А.Ш. БЛОХ

ГРАФ-СХЕМЫ И АЛГОРИТМЫ

Допущено
Министерством просвещения БССР
в качестве учебного пособия для студентов
физико-математических факультетов
педагогических институтов

МИНСК
"ВЫШЭЙШАЯ ШКОЛА"
1987.

ББК 22.12я73 + 22.174я73
Б70
УДК 510.51+519.17] (075.8)

Р е ц е н з е н т ы: кафедра алгебры Брестского государственного педагогического института им. А.С.Пушкина; *В.С.Танаев*, доктор физико-математических наук, профессор

Блох А.Ш.
Б70 Граф-схемы и алгоритмы: Учеб. пособие для физ.-мат. фак. пед. интов. – Мн.: Выш. шк., 1987. – 144 с.: ил.

Излагаются основные сведения по теории граф-схем, рассматриваются классические алгоритмические системы, вопросы построения алгоритмов и их программ.

Может быть полезно студентам математических специальностей вузов, а также преподавателям математики и информатики средней школы.

Б $\frac{1702020000-113}{M304\,(03)-87}$ 21–87

ББК 22.12я73+22.174я73

ПРЕДИСЛОВИЕ

В связи с изменениями учебных планов средней и высшей школ, вызванными введением нового курса "Информатика и вычислительная техника", появилась настоятельная необходимость в разработке учебных пособий, отражающих общую тенденцию расширения и углубления конструктивного направления в математике, основой которого является теория алгоритмов.

Определение алгоритма было сформулировано в середине нашего столетия в форме трех классических алгоритмических систем: машин Тьюринга и Поста, нормальных алгоритмов Маркова и рекурсивных функций. Классические алгоритмические системы были в основном разработаны до появления электронных вычислительных машин, исходя из потребностей самой математики. Практика решения задач на компьютерах вскрыла широкий круг важных проблем, весьма далеких от рассматриваемых в классических системах алгоритмов. Исследования по этим проблемам составили новый раздел теории алгоритмов, представляющий интерес не только для самой математики, но и для ее приложений.

Литература по теории алгоритмов немногочисленна, несмотря на важность этой проблематики. Кроме того, во всех изданных книгах излагаются только классические алгоритмические системы.

В основе данного пособия лежит цикл лекций по теории граф-схем, которые автор в течение ряда лет читал на математическом факультете Минского государственного педагогического института им. А.М.Горького. Оно соответствует программе спецкурса "Граф-схемы и их приложения", утвержденной Министерством просвещения БССР, и содержит результаты последних научных исследований.

Отличительной чертой данного пособия является то, что в нем описание алгоритмов проводится с помощью граф-схем. Это, во-первых, упрощает изложение основ теории машин Тьюринга и, во-вторых, что более важно, позволяет выйти за рамки классической теории алгоритмов благодаря тому, что в теорию включаются вопросы, поставленные практикой использования компьютеров.

Книга состоит из четырех глав. В первой излагается теория граф-схем, необходимая для изучения последующего материала. Вторая глава посвящена классическим системам алгоритмов. В третьей главе дается общее определение алгоритма и приводится перечень правил построения оптимальных схем алгоритмов и их программ. В четвертой главе рассматриваются вопросы построения алгоритмов по неполной информации. Третья и четвертая главы не связаны непосредственно со второй главой. Изложение теоретического материала иллюстрируется примерами.

Автор выражает благодарность рецензентам: кафедре алгебры Брестского государственного педагогического института им. А.С. Пушкина, возглавляемой кандидатом физико-математических наук доцентом Ю.И.Дежурко, и заведующему лабораторией Института технической кибернетики АН БССР доктору физико-математических наук профессору В.С.Танаеву за ценные советы и замечания.

Все отзывы и пожелания, направленные на улучшение учебного пособия, просьба присылать по адресу: 220048, Минск, проспект Машерова, 11, издательство "Вышэйшая школа".

Автор

1. ГРАФ-СХЕМЫ

1.1. ЭЛЕМЕНТАРНЫЕ ГРАФ-СХЕМЫ

Переменная, принимающая только два значения: 0 или 1, называется *двоичной переменной,* а функция от двоичных независимых переменных – *двоичной функцией.* Процессы управления, за редким исключением, описываются двоичными функциями.

Двоичную функцию естественно задавать таблицей. Рассмотрим несколько примеров.

Пример 1.1. Имеются источник воды и три потребителя: A_1, A_2 и A_3. Потребители получают воду в зависимости от наличия заявок:

1) если заявки от потребителей A_1 и A_2 отсутствуют, воду получает A_3;

2) если есть заявки от A_1 или A_2, то при наличии заявки от A_3 воду получает A_2, а при отсутствии таковой – A_1.

Найти двоичную функцию распределения воды.

Р е ш е н и е. Обозначим заявки соответственно через x_1, x_2, x_3. Условимся считать $x_1 = 1$ при наличии заявки от A_1 и $x_1 = 0$ при ее отсутствии. Аналогичные условия примем для x_2, x_3. Требуется найти функцию $y = f(x_1, x_2, x_3)$, где значениями y будут A_1, или A_2, или A_3.

Так как набор (x_1, x_2, x_3) принимает восемь значений, таблица функции $f(x_1, x_2, x_3)$ содержит восемь строк. По условиям задачи для каждой строки (x_1, x_2, x_3) в столбце y записывают значения $f(x_1, x_2, x_3)$. Искомая двоичная функция задана табл. 1.1. Например, шестая строка таблицы означает, что при наличии заявок от A_1 и A_3 и отсутствии заявки от A_2, т.е. при $x_1 = 1, x_2 = 0, x_3 = 1$, воду получает потребитель A_2.

Таблица 1.1

Номер строки	x_1	x_2	x_3	y	Номер строки	x_1	x_2	x_3	y
1	0	0	0	A_3	5	1	0	0	A_1
2	0	0	1	A_3	6	1	0	1	A_2
3	0	1	0	A_1	7	1	1	0	A_1
4	0	1	1	A_2	8	1	1	1	A_2

Двоичную функцию изображают фигурой, которую называют *элементарной граф-схемой.* На рис. 1.1 приведена граф-схема функции, определяемой табл. 1.1. В вершинах записаны переменные. Из каждой вершины выходят две дуги: *0* и *1.* Вершина и две выходящие из нее дуги образуют *куст.*

По элементарной граф-схеме значение функции находят следующим образом. Пусть заданы значения $x_1 = 0, x_2 = 1, x_3 = 1$. Из вершины x_1 граф-схемы по дуге $x_1 = 0$ попадают в вершину x_2. Так как $x_2 = 1$, то по дуге $x_2 = 1$ попа-

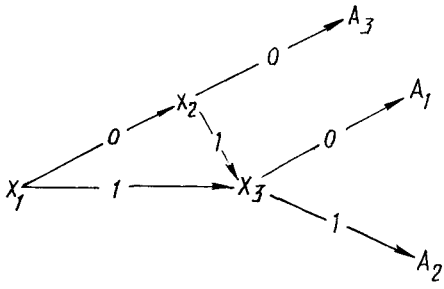


Рис. 1.1

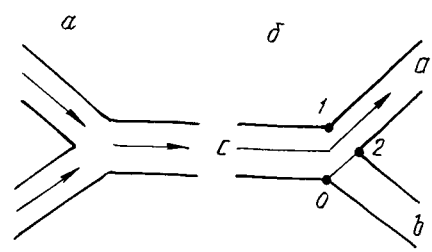


Рис. 1.2

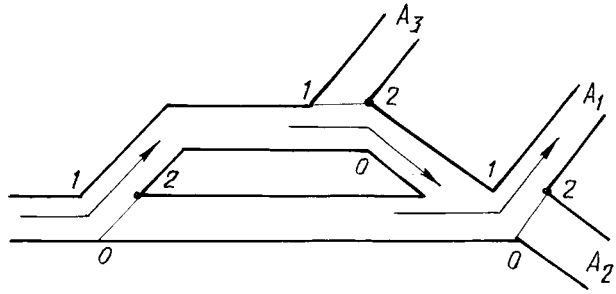


Рис. 1.3

дают в вершину x_3 , затем по дуге $x_3 = 1$ попадают в конечную вершину A_2 . Итак, значение функции для $x_1 = 0, x_2 = 1, x_3 = 1$ равно A_2.

