[i]$. Величина $L[i]$ увеличивается на единицу на каждом такте работы алгоритма, а после того как ее значение превысит некоторый заранее заданный порог l, соответствующий автомат переходит в нулевое состояние, т. е., попросту говоря, уничтожается (имитируя тем самым естественную смерть).

Величина $H[i]$ увеличивается на единицу, если $F[i] = 0$ (т. е. в том случае, когда автомат находится в точке пространства, лишенной «пищи»), и уменьшается на единицу в противном случае, не принимая, однако, отрицательных значений (в случае, когда $H[i] = 0$, полагаем $H[i] - 1 = 0$). При превышении величиной $H[i]$ некоторого фиксированного заранее порога h соответствую-

щий автомат переходит в нулевое состояние (имитируя тем самым смерть от голода).

Входными сигналами автомата, находящегося в точке i, являются состояния $s[bi]$ и $s[fi]$ соседних с ним точек, а также сигналы $F[bi]$, $F[i]$, $F[fi]$ о наличии или отсутствии пищи как в самой точке i, так и в соседних с нею точках. Выходным сигналом m является так называемое *движение* автомата, т. е., иными словами, приращение номера занимаемой им точки пространства за данный такт работы алгоритма. Будем считать, что величина m может принимать лишь три различных значения: 0, 1 и —1.

Ввиду наличия пяти входных каналов, таблицы переходов и выходов каждого автомата могут быть заданы в виде шестимерных массивов. Для задания таблиц переходов и выходов всех автоматов сразу используем семимерные массивы $SP[1:n, 1:k; 0:k, 0:k, 0:1, 0:1, 0:1]$ и $M[1:n, 1:k, 0:k, 0:k, 0:1, 0:1, 0:1]$ соответственно. Первый индекс в каждом из этих массивов указывает номер ячейки i, занимаемой автоматом, второй — состояние $s[i]$ этого автомата, третий и четвертый — состояния соседних точек $s[bi]$ и $s[fi]$, пятый, шестой и седьмой индексы представляют собой сигналы $F[i]$, $F[bi]$ и $F[fi]$ о наличии или об отсутствии «пищи» в самой точке и в соседних с нею точках. Для движения, задаваемого таблицей выходов M, не будем вводить никаких ограничений в самой таблице, однако выполнение соответствующего движения будет осуществляться лишь в том случае, когда точка, в которую смещается автомат, не занята каким-либо другим автоматом.

Изменение показаний счетчиков «жизни» и «голода» производится после выполнения движения и перевода автомата в новое состояние. Если при этом не произойдет «гибели» автомата, а его движение окажется нетривиальным (т. е. автомат не останется на прежнем месте), то при выполнении некоторых дополнительных условий происходит «размножение» автомата путем деления. При этом сдвинувшийся автомат A полностью сохраняет свою структуру, за исключением того, что на его счетчике «жизни» устанавливается значение, равное нулю. А на том месте, которое автомат A занимал перед этим, возникает его «двойник», отличающийся от A лишь тем, что в каждом из двух массивов, задающих переходы и выходы автомата A, одно число (соответственно новое состояние или движение автомата) заменяется случайным числом. Счетчик «жизни» нового автомата также устанавливается в нуль.

Дополнительные условия для возможности размножения, о которых шла речь выше, состоят в том, чтобы показание счетчика «жизни» автомата было заключено между двумя фиксированными заранее числами ll и lu, а показание счетчика «голода» не превышало некоторого числа hu, также фиксированного заранее.

Для образования случайных чисел фиксируем специальную целочисленную процедуру *random* (a, b), выдающую при каждом ее вызове некоторое целое число, расположенное на замкнутом отрезке $[a, b]$. Все целые числа указанного отрезка считаются при

этом равновероятными. Способы построения подобных процедур были описаны выше.

Описанный нами алгоритм за один такт работы должен осуществить просмотр всех точек нашего пространства, произведя в них перечисленные выше изменения. Выполнив очередной такт, алгоритм должен прочитать новые значения всех компонент массива $F[1:n]$ и приступить к выполнению следующего такта. Будем осуществлять счет числа тактов с помощью специальной величины t. При достижении этой величиной фиксированного заранее значения p алгоритм должен прекратить свою работу.

Для облегчения программирования описанного алгоритма на языке АЛГОЛ введем три блока, описывающих процесс перемещения автомата, находящегося в i-ой точке, процесс его «гибели» и процесс его «размножения». Обозначим для краткости эти блоки через $\mathfrak{B}_1$, $\mathfrak{B}_2$ и $\mathfrak{B}_3$ соответственно и выпишем для каждого из них соответствующую программу на АЛГОЛе.

Блок $\mathfrak{B}_1$:

```
begin integer m, j, bs, fs, f, bf, ff;
      m: = M[i, s[i], s[bi], s[fi], F[i], F[bi] F[fi]];
      pi: = if i + m > n then 1 else i + m;
      if s[pi] ≠ 0 then pi: = i comment pi есть номер точки,
      в которую смещается рассматриваемый автомат;
      L[pi]: = L[i] + 1;
      H[pi]: = if F[pi] = 0 then H[i] + 1 else if H[i] ≠ 0
      then H[i] — 1 else 0;
      if pi ≠ i then
      for j: = 1 step 1 until k do
      for bs: = 0 step 1 until k do
      for fs: = 0 step 1 until k do
      for f: = 0,1 do for bf = 0,1 do for ff: = 0.1 do
      begin SP[pi, j, bs, fs, f, bf, ff]: = SP[i, j, bs, fs, f, bf, ff];
      M[pi, j, bs, fs, f, bf, ff]: = M[i, j, bs, fs, f, bf, ff]
      end;
      s[pi]:=SP[i, s[i], s[bi], s[fi], F[i], F[bi], F[fi]]
end
```

Блок $\mathfrak{B}_1$ осуществляет передвижение автомата из точки i в точку pi, переход его в новое состояние (определяемое ситуацией в момент нахождения автомата в точке i), изменение показаний счетчиков «жизни» и «голода» и переписывание массивов, задающих функции переходов и выходов автоматов, с целью привести их в соответствие с новым месторасположением рассматриваемого автомата. При выходе из блока сохраняются значения i и pi.

Блок $\mathfrak{B}_2$ весьма прост: **begin** $s[pi] := 0$ **end.**

Заметим, что программа будет строиться нами таким образом, чтобы сохраняющиеся после гибели автомата в точке pi значения $L[pi]$ и $H[pi]$, как и значения соответствующих компонент массивов SP и M, не могли привести к ошибкам в будущем. Это достигается за счет того, что при повторении программы перечисленные величины, прежде чем быть использованными, определяются заново, поскольку они обязательно попадут предварительно в левые части соответствующих операторов присваивания.

Блок «размножения» $\mathfrak{B}_3$:

```
begin integer rj, rbs, rfs, rf, rbf, rff, j, bs, fs, f, bf, ff;
        Boolean B;
        rj: = random (1, k);
        rbs: = random (0, k);
        rfs: = random (0, k);
        rf: = random (0, 1);
        rbf: = random (0, 1);
        rff: = random (0,1);
        for j: = 1 step 1 until k do
        for bs: = 0 step 1 until k do
        for fs: = 0 step 1 until k do
        for f: = 0,1 do for bf: = 0,1 do for ff: = 0.1 do
    begin B: = j = rj ∧ bs = rbs ∧ fs = rfs ∧ f = rf ∧ bf = rbf ∧ ff = rff;
        SP[i, j, bs, fs, f, bf, ff]: = if B then random (1, k)
        else SP[pi, j, bs, fs, f, bf, ff];
        M[i, j, bs, fs, f, bf, ff]: = if B then random (— 1, 1)
        else M[pi, j, bs, fs, f, bf, ff]
    end
    end.
```

Вся программа моделирования процесса эволюции представляется теперь следующим образом:

```
begin integer t, i, bi, fi, p, q, pi;
      integer array F[1:n], S[1:n], L[1:n], H[1:n],
      SP[1:n, 1:k, 0:k, 0:k, 0:1, 0:1, 0:1], M[1:n, 1:k, 0:k,
      0:k, 0:1, 0:1, 0:1];
      for q = 1 step 1 until n do
      L[q]: = H[q]:=0;
      t: = 0; i: = 1; read (S); read (SP); read (M);
      Q: read (F); fi: = if i ≠ n then i + 1 else 1;
      if S[i] = 0 then begin i: = fi; go to P end;
      bi: = if i ≠ 1 then i — 1 else n;
                        |Блок 𝔅1|;
      if L[pi] > l ∨ H[pi] > h then
                        |Блок 𝔅2|;
      if pi ≠ i∧H[pi] ⩽ hu∧L[pi]⩽lu∧L[pi] ⩾ ll then
                        |Блок 𝔅3|;
    if pi ≠ fi then i: = fi else i: = if fi ≠ n then fi + 1 else 1;
      P:if i = 1∨pi = 1 then t: = t + 1; if t ≠ p then go to Q
end.
```

Заметим, что построенная нами программа неэкономна с точки зрения расходования памяти и необходимости перезаписывания многомерных массивов. Можно добиться гораздо более экономного построения программы, если ввести нумерацию автоматов и использовать в массивах L, H, SP и M вместо номера точки пространства номер соответствующего автомата.

При реальном моделировании эволюционного процесса на универсальной электронной цифровой машине в работе [43] использовалась программа с более ограниченными возможностями и, тем не менее, проведенные опыты показали, что даже при этих условиях качество моделирования было достаточно удовлетворительным. Для относительно простых «законов природы» процесс приспособления автоматов к окружающей среде и образование устойчивых «видов» наблюдались через несколько десятков тысяч тактов работы алгоритма и смены соответствующего числа «поколений». Первоначально таблицы переходов и выходов автоматов (массивы M и SP в нашем случае) задавались произвольно. В процессе эволюции происходило «вымирание» плохо устроенных автоматов и возникновение форм, более приспособленных для «жизни» в заданных условиях.

Глава VI

ИСЧИСЛЕНИЕ ПРЕДИКАТОВ И ПРОБЛЕМА АВТОМАТИЗАЦИИ ПРОЦЕССОВ НАУЧНОГО ТВОРЧЕСТВА

§ 1. Основные понятия исчисления предикатов

Как отмечалось в гл. II, простейшая составная часть математической логики — исчисление высказываний — не проникает по существу в строение элементарных высказываний, ограничивая тем самым свои возможности в формализации сколько-нибудь сложных мыслительных процессов. Гораздо более сильными выразительными средствами обладает следующая, более сложная ступень математической логики, называемая *узким исчислением предикатов*, или *исчислением предикатов первой ступени*.

Особенностью исчисления предикатов является прежде всего то, что, наряду с переменными высказываниями, принимающими лишь два возможных значения («истина» и «ложь»), вводятся в рассмотрение так называемые *предметные переменные*, пробегающие некоторую, вообще говоря, бесконечную область значений, которую принято называть *предметной областью*. Составляющие эту область значения называются обычно *предметами*, или *объектами*.

Фиксируя ту или иную предметную область, получаем возможность строить *пропозициональные функции* предметных переменных, называемые обычно *предикатами*: n-местный предикат $P(x_1, x_2, \ldots, x_n)$ представляет собой переменное высказывание, истинность или ложность которого определяется наборами значений предметных переменных $x_1, x_2, \ldots, x_n$. Если предикат P не является *тождественно истинным* или *тождественно ложным*, то на одних наборах значений предметных переменных он принимает значение «истина», а на других — значение «ложь».

В классической теории предикатов предикатами (или свойствами) назывались лишь одноместные предикаты. Для многоместных предикатов употреблялся специальный термин «*отношение*»: двух-

местные предикаты назывались *бинарными* отношениями, трехместные — тернарными отношениями и т. д. Для наших целей нет необходимости в специальном подчеркивании указанного различия, поэтому будем называть предикатами пропозициональные функции от любого (большего нуля) числа предметных переменных.

Использование предикатов позволяет построить формальный язык, аналогичный исчислению высказываний, но, в отличие от него, проникающий в структуру элементарных высказываний. Например, высказывание «четыре больше двух» в исчислении высказываний является неразложимым. Если же ввести предикат $P(x, y)$ с множеством целых неотрицательных чисел в качестве предметной области, истинный тогда и только тогда, когда выполняется неравенство $x > y$, то приведенное высказывание запишется в форме $P(4, 2)$, которая дает уже представление о внутренней структуре высказывания.

Точно таким же образом может быть выявлена внутренняя структура высказывания «кислород является газом». Для этой цели достаточно ввести предикат «быть газом», принимающий значение «истина» тогда и только тогда, когда в него подставляется объект предметной области, действительно являющийся газом. Если обозначить указанный предикат через $Q(x)$, то приведенная нами фраза может быть записана в виде Q (кислород).

При формальном построении исчисления предикатов обычно не интересуются тем, из каких именно объектов состоит та или иная предметная область, достаточно знать лишь общее число всех этих объектов или, выражаясь более точно, мощность множества всех объектов, составляющих предметную область. Если предметная область конечна или счетна, составляющие ее объекты могут быть заменены их номерами. Тем самым предметные области сводятся к некоторым числовым множествам, что облегчает задачу конкретного выражения соответствующих предикатов. Поскольку предикаты представляют собой переменные высказывания, с ними можно производить все операции, применявшиеся во второй главе при построении исчисления высказываний. Вместе с тем наличие предметных переменных позволяет ввести ряд новых операций, специфических для исчисления предикатов. Построение указанных операций осуществляется с помощью так называемых *кванторов*. Обычно ограничиваются лишь двумя видами кванторов, называемых *кванторами существования* и *кванторами общности*. Для их обозначения будем использовать знаки $\exists x$ и $\forall x$ соответственно, где x указывает ту переменную, на которую действует квантор.