Граф-схема задает функцию более компактно и наглядно, чем таблица. Но главное – это то, что в многочисленных приложениях необходимо знание граф-схемы.

Пусть, например, требуется практически реализовать условия примера 1.1. При этом можно использовать трубы, тройники для соединения двух труб (рис. 1.2, *а*) и переключатели (рис. 1.2, *б*), обеспечивающие движение воды в одном из двух возможных каналов: *a* или *b*. В переключателе имеется заслонка, вращающаяся на оси *2* и занимающая положение *0* или *1*. В положении *0* открыт выход *a* , в положении *1* – выход *b* . Чтобы получить требуемое распределительное устройство, нужно в граф-схеме, приведенной на рис. 1.1, кусты заменить переключателями, соединение – тройником, дуги – каналами.

На рис. 1.3 изображено требуемое устройство. Положение заслонок соответствует $x_1 = 0, x_2 = 1, x_3 = 0$. Изменяя вручную или автоматически положение заслонок, можно добиться требуемого распределения воды. Такая система каналов пригодна не только для воды, но и для любой жидкости или газа и даже электрического тока. В последнем случае имеем электрическую схему управления, состоящую из проводов и электрических переключателей.

На рис. 1.4 дано схематическое изображение электрического переключателя. Стержень, вращаясь вокруг оси *2* , занимает положение *0* или *1*. В положении *0* замкнута цепь *ca* и разомкнута цепь *cb*, в положении *1* разомкнута цепь *ca* и замкнута цепь *cb* . На одной оси *2* может быть расположен комплект переключателей, находящихся в одинаковом состоянии.

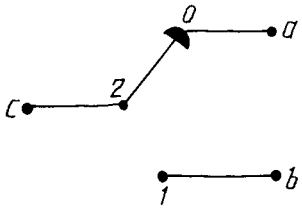


Рис. 1.4

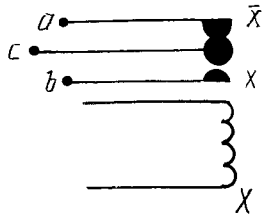


Рис. 1.5

Электрический переключатель выполняют также в виде пары контактов. Если один контакт замыкает свою цепь, то второй размыкает свою цепь, и наоборот. Переключение контактов осуществляется либо вручную, либо автоматически с помощью реле. Электромагнитное реле – это устройство, у которого обмотка является приемным элементом, а контакты – исполнительными элементами. Контакты могут быть замыкающими и размыкающими (рис.1.5). Контакт называется *замыкающим,* если при подаче напряжения на обмотку реле цепь между клеммами *b* и *c* замыкается, а в спокойном состоянии реле цепь разомкнута. Контакт называется *размыкающим,* если при подаче напряжения на обмотку реле цепь между клеммами *a* и *c* размыкается, а в спокойном состоянии реле цепь замкнута.

Реле (обмотку или несколько обмоток) обозначают прописными буквами латинского или русского алфавита: *X, Y, A, Б,* ...; замыкающие контакты – строчными буквами: *x, y, a, б,* ...; размыкающие контакты – строчными буквами с чертой сверху: $\bar{x}, \bar{y}, \bar{a}, \bar{б}, \ldots$. При подаче напряжения на обмотку реле *X* считаем $X = 1$, в невозбужденном состоянии $X = 0$. Если контакт замкнут, то $x = 1$, если разомкнут – $x = 0$. Итак, если $X = 1$, то $x = 1, \bar{x} = 0$, если же $X = 0$, то $x = 0, \bar{x} = 1$.

Реализацию граф-схемы от двоичных переменных на электромагнитных реле назовем *контактной схемой.* В контактной схеме изображают только контакты реле, так как лишь они замыкают и размыкают электрические цепи исполнительных элементов. Если цепь замкнута, считается, что ее структурная проводимость равна 1, если разомкнута – 0. От значения структурной проводимости зависит состояние исполнительного элемента.

Пример 1.2. В коридор, освещаемый одним светильником, выходят четыре двери. Возле каждой двери имеется кнопка, с помощью которой изменяют состояние одного переключателя или их комплекта. При входе в коридор из какой-либо двери изменяют положение кнопки возле этой двери и включают светильник. Выйдя из коридора через эту же или другую дверь, снова меняют положение кнопки и выключают светильник. Найти двоичную функцию управления светильником и ее граф-схему.

Р е ш е н и е. Обозначим двери и их переключатели через a, b, c, d , а светильник – через *Л* . Переключатели могут находиться в двух состояниях: 0 или 1. Условимся считать $Л = 1$, если светильник включен, и $Л = 0$ в противном случае.

Табл. 1.2 искомой функции $Л = f(a, b, c, d)$ содержит шестнадцать строк по числу двоичных наборов (a, b, c, d). Столбцы a, b, c, d заполняют заранее, до анализа задачи, так, чтобы получить таблицу стандартного вида. Это означает, что в столбце a верхняя половина строк заполнена нулями, а нижняя – единицами. В столбце b четверти столбца чередуются, начиная с нулей, и т. д. (Укажем другое правило заполнения столбцов a, b, c, d. Считая a, b, c, d разрядами числа, их располагают в порядке возрастания – сверху вниз.) Столбец *Л* заполняют исходя из условий задачи. Так, например, для нулевой строки $a = b = c = d = 0$, и из условия задачи следует $Л = 0$.

Таблица 1.2

a	b	c	d	$Л$
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Рассмотрим, например, ситуацию, когда в коридор входят через дверь a и зажигают свет, изменяя положение переключателя a. Это означает, что в строке $a = 1, b = c = d = 0$ имеем $Л = 1$. (Аналогично получаем $Л = 1$ в строках с одной единицей.) Затем из коридора выходят через дверь b, гасят свет, изменяя положение переключателя b. Поэтому в строке $a = b = 1, c = d = 0$ имеем $Л = 0$. (Аналогично получаем $Л = 0$ для всех строк с двумя единицами.) Снова входя в коридор через дверь c, зажигают свет, изменяя положение переключателя c. Имеем $Л = 1$ при $a = b = c = 1, d = 0$. (Аналогично получаем $Л = 0$ в строках с тремя единицами.) И, наконец, выходя из коридора через дверь d, гасят свет. Это означает, что при $a = b = c = d = 1$ $Л = 0$.

Легко убедиться, что на рис. 1.6, *а* изображена граф-схема функции $Л = f(a, b, c, d)$.

Чтобы получить электрическую схему, нужно каждый куст граф-схемы заменить переключателем, к выходу *1* подключить светильник *Л*. Полюс (выход) *0* можно вместе с входящими дугами исключить. Электрическая схема дана на рис. 1.6, *б*.

Как известно, в зависимости от коэффициентов и свободных членов система уравнений

$$\left.\begin{aligned} a_1 x + b_1 y &= c_1; \\ a_2 x + b_2 y &= c_2 \end{aligned}\right\}$$

может не иметь ни одного решения, иметь только одно решение и, наконец, иметь бесчисленное множество решений.

Рассмотрим более простую систему уравнений.

Пример 1.3. Дана система уравнений

$$\left.\begin{aligned} ax &= b; \\ x + cy &= 1. \end{aligned}\right\}$$

Указать двоичную функцию, определяющую решение этой системы уравнений, и ее граф-схему.

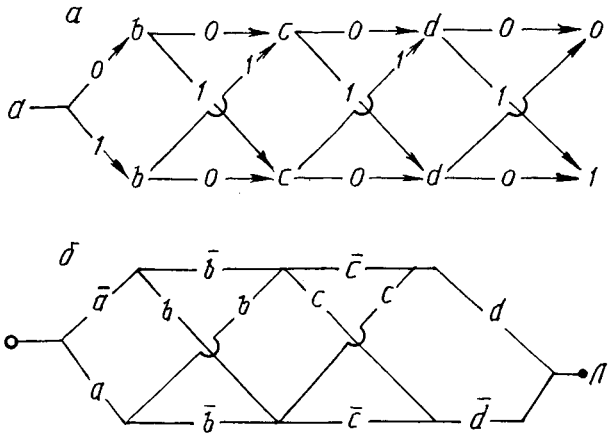


Рис. 1.6

Р е ш е н и е. Рассмотрим возможные случаи:

1) если $a \neq 0, c \neq 0$, то $x = b/a$, $y = (a - b)/(ac)$;

2) если $a \neq 0, c = 0, a = b$, то $x = 1$, y – любое число;

3) если $a \neq 0$, $c = 0, a \neq b$, система не имеет решений;

4) если $a = 0$, $b = 0$, то $x = 1 - cy$, y – произвольное число;

5) если $a = 0$, $b \neq 0$, система не имеет решений.

В случаях 4 и 5 можно условия $b = 0$ и $b \neq 0$ заменить соответственно $b = a$ и $b \neq a$, ибо $a = 0$.

Введем двоичные переменные:

$$p_1 = \begin{cases} 0, \text{ если } a \neq 0; \\ 1, \text{ если } a = 0; \end{cases} \qquad p_2 = \begin{cases} 0, \text{ если } c \neq 0; \\ 1, \text{ если } c = 0; \end{cases} \qquad p_3 = \begin{cases} 0, \text{ если } b \neq a; \\ 1, \text{ если } b = a. \end{cases}$$

Обозначим через A_3, A_2 соответственно решения ($x = b/a$, $y = (a-b)/(ac)$) и ($x = 1 - cy$, y – любое число), а через A_1 – тот факт, что система уравнений противоречива.

Двоичную функцию зададим табл. 1.3 стандартного вида. В столбце A укажем соответствующие значения A_1, A_2, A_3. Первые две строки определяются случаем 1, третья строка – случаем 3, четвертая – случаем 2, пятая и седьмая строки – случаем 5, шестая и восьмая строки – случаем 4.

Таблица 1.3

Номер строки	p_1	p_2	p_3	A	Номер строки	p_1	p_2	p_3	A
1	0	0	0	A_3	5	1	0	0	A_1
2	0	0	1	A_3	6	1	0	1	A_2
3	0	1	0	A_1	7	1	1	0	A_1
4	0	1	1	A_2	8	1	1	1	A_2

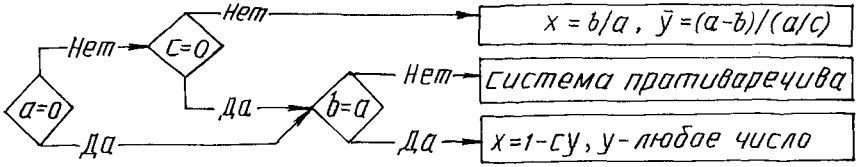


Рис. 1.7

Так как двоичные функции из примеров 1.1 и 1.3 совпадают, то и их граф-схемы одинаковы (см. рис. 1.1). Поэтому при решении примеров указанного типа можно воспользоваться граф-схемой, приведенной на рис. 1.1. Для облегчения пользования граф-схемой раскроем содержание p_1 , p_2 , p_3 , A_1 , A_2 , A_3. Получим фигуру (рис. 1.7), которую называют *схемой алгоритма.* В данном примере это схема алгоритма решения системы уравнений вида $ax = b$, $x + cy = 1$. Здесь, например, вместо p_1 записано $a = 0$, вместо 0(1) на дугах – "Нет" ("Да"). Если выполняется условие $a = 0$, то исполнитель перемещается по дуге "Да", в противном случае – по дуге "Нет".

Пусть дана система $\left.\begin{matrix} 3x = 7; \\ 1 + 2y = 1. \end{matrix}\right\}$ Так как $a = 3$, $c = 2$, то условия $a = 0$, $c = 0$ не выполняются. Поэтому исполнитель перемещается из начала схемы алгоритма по верхним дугам и находит $x = \frac{7}{3}$, $y = \frac{3-7}{3 \cdot 2} = -\frac{2}{3}$.

Отметим, что одна и та же граф-схема в примерах 1.1 и 1.3 получила различные представления: в виде схемы управления потоками воды и в виде алгоритма решения одного вида системы линейных уравнений.

Уже на этих весьма простых примерах видно, что граф-схемы имеют важные применения. Поэтому необходимо научиться по двоичной функции строить ее граф-схему, и притом возможно наиболее простую, так как для данной функции существует не одна элементарная граф-схема.

1.2. ПОСТРОЕНИЕ ЭЛЕМЕНТАРНЫХ ГРАФ-СХЕМ

Обычно двоичную функцию задают таблицей, поэтому рассмотрим *синтез элементарных граф-схем* по заданной таблице. Считаем, что таблица задана в стандартной форме, а именно: строка α_1 , α_2 , ..., α_n предшествует строке β_1 , β_2 , ..., β_n , если число $\alpha_1\alpha_2...\alpha_n$ меньше числа $\beta_1\beta_2...\beta_n$. Например, строка (0, 1, 0, 1, 1) предшествует строке (1, 0, 0, 1, 0), так как $1011 < 10010$. Строку переменных можно считать числом в двоичной системе счисления, называемым *числовым эквивалентом строки.* Например, числовыми эквивалентами строк (0, 1, 0), (1, 0, 1), (0, 1, 1) будут соответственно 10, 101, 11 (или 2, 5, 3 в десятичной системе счисления). В таблице стандартной формы числовой эквивалент равен номеру строки, считая с верхней (нулевой) строки.

Табл. 1.4 двоичной функции задана в стандартной форме. Для того чтобы сохранить эту стандартную форму при перестановке столбцов, т. е. при изменении порядка следования переменных, необходимо сделать соответствующую перестановку строк.

Таблица 1.4

x_2	x_3	x_1	y	x_2	x_3	x_1	y
0	0	0	3	1	0	0	5
0	0	1	5	1	0	1	4
0	1	0	3	1	1	0	3
0	1	1	3	1	1	1	5

Табл. 1.5 получена из табл. 1.4 в результате перестановки столбцов и, следовательно, строк.