Выражение $\exists x P(x)$ представляет собой условное обозначение для высказывания «существует такой объект x, для которого предикат P является истинным». Аналогичным образом выражение $\forall x P(x)$ обозначает высказывание «для всех объектов x предикат P является истинным». При этом подразумевается, что объекты,

о которых здесь идет речь, принадлежат той или иной фиксированной предметной области M. Если область M состоит из конечного числа объектов $x_1, x_2, \ldots, x_k$, выражение $\exists x P(x)$ сводится к дизъюнкции $P(x_1) \vee P(x_2) \vee \ldots \vee P(x_k)$, а выражение $\forall x P(x)$ — конъюнкции $P(x_1) \wedge P(x_2) \wedge \ldots \wedge P(x_k)$. В случае же бесконечной предметной области подобное сведение оказывается невозможным, поскольку конструктивный характер наших построений исключает возможность пользования бесконечными дизъюнкциями и конъюнкциями.

При неконструктивном (так называемом теоретико-множественном) подходе к построению исчисления предикатов можно всегда представлять себе выражения $\exists x P(x)$ и $\forall x P(x)$ как дизъюнкцию и конъюнкцию, распространенные на все объекты x, составляющие заданную предметную область M. При конструктивном подходе подобное представление может использоваться лишь для эвристических (наводящих) рассуждений и построений, но отнюдь не как способ строго формального доказательства.

В выражениях $\exists x P(x)$ и $\forall x P(x)$ переменная x оказывается *связанной* соответствующим квантором. В отличие от *свободных* (несвязанных) переменных, например переменных x и y в выражении $Q(x, y)$, связанные переменные не имеют самостоятельного (индивидуального) значения, поскольку высказывание, содержащее связанные переменные, фактически *не зависит* от этих переменных. Роль связанных переменных в исчислении предикатов в этом смысле совершенно аналогична роли, которую играет переменный индекс i при вычислении суммы $\sum_{i=1}^{n} f(i)$ или переменная интегрирования x при вычислении определенного интеграла $\int_a^b f(x)dx$. Можно, в частности, заменять связанную переменную любой другой переменной, совершенно не меняя при этом смысла и значения соответствующего выражения.

Произвольная формула узкого исчисления предикатов строится с помощью четырех операций исчисления высказываний (отрицания, дизъюнкции, конъюнкции и импликации) и двух операций связывания с помощью предметных кванторов (общности и существования) из элементарных высказываний, в качестве которых выступают обычные переменные высказывания (пропозициональные буквы) и определенные выше пропозициональные функции (предикаты). При этом предметная область предполагается фиксированной, а общее количество символов, составляющих любую формулу, должно быть непременно конечным. Для единства терминологии элементарные переменные высказывания, не зависящие от предметных переменных (т. е. пропозициональные буквы), удобно рассматривать как нульместные предикаты (пропозициональные функции пустого множества предметных переменных).

Как и в исчислении высказываний, в исчислении предикатов для обозначения порядка действий в формулах могут употребляться круглые скобки. С помощью таких скобок устанавливаются также *области действия* входящих в формулу кванторов. Например, в выражении $\exists x(P(x,y) \supset Q(x)) \supset R(x)$ область действия квантора существования $\exists x$ включает в себя лишь выражение $P(x,y) \supset Q(x)$. Переменная x, входящая в последнее выражение, связана указанным квантором, в то же время переменная x в предикате $R(x)$ должна рассматриваться как свободная переменная.

Во избежание путаницы при различного рода преобразованиях формул в исчислении предикатов обычно предпочитают переобозначать связанные переменные так, чтобы их обозначения отличались как друг от друга, так и от обозначений всех свободных переменных, входящих в ту же самую формулу. В таком случае можно считать, что область действия каждого квантора распространяется от места его вхождения до самого конца формулы. Тем самым употребление скобок для обозначения областей действия кванторов может быть сделано излишним. В дальнейшем будем придерживаться, как правило, именно такого толкования областей действия кванторов.

Заметим, что в узком исчислении предикатов разрешается связывать с помощью кванторов лишь предметные переменные. Входящие в формулу предикаты предполагаются при этом неизменными. Подобное ограничение, естественно, суживает область применений строящегося нами логического исчисления, чем и объясняется включение в его название термина «узкое». В так называемом *расширенном исчислении предикатов* употребляются переменные предикаты и предикатные кванторы. Иначе говоря, допускаются выражения вида «для всякого предиката P справедливо $P(x)$» или «существует такой предикат Q, для которого истинно высказывание $Q(x)$» и т. п. При неограниченном употреблении предикатных кванторов возникает возможность построения внутренне противоречивых формул и появления парадоксов. Все это вызывает необходимость дальнейшего усложнения соответствующих исчислений. Не будем, однако, заниматься подробным изучением расширенного исчисления предикатов, сосредоточив свое внимание на узком исчислении предикатов. Поэтому условимся в дальнейшем под исчислением предикатов подразумевать всегда (если не оговорено противное) именно узкое исчисление предикатов. Не будем также рассматривать и другие возможные обобщения исчисления предикатов, например исчисления предикатов не с одной, а с несколькими предметными областями и т. п.

Как и в случае исчисления высказываний, при построении исчисления предикатов недостаточно указать лишь способ записи формул. Необходимо задать еще правила преобразования формул, выражаемые аксиомами. В число аксиом исчисления предикатов включаются все 11 аксиом исчисления высказываний, выписанные в § 5 гл. II. Кроме них, вводится еще четыре постулата, специфи-

ческих для исчисления предикатов, которым присвоим номера от 12 до 15 включительно:

12. $\dfrac{C \supset P(x)}{C \supset \forall x P(x)}$.

13. $\forall x P(x) \supset P(t)$.

14. $P(t) \supset \exists x P(x)$.

15. $\dfrac{P(x) \supset C}{\exists x P(x) \supset C}$.

С помощью указанных постулатов (аксиом) можно осуществлять *формальный вывод* новых формул совершенно таким же способом, как и в случае исчисления высказываний. Заметим лишь, что выражения $P(x)$ и $P(t)$ в аксиомах 12—15 должны пониматься не только как элементарные одноместные предикаты, но и как *произвольные формулы* исчисления предикатов, содержащие буквы x и t в качестве свободных переменных. При этом не исключается, что в соответствующие формулы входят и другие свободные переменные. При формально-аксиоматическом построении исчисления предикатов не пользуются обычно символами индивидуальных объектов или индивидуальных предикатов, так что предикаты, входящие в формулы, принимаются как произвольные, а не фиксированные предикаты. Вместо пропозициональных букв A, B, C в аксиомы 1 —15 могут подставляться произвольные формулы исчисления предикатов, в том числе и такие, которые содержат свободные переменные.

При всех подстановках, о которых здесь идет речь, подразумевается, что свободные и связанные переменные, как и различные связанные переменные, в формулах, получаемых в результате подстановок, должны быть обозначены различными буквами. Такое условие позволяет избежать так называемой *коллизии переменных*, вызывающей непредвиденное связывание переменных, которые должны были бы остаться свободными. Действительно, если, скажем, в формулу $\exists x P(x) \supset C$ вместо C подставить формулу $Q(x)$, то квантор существования $\exists x$ связал бы не только переменную в предикате P, но и переменную в предикате Q. Коллизии переменных всегда можно избежать за счет переименования связанных переменных. В дальнейшем в случае необходимости такого переименования будем предполагать его каждый раз выполненным.

При условии, когда принимаются необходимые меры предосторожности, чтобы избежать коллизии переменных, в исчисление предикатов переносятся все результаты о выводимости одних формул из других, полученные ранее в исчислении высказываний (см. формулы 1 —7 из § 5 гл. II). В исчислении предикатов остается справедливой (с соответствующими оговорками) и теорема о дедукции. В частности, если формула $\mathfrak{B}$ выводима (в исчислении предикатов) из формулы $\mathfrak{A}$, то формула $\mathfrak{A} \supset \mathfrak{B}$ (при условии принятия мер против возникновения коллизии переменных) будет выводимой в исчислении предикатов.

Из аксиом 12 —15 легко выводятся следующие правила выводимости:

введение квантора общности ($\forall$-введение)

$$A(x) \overset{x}{\vdash} \forall x A(x); \tag{115}$$

введение квантора существования ($\exists$-введение)

$$A(t) \vdash \exists x A(x); \tag{116}$$

удаление квантора общности ($\forall$-удаление)

$$\forall x A(x) \vdash A(t); \tag{117}$$

так называемое $\exists$-удаление: если $\Gamma, A(x) \vdash C$, то (см. С. К. Клини [42])

$$\Gamma, \exists x A(x) \overset{x}{\vdash} C. \tag{118}$$

Значок переменной x, который пишется над символом выводимости $\vdash$, означает, что соответствующая переменная в процессе вывода *варьируется* (превращаясь из свободной переменной в связанную) в соответствии с аксиомами (правилами вывода) 12 и 15. Свободные переменные, не варьирующиеся в процессе вывода, принято называть *фиксированными*. Последнее понятие можно использовать для уточнения формулировки теоремы о дедукции.

Если имеет место выводимость $\Gamma, A \vdash B$, причем в процессе вывода свободные переменные, входящие в формулу A, остаются фиксированными, то имеет место выводимость $\Gamma \vdash A \supset B$.

Используя символ эквивалентности $\sim$ в том же смысле, что и в исчислении высказываний, и обозначая через A любую формулу, не содержащую свободной переменной x, можно легко установить следующие соотношения:

$$\vdash \forall x A \sim A, \quad \vdash \exists x A \sim A; \tag{119}$$

$$\vdash \forall x \forall y P(x, y) \sim \forall y \forall x P(x, y); \tag{120}$$

$$\vdash \exists x \exists y P(x, y) \sim \exists y \exists x P(x, y); \tag{121}$$

$$\vdash \forall x P(x) \supset \exists x P(x); \tag{122}$$

$$\vdash \exists x \forall y P(x, y) \supset \forall y \exists x P(x, y). \tag{123}$$

Правила (120) и (121) показывают возможность изменения порядка применения одноименных кванторов. Для разноименных кванторов подобное обстоятельство уже не имеет места, поскольку соотношение $\vdash \forall x \exists y P(x,y) \supset \exists y \forall x P(x,y)$, двойственное соотношению (123), в общем случае уже не имеет места в исчислении предикатов. Чтобы убедиться в этом, достаточно рассмотреть в качестве предметной области множество всех натуральных чисел, а в качестве предиката $P(x,y)$ предикат, истинный тогда и только тогда, когда $x < y$. Тогда формула $\forall x \exists y P(x,y)$ выражает собой выска-

зывание «для всякого натурального числа существует натуральное число y, большее, чем x». В то же время формула $\exists y \forall x P(x,y)$ представляет собой высказывание «существует натуральное число, большее всех натуральных чисел». Первое высказывание является истинным, а второе ложным. Поэтому высказывание $\forall x \exists y P(x,y) \supset \supset \exists y \forall x P(x,y)$ при рассматриваемой интерпретации является ложным и не должно быть выводимым в (содержательно) непротиворечивом исчислении.

Содержательная непротиворечивость исчисления предикатов (как и исчисления высказываний) понимается нами в том смысле, что выводимыми в этом исчислении могут быть лишь *тождественно истинные* формулы, т. е. такие формулы, которые остаются истинными для любой предметной области и для любой конкретной интерпретации входящих в них предикатов. Не связывая себя требованием конструктивности рассуждений (т. е. оставаясь в рамках теоретико-множественного подхода к исчислению предикатов), легко понять, что формулы, выражаемые аксиомами 13 и 14, как и формулы, выражаемые аксиомами 1 — 10 исчисления высказываний, являются тождественно истинными формулами.

Правила вывода (11, 12 и 15) также приводят к тождественно истинным формулам при условии тождественной истинности их посылок. В качестве примера рассмотрим правило вывода 12. Тождественная истинность посылки $C \supset P(x)$ может иметь место лишь в двух случаях: либо, когда высказывание C ложно, либо когда высказывание $P(x)$ всегда истинно. В обоих этих случаях имеет, очевидно, место также истинность высказывания $C \supset \forall x P(x)$.

Подобными рассуждениями доказывается (хотя и не вполне конструктивно) содержательная непротиворечивость исчисления предикатов. Неконструктивность, о которой здесь идет речь, связана с неявно предполагавшейся возможностью перебора всех значений предметных переменных при определении тождественной истинности, что в случае бесконечной предметной области требует бесконечного числа шагов, не согласующегося с требованием финитности, обязательным для строго конструктивных построений.

Ограничиваясь лишь конечными предметными областями, легко придать доказательству непротиворечивости исчисления предикатов вполне конструктивный характер. При таком ограничении исчисление предикатов по существу сводится к исчислению высказываний, поскольку связывание кванторами представляет собой в этом случае просто краткую форму записи распространенных на все объекты предметной области конъюнкций и дизъюнкций, а соотношения, выражаемые аксиомами 12 —15, оказываются выводимыми из аксиом 1 —11. Благодаря возможности подобной интерпретации вопрос о непротиворечивости исчисления предикатов сводится к соответствующему вопросу для исчисления высказываний, решенному ранее.

Подобным путем устанавливается и *формальная непротиворечивость* (называемая также простой непротиворечивостью) исчисле-

ния предикатов, т. е. невозможность вывести в этом исчислении некоторую формулу вместе с ее отрицанием.

Проблема *содержательной полноты* исчисления предикатов, т. е. возможность формального вывода в этом исчислении любой тождественно истинной формулы, была решена в положительном смысле К. Гёделем [16]. Разумеется, в случае бесконечных предметных областей установление содержательной полноты исчисления предикатов требует привлечения средств, выходящих за пределы финитной математики. В случае конечных предметных областей вопрос о содержательной полноте исчисления предикатов сводится к соответствующему вопросу для исчисления высказываний и потому решается конструктивно.