Таблица 1.5

x_1	x_2	x_3	y	x_1	x_2	x_3	y
0	0	0	3	1	0	0	5
0	0	1	3	1	0	1	3
0	1	0	5	1	1	0	4
0	1	1	3	1	1	1	5

При использовании *канонического метода синтеза граф-схем* по таблице значений функции сначала строят каноническую таблицу, а затем преобразуют ее в граф-схему.

Каноническая таблица для функции от n переменных состоит из n+1 столбцов. В n-м столбце (столбце значений функции) имеется 2^n чисел, в нулевом столбце – одно число, в промежуточном, i-м, столбце – 2^i чисел. Числа, образующие столбцы канонической таблицы, называются *номерами.*

Опишем методику построения канонической таблицы. Из таблицы, заданной в стандартной форме, выпишем столбец значений функции $y = f(x_1, x_2, ..., x_n)$ и строку, указывающую порядок следования переменных. Пусть выбран такой порядок: $x_1, x_2, ..., x_n$. Разобьем номера последнего, n-го, столбца на пары и каждую полученную пару пронумеруем. Номер запишем левее и посередине пары. Полученные 2^{n-1} номера образуют $(n-1)$-й столбец канонической таблицы и соответствуют x_n.

При нумерации пар необходимо придерживаться такого правила: различные пары (a, b) нумеруются разными номерами c, одинаковые пары – одинаковыми номерами, пара из равных номеров (a, a) – этим же номером a.

Каждый номер c соединим дугами с номерами соответствующей пары (a, b). Одну из дуг $(x = 1)$ назовем *нижней,* другую $(x = 0)$ – *верхней.* Две дуги образуют *куст с входной вершиной* c и *выходными вершинами*: верхней a, нижней b. Обозначим куст $(c; x_n, a, b)$.

Пусть i-й столбец канонической таблицы уже построен. Для построения следующего, $(i-1)$-го, столбца разобьем на пары номера i-го столбца и пронумеруем их по тем же правилам. Каждый номер c соединим двумя дугами с номерами a, b пары, которой он поставлен в соответствие.

За исключением случая куста $(a; x, a, a)$, одинаковые пары (a, b) раз-

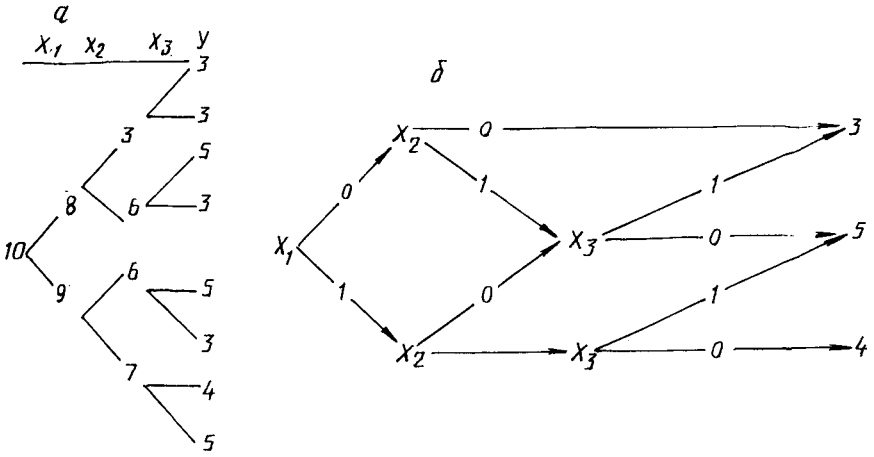


Рис. 1.8

личных столбцов нумеруются разными номерами c . Для удобства номера́ можно брать в порядке возрастания.

Построенную таким образом треугольную таблицу называют *канонической.* Для табл. 1.5 получим каноническую таблицу, приведенную на рис. 1.8,*а*.

Каноническая таблица реализуется граф-схемой из последовательно соединенных блоков, причем в каждом блоке содержатся кусты одной переменной. Преобразование канонической таблицы выполняется по столбцам от входной вершины к столбцу значений функции. Пусть в процессе построения граф-схемы мы дошли до i-го столбца. Найдем в нем номер (если такой есть), совпадающий с одним из значений функции; этот номер оставим, а все кусты, расположенные правее его и связанные с ним путями, исключим.

Среди оставшихся в столбце номеров отыщем группу одинаковых (если такие есть) и один из них отметим. У неотмеченных номеров из этой группы исключим кусты, расположенные правее их и связанные с ними путями. Соединим неотмеченные номера с отмеченными. Так поступим с каждой группой одинаковых номеров. В вершинах кустов запишем переменную этого столбца. Перейдем к $(i+1)$-му столбцу. На рис. 1.8,*б* изображена граф-схема, построенная по канонической таблице, приведенной на рис. 1.8, *а* .

Таблица 1.6

x_1	y_1	x_0	y_0	z	x_1	y_1	x_0	y_0	z
0	0	0	0	2	1	0	0	0	1
0	0	0	1	3	1	0	0	1	1
0	0	1	0	1	1	0	1	0	1
0	0	1	1	2	1	0	1	1	1
0	1	0	0	3	1	1	0	0	2
0	1	0	1	3	1	1	0	1	3
0	1	1	0	3	1	1	1	0	1
0	1	1	1	3	1	1	1	1	2

Граф-схемы сравнения реализуют операцию сравнения чисел.

Пример 1.4. Пусть заданы числа x, y в двоичной системе счисления: $x = x_0 + 2x_1 + \ldots + 2^{n-1}x_{n-1}$, $y = y_0 + 2y_1 + \ldots + 2^{n-1}y_{n-1}$, где $x_i = \{1, 0\}$; $y_i = \{1, 0\}$. Построить граф-схемы сравнения.

Р е ш е н и е. Первая граф-схема сравнения реализует функцию $z = F_1(x, y)$, которая при $x > y$, $x = y$, $x < y$ принимает соответственно значения 1, 2, 3. Представим эту функцию как функцию двоичных переменных: $z = f(x_0, y_0, \ldots, x_{n-1}, y_{n-1})$. Построим граф-схему сравнения двухразрядных чисел x и y. Функция $z = f(x_0, y_0, x_1, y_1)$ задана табл. 1.6 стандартного вида.

На рис. 1.9 приведена каноническая таблица функции F_1, а на рис. 1.10 – ее граф-схема.

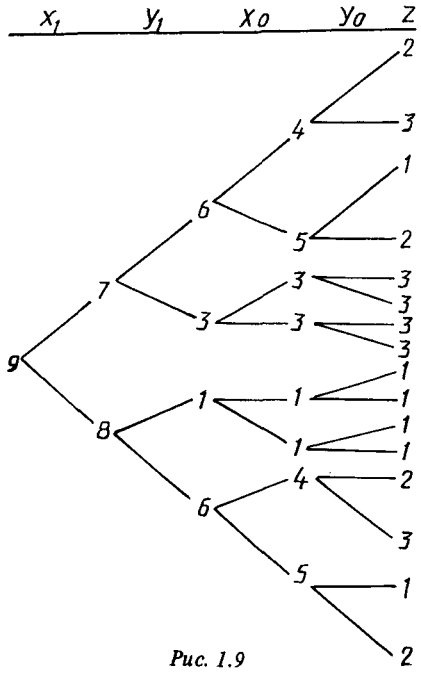


Рис. 1.9

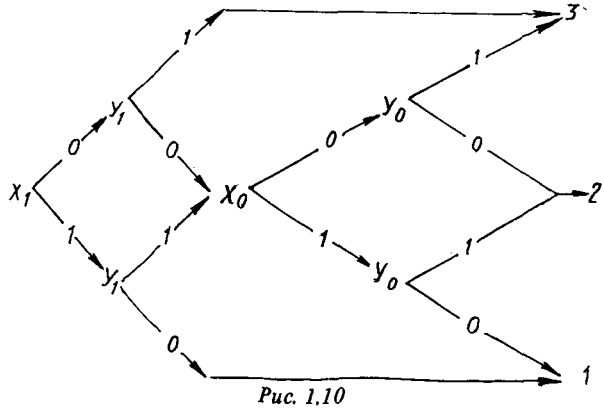


Рис. 1.10

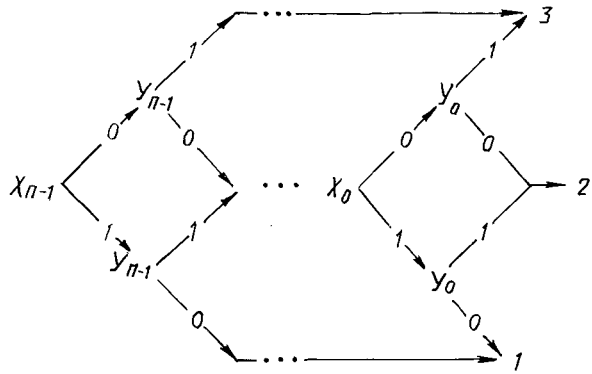


Рис. 1.11

Заметим, что граф-схема сравнения состоит из двух одинаковых блоков. Если по таблице построить граф-схему для трехразрядных чисел, она будет содержать три таких блока. Легко доказать по индукции, что для n-разрядных чисел граф-схема содержит n блоков (рис. 1.11).

Вторая граф-схема сравнения изображает функцию $z = F_2(x, y)$, которая при $x \geqslant y$, $x < y$ принимает соответственно значения 1 и 3. Такую граф-схему можно получить из первой граф-схемы сравнения, записав на втором выходе номер 1. Затем, проведя перенумерацию согласно изложенным выше правилам, получим вторую граф-схему сравнения. Очевидно, что только последний блок этой граф-схемы будет отличаться от типового.

Третья граф-схема сравнения изображает функцию $z = F_3(x, y)$, которая при $x \neq y$, $x = y$ принимает соответственно значения 1 и 2. Ее можно получить из первой, записав на третьем выходе номер 1. Последующей перенумерации не будет.

Покажем, что первая граф-схема сравнения содержит минимальное число кустов. Если в граф-схеме придать всем переменным, кроме x_k и y_k, нулевые значения, получим граф-схему сравнения двух одноразрядных чисел x_k, y_k. Таким образом, первая граф-схема обязательно должна содержать все одноразрядные граф-схемы сравнения. А так как она содержит только эти блоки и каждый блок минимален, ибо состоит из трех кустов, то первая граф-схема сравнения минимальная по числу кустов. Минимальность остальных граф-схем сравнения доказывается несколько сложнее.

Пример 1.5. Построить граф-схему одного разряда двоичного сумматора.

Р е ш е н и е. Пусть x_i, y_i – двоичные разряды слагаемых x, y, а z_i – перенос в i-й разряд. Обозначим i-й разряд суммы $x + y$, а перенос в $(i+1)$-й разряд – соответственно через s_i, z_{i+1}. В табл. 1.7 указаны значения s_i, z_{i+1}, вычисляемые по известным правилам

Таблица 1.7

Номер строки	x_i	y_i	z_i	s_i	z_{i+1}	Номер строки	x_i	y_i	z_i	s_i	z_{i+1}
1	0	0	0	0	0	5	1	0	0	1	0
2	0	0	1	1	0	6	1	0	1	0	1
3	0	1	0	1	0	7	1	1	0	0	1
4	0	1	1	0	1	8	1	1	1	1	1

сложения для двоичной системы счисления. Например, для четвертой строки имеем $x_i = 0$, $y_i = 1$, $z_i = 1$. Отсюда $0 + 1 + 1 = 10$, т.е. $s_i = 0$, $z_{i+1} = 1$. Для восьмой строки имеем $1 + 1 + 1 = 11$, поэтому $s_i = 1$, $z_{i+1} = 1$.

По табл. 1.7 строим каноническую таблицу (рис. 1.12), а затем граф-схему (рис. 1.13, *а*) и контактную схему (рис. 1.13, *б*). Исполнительными элементами являются реле S и Z'. Контакты реле Z' находятся в блоке следующего разряда. Соединив n таких схем, получим контактную схему n-разрядного сумматора.

Пример 1.6. Построить граф-схему одного разряда двоичного вычитателя.

Р е ш е н и е. Пусть x_i, y_i – двоичные разряды чисел x, y, а z_i – "долг" в i-м разряде. Обозначим i-й разряд разности $x - y$ и "долг" $(i + 1)$-го разряда соответственно через u_i, z_{i+1}.

В табл. 1.8 указаны значения u_i, z_{i+1}, вычисляемые по известным правилам вычита-

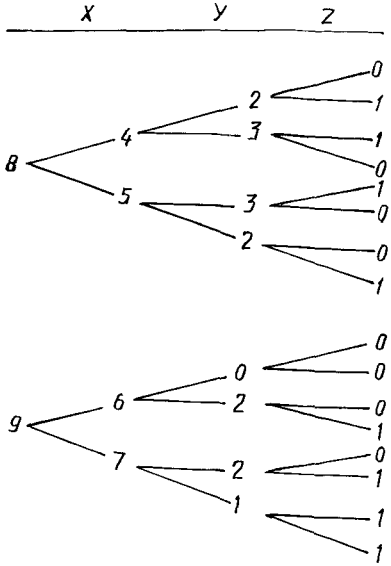


Рис. 1.12

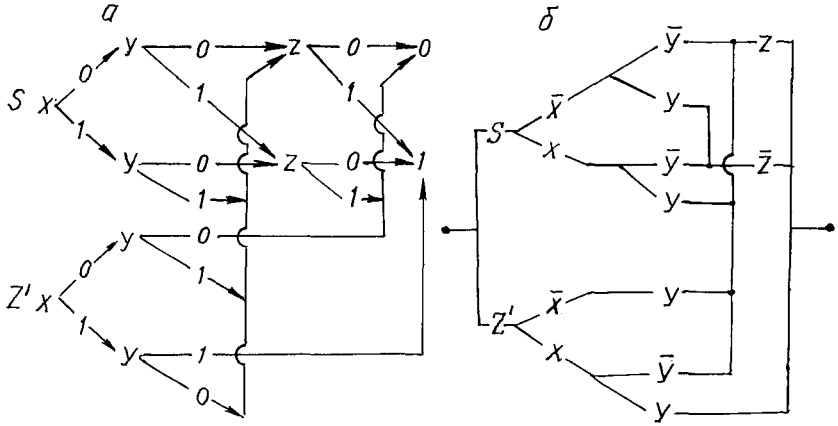


Рис. 1.13

Таблица 1.8

Номер строки	x_i	y_i	z_i	s_i	z_{i+1}	Номер строки	x_i	y_i	z_i	s_i	z_{i+1}
1	0	0	0	0	0	5	1	0	0	1	0
2	0	0	1	1	1	6	1	0	1	0	0
3	0	1	0	1	1	7	1	1	0	0	0
4	0	1	1	0	1	8	1	1	1	1	1