В противоположность содержательной полноте, называемой также *полнотой в широком смысле*, *полнота в узком смысле* уже не имеет места в исчислении предикатов. Действительно, к числу аксиом исчисления предикатов может быть присоединена формула $\exists x P(x) \supset \forall x P(x)$, не выводимая в этом исчислении и не приводящая к возникновению противоречия. Непротиворечивость возникающей в результате указанного присоединения системы аксиом становится ясной при рассмотрении предметной области, состоящей из единственного объекта. Вновь присоединенная аксиома обращается при этом в тождественно истинную формулу. В то же время для предметной области, состоящей уже из двух объектов x и y, указанная аксиома превращается в формулу $P(x) \vee P(y) \supset P(x) \wedge P(y)$, не являющуюся тождественно истинной и потому не выводимую из остальных аксиом.

При теоретико-множественном подходе к построению исчисления предикатов удается доказать следующую интересную теорему, принадлежащую А. И. Мальцеву [52].

Теорема 1. *Если бесконечная дизъюнкция (конечных) формул узкого исчисления предикатов есть тождественно истинная формула, то тождественно истинной является некоторая конечная дизъюнкция указанных формул.*

Эта теорема может с успехом применяться при доказательствах так называемых *локальных теорем*, позволяющих в ряде случаев переносить на бесконечные множества свойства, справедливые для всех их конечных подмножеств.

Наряду с тождественно истинными формулами, в исчислении предикатов целесообразно рассматривать так называемые *выполнимые* формулы. *Выполнимой будем называть такую формулу, которая может быть сделана истинной при выборе подходящей предметной области и должном определении заданных на ней предикатов.* Подразумевается, что формула, о которой здесь идет речь, не содержит символов индивидуальных предметов и индивидуальных предикатов.

Всякая тождественно истинная формула является вместе с тем и выполнимой, но обратное в общем случае, разумеется, неверно. Примером выполнимой, но не тождественно истинной формулы

может служить рассмотренная выше формула $\exists x P(x) \supset \forall x P(x)$. Эта формула является тождественно истинной лишь на таких предметных областях, которые состоят из одного-единственного объекта.

Ясно, что формула, не выполнимая на некоторой предметной области, *тождественно ложна* на этой области. Отрицания тождественно ложных формул представляют собой тождественно истинные формулы. Тем самым устанавливается связь между понятиями выполнимости и тождественной истинности формул исчисления предикатов.

Можно построить примеры формул, не выполнимых ни на каких конечных предметных областях, но выполнимых на бесконечной предметной области. Вместе с тем справедлива теорема, принадлежащая Левенгейму.

Теорема 2. *Если формула исчисления предикатов выполнима на какой-либо бесконечной предметной области, то она выполнима также и на счетной предметной области.*

Проблема разрешимости для исчисления предикатов состоит в том, *чтобы указать единый эффективный прием (алгоритм) для определения выполнимости или невыполнимости любой заданной формулы исчисления предикатов (на какой-либо предметной области).* В отличие от исчисления высказываний, где подобный алгоритм строился без всякого труда, проблема разрешимости в общем случае для исчисления предикатов, как показали Черч и Тьюринг, вообще не имеет решения. Иными словами, не существует единого конструктивного приема для установления выполнимости или невыполнимости любой формулы исчисления предикатов.

Весьма часто проблема разрешимости для исчисления предикатов формулируется несколько в иной форме: *найти алгоритм для определения истинности (т. е. тождественной истинности) любой данной формулы в этом исчислении.*

Ввиду содержательной полноты исчисления предикатов, алгоритм, отличающий истинные формулы исчисления от ложных, одновременно решает задачу различения доказуемых и недоказуемых формул этого исчисления. Заметим также, что из истинности любой формулы вытекает невыполнимость ее отрицания. Поэтому, если бы можно было решить вопрос о выполнимости или невыполнимости всех формул исчисления предикатов, можно было бы получить возможность решить также вопрос об истинности любой формулы. К сожалению, как первый, так и второй вопросы в общем случае не имеют решения.

Таким образом, в отношении проблемы разрешимости исчисление предикатов коренным образом отличается от исчисления высказываний. Впрочем, если ограничиться теми или иными частными видами формул, то разрешающий алгоритм может быть построен и в случае исчисления предикатов.

Подобный алгоритм можно, например, построить для формул исчисления предикатов, содержащих лишь одноместные предикаты.

Это обстоятельство представляет собой простое следствие того факта, что для установления выполнимости или невыполнимости формулы, содержащей n одноместных предикатов, достаточно ограничиться рассмотрением предметных областей, состоящих не более чем из 2^n объектов. В результате проверка выполнимости (или тождественной истинности) предикатной формулы сводится (после замены кванторов дизъюнкциями и конъюнкциями) к проверке выполнимости (или, соответственно, тождественной истинности) соответствующей формулы исчисления высказываний.

В общем случае при решении вопроса о выполнимости или невыполнимости той или иной конкретной формулы может оказать существенную помощь предварительное приведение этой формулы к так называемой *нормальной форме.* Будем различать два вида нормальных форм: так называемую *предваренную форму* и *нормальную форму Сколема.* Предваренная форма отличается тем, что все кванторы (если они имеются) должны быть вынесены в самое начало формулы, а область действия каждого из них должна распространяться до конца этой формулы. В нормальной форме Сколема требуется дополнительно, чтобы все кванторы существования предшествовали всем кванторам общности.

Если формула записана в предваренном виде, то ее часть, стоящая после кванторов (бескванторная часть формулы), может рассматриваться как формула исчисления высказываний (каждый предикат рассматривается при этом просто как переменное высказывание). Но тогда в этой формуле можно исключить все знаки импликации (заменяя $A \supset B$ через $\neg A \vee B$) и привести ее после этого к дизъюнктивной нормальной форме. Подобное преобразование приводит исходную предикатную формулу к некоторой эквивалентной ей предикатной формуле. В ряде случаев в понятие нормальной формы предикатной формулы включается не только условие предварения кванторов, но и обязательное приведение ее бескванторной части к совершенной дизъюнктивной нормальной форме.

Справедлива следующая теорема.

Теорема 3. *Для всякой формулы* $\mathfrak{A}$ (*узкого*) *исчисления предикатов существует эквивалентная ей формула* $\mathfrak{B}$, *записанная в предваренной форме. Существует единый конструктивный прием* (*алгоритм*) *приведения произвольной* (*предикатной*) *формулы к предваренной форме.*

Справедливость сформулированной теоремы вытекает из легко проверяемых соотношений

$$\vdash \neg \forall x P(x) \sim \exists x \neg P(x); \tag{124}$$

$$\vdash \neg \exists x P(x) \sim \forall x \neg P(x); \tag{125}$$

$$\vdash Q \wedge \forall x P(x) \sim \forall x (Q \wedge P(x)); \tag{126}$$

$$\vdash Q \wedge \exists x P(x) \sim \exists x (Q \wedge P(x)); \tag{127}$$

$$\vdash Q \vee \forall x P(x) \sim \forall x (Q \vee P(x)); \quad (128)$$

$$\vdash Q \vee \exists x P(x) \sim \exists x (Q \vee P(x)). \quad (129)$$

Поскольку импликация может быть заменена операциями дизъюнкции и отрицания, то с помощью выписанных формул при соблюдении условий, исключающих возможность возникновения коллизии переменных, можно осуществлять последовательную перестановку кванторов со всеми составляющими формулу (отличными от кванторов) символами, пока все кванторы не окажутся в левой части формулы. Например, формула $P(x) \vee \neg \forall y Q(x,y)$ может быть сначала преобразована в эквивалентную ей формулу $P(x) \vee \exists y \neg Q(x,y)$, а затем (также в эквивалентную ей) формулу $\exists y (P(x) \vee \neg Q(x,y))$, которая и будет представлять собой требуемую предваренную форму исходной формулы.

Для нормальной формы Сколема прямого аналога теоремы 3 не существует: не для всякой формулы исчисления предикатов существует эквивалентная ей формула, имеющая нормальную форму Сколема.

Однако понятие эквивалентности может быть так обобщено, что любую формулу исчисления предикатов можно будет привести к нормальной форме Сколема. Такое обобщение дается понятием так называемой *дедуктивной эквивалентности* [61].

Формула $\mathfrak{A}$ *называется дедуктивно эквивалентной формуле* $\mathfrak{B}$, *если, присоединив формулу* $\mathfrak{A}$ *к числу аксиом исчисления, получаем возможность вывести формулу* $\mathfrak{B}$ *из расширенной таким образом системы аксиом и, наоборот, присоединив к числу аксиом формулу* $\mathfrak{B}$, *получаем возможность вывести формулу* $\mathfrak{A}$.

Это определение применимо не только к исчислению предикатов, но и к любому другому логическому исчислению, в частности к исчислению высказываний. Поскольку в исчислении высказываний присоединение любой невыводимой формулы к числу аксиом делает выводимыми все формулы, то любые две невыводимые формулы исчисления высказываний оказываются дедуктивно эквивалентными. Ясно также, что любые выводимые формулы (в любом исчислении) являются дедуктивно эквивалентными. В то же время дедуктивная эквивалентность выводимой и невыводимой формул невозможна, поскольку присоединение первой формулы к системе аксиом не сделает выводимой вторую формулу.

Таким образом, в исчислении высказываний дедуктивно эквивалентными являются как все выводимые, так и все невыводимые формулы. Легко понять также, что в исчислении предикатов (как, впрочем, и в исчислении высказываний) обычная эквивалентность формул влечет за собой их дедуктивную эквивалентность. Обратное же, вообще говоря, неверно, поскольку, например, два элементарных переменных высказывания (пропозициональные буквы) P и Q, не эквивалентные между собой, являются вместе с тем дедуктивно эквивалентными

Справедлива следующая (принадлежащая Сколему) теорема.

Теорема 4. *Для всякой формулы (узкого) исчисления предикатов существует дедуктивно эквивалентная ей формула, записанная в нормальной форме Сколема. Существует единый конструктивный прием (алгоритм), позволяющий осуществлять приведение любой предикатной формулы к дедуктивно эквивалентной ей сколемской форме.*

Можно показать, что если две формулы дедуктивно эквивалентны, то из тождественной истинности одной из них вытекает тождественная истинность другой. Поскольку существует прием приведения произвольной формулы узкого исчисления предикатов к дедуктивно эквивалентной ей формуле в нормальной форме Сколема, то при решении проблемы об установлении тождественной истинности тех или иных формул, можно заменять эти формулы соответствующими им нормальными формами Сколема. Это обстоятельство может быть использовано также при доказательстве содержательной полноты исчисления предикатов, поскольку для такого доказательства достаточно, в силу сказанного выше, установить выводимость всех тождественно истинных формул, записанных в нормальной форме Сколема. Действительно, установив выводимость всех указанных формул, мы установим тем самым и выводимость всех дедуктивно эквивалентных им формул, т. е. всех тождественно истинных формул исчисления предикатов.

Если некоторая формула исчисления предикатов содержит свободные переменные, то ее называют *открытой* формулой. Формулы, не содержащие свободных переменных, принято называть *замкнутыми*. Если $x_1, x_2, \ldots, x_n$ — все свободные переменные какой-либо открытой формулы $\mathfrak{A}$, то замкнутая формула $\forall x_1 \forall x_2 \ldots \forall x_n \mathfrak{A}$ называется *замыканием формулы* $\mathfrak{A}$. Любая формула $\mathfrak{B}$ оказывается дедуктивно эквивалентной своему замыканию $\mathfrak{B}'$, и потому эти две формулы являются либо одновременно тождественно истинными, либо одновременно не тождественно истинными.

Если проблему разрешимости понимать в смысле нахождения алгоритма, отличающего истинные формулы исчисления предикатов от неистинных, то процедура замыкания формул, как и процедуры приведения формул к нормальным формам (предваренной или сколемовской), осуществляет *редукцию* (сведение) общей проблемы разрешимости к соответствующей проблеме для формул некоторого специального вида. Разумеется, подобная редукция не помогает решить проблему разрешимости *для всех* формул узкого исчисления предикатов. Можно, однако, выделить ряд довольно широких классов формул, для которых существуют разрешающие процедуры. Один класс такого рода (формулы, содержащие только одноместные предикаты) был рассмотрен нами выше.

Проблема разрешимости имеет положительное решение для случая замкнутых формул, записанных в предваренной форме либо с одними кванторами общности (A-формулы), либо с одними кванторами существования (E-формулы). Если обозначить число этих кванторов через m, то справедливы следующие теоремы [1].

Теорема 5. *Для замкнутых A-формул с m кванторами истинность должна быть установлена лишь для предметных областей, содержащих не более чем m предметов. Если такая формула истинна в области, состоящей из m предметов, то она является тождественно истинной формулой.*

Теорема 6. *Замкнутая E-формула тождественно истинна, если она истинна в предметной области, содержащей лишь один единственный предмет. Если она истинна в некоторой области, то она истинна и в любой другой области с большим числом предметов.*

Из сформулированных теорем непосредственно вытекают разрешающие процедуры для замкнутых A-формул и E-формул: как и в случае формул с одноместными кванторами, конечность предметной области позволяет сводить вопрос об истинности предикатных формул к вопросу об истинности соответствующих формул исчисления высказываний.

Проблема разрешимости имеет положительное решение также для всех AE-формул, т. е. для таких замкнутых формул узкого исчисления предикатов, в предваренной нормальной форме которых все кванторы общности предшествуют всем кванторам существования (в сколемовской нормальной форме порядок кванторов обратный). Разрешимыми оказываются все замкнутые AEA-формулы, у которых число кванторов существования не превышает двух.

Заметим, что во всех случаях, о которых здесь шла речь, разрешимость понималась в смысле установления истинности или неистинности формул. При переходе к понятию разрешимости в смысле установления выполнимости или невыполнимости формул разрешимые классы формул получаются из перечисленных выше (разрешимых в первом смысле) классов формул заменой всех кванторов существования кванторами общности и наоборот. Таким образом, разрешимыми (в смысле установления невыполнимости или выполнимости) будут, например, класс всех EA-формул и класс всех EAE-формул, содержащих не более двух кванторов общности.