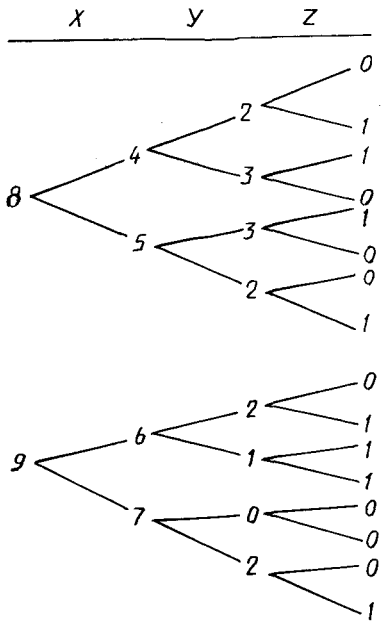


Рис. 1.14

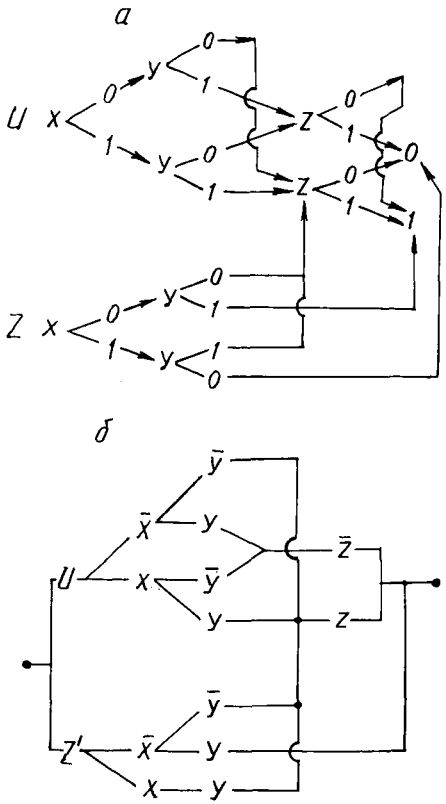


Рис. 1.15

ния для двоичной системы счисления. Например, для второй строки $x_i = 0, y_i = 1$. Занимаем в следующем разряде 10 (т. е. 2) и находим $10 + 0 - 0 - 1 = 1$, т. е. $z_{i+1} = 1, u_i = 1$. Для седьмой строки $1 - 1 - 0 = 0$, т. е. $z_{i+1} = 0, u_i = 0$. Для последней строки $10 + 1 - 1 - 1 = 1$, следовательно, $z_{i+1} = 1, u_i = 1$.

По табл. 1.8 строим каноническую таблицу (рис. 1.14), граф-схему (рис. 1.15, *а*) и контактную схему (рис. 1.15, *б*). Соединив n таких схем, получим n-разрядную схему вычитания натуральных чисел.

Так как канонический метод синтеза состоит из последовательного сравнения чисел, то его сложность оценивается числом таких сравнений. Легко доказать, что $P_n < Q_n^2/3 + Q_n$, где $Q_n = 2^n$ – длина обрабатываемого столбца таблицы от n переменных.

Рассмотрение более сложных граф-схем и уточнение самого этого понятия требуют изложения некоторых сведений о графах.

1.3. ПОНЯТИЕ О ГРАФАХ

Граф $\Gamma(X, U)$ определяется множеством X точек, называемых *вершинами* (или *узлами*) графа, и некоторым множеством U линий, соединяющих все или некоторые пары точек. Элементы из U называются *ребрами* графа. В случае,

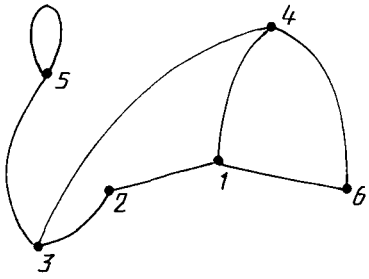


Рис. 1.16

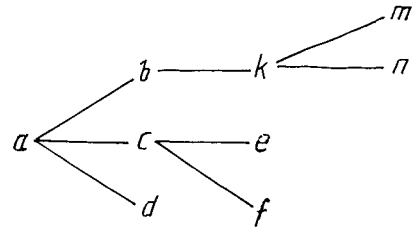


Рис. 1.17

когда множества вершин X и ребер U конечны, граф Γ называют *конечным*.

На рис. 1.16 изображен граф $\Gamma(X, U)$, где X – множество точек $\{1, 2, 3, 4, 5, 6\}$, а U состоит из ребер $(1,2)$, $(2,3)$, $(3,4)$, $(4,6)$, $(6,1)$, $(3,5)$, $(5,5)$. Точки пересечения ребер не считаются вершинами графа.

Не всякий граф может быть изображен на плоскости без пересечения ребер. В качестве примера приведем граф, связанный с решением известной задачи о трех домах и трех колодцах, в которой требуется от каждого дома проложить к каждому колодцу тропинки таким образом, чтобы тропинки не пересекались друг с другом.

Последовательность ребер графа, таких, что конец одного ребра является началом следующего, называется *цепью*. Так, в графе, изображенном на рис. 1.16, цепью является последовательность ребер $(2, 3)$, $(3, 4)$, $(4, 1)$, $(1, 6)$. Вершины $2, 6$ являются соответственно началом и концом цепи. Иначе говоря, цепь соединяет вершины 2 и 6. Цепь называется *простой*, если в ней никакое ребро не встречается дважды, и *элементарной*, если в ней никакая вершина не встречается дважды.

Для любого ребра (вершины), принадлежащего цепи, будем считать, что цепь проходит через это ребро (вершину). Если начало и конец цепи совпадают, она называется *циклом*. В частности, цикл из одной вершины называется *петлей*. В графе, приведенном на рис. 1.16, петлей является ребро $(5, 5)$, одним из циклов – $(1, 4)$, $(4, 6)$, $(6, 1)$.

Граф называется *связным*, если любые две его вершины соединены цепью. Если вершина a соединена цепью с вершиной b, а вершина b в свою очередь соединена некоторой цепью с вершиной c, то говорят, что существует цепь, соединяющая вершины a и c. Поэтому несвязный граф состоит из нескольких отдельных связных графов (связных компонентов).

Связный граф без циклов, содержащий не менее двух вершин, называется *деревом*. Любые две вершины дерева соединены ровно одной цепью, а любая его цепь элементарна. На плоскости дерево Γ наглядно можно изобразить следующим образом. Зафиксируем некоторую вершину a дерева Γ (рис. 1.17) и поместим справа от нее все вершины, соединенные с ней ребром (это вершины первого яруса). Возьмем далее произвольную вершину b первого яруса и расположим справа от нее все вершины (исключая a), соединенные с ней ребром. Получим вершины второго яруса. Поступив так с каждой вершиной второго яруса, мы получим вершины третьего яруса и т. д. В любых двух ярусах будут расположены разные вершины, иначе существовал бы цикл.

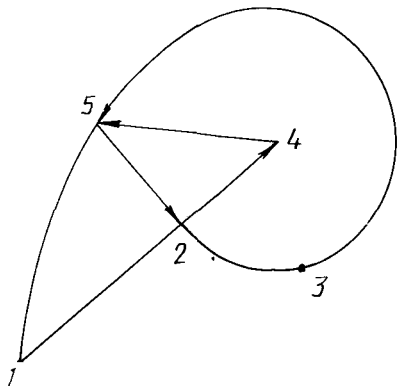


Рис. 1.18

Если порядок следования вершин a, b ребра определен однозначно, т.е. на ребре стрелкой указано направление, то в этом случае ребро (a, b) называют *ориентированным ребром* или *дугой*, выходящей из вершины a и входящей в веришину b, т.е. дугой с началом в a и концом в b.