В настоящее время установлено большое число классов формул, для которых проблема разрешимости решается положительно. Ограничения, с помощью которых выделяются указанные классы, касаются не только характера, числа и порядка расположения кванторов, но и вида бескванторных частей формул (записанных в предваренной нормальной форме).

Исследовались также возможности построения разрешающих процедур за пределами узкого исчисления предикатов, в частности процедур для разрешения некоторых формул *исчисления предикатов второй ступени*. В исчислении предикатов второй ступени используются не только предметные, но и предикатные кванторы («для любого предиката P», «существует такой предикат P»), однако предикаты могут зависеть только от предметных переменных и не могут быть включены в число предметов, составляющих предметную область.

Исчисление предикатов второй ступени в общем виде не только неразрешимо, но и (как показал К. Гёдель) не может иметь никакой полной системы аксиом. Тем не менее и в этом исчислении существуют довольно обширные разрешимые части. Такой частью является, например, так называемое *исчисление* И. В этом исчислении предметной областью служит множество всех натуральных чисел, а все предикаты — одноместные. Соответствующий результат был анонсирован Бюхи, который несколько ранее построил разрешающий алгоритм для ослабленного варианта исчисления И, находящего применение в теории конечных автоматов.

§ 2. Формальная арифметика и теорема Гёделя

На базе (узкого) исчисления предикатов может быть построена формальная арифметика. Объектами, с помощью которых строится формальная арифметика, служат целые неотрицательные числа 0,1,2,3,... . На множестве всех таких чисел определяются обычные арифметические операции сложения и умножения, а также *операция непосредственного следования* $a' = a + 1$ $(a = 0,1,2,...)$. Последняя операция дает способ единообразного представления всех натуральных чисел: $1 = 0'$, $2 = 1' = 0''$, $3 = 2' = 0'''$ и т. д.

С помощью указанных операций из целых неотрицательных чисел и переменных, пробегающих целые неотрицательные значения, составляются арифметические выражения, которые принято называть *термами*. Примерами термов могут служить выражения x, $0'$, $x \cdot y' + a'' \cdot z$. Заметим, что в случае отсутствия скобок, определяющих тот или иной порядок действий, правом старшинства пользуется операция непосредственного следования. За ней идет операция умножения, затем — сложения. Таким образом, например, выражение $x \cdot y' + z'$ должно пониматься как $((x) \cdot (y')) + (z')$, а не как-нибудь иначе.

Соединив два терма знаком равенства, получим некоторое высказывание, истинное или ложное в зависимости от того, верно или неверно указанное равенство. Все подобные высказывания составляют набор так называемых элементарных формул формальной арифметики. Если в составляющих высказывание термах содержатся переменные, то оно (это высказывание) будет представлять собой некоторый предикат, который естественно назвать *элементарным арифметическим предикатом*. Из подобных элементарных предикатов с помощью операций (узкого) исчисления предикатов (включая операции связывания кванторами) строятся более сложные арифметические предикаты. Все предикаты, которые можно построить таким способом, называются *арифметическими по Гёделю*.

Формулы строящейся нами формальной арифметической системы исчерпываются формулами, которые можно построить из элементарных формул с помощью операций (узкого) исчисления предикатов. Система же аксиом формальной арифметики получается путем

дополнения системы аксиом (узкого) исчисления предикатов (аксиомы 1—15) специфически арифметическими аксиомами:

16. $P(0) \wedge \forall x((P(x) \supset P(x')) \supset P(x))$ (аксиома математической индукции).

17. $a' = b' \supset a = b$.

18. $\neg a' = 0$.

19. $a = b \supset (a = c \supset b = c)$.

20. $a = b \supset (a' = b')$.

21. $a + 0 = a$.

22. $a + b' = (a + b)'$.

23. $a \cdot 0 = 0$.

24. $a \cdot b' = a \cdot b + a$.

Для правильного понимания выписанных соотношений необходимо заметить, что в целях экономии скобок в формулах построенной нами формальной арифметической системы устанавливается определенный порядок старшинства операций: все арифметические операции (непосредственное следование, умножение и сложение) являются старшими по отношению к равенству, а последнее старше всех логических операций.

Имея систему аксиом (включающую в себя правила вывода 11, 12 и 15), можно перенести на формальную арифметику понятие (формальной) доказуемости (выводимости) и недоказуемости (невыводимости) формул, а также понятия формального вывода, тождественно истинных и тождественно ложных формул и т. д.

С помощью выписанных аксиом можно строго формальным путем установить такие законы арифметики, как коммутативный и ассоциативный законы для сложения и умножения, дистрибутивный закон для умножения по отношению к сложению и т. п. Можно доказать справедливость соотношений $\vdash a + 1 = a'$, $\vdash a \cdot 0' = a$, $\vdash a + b = 0 \supset a = 0 \wedge b = 0$, $\vdash a \cdot b = 0 \supset a = 0 \vee \vee b = 0$ и др.

С помощью аксиомы 16 можно вывести следующее общее правило доказательства методом индукции.

Пусть Γ — *некоторое множество формул* (*формальной*) *арифметики, не содержащих переменную* x *в качестве свободной переменной, а* $P(x)$ — *формула, в которую переменная* x *входит свободно. Тогда, если* $\Gamma \vdash P(0)$ *и* $\Gamma, P(x) \vdash P(x')$ (*без варьирования свободных переменных в* $P(x)$), *то* $\Gamma \vdash P(x)$.

Продолжая рассуждения подобным образом, можно строго формальным путем доказать все основные теоремы и обосновать все основные приемы доказательств, употребляющихся в строящейся содержательным путем элементарной арифметике.

Решение таких основных вопросов, как непротиворечивость и полнота системы аксиом, в случае формальной арифметики ока-

зывается гораздо более сложным, чем в узком исчислении предикатов. Так, для доказательства непротиворечивости приходится выходить за рамки строго финитных методов. Полнота же вообще не имеет места для построенной нами формальной арифметической системы. Более того, неполнота сохраняется для любого непротиворечивого *расширения* этой системы, получающегося в результате дополнения выписанной нами системы аксиом любым конечным числом совместимых с ней (т. е. не приводящих к противоречию) новых аксиом. В этом состоит смысл знаменитой теоремы Гёделя о неполноте арифметики, заставившей по-новому взглянуть на всю проблему обоснования математики и автоматизации (на базе полной формализации) процесса вывода новых теорем в дедуктивно строящихся теориях.

Для того чтобы уяснить основную идею доказательства теоремы о неполноте арифметики, необходимо предварительно познакомиться с рядом понятий и вспомогательных результатов. Прежде всего нужно определить формально само понятие полноты.

Для установления класса выводимых (доказуемых) формул арифметики достаточно ограничиться рассмотрением одних лишь замкнутых формул. Действительно, в силу легко проверяемых соотношений $P(x_1,x_2,\ldots,\ x_n)\ \vdash \forall x_1\forall x_2\ldots\forall x_n\,P(x_1,x_2,\ \ldots,\ x_n)$, $\forall x_1\ \forall x_2\ldots\ \forall x_nP(x_1,x_2,\ldots,x_n)\vdash P(x_1,x_2,\ldots,x_n)$ исчисления предикатов, из доказуемости некоторой формулы следует доказуемость ее замыкания и наоборот.

Формальная арифметическая система называется (просто) полной, если каждая ее замкнутая формула $\mathfrak{A}$ *формально разрешима, т. е. если одна из формул* $\mathfrak{A}$ *или* $\neg\,\mathfrak{A}$ *является доказуемой формулой.*

Если для некоторой формулы $\mathfrak{A}$ доказуемо ее отрицание $\neg\,\mathfrak{A}$, то сама формула $\mathfrak{A}$ называется (формально) *опровержимой. Формальная разрешимость* замкнутой формулы означает, таким образом, что эта формула либо доказуема, либо опровержима. Поскольку с наивной содержательной точки зрения замкнутая формула должна быть либо истинной, либо ложной, то условие ее формальной разрешимости является весьма естественным критерием для решения вопроса о полноте соответствующей формальной системы. Основная идея доказательства теоремы о неполноте арифметики как раз и состоит в фактическом построении формально неразрешимой замкнутой формулы. Для такого построения нам потребуется ряд вспомогательных результатов, которые мы и рассмотрим.

В содержательном понимании *арифметический предикат* представляет собой произвольный (не обязательно конструктивно определенный) предикат на множестве всех целых неотрицательных чисел. Построенная нами формальная арифметическая система дает способ конструктивного задания некоторых арифметических предикатов. Для установления этого способа введем следующее определение.

Арифметический предикат $P(x_1,x_2,\ldots,x_n)$ называется нумерически выразимым, если имеется формула $P_1(x_1,x_2,\ldots,x_n)$ рассматри-

ваемой нами формальной арифметической системы, не содержащая никаких других свободных переменных, кроме $x_1, x_2, \ldots, x_n$, и такая, что для любого конкретного набора из n целых неотрицательных чисел $a_1, a_2, \ldots, a_n$ выполняются следующие условия:

1) если высказывание $P(a_1, a_2, \ldots, a_n)$ истинно, то $\vdash P_1(a_1, a_2, \ldots, a_n)$;

2) если высказывание $P(a_1, a_2, \ldots, a_n)$ ложно, то $\vdash \neg P_1(a_1, a_2, \ldots, a_n)$.

При этом говорят, что формула $P_1(x_1, x_2, \ldots, x_n)$ нумерически выражает предикат $P(x_1, x_2, \ldots, x_n)$.

Легко показать, что арифметические предикаты $x = y$ и $x < y$ нумерически выражаются формулами $x = y$ и $\exists z(z' + x = y)$ соответственно.

Формула $P_1(x_1, x_2, \ldots, x_n)$, о которой идет речь в только что приведенном определении, разрешима при любом конкретном наборе значений ее свободных переменных $x_1, x_2, \ldots, x_n$. Формулы, обладающие подобным свойством, принято называть *нумерически разрешимыми* формулами. Проверка истинности предиката, нумерически выражаемого такой формулой, при любом наборе значений предметных переменных может быть проведена, в силу сказанного, конструктивным путем. В построенной нами формальной арифметической системе каждая формула без переменных является разрешимой, а каждая формула без кванторов — нумерически разрешимой формулой.

Понятие нумерической выразимости может быть установлено не только для арифметических предикатов, но и для (содержательно определяемых) *арифметических функций* (значениями которых служат целые неотрицательные числа). *Говорят, что формула* $P(x_1, x_2, \ldots, x_n, y)$ *формальной арифметической системы нумерически представляет арифметическую функцию* $f(x_1, x_2, \ldots, x_n)$, *если для любого набора из n целых неотрицательных чисел выполняются следующие условия*:

1) *если* $f(a_1 a_2, \ldots, a_n) = b$, *то* $\vdash P(a_1, a_2, \ldots, a_n, b)$;

2) $\vdash \exists y (P(a_1, a_2, \ldots, a_n, y) \wedge \forall z (P(a_1, a_2, \ldots, a_n, z) \supset z = y))$.

Второе из приведенных условий представляет собой выраженное на языке исчисления предикатов условие однозначности задания функции f с помощью предиката P.

Заметим, что возможность эффективного задания предикатов не связана непременно с использованием аппарата формальной арифметики. Произвольный n-местный арифметический предикат $P(x_1, x_2, \ldots, x_n)$ может быть задан с помощью n-местной арифметической функции $\varphi(x_1, x_2, \ldots, x_n)$, принимающей значение 0 на всех наборах значений переменных $x_1, x_2, \ldots, x_n$, на которых предикат 0 истинен, и значение 1 — на всех тех наборах, на которых предикат P ложен. Эта функция называется *представляющей функцией* рассматриваемого предиката P. *Предикат, представляющая функция*

которого примитивно рекурсивна или общерекурсивна, называется соответственно примитивно рекурсивным или общерекурсивным предикатом.

К. Гёделем установлен следующий результат.

Теорема 1. *Если арифметическая функция* $\varphi(x_1,x_2,\ldots,x_n)$ *примитивно рекурсивна, то* $(n+1)$*-местный предикат* $\varphi(x_1,x_2,\ldots,x_n)=y$ *является арифметическим по Гёделю, т. е. выразимым средствами формальной арифметики.*

Из этой теоремы вытекает, в частности, что все примитивно рекурсивные предикаты относятся к числу арифметических по Гёделю предикатов.

С помощью теоремы 1 можно легко установить справедливость следующего предложения.

Теорема 2. *Каждый примитивно рекурсивный предикат нумерически выразим в формальной арифметике.*

В отношении общерекурсивных предикатов Клини [42] и Постом был установлен следующий результат.

Теорема 3. *При любом* $n\geqslant 0$ *всякий общерекурсивный предикат* $P(x_1,x_2,\ldots,x_n)$ *может быть представлен как в виде* $\exists y R(x_1,x_2,\ldots,x_n,y)$, *так и в виде* $\forall y S(x_1,x_2,\ldots,x_n,y)$, *где предикаты* R *и* S *примитивно рекурсивны. И наоборот, всякий предикат, представимый в каждой из этих двух форм, является общерекурсивным, причем он остается общерекурсивным и в том случае, когда предикаты* R *и* S *не примитивно рекурсивны, а лишь общерекурсивны.*

Справедлива также следующая теорема, принадлежащая Клини [42].

Теорема 4. *Все общерекурсивные предикаты являются арифметическими по Гёделю.*

Для различного рода сложных построений и доказательств в формальной арифметике целесообразно ввести специальную нумерацию для всех ее формул и доказательств этих формул (средствами построенной нами формальной арифметической системы). Такая нумерация была предложена К. Гёделем и поэтому называется *гёделевской нумерацией*. Для ее осуществления существует много различных способов. Рассмотрим один из них.

Прежде чем определить нумерацию формул, целесообразно несколько видоизменить способ их записи, рассматривая не только сами формулы, но и отдельные их части как некоторые формальные объекты, называемые *вещами*. К элементарным вещам относятся, во-первых, все логические символы ($\supset$, $\wedge$, $\vee$, $\neg$, $\forall$, $\exists$), символ равенства ($=$), символы арифметических операций ($+,\cdot,'$), символ нуля (0) и два символа для обозначения различных предметных переменных (x, $|$). Различным предметным переменным $x,y,z,\ldots$ сопоставляются вещи x, $(|,x)$, $(|,(|,x))$ и т. д. Термам и формулам вида $r+\mathrm{s}$, r', $r=\mathrm{s}$, $A\vee B$, $A\wedge B$, $\neg A$, $\forall u A(u)$, $\exists u A(u)$ сопоставляются соответственно вещи $(+,r,\mathrm{s})$, $(',r)$, $(=,r,\mathrm{s})$, $(\vee,A,B)$, $(\wedge,A,B)$, $(\neg A)$, $(\forall,u,A(u))$, $(\exists,u,A(u))$. Для упрощения обозначений символы $r,\mathrm{s},A,B,u,A(u)$ и соответствующие им вещи здесь

не различаются. Используя приведенные определения, можно последовательно, шаг за шагом строить вещи, соответствующие различным формулам и их составным частям. Например, формуле $\forall x(0' + x = y)$ соответствует вещь $(\forall,x(=,(+('),0),x),(|,x)))$, терму $0' + x$ — вещь $(+,('),0),x)$, формуле $x = y$ — вещь $(=,x,(|,x))$ и т. д.

Сопоставим теперь элементарным вещам различные нечетные числа (гёделевские номера этих вещей)

$$\begin{array}{cccccccccccccc} \supset & \wedge & \vee & \neg & \forall & \exists & = & + & \cdot & ' & 0 & x & | \\ 3 & 5 & 7 & 9 & 11 & 13 & 15 & 17 & 19 & 21 & 23 & 25 & 27 \end{array}.$$

Обозначая через $p_0,p_1,p_2,\ldots$ последовательность всех простых чисел ($p_0 = 2$, $p_1 = 3$, $p_2 = 5,\ldots$ и т. д.) и предполагая, что вещам $a_0,a_1,\ldots,a_m$ уже сопоставлены гёделевские номера $n_0,n_1,\ldots,n_m$, сопоставляем вещи $(a_0,a_1,\ldots,a_m)$ гёделевский номер $p_0^{n_0}p_1^{n_1}\ldots p_m^{n_m}$. Сопоставляя каждой формуле рассматриваемой нами формальной арифметической системы гёделевский номер соответствующей ей вещи, получим искомую гёделевскую нумерацию для всех этих формул. Например, формула $x = y$, которой соответствует вещь $(=, x, (|,x))$, имеет гёделевский номер $2^{15}\cdot 3^{25}\cdot 5^{2^{27}\cdot 3^{25}}$, терму $0' + x$ с соответствующей ему вещью $(+,('),0),x)$ должен быть сопоставлен гёделевский номер $2^{17}\cdot 3\cdot 2^{21\cdot 3^{23}}\cdot 5^{25}$ и т. д.

Легко видеть, что гёделевский номер всякой неэлементарной вещи непременно чётный. Имея в виду этот факт, а также однозначность разложения всякого натурального числа на простые множители, нетрудно понять, что по гёделевскому номеру можно однозначно восстановить соответствующую ему формулу. Это означает, что соответствие между формулами рассматриваемой нами формальной арифметической системы и их гёделевскими номерами является взаимно однозначным. Таким образом, в случае необходимости можно вместо формул этой системы пользоваться лишь их гёделевскими номерами.

По аналогии с гёделевской нумерацией формул арифметики можно ввести гёделевскую нумерацию для всевозможных конечных последовательностей таких формул, среди которых будут находиться, в частности, доказательства всех доказуемых арифметических формул.

Условимся для любого целого неотрицательного числа a, понимаемого содержательно, через $\hat{a}$ обозначать представление этого числа в формальной арифметике ($\hat{a}$ представляет собой символ 0 с a штрихами). Фиксируя какую-нибудь гёделевскую нумерацию формул арифметики, условимся для любого гёделевского номера n через P_n обозначать ту формулу, которая имеет в нашей нумерации номер n. Выделим в формуле P_n переменную x (от которой формула может в действительности и не зависеть), записывая формулу $P_n(x)$.

Определим теперь два арифметических предиката $A(a,b)$ и $B(a,b)$, считая первый предикат истинным тогда и только тогда, когда число a является гёделевским номером формулы $P_a(x)$, такой,

что формула $P_a(\hat{a})$ доказуема, а число b есть гёделевский номер некоторого ее доказательства.

Аналогично предикат $B(a, b)$ считается истинным тогда и только тогда, когда число a является гёделевским номером формулы $P_a(x)$, для которой формула $P_a(\hat{a})$ опровержима, а число b есть гёделевский номер доказательства формулы $\neg P_a(\hat{a})$.

Используя сформулированные выше теоремы, можно доказать справедливость следующей важной леммы.

Лемма. *Определенные нами арифметические предикаты* $A(a, b)$ *и* $B(a, b)$ *в случае фиксированной выше гёделевской нумерации нумерически выразимы в формальной арифметической системе с аксиомами 1—24.*

Построим формулы $A_1(a, b)$ и $B_1(a, b)$, которые нумерически выражают предикаты $A(a, b)$ и $B(a, b)$ соответственно, и рассмотрим формулу $\forall y \neg A_1(x, y)$. Эта формула имеет некоторый гёделевский номер p и совпадает поэтому с формулой, которую мы условились обозначать через $P_p(x)$. Рассмотрим теперь формулу $P_p(\hat{p})$, которая не имеет свободных переменных. Эту формулу, в явном виде представляющуюся как $\forall y \neg A_1(\hat{p}, y)$, можно рассматривать как высказывание, выражающее свою собственную недоказуемость. Действительно, это высказывание представляет собой утверждение о том, что никакое число не может являться гёделевским номером доказательства формулы, получающейся из формулы $P_p(x)$ в результате замены x на p. Но указанная замена как раз и превращает формулу $P_p(x)$ в построенную нами формулу $\forall y \neg A_1(\hat{p}, y)$.

Оказывается, что при некоторых дополнительных предположениях из указанного свойства формулы $\forall y \neg A_1(\hat{p}, y)$ вытекает ее неразрешимость, что и доказывает неполноту построенной нами формальной арифметической системы. Дополнительное предположение, о котором здесь идет речь, сводится к тому, что формальная арифметика предполагается *ω-непротиворечивой.*

Под ω-непротиворечивостью формальной арифметической системы подразумевается следующее ее свойство: *ни для какой формулы* $P(x)$, *для которой доказуема формула* $\neg \forall x P(x)$, *не могут быть доказаны все формулы вида* $P(0)$, $P(1)$, $P(2)$,... .

Из ω-непротиворечивости формальной арифметики вытекает ее простая непротиворечивость. В самом деле, пусть P — любая доказуемая формула, не содержащая свободных переменных (например, формула $0 = 0$). Вводя в эту формулу фиктивную переменную x, от которой P фактически не зависит, запишем ее в виде $P(x)$. Тогда все формулы $P(0)$, $P(1)$,... совпадают с P и, следовательно, доказуемы. В силу ω-непротиворечивости системы это означает, что формула $\neg \forall x P(x)$, совпадающая фактически с формулой $\neg P$, недоказуема. При наличии же противоречия в системе, благодаря свойству слабого $\neg$-удаления (см. формулу (7) § 5 гл. II), все формулы этой системы были бы доказуемы. Поскольку, однако, формула $\neg P$ недоказуема, то наша система (просто) непротиворечива.

Теперь можно доказать теорему Гёделя о неполноте арифметики в первоначальной (слабой) форме.

Теорема 5. *Если формальная арифметика ω-непротиворечива, то формула* $\forall y \neg A_1(\hat{p}, y)$, *построенная выше, представляет собой пример неразрешимой формулы.*

Доказательство. Предположим сначала, что формула $\forall y \neg A_1(\hat{p}, y)$ доказуема. Обозначим через k гёделевский номер доказательства этой формулы. Тогда, предложение $A(p,k)$ истинно и, следовательно, выводима формула $A_1(\hat{p}, \hat{k})$. Используя операцию $\exists$-введения (см. § 1 настоящей главы), получим $\vdash \exists y A_1(\hat{p}, y)$, или, используя формулу (125), получим $\vdash \neg \forall y \neg A_1(\hat{p}, y)$. Но тогда, в силу сделанного вначале предположения, рассматриваемая нами формальная арифметическая система будет (просто) противоречивой, что исключается в силу условия ее ω-непротиворечивости.

Предположим теперь, что доказуема формула $\neg \forall y \neg A_1(\hat{p}, y)$. Как было доказано, формула $\forall y \neg A_1(\hat{p}, y)$ недоказуема. Поэтому никакое из чисел 0,1,2,... не является гёделевским номером доказательства последней формулы. Это означает, что все высказывания $A(p,0)$, $A(p,1)$, $A(p,2)$,... ложны и, следовательно, ввиду нумерической выразимости предиката A, выводимы все формулы вида $\neg A_1(p,i)$ для $i=0,1,2,\ldots$. Но тогда, в силу предположения об ω-непротиворечивости системы, формула $\neg \forall y \neg A_1(\hat{p}, y)$ недоказуема, что противоречит принятому нами допущению о ее доказуемости.

Таким образом, формула $\forall y \neg A_1(\hat{p}, y)$ не может быть ни доказуемой, ни опровержимой формулой и, следовательно, является примером неразрешимой формулы. Тем самым теорема доказана.

Как видно из приведенного доказательства, для установления недоказуемости построенной нами формулы достаточно предположения о простой непротиворечивости формальной арифметической системы, а предположение об ω-непротиворечивости системы используется лишь при доказательстве неопровержимости этой формулы.

Россер показал [71], что можно построить пример формулы, неразрешимость которой устанавливается без предположения об ω-непротиворечивости формальной арифметики. Для этого достаточно ее простой непротиворечивости. Формула, о которой здесь идет речь, строится следующим способом. Сначала из определенных выше предикатов $A(a, b)$ и $B(a, b)$ строится формула $\forall y(\neg A_1(x,y) \vee \vee \exists z(z \leqslant y \wedge B_1(x, z)))$ (где формулы A_1 и B_1 нумерически выражают предикаты A и B соответственно). Если обозначить эту формулу через $P_q(x)$ (q — ее гёделевский номер), то формула $P_q(\hat{q})$ представляет собой искомый пример формулы, неразрешимость которой устанавливается с помощью предположения о простой непротиворечивости формальной арифметики. Справедливость же последнего предположения была установлена Аккерманом, Нейманом при некотором ограничении, а Г. Генценом — в общем случае.

П. С. Новиковым [59] была доказана не только простая непротиворечивость, но даже ω-непротиворечивость арифметики, хотя

для этого потребовалось привлечение средств, выходящих за рамки самой формальной арифметики. Из этого результата и теоремы 5 вытекает теорема Гёделя о неполноте арифметики:

Теорема 6. *Формальная арифметическая система с аксиомами 1—24 неполна в том смысле, что в ней существуют постоянные высказывания (формулы, не содержащие свободных переменных), которые нельзя ни доказать, ни опровергнуть средствами этой системы.*

Можно было бы подумать, что результат Гёделя вскрывает лишь недостаточную полноту выбранной нами системы аксиом формальной арифметики и что при разумном пополнении этой системы аксиом новыми аксиомами неполнота арифметики (при сохранении ее непротиворечивости) уже не будет иметь места. В действительности дело обстоит далеко не так просто. Как показывает подробный анализ, проведенный Гёделем, при любом непротиворечивом расширении системы аксиом формальная арифметика продолжает оставаться неполной и в ней по-прежнему будут существовать неразрешимые замкнутые формулы. Более того, всякая формальная система, удовлетворяющая некоторым довольно общим условиям (наличие достаточно богатого набора формул и объектов), в случае ее непротиворечивости будет обязательно неполной.

Как отмечалось выше, доказательство непротиворечивости построенной нами формальной арифметической системы S требовало привлечения средств, выходящих за рамки этой системы. Оказывается, что этот факт не случаен: можно показать, что доказательство непротиворечивости системы S средствами, формализуемыми в самой этой системе, невозможно.

В самом деле, в системе S оказывается невозможным доказать формулу $1=0$. В случае противоречивости этой системы все ее формулы и, в частности, формула $1=0$, становятся доказуемыми. Верно и обратное: из доказуемости формулы $1=0$ вытекает как следствие противоречивость системы S. Пусть r — гёделевский номер формулы $1=0$. Тогда формула $\neg\exists y A_1(\hat{r}, y)$, которую для краткости обозначим через $\mathfrak{A}$, в силу приведенного выше определения предиката $A(x, y)$ с нумерическим выражением $A_1(x, y)$, представляет собой формальное выражение непротиворечивости системы S.

Можно показать, что формализация (в системе S) доказательства теоремы 5 может быть сведена к выводу (в S) формулы $\mathfrak{A} \supset \forall y \neg A_1(\hat{p}, y)$, а формализация (в S) доказательства непротиворечивости системы S—к выводу (в S) формулы $\mathfrak{A}$. Но в случае существования обоих указанных выводов, по правилу вывода, выражаемому аксиомой 11 исчисления высказываний (см. гл. II, § 5), должна быть выводимой (доказуемой) также и формула $\forall y \neg A_1(\hat{p}, y)$. Поскольку это противоречит теореме 5, то формула $\mathfrak{A}$ не может быть доказуемой в системе S, что и показывает невозможность доказательства непротиворечивости формальной арифметической системы средствами самой этой системы.