Граф называется *ориентированным* (или *орграфом*), если он содержит только дуги, и *неориентированным*, если он содержит только ребра. Если в графе есть и ребра и дуги, его называют *смешанным*. На рис. 1.18 изображен смешанный граф.

Путем называется цепь, состоящая только из дуг, таких, что общая вершина двух дуг является обязательно концом одной и началом другой дуги. В графе, изображенном на рис. 1.18, цепь $(2, 4)$, $(4, 5)$ есть путь, а цепи $(4, 5)$, $(5, 3)$ и $(2, 3)$, $(3, 5)$, $(5, 1)$ не являются путями.

Контуром называется путь, начальная и конечная вершины которого совпадают. Очевидно, что последовательность дуг $(2, 4)$, $(4, 5)$, $(5, 2)$ является контуром графа, приведенного на рис. 1.18. Контур элементарен, если все его вершины различны, за исключением начальной и конечной, которые совпадают.

Таким образом, понятия ребра, цепи, цикла отличаются от понятия дуги, пути и контура только тем, что для последних принимается во внимание ориентация.

Вершину орграфа, в которую не входит ни одна дуга, назовем *начальной* или *входом*, вершину, из которой ни одна дуга не выходит, – *конечной* или *выходом*, а остальные вершины – *внутренними*. Вершину b назовем *достижимой* из вершины a, если существует путь, соединяющий a с b.

Все вершины в орграфе без контуров с одним входом достижимы из начальной вершины. В самом деле, если b – внутренняя вершина графа, то существует входящая в нее дуга (a, b). Если вершина a не является начальной, то, повторив относительно нее рассуждения, найдем вершину a_1. Вершина b достижима из вершины a_1. Продолжив выделение вершин, определим путь $(a_k, a_{k-1}), \ldots, (a_1, a), (a, b)$. Процесс выделения вершин завершится, ибо вершин в графе конечное число, и последовательность $a_k, a_{k-1}, \ldots, a_1, a, b$ содержит разные вершины (потому, что в графе нет контуров). Процесс прекращается только в случае совпадения одной из вершин a_k с входом графа.

Вершину орграфа вместе с выходящими из нее дугами назовем *кустом орграфа*, саму вершину – *входом куста* или *логической вершиной*, концы дуг – *выходами*, число дуг – *значностью куста*. Дугу графа, не принадлежащую кусту, назовем *простой дугой*.

Граф-схемой в алфавите B от совокупности X переменных называется орграф, у входа каждого куста которого записано по одной переменной, а на дугах его – различные значения этой переменной; в остальных вершинах орграфа записано по букве из алфавита B. Входы граф-схемы пронумерованы.

1.4. ПРЕОБРАЗОВАНИЯ ЭЛЕМЕНТАРНЫХ ГРАФ-СХЕМ

Элементарной граф-схемой называется граф-схема без контуров, каждая внутренняя вершина которой является логической.

На рис. 1.1, 1.6, *а* , 1.10, 1.11, 1.14, *а*,1.15, *а* изображены элементарные граф-схемы от двоичных переменных: на рис. 1.1 – в алфавите $\{A_1, A_2, A_3\}$, на рис. 1.6, 1.15 – в алфавите $\{0,1\}$, на рис. 1.10, 1.11 – в алфавите $\{1, 2, 3\}$. На рис. 1.19 изображена элементарная граф-схема от переменных $x_1 = \{2, 3, 6\}$, $x_2 = \{0,3\}$, $x_3 = \{1, 5\}$ в алфавите $B = \{7, 9, a, b\}$.

Вычисление по элементарной граф-схеме проведем следующим образом. Пусть задан набор $x_1 = \alpha_1$, $x_2 = \alpha_2$, ..., $x_n = \alpha_n$ и указан номер ν . Вычисления начнем с ν-й начальной вершины. Определим, какая переменная записана в этой вершине, и перейдем к следующей вершине по той дуге начального куста, на которой записано заданное значение переменной. Затем этот процесс повторим относительно вершины, в которую мы переместились, и так до тех пор, пока не попадем в конечную вершину. В конечной вершине прочтем записанную там букву алфавита B и поставим ее в соответствие номеру ν и данному набору значений переменных. Вычисления на этом прекратим.

Граф-схема с k входами определяет k функций, иначе говоря, изображает эти функции. Например, граф-схема, приведенная на рис. 1.19, изображает функцию, заданную табл. 1.9.

Каждая внутренняя вершина является входом соответствующей внутренней граф-схемы. Пусть M – вершина элементарной граф-схемы. Все пути с началом в M образуют элементарную граф-схему Γ_M , определенную вершиной M.

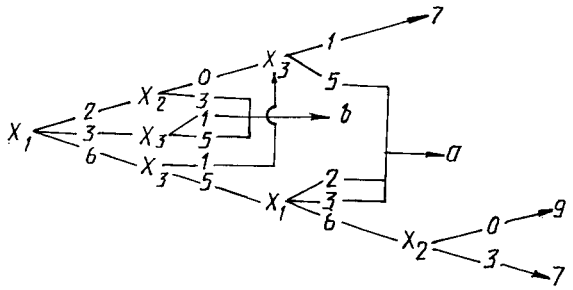


Рис. 1.19

Таблица 1.9

x_1	x_2	x_3	y	x_1	x_2	x_3	y
2	0	1	7	3	3	1	*b*
2	0	5	*a*	3	3	5	*b*
2	3	1	*b*	6	0	1	7
2	3	5	*b*	6	0	5	9
3	0	1	*b*	6	3	1	2
3	0	5	*b*	6	3	5	2

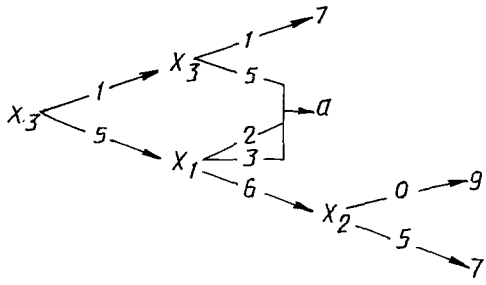


Рис. 1.20

На рис. 1.20 изображена граф-схема, которая определена одной из вершин переменной x_3 граф-схемы, приведенной на рис. 1.19.

Функция, определяемая внутренней граф-схемой, не обязательно зависит от всех переменных. Граф-схему, которая не зависит от переменных, называют *нулевой.*

Вершины M и N называются *эквивалентными,* если определяемые ими граф-схемы Γ_M и Γ_N изображают одну и ту же функцию. Вершины граф-схемы распределяются по непересекающимся классам эквивалентных вершин.

Граф-схема называется *простой,* если каждый ее класс эквивалентных вершин содержит только одну вершину. Две элементарные одновходовые граф-схемы называют *эквивалентными,* если они изображают одну и ту же функцию, т. е. их начальные вершины эквивалентны.

В общем случае говорят, что две элементарные граф-схемы *эквивалентны,* если для каждого входа одной граф-схемы можно указать эквивалентный ему вход другой граф-схемы.

Эквивалентность вершин и граф-схем обозначают символом $\sim$, например $M \sim N$, $\Gamma_M \sim \Gamma_N$.

Путь, соединяющий начальную и конечную вершины граф-схемы, определяет некоторый порядок следования дуг пути, а значит, и соответствующих переменных пути.