§ 3. Понятие об автоматизации доказательств и построения дедуктивных теорий

Построенная в предыдущем параграфе формальная арифметическая система представляет собой пример формализации математической теории на базе исчисления предикатов. Подобная формализация позволяет разложить на точно определенные элементарные составные части процесс доказательства всех доказуемых в рамках данной теории предложений. Закладывая в программу универсальной электронной цифровой машины все аксиомы и правила вывода рассматриваемой теории, а также формулу, выражающую подлежащее доказательству предложение, можно организовать систему случайного поиска доказательства этой формулы.

Если количество элементарных шагов, позволяющих осуществить доказательство требуемой формулы, относительно невелико, то большая скорость работы электронной цифровой машины позволяет находить доказательство методом простого перебора всех вариантов. Однако для сколько-нибудь сложных предложений такой метод поиска доказательства становится практически непригодным ввиду того, что количество перебираемых вариантов оказывается чудовищно большим, так что их полный перебор за разумное время невозможен даже на современных быстродействующих электронных цифровых машинах. В этих случаях приходится пользоваться различного рода приемами, позволяющими резко сократить число перебираемых вариантов. К подобным приемам относится укрупнение правил вывода, благодаря чему доказательство строится из более крупных блоков и становится в результате значительно короче. Другой прием сокращения перебора состоит в разработке различных эвристических методов, позволяющих ставить промежуточные цели и тем самым разбивать процесс поиска доказательства на отдельные этапы. Такие этапы должны быть настолько мелкими, чтобы внутри них оказывалось возможным осуществление полного перебора.

Обычно при автоматизации доказательств предпочитают пользоваться системой формализации исчисления предикатов, несколько отличной от той, которая была развита в первом параграфе настоящей главы. Такую систему формализации предложил в своей работе Г. Генцен [15]. Она позволяет в некотором смысле нормализовать процесс доказательства, использующий формальный аппарат исчисления предикатов. Изложим некоторые основные положения *генценовской системы* формализации исчисления предикатов, которую, в отличие от ранее рассмотренной так называемой гильбертовской системы H, будем обозначать через G или, точнее, через $G1$.

Одним из существенных понятий в генценовской системе является понятие так называемой *секвенции*. Секвенцией называется формальное выражение вида $\mathfrak{A}_1, \mathfrak{A}_2, \ldots, \mathfrak{A}_m \rightarrow \mathfrak{B}_1, \mathfrak{B}_2, \ldots, \mathfrak{B}_n$, где

$\mathfrak{A}_i$ и $\mathfrak{B}_i$ — формулы, а стрелкой обозначен некоторый новый формальный символ. Секвенция $\mathfrak{A}_1, \mathfrak{A}_2, \ldots, \mathfrak{A}_m \to \mathfrak{B}_1, \mathfrak{B}_2, \ldots, \mathfrak{B}_n$ имеет ту же самую интерпретацию, что и формула $\mathfrak{A}_1 \wedge \mathfrak{A}_2 \wedge \ldots \wedge \mathfrak{A}_m \supset \supset \mathfrak{B}_1 \vee \mathfrak{B}_2 \vee \ldots \vee \mathfrak{B}_n$ в гильбертовской системе, причем конъюнкция пустого множества формул считается истинной, а дизъюнкция пустого множества формул — ложной.

Часть секвенции, стоящая слева от символа $\to$, называется *антецедентом*, а часть, стоящая справа от этого символа, *сукцедентом* рассматриваемой секвенции. Для сокращения записи конечные последовательности формул обозначают прописными греческими буквами (Γ, Θ, Δ, Λ и т. п.), а отдельные формулы — прописными латинскими буквами.

В генценовской системе $G1$ имеется единственная аксиома (схема аксиом)

$$C \to C, \tag{130}$$

а также целый ряд *правил вывода*, подразделяющихся на правила вывода для исчисления высказываний и дополнительные правила вывода для исчисления предикатов.

Правила вывода для исчисления высказываний:

$$\frac{A, \Gamma \to \Theta, B}{\Gamma \to \Theta, A \supset B} \quad (\supset\text{-введение в сукцедент}); \tag{131}$$

$$\frac{\Delta \to \Lambda, A \quad B, \Gamma \to \Theta}{A \supset B, \Delta, \Gamma \to \Lambda, \Theta} \quad (\supset\text{-введение в антецедент}); \tag{132}$$

$$\frac{\Gamma \to \Theta, A \quad \Gamma \to \Theta, B}{\Gamma \to \Theta, A \wedge B} \quad (\wedge\text{-введение в сукцедент}); \tag{133}$$

$$\frac{A, \Gamma \to \Theta}{A \wedge B, \Gamma \to \Theta} \; A \; \frac{B, \Gamma \to \Theta}{A \wedge B, \Gamma \to \Theta} \quad (\wedge\text{-введение в антецедент}); \tag{134}$$

$$\frac{\Gamma \to \Theta, A}{\Gamma \to \Theta, A \vee B} \; \frac{\Gamma \to \Theta, B}{\Gamma \to \Theta, A \vee B} \quad (\vee\text{-введение в сукцедеит}); \tag{135}$$

$$\frac{A, \Gamma \to \Theta \quad B, \Gamma \to \Theta}{A \vee B, \Gamma \to \Theta} \quad (\vee\text{-введение в антецедент}); \tag{136}$$

$$\frac{A, \Gamma \to \Theta}{\Gamma \to \Theta, \neg A} \quad (\neg\text{-введение в сукцедент}); \tag{137}$$

$$\frac{\Gamma \to \Theta, A}{\neg A, \Gamma \to \Theta} \quad (\neg\text{-введение в антецедент}). \tag{138}$$

Дополнительные правила вывода для исчисления предикатов:

$$\frac{\Gamma \to \Theta, A(b)}{\Gamma \to \Theta, \forall x A(x)} \quad (\forall\text{-введение в сукцедент}); \tag{139}$$

$$\frac{A(t), \Gamma \to \Theta}{\forall x A(x), \Gamma \to \Theta} \quad (\forall\text{-введение в антецедент}); \tag{140}$$

$$\frac{\Gamma \to \Theta, A(t)}{\Gamma \to \Theta, \exists x A(x)} \quad (\exists\text{-введение в сукцедент}); \tag{141}$$

$$\frac{A(b), \Gamma \to \Theta}{\exists x A(x), \Gamma \to \Theta} \quad (\exists\text{-введение в антецедент}). \tag{142}$$

В правилах (139) и (142) должно соблюдаться определенное ограничение, заключающееся в следующем: переменная b не должна входить свободно в заключения (т. е. в выражения, стоящие под чертой) (139) и (142).

Заметим, что когда формула $A(x)$ не содержит в действительности свободной переменной x, то $A(b)$ совпадает с $A(x)$. В этом случае переменная b может быть любой, так что в качестве b всегда можно выбрать переменную, не входящую в заключение, и тем самым соблюсти требуемое ограничение.

Кроме приведенных правил, в генценовской системе имеется еще семь так называемых *структурных* правил вывода:

$$\frac{\Gamma \to \Theta}{\Gamma \to \Theta, C} \quad \text{(утончение в сукцеденте)}; \tag{143}$$

$$\frac{\Gamma \to \Theta}{\Gamma, C \to \Theta} \quad \text{(утончение в антецеденте)}; \tag{144}$$

$$\frac{\Gamma \to \Theta, C, C}{\Gamma \to \Theta, C} \quad \text{(сокращение в сукцеденте)}; \tag{145}$$

$$\frac{C, C, \Gamma \to \Theta}{C, \Gamma \to \Theta} \quad \text{(сокращение в антецеденте)}; \tag{146}$$

$$\frac{\Gamma \to \Lambda, C, D, \Theta}{\Gamma \to \Lambda, D, C, \Theta} \quad \text{(перестановка в сукцеденте)}; \tag{147}$$

$$\frac{\Delta, C, D, \Gamma \to \Theta}{\Delta, D, C, \Gamma \to \Theta} \quad \text{(перестановка в антецеденте)}; \tag{148}$$

$$\frac{\Delta \to \Lambda, C \quad C, \Gamma \to \Theta}{\Delta, \Gamma \to \Lambda, \Theta} \quad \text{(сечение)}. \tag{149}$$

Для обозначения доказуемости той или иной секвенции S в системе $G1$ употребляется сокращенное обозначение $\vdash S$, аналогичное соответствующему обозначению в гильбертовской системе H.

Генценовская система $G1$ в известном смысле слова эквивалентна гильбертовской системе H, поскольку, как показал Генцен, справедлива следующая теорема:

Теорема 1. *Если некоторая формула A выводима из конечного множества формул Γ в гильбертовской системе H и все переменные остаются фиксированными, то в генценовской системе G1 выводима секвенция* $\Gamma \to A$. *И, наоборот, если в системе G1 выводима секвенция* $\Gamma \to A$, *то формула A оказывается выводимой из множества формул Γ и при этом все переменные остаются фиксированными.*

Сходство между гильбертовской и генценовской системами оказывается столь большим, что если в выводе, проведенном в одной системе, использовались правила (аксиомы) лишь для части логических операций $\subset, \neg, \vee, \wedge, \forall, \exists$, то в соответствующем ему выводе в другой системе также ограничивались лишь правилами с теми же самыми символами, за исключением, быть может, символа импликации $\supset$.

Генцен установил результат, позволяющий устранять из доказательств в системе *G1* использование сечений (правила вывода (149)). Это так называемая теорема Генцена *о нормальной форме*, или *элиминационная теорема*.

Теорема 2. *Пусть в системе G1 дано доказательство какой-либо секвенции, в которое никакая переменная не входит одновременно свободно и связано. Тогда в G1 найдется такое доказательство той же секвенции, которое не использует сечений (правило* (149)) *и использует лишь логические правила, применяющиеся в исходном доказательстве.*

Наряду с системой *G*1, Генценом были построены и другие формальные системы (системы *G*2, *G*3).

Хао-Ванг [77] использовал генценовскую систему *G1* для автоматизации (с помощью универсальной электронной цифровой машины) процесса доказательства большого числа теорем не только из исчисления высказываний, но и из (узкого) исчисления предикатов. Эксперименты, проведенные Хао-Вангом, показали, что, несмотря на отсутствие универсальной разрешающей процедуры для исчисления предикатов, можно построить частную разрешающую процедуру, которая позволяет доказать все теоремы, обычно включаемые в руководство по математической логике.

В случае исчисления высказываний для доказательства той или иной секвенции правила вывода (131) — (138) генценовской системы (дополненные правилами введения символа эквивалентности в сукцедент и антецедент) применяются Хао-Вангом в обратном направлении (заключение заменяется посылкой). При этом производится последовательное (начиная с левого конца секвенции) исключение логических связок. В результате применения указанной процедуры через конечное число шагов получим секвенции вида $A_1, A_2, \ldots, A_m \to B_1, B_2, \ldots, B_n$, где A_i и B_j — так называемые *атомарные формулы*, т. е., попросту говоря, пропозициональные буквы. Подобные секвенции, которые естественно называть *элементарными*, доказуемы в том и только в том случае, когда в их левой и правой частях встречается одна и та же атомарная формула.

Если все элементарные секвенции, полученные в результате

описанной процедуры, доказуемы, то доказуема, очевидно, и исходная секвенция. Для ее доказательства достаточно повторить все шаги, которые привели к появлению указанных элементарных секвенций, в обратном порядке.

Если подлежащая доказательству теорема записана в виде формулы в гильбертовской системе H, то для превращения ее в секвенцию достаточно поставить перед ней стрелку. Если последняя операция, выполняемая в исходной формуле, является импликацией, то формулу можно превратить в секвенцию, заменив соответствующий символ $\supset$ стрелкой. Подобный способ превращения формулы в секвенцию обычно приводит к более коротким доказательствам, чем при записи стрелки впереди формулы. В силу определения смысла символа $\rightarrow$ в секвенции и теоремы 1 настоящего параграфа, доказательство полученной любым из двух указанных способов секвенции представляет собой вместе с тем и доказательство исходной формулы.

Рассмотрим в качестве примера формулу $\neg(A \vee B) \supset \neg A$ исчисления высказываний. Заменяя символ импликации стрелкой, превратим ее в секвенцию $\neg (A \vee B) \rightarrow \neg A$. Крайней слева логической связкой является символ отрицания $\neg$. Обращение правила $\neg$-введения в антецедент (правило (138)) приводит нашу секвенцию к виду $\rightarrow \neg A, A \vee B$. Элиминация следующей логической связки (которой является снова отрицание) приводит (с помощью обращения правил (147) и (137)) к секвенции $A \rightarrow A \vee B$. Наконец, обращение правила (135) приводит нашу секвенцию к виду $A \rightarrow A, B$, представляющему собой элементарную секвенцию. Поскольку буква A входит как в левую, так и в правую часть последней секвенции, эта секвенция доказуема. Выписывая в обратном порядке все шаги, которые привели нас к секвенции $A \rightarrow A, B$, придем к доказательству исходной секвенции $\neg(A \vee B) \rightarrow \neg A$.

Для правильного понимания последнего шага в описанном примере последовательной элиминации логических связок заметим, что правило вывода (135) можно (что и делает Хао-Ванг) записать

$$\frac{\Gamma \rightarrow \Theta, A, B}{\Gamma \rightarrow \Theta, A \vee B}. \qquad (150)$$

Аналогично в правиле (134) $\wedge$-введения в антецедент две посылки могут быть заменены одной посылкой вида $\Gamma, A, B, \rightarrow \Theta$. Правомочность указанных изменений правил (134) и (135) легко обосновать с помощью правил утончения в сукцеденте и антецеденте (правила (143) и (144)).

Разумеется, для исчисления высказываний можно построить более эффективные процедуры доказательств, однако описанная процедура хороша тем, что допускает обобщение на случай исчисления предикатов. При подобном обобщении используется прием элиминации кванторов с помощью обращения правил вывода (139) — (142), совершенно аналогичный описанному выше приему

элиминации логических связок с помощью обращения правил вывода (131) — (138).

Процедура разрешения, построенная Хао-Вангом, охватывает, естественно, лишь часть формул исчисления предикатов (поскольку для исчисления предикатов в целом разрешающей процедуры не существует). Однако она оказывается достаточной для того, чтобы охватить почти все теоремы исчисления предикатов, включенные в такую капитальную монографию, как «Principia mathematica» Уайтхеда и Рассела.

Увеличение эффективности процедуры разрешения достигается за счет использования ряда дополнительных приемов. Среди этих приемов важное место занимает приведение формул к так называемому *минисферному* виду. В противоположность предваренной форме, у которой область действия кванторов является максимально возможной, минисферный вид формул предусматривает по возможности большее сужение областей действия кванторов. В случае приведения формул к минисферному виду обычно предварительно операции импликации и эквивалентности выражаются через операции дизъюнкции, конъюнкции и отрицания. В этом случае понятие минисферной формы может быть уточнено посредством следующих определений.

Во-первых, отдельные пропозициональные буквы и элементарные предикаты представляют собой минисферные формулы. Во-вторых, если формулы A и B имеют минисферную форму, то формулы $A \vee B$, $A \wedge B$ и $\neg A$ также являются минисферными. В-третьих, если $P(x)$ есть дизъюнкция (или, соответственно, конъюнкция) минисферных формул, то минисферный вид будет иметь также формула $\forall x P(x)$ (или, соответственно, формула $\exists x P(x)$). В-четвертых, если формула $P(x)$ в $\forall x P(x)$ (либо $Q(x)$ в $\exists x Q(x)$) начинается с квантора существования (или, соответственно, с квантора общности) и формула $P(x)$ (и $Q(x)$) минисферна, то минисферной будет также формула $\forall x P(x)$ (и $\exists x Q(x)$). Наконец, в-пятых, формула, которая начинается с цепочки однородных кванторов, имеет минисферную форму, если каждая формула, получающаяся из нее при перестановках этих кванторов и опускании первого из них, имеет минисферную форму.

Процедура приведения формул исчисления предикатов к минисферному виду часто оказывается достаточно простой. В этом случае разрешающую процедуру целесообразно начинать с приведения обеих частей заданной секвенции к минисферной форме и одновременной элиминации (всюду, где это только возможно) всех логических связок с помощью обращения правил вывода (131) — (138).

Для элиминации кванторов вместо требующего известных оговорок применения (обращенных) правил вывода (139) — (142) часто оказывается целесообразным применять более простой метод, основанный на понятии *о знаках кванторов*, входящих в ту или иную секвенцию. При определении этого понятия предварительно рассматривается вопрос о приписывании знаков различным частям формул исчисления предикатов. Прежде всего каждая формула,

рассматриваемая как вхождение в самое себя, считается положительной. Если P — положительная (отрицательная) часть формулы Q или формулы R, то P будет положительной (или, соответственно, отрицательной) частью в формулах $Q \wedge R$, $Q \vee R$, $\forall x Q$, $\exists x Q$. Если D — положительная (отрицательная) часть в формуле S, то D будет отрицательной (соответственно положительной) частью формул $\neg S$ и $S \supset Q$, вместе с тем D будет положительной (соответственно отрицательной) частью формулы $Q \supset S$.

Если частью секвенции считать часть какой-нибудь формулы из числа составляющих секвенцию формул, то любая часть в секвенции будет иметь тот же самый знак, что и в соответствующей формуле, если эта формула входит в сукцедент, и противоположный знак, если эта формула входит в антецедент рассматриваемой секвенции. Всякому квантору общности в секвенции приписывается тот знак, который в этой секвенции имеет область его действия (рассматриваемая как часть секвенции). Знаки кванторов существования считаются противоположными знакам их областей действия. Например, в секвенции $\forall x \exists y P(x,y), \neg \forall v Q(v) \rightarrow \forall z R(z) \wedge \wedge \exists u S(u)$ кванторы $\forall z$, $\forall v$ и $\exists y$ положительны, а кванторы $\forall x$ и $\exists u$ — отрицательны. При установлении знаков кванторов необходимо, чтобы все переменные, связанные кванторами, были попарно различными. Их обозначения должны также отличаться от обозначений всех свободных переменных. При выполнении этих условий для секвенций, находящихся в AE-форме, т. е. состоящих из формул, в которых ни один квантор существования не может включать в область своего действия кванторы общности, может быть построена следующая процедура разрешения.

Сначала все входящие в секвенцию формулы приводятся к минисферному виду. Затем с помощью обращения правил (131) — (138) элиминируются все логические (пропозициональные) связки, допускающие такую элиминацию. Получающиеся в результате секвенции должны быть в AE-форме (поскольку иначе исходная секвенция не была бы AE-секвенцией). Во всех этих секвенциях опускаются все кванторы, причем переменные, связанные отрицательными кванторами, заменяются попарно различными числами, а переменные, связанные положительными кванторами, сохраняются без изменения.

Снова применяя обращения правил (131) — (138), приводят полученные секвенции к элементарному виду, т. е. к виду, не содержащему ни кванторов, ни пропозициональных логических связок. Отбрасываются истинные элементарные секвенции (т. е. такие секвенции, у которых имеется хотя бы одна общая для сукцедента и антецедента формула). Производя всевозможные (не обязательно взаимно однозначные) подстановки переменных вместо чисел в остающихся секвенциях, стараются сделать все их истинными. Если это удается, то исходная секвенция была истинной, если не удается — то она была ложной.

Рассмотрим в качестве примера две секвенции: $\forall x P(x) \rightarrow \exists y P(y)$

и $ExP(x) \to \forall y P(y)$. В первой секвенции оба квантора отрицательны. Поэтому описанная процедура удаления кванторов приведет ее к виду $P(1) \to P(2)$. Совершая подстановку переменной x вместо чисел 1 и 2, превратим последнюю секвенцию $P(x) \to P(x)$. Следовательно, первая из числа данных нам вначале секвенций является истинной. Оба же квантора второй секвенции положительны. Процедура исключения кванторов приведет ее к виду $P(x) \to P(y)$, представляющему собой в общем случае (для произвольного предиката P и нетривиальной предметной области) ложную секвенцию. Следовательно, вторая из исходных секвенций ложна.

Полученные результаты совпадают с результатами прямой проверки заданных секвенций, которую в рассматриваемом случае нетрудно осуществить. Заметим, что, если бы, вопреки оговоренному выше условию, во второй секвенции обе связанные переменные были обозначены одной и той же буквой, мы пришли бы к неправильному заключению, признав эту секвенцию истинной. Полезно отметить также, что описанная процедура даже без предварительного приведения формул к минисферному виду пригодна для разрешения всех AE-секвенций, содержащих не более чем по одному положительному квантору.

Наряду с описанной выше процедурой разрешения для исчисления высказываний (процедурой I), только что описанная процедура без приведения формул к минисферному виду (процедура II) была запрограммирована Хао-Вангом для универсальной электронной цифровой машины ИБМ-704.

Работая по программе I, машина в течение примерно трех минут доказала все 220 теорем исчисления высказываний, составляющие первые пять глав из монографии «Principia mathematica». Общее время работы машины (с учетом времени ввода данных и вывода результатов) составляло около 37 *мин*. С помощью программы II примерно за час работы машина доказала около 130 теорем исчисления предикатов из 158 теорем, составляющих следующие пять глав той же монографии. Всего же программа II могла доказать 139 теорем, хотя при этом время решения значительно возрастало.

Если дополнить процедуру II приемом элиминации кванторов на основе обращения правил (139) — (142) и ввести в нее некоторые предварительные преобразования формул, составляющих исходную секвенцию, то доказуемыми становятся все 158 указанных выше теорем. Предварительные преобразования, о которых здесь идет речь, сводятся к тому, что к составляющим исходную секвенцию формулам применяются (до тех пор, пока это возможно) следующие правила замены:

$$\forall x (P(x) \wedge Q(x)) \text{ заменяется на } \forall x P(x) \wedge \forall x Q(x); \quad (151)$$

$$\exists x (P(x) \vee Q(x)) \text{ заменяется на } \exists x P(x) \vee \exists x Q(x); \quad (152)$$

$$\forall x (P(x) \supset (Q(x) \wedge R(x))) \text{ заменяется на } \forall x (P(x) \supset Q(x)) \wedge \wedge \forall x (P(x) \supset R(x)). \quad (153)$$

Перечисленные правила в какой-то мере заменяют процедуру приведения формул к минисферному виду, которая в общем случае оказывается достаточно сложной. Если после применения этих правил и элиминации логических связок с помощью обращения правил (131) — (138) все полученные секвенции будут AE-секвенциями и будут к тому же либо минисферными, либо содержащими не более одного положительного квантора, то разрешение секвенции может быть, как правило, проведено процедурой II.

Хао-Ванг предложил также дальнейшие усовершенствования описанных процедур, позволяющие выйти за пределы одних лишь AE-формул. Заметим, что с помощью одной из таких усовершенствованных процедур машина ИБМ-704 провела доказательство 350 теорем из первых девяти глав «Principia mathematica» за 8,5 *мин*. Процедуры, построенные Хао-Вангом, по-видимому, легко могут быть превращены в квазиразрешающие процедуры для всего узкого исчисления предикатов в том смысле, что они могут (после соответствующего дополнения) доказать любую доказуемую формулу этого исчисления и опровергнуть «почти все» недоказуемые формулы. Выражение «почти все» понимается здесь в сугубо практическом смысле и не может, разумеется, пониматься как «все, за исключением конечного числа».

Следует подчеркнуть различие между чисто теоретическим и практическим подходом к решению проблемы разрешимости. В теоретическом плане важен прежде всего сам факт существования или несуществования разрешающей процедуры для того или иного класса формул. Процедуры разрешения, которые строятся для этой цели, в большинстве случаев оказываются совершенно не пригодными для автоматизации доказательств теорем, поскольку они приводят к чересчур громоздким и длинным построениям.

Наоборот, при практическом подходе к построению разрешающих процедур основное внимание уделяется вопросам быстроты и легкости выполнения этих процедур. В то же время зачастую мирятся с тем, что построенная процедура разрешения не охватывает абсолютно всех формул заданного класса, лишь бы при практическом ее применении случаи, когда она не дает ответа (за некоторое, ограниченное заранее, время), были сравнительно редкими. Таким образом, практические процедуры разрешения могут являться не в точном смысле слова разрешающими, а лишь *квазиразрешающими* процедурами.

Нет поэтому ничего удивительного, что практически эффективные процедуры разрешения могут быть построены не только в разрешимых, но и в неразрешимых теориях. Не следует забывать, что человек, успешно работающий в области той или иной неразрешимой теории (например, в арифметике натуральных чисел), пользуется конечным (и, зачастую, даже не очень большим) числом приемов для проведения доказательств и построения опровергающих примеров. Задача практических процедур разрешения как раз и состоит в том, чтобы формализовать указанные приемы.

Разумеется, решение этой задачи упрощается, если область применения разрешающей процедуры заранее ограничивается некоторой достаточно узкой областью. В то же время предварительное установление принципиальной возможности решения проблемы разрешимости в соответствующей области, вообще говоря, не упрощает проблемы построения практического разрешающего алгоритма.

В настоящее время построен ряд разрешающих процедур для относительно простых разделов математики (алгебра вещественных полиномов, элементарная геометрия, теория абелевых групп с конечным числом образующих и др.). Однако все эти процедуры строились, как правило, в сугубо теоретическом аспекте и для превращения их в практические разрешающие алгоритмы предстоит затратить еще немало усилий.

Большой интерес представляет проблема построения алгоритмов, которые бы не просто доказывали или опровергали задаваемые человеком предложения, но и сами отыскивали новые интересные теоремы в той или иной области. Для построения такого рода алгоритмов необходимо выработать достаточно хороший критерий оценки степени нетривиальности теоремы.

Одна из первых попыток в этом направлении была сделана Хао-Вангом [77], построившим программу для отбора (с последующим доказательством) теорем в исчислении высказываний. Эта попытка, однако, оказалась не вполне удачной, ввиду бедности заложенного в программу критерия нетривиальности: машина печатала слишком большое число теорем, не производя должного отсева неинтересных (тривиальных) теорем.

Первый критерий нетривиальности, который обычно приходит в голову, состоит в том, что нетривиальная теорема должна относительно хорошо формулироваться (выражаться короткой формулой) и не иметь вместе с тем коротких доказательств. Установление еще более естественных критериев (совпадающих с обычными представлениями о нетривиальности теорем) становится возможным, если процесс отбора новых теорем и их доказательств построить на принципах самоорганизации. Достигнуть этого можно за счет пополнения исходной системы аксиом отбираемыми программой нетривиальными теоремами. Сложность теоремы естественно оценивать минимальным числом шагов, с помощью которых может быть осуществлено ее доказательство. Нетривиальными предложениями назовем исходные аксиомы и все теоремы, сложность которых превосходит некоторый выбираемый заранее порог. Каждое вновь доказанное нетривиальное предложение присоединяется к системе аксиом, в результате чего происходит переоценка сложности всех ранее полученных теорем. Исключая из системы аксиом теоремы, ставшие тривиальными, ищем новую нетривиальную теорему, присоединяем ее к системе аксиом, снова исключаем теоремы, ставшие тривиальными, и т. д.