Пусть для определенности переменные граф-схемы упорядочены так: $x_1, x_2, ..., x_n$. Порядок следования переменных пути будем называть *согласованным,* если он такой же, как для граф-схемы, т. е. переменная x_k не следует за переменной x_i в случае $k \leqslant i$ (при этом не обязательно, чтобы на пути встречались все переменные).

Граф-схему называют *правильной,* если у нее порядки переменных любого пути согласованы. На рис. 1.6, *а* приведена правильная граф-схема.

Пусть правильная граф-схема Γ изображает функцию $y = f(x_1, x_2, ..., x_n)$. Если путь, соединяющий вход с вершиной M, определяется набором $x_1 = a_1, x_2 = a_2, ..., x_k = a_k$, то граф-схема Γ_M изображает функцию $\varphi_M(x_{k+1}, ..., x_n) = f(a_1, a_2, ..., a_k, x_{k+1}, ..., x_n)$. Это объясняется тем, что вычисление функции φ_M для заданных значений $x_{k+1} = \beta_{k+1}, ..., x_n = \beta_n$ можно провести по граф-схеме Γ, взяв при этом $x_1 = a_1, x_2 = a_2, ..., x_k = a_k$.

Пусть $\Gamma_1 \sim \Gamma_2$ и задан набор $x_1 = a_1, x_2 = a_2, ..., x_k = a_k$. Возьмем две вершины $M \in \Gamma_1$ и $N \in \Gamma_2$, определяемые этим набором. Граф-схемы Γ_M и

Γ_N изображают равные функции $\varphi_N(x_{k+1}, ..., x_n) = \varphi_M(x_{k+1}, ..., x_n)$. Это следует из того, что $f_1(x_1, x_2, ..., x_n) = f_2(x_1, x_2, ..., x_n)$, так как $\Gamma_1 \backsim \Gamma_2$ и

$$\varphi_M(x_{k+1}, ..., x_n) = f_1(a_1, a_2, ..., a_k, x_{k+1}, ..., x_n);$$

$$\varphi_N(x_{k+1}, ..., x_n) = f_2(a_1, a_2, ..., a_k, x_{k+1}, ..., x_n).$$

Таким образом, если правильные граф-схемы Γ_1 и Γ_2 эквивалентны, то для каждой вершины $M \in \Gamma_1$ найдется вершина $N \in \Gamma_2$, такая, что $M \backsim N$. Это значит, что для каждого класса эквивалентных вершин одной граф-схемы существует класс эквивалентных ему вершин второй граф-схемы. Следовательно:

1) у эквивалентных элементарных граф-схем с одинаковым порядком переменных число классов эквивалентных вершин одинаковое;

2) простая элементарная граф-схема имеет минимальное число вершин среди всех эквивалентных ей элементарных граф-схем с тем же порядком переменных.

Согласно каноническому методу синтеза элементарных граф-схем, сначала по столбцу таблицы стандартного вида одной или нескольких функций строят каноническую таблицу. По существу этим уже построена одна из требуемых граф-схем. Однако такая граф-схема содержит обычно большое количество кустов. Поэтому следует перейти к менее сложной граф-схеме, эквивалентной ей. Суть указанных выше правил нумерации заключается в выявлении эквивалентных вершин и переходе к простой граф-схеме для указанного порядка переменных.

Теорема 1.1. *Канонический метод синтеза элементарных правильных граф-схем доставляет простые граф-схемы.*

Доказательство. Возьмем две вершины M, N последнего яруса исходной граф-схемы, т. е. вершины, относящиеся к x_n. Согласно правилам нумерации вершин, равные номера присваиваются им тогда и только тогда, когда соответствующие выходы граф-схемы отмечены одинаковыми буквами алфавита B, т. е. определяемые ими функции равны: $\varphi_M(x_n) = \varphi_N(x_n)$.

Итак, если $M \backsim N$, то вершинам последнего яруса канонической таблицы присваиваются одинаковые номера, и наоборот. Предположим, что это справедливо для столбцов x_i, где $k + 1 \leqslant i \leqslant n$, и докажем, что в этом случае утверждение верно для столбца x_k. Тем самым будет доказана теорема.

Рассмотрим в столбце x_k две вершины M и N, причем вершина M – вход куста $(c; x_k, a, b)$, а вершина N – вход куста $(c_1; x_k, a_1, b_1)$. (На рис.1.21 изображены два куста.) Если $c = c_1$, из правил нумерации вершин следует, что $a = a_1$, $b = b_1$. Согласно предположению относительно вершины столбца x_{k+1}, имеем $\varphi_{M_1} = \varphi_{N_1}$, $\varphi_{M_2} = \varphi_{N_2}$. Отсюда $\varphi_M = \varphi_N$, т. е. $M \backsim N$.

Пусть теперь $M \backsim N$. Следовательно, $\varphi_M = \varphi_N$. Но тогда $\varphi_{M_1} = \varphi_{N_1}$, $\varphi_{M_2} = \varphi_{N_2}$. Согласно предположению относительно столбца x_{k+1}, имеем $a = a_1$, $b = b_1$. Из правил нумерации следует, что $c = c_1$. Объединив затем

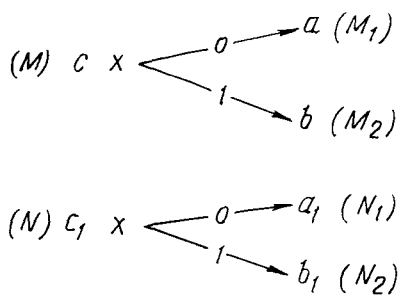


Рис. 1.21

вершины с одним номером в одну вершину, получим простую граф-схему. Теорема доказана.

Сложностью граф-схемы назовем число ее вершин, граф-схему минимальной сложности – *минимальной граф-схемой.*

Даже среди правильных граф-схем с различным порядком переменных простых граф-схем, эквивалентных данной, может быть несколько. Однако минимальная граф-схема будет простой. Если простая граф-схема является единственной, то она будет минимальной.

Теперь оценим число минимальных элементарных граф-схем от n переменных, получаемых каноническим методом, в множестве таких граф-схем, у которых на любом пути нет двух кустов одной переменной.

Пусть булева функция $y = f(x_1, x_2, ..., x_n)$ реализована дерево-схемой D, каждый путь которой содержит n неповторяющихся переменных. Переменные дерево-схемы располагаются в n столбцах, причем в каждом столбце находится не обязательно одна переменная.

Пронумеруем вершины дерево-схемы D, придерживаясь следующих правил:

1) конечные вершины D уже пронумерованы значениями функции;

2) если номера выходных вершин куста составляют пару (a, a) из одинаковых номеров, то номер начальной вершины тоже будет a;

3) если выходные вершины двух кустов одного столбца пронумерованы одинаковыми парами (a, b) и кусты относятся к одной переменной, то их начальные вершины получают одинаковые номера c;

4) во всех других случаях начальные вершины кустов получают различные номера.

Дерево-схема D функции $y = f(x_1, x_2, ..., x_n)$, вершины которой пронумерованы согласно указанным правилам, называется *обобщенной канонической таблицей.*

Обобщенную каноническую таблицу K называют *адекватной* граф-схеме Г, если Г получают из K *"склеиванием"* вершин с одинаковыми номерами. В дальнейшем мы не будем различать граф-схему и адекватную ей обобщенную каноническую таблицу.

Теорема 1.2. *Число $\mu(n)$ булевых функций от n переменных, для которых канонический метод синтеза доставляет минимальные граф-схемы в классе G, удовлетворяет