Описанная самоорганизующаяся система построения формальных дедуктивных теорий в большой мере напоминает процесс по-

строения таких теорий человеком. Следует отметить, что переход к процессам построения дедуктивных теорий на принципах самоорганизации заставляет по-новому подойти к проблеме их разрешимости. Разумеется, если указанный процесс протекает изолированно от внешнего мира, то он оказывается в конце концов эквивалентным некоторому «жесткому» (несамоорганизующемуся) алгоритму, так что в постановке проблемы разрешимости ничего фактически не изменяется. То же самое будет, очевидно, и в том случае, когда на процесс воздействует некоторый внешний по отношению к нему алгоритм (случай «конструктивного внешнего мира»).

В случае «неконструктивного внешнего мира», когда внешние воздействия на рассматриваемый нами процесс не могут быть сведены к алгоритму, положение принципиально изменяется. В самом деле, предположим, что процесс, о котором идет речь, может накапливать информацию, поступающую извне, и осуществлять сравнение с нею задаваемых ему формул узкого исчисления предикатов. Предположим далее, что поступающая извне информация представляет собой две последовательности формул узкого исчисления предикатов, расположенных в порядке возрастания их сложности (оцениваемой числом составляющих формулы символов). Если первая последовательность содержит все истинные, а вторая — все неистинные формулы исчисления предикатов (что в случае «неконструктивной среды» не является невозможным), то нетрудно построить вполне конструктивную процедуру разрешения для (узкого) исчисления предикатов, основанную на накоплении все большего и большего количества внешней информации и сравнении с нею формул, подлежащих разрешению.

Само собой разумеется, что «неконструктивная среда» вовсе не обязана полностью брать на себя задачу разрешения, как это фактически имело место в приведенном примере. Выдаваемые ею неконструктивные последовательности могут и не быть непосредственно последовательностями формул. Они должны обладать лишь одной особенностью — возможностью их конструктивного преобразования (в рамках рассматриваемой самоорганизующейся процедуры разрешения) в соответствующим образом упорядоченные последовательности доказуемых (истинных) и недоказуемых (неистинных) формул узкого исчисления предикатов.

Возможно, что указанные соображения могут послужить в будущем основой для создания систем весьма далеко идущей автоматизации процессов научного творчества и прежде всего автоматизации процесса построения сложных дедуктивных теорий.

ЛИТЕРАТУРА

1. Ackermann W., Solvable cases of the decision problem, Amsterdam, 1954.
2. Аттли О. М., Машины условиой вероятности и условные рефлексы, в сб. «Автоматы», ИЛ, М., 1956.
3. Aufenkamp D. D., Analysis of sequential machines II, IRE Trans., EC-7, 1958, № 4.
4. Aufenkamp D. D., Hohn F. S., Analysis of sequential machines I, IRE Trans., EC-6, 1957, № 4.
5. Бауэр Ф. Л., Бэкус Дж. В. и др., Сообщение об алгоритмическом языке АЛГОЛ-60, Изд-во АН СССР, М., 1960.
6. Барсов А. С., Что такое линейное программирование, Физматгиз, 1959.
7. Беллман Р., Динамическое программирование, ИЛ, М., 1960.
8. Blake A., Canonical expression in Booleau algebra, Chicago, 1938.
9. Блох А. Ш., О задачах, решаемых последовательностными машинами, Проблемы кибериетики, 1960, вып. 3.
10. Bottenbruch H., Structure and use of ALGOL-60, Journ. Assoc. Comp. Mach., 1962, N 2.
11. Браверман Э. М., Опыты по обучению машины распознаванию зрительных образов, Автоматика и телемеханика, 1962, № 3.
12. Vaida S., The theory of games and linear programming, Princeton, 1956; в сб. «Линейные неравенства и смежные вопросы», ИЛ, М., 1959.
13. Veitch E. W., A chart method for simplifying truth functions, Proc. Assoc.Comp. Mach., 1952, N 2 — 3.
14. Гантмахер Ф. Р., Теория матриц, Физматгиз, М., 1953.
15. Gentzen G., Untersuchungen über das logische Schliessen, Math. Zeitschrift, 1934—1935, т. 39.
16. Gödel K., Die Vollständigkeit der Axiome des logischen Funktionenkalkülus, Monatshefte Math. und Physik., 1930, т. 37.
17. Gödel K., On undecidable propositions of formal mathematical systems, Princeton, 1934.
18. Гельфанд И. М. и Цетлин М. Л., О иекоторых способах управления сложными системами, УМН, 1962, т. 17 вып. 1.
19. Ginsburg S., Synthesis of minimal state machines, IRE Trans., EC-8,1959.
20. Гильберт Д. и Аккерман В., Основы теоретической логики, ИЛ, 1947.
21. Глушков В. М., Об одном алгоритме синтеза абстрактных автоматов, УМЖ, 1960, т. 12, № 2.
22. Глушков В. М., Об одном методе анализа абстрактиых автоматов, ДАН УССР, 1960, № 9.
23. Глушков В. М., Некоторые проблемы синтеза цифровых авто-

матов, Журнал вычислительной математики и математической физики, 1961, т. 1, № 3.
24. Глушков В. М., Об одном методе автоматизации программирования, Проблемы кибернетики, 1959, вып. 2.
25. Глушков В. М., Самоорганизующиеся системы и абстрактная теория автоматов, Журнал вычислительной математики и математической физики, 1962, т. 2, № 3.
26. Глушков В. М., Синтез цифровых автоматов, Физматгиз, 1962.
27. Глушков В. М., Теория обучения одного класса дискретных персептронов, Журнал вычислительной математики и математической физики, 1962, т. 2, № 2.
28. Глушков В. М., К вопросу о самообучении в персептроне, Журнал вычислительной математики и математической физики, 1962, т. 2, № 6.
29. Глушков В. М., Ковалевский В. А., Рыбак В. И., Алгоритм обучения машины распознаванию простейших геометрических фигур, в сб. «Принципы построения самообучающихся систем», Гостехиздат, К., 1962.
30. Глушков В. М., Грищенко Н. М., Стогний А. А., Алгоритм распознавания осмысленных предложений, в сб. «Принципы построения самообучающихся систем», Гостехиздат, К., 1962.
31. Гнеденко Б. В., Королюк В. С., Ющенко Е. Л., Элементы программирования, Физматгиз, 1961.
32. Голдман С., Теория информации, ИЛ, М., 1957.
33. Детловс В. К., Нормальные алгоритмы и рекурсивные функции, ДАН СССР, 1953, т. 90, № 5.
34. Joseph A. D., On predicting perceptron performance, IRE Intern. Conv. Rec., 1960, v. 8, N 2.
35. Жегалкин Н. И., О технике вычислений предложений в символической логике, Матем. сб., 1927, т. 34.
36. Журавлев Ю. И., О невозможности построения минимальных дизъюнктивных нормальных форм функций алгебры логики в одном классе алгоритмов, ДАН СССР, т. 132, № 3, 1960.
37. Калужнин Л. А., Об алгоритмизации математических задач, Проблемы кибернетики, 1959, вып. 2.
38. Канторович В. Л., Математические методы организации и планирования пронзводства, Изд-во ЛГУ, Л., 1939.
39. Китов А. И., Криницкий Н. А., Электронные цифровые машины и программирование, Физматгиз, М., 1959.
40. Клини С., Представление событий в нервных сетях и конечных автоматах, в сб. «Автоматы», ИЛ, 1956.
41. Kleene S. C., Recursive predicates and quantifiers, Trans. Amer. Math. Soc., 1943, v. 53.
42. Клини С. К., Введение в метаматематику, ИЛ, М., 1957.
43. Колмогоров А. Н., Успенский В. А., К определению алгоритма, УМН, 1958, т. 13, вып. 4(82).
44. Королюк В. С., Ющенко Е. Л., Вопросы теории и практики программирования, Сборник трудов по вычислительной математике и технике, Изд-во АН УССР, К., 1961.
45. Copi I. M., Elgot C., Wright J. B., Realisation of ebents by logical nets, Journ. Assoc. Comp. Mach., 1958, v. 5, N 2.
46. Котельников В. А., О пропускной способности эфира и проволоки в электросвязи, в сб. «Материалы к 1-му Всесоюзному съезду по вопросам технической реконструкции дела связи и развития слаботочной промышленности», Изд. Ред. Упр. Связи РККА, 1933.
47. Кобринский Н. Е., Трахтенброт Б. А., Введение в теорию конечных автоматов, Физматгиз, М., 1962.
48. Крамер Г., Математические методы статистики, ИЛ, М., 1948.
49. Летичевский А. А., Дородницына А. А., Моделирование естественного отбора на вычислительной машине, в сб. «Принципы построения самообучающихся систем», Гостехиздат, К., 1962.

50 Ляпунов А. А., О логических схемах программ, Проблемы кибернетики, 1958, вып. 1.
51. Лупанов О. Б., О возможности синтеза схем из произвольных элементов, Труды математического института им. В. А. Стеклова, Изд-во АН СССР, т. 51, 1958.
52. Maltzew A. I., Untersuchungen aus dem Gebiete der mathematischen Logik, Матем. сб. 1936, т. 1(43).
53. Марков А. А., Теория алгорифмов, Труды математического института им. В. А. Стеклова, Изд-во АН СССР, т. 42, 1954.
54. Медведев Ю. Т., О классе событий, допускающих представление в конечном автомате, в сб. «Автоматы», ИЛ, 1956.
55. Mealy G. H., A method for synthesizing sequential circuits, Bell System Techn. J., 1955, v. 34.
56. Михалевич В. С., Шор Н. З., Численное решение многовариантных задач по методу последовательного анализа вариантов, в сб. «Научно-методические материалы экономико-математического семинара», вып. 1, Изд-во АН СССР, М., 1962.
57. Мур Э. Ф., Умозрительные эксперименты с последовательностными машинами, в сб. «Автоматы», ИЛ, 1956.
58 Нагорный Н. М., К усилению теоремы приведения теории нормальных алгоритмов, ДАН СССР, 1953, т. 90, № 3.
59. Novicov P. S., On the consistensy of certain logical calculus, Матем. сб. 1943, т. 12(54).
60. Новиков П. С., Об алгоритмической неразрешимости проблемы тождества слов в теории групп, Труды математического института им. В. А. Стеклова, Изд-во АН СССР, т. 44, 1955.
61. Новиков П. С., Элементы математической логики, Физматгиз, 1962.
62. Post E. L., Introduction to the general theory of elementary propositions, Amer. J. Math., 1921, v. 43.
63. Post E. L., Finite combinatory processes. Formulation 1, J. Symb. Logic, 1936, v. 1.
64. Post E. L., Recursive unsolvability a problem of Thue, J. Symb. Logic, 1947, v. 12.
65. Поваров Г. П., Математическая теория синтеза контактных n-полюсников, ДАН СССР, 1959, т. 100, № 5.
66. Prawitz D., Prawitz H., Voghera N., A mechanical proof procedure and its realization in an electronic computer, Journ. Ass. Comp. Mach., 1960, v. 7 № 2.
67 Roberts I. O., Pattern recognition with an adaptive netvoork, IRE Intern. Conv. Rec., 1960, № 2.
68. Романовский В. И., Дискретные цепи Маркова, ГТТИЛ, М.—Л., 1949.
69. Rosenblatt F., Two theorems of statistical separability in the Perceptron, Symp. Mechaniz. Thought Proc., Nat. Phys. Lab., Teddington, England, 1958.
70 Rosenblatt F., Perceptron simulation experiments, Proc. IRE, 1960, v. 48, N 3.
71. Rosser B., Extensions of some theorems of Godel and Church, Journ. Symb. Logic, 1936, v. 1.
72. Selfridge O. G., Pandemonium: a paradigm for learning, Symp. Mechaniz. Thought Proc, Nat. Phys Lab., Teddington, England, 1958.
73. Turing A. M., On computable numbers with an application to Entscheidungsproblem, Proc. London Math. Soc. 1936, v. 42(2); 1937, v. 43.
74. Whitehead A. and Russel B., Principia mathematica, Cambridge, 1910, v. 1; 1912, v. 2; 1913, v. 3.
75 Wilkes M. V., Stringer J. B., Microprogramming and the design of the control circuits in an electronic digital computer, Proc. Cambridge Phil. Soc., 1953, v. 49, № 2.

76. Успенский В. А., Лекции о вычислимых функциях, Физматгиз, М., 1960.
77. Hao-Wang, Tomards mechanical mathematics, IBM Journ., 1960.
78. Hao-Wang, Proving theorems by pattern recognition I, Communic. Assoc. Comp. Mach., 1960, v. 3, № 4.
79. Hao-Wang, Proving theorems by pattern recognition II, Bell System Techn. Journ., 1961, v. 40, № 1.
80. Shannon C., The synthesis of two-terminal switching circuits, Bell System Techn. Journ., 1949, v. 28, № 1.
81. Феллер В., Введение в теорию вероятностей и ее приложения, ИЛ, М., 1952.
82. Цетлин М. Л., Некоторые задачи о поведении конечных автоматов, ДАН СССР, 1961, т. 139, № 4.
83. Яблонский С. В., Функциональные построения в k-значной логике, Труды математического института им. В. А. Стеклова, Изд-во АН СССР, т. 51, 1958.

ОГЛАВЛЕНИЕ

Виктор Михайлович Глушков

Введение в кибернетику

Печатается по постановлению научного совета по кибернетике АН Украинской ССР

Редактор *Е. Л. Орлик*

Художественный редактор *И. П. Антонюк*

Оформление художника *П. В. Логвинова*

Технический редактор *А. А. Матвейчук* Корректоры *М. М. Азаренко, Г. М. Столярчук*

БФ 31895. Зак. № 1302. Изд. № 342. Тираж 15 000. Формат бумаги 60×92¹/₁₆. Печ. физ. листов 20,25. Услов. печ. листов 20,25. Учетно-изд. листов 20,69. Подписано к печати 7.XII 1963 г. Цена 1 руб. 25 коп.

Отпечатано с матриц типографии Издательства АН УССР, в Киевской областной типографии, Киев, ул. Ленина, 19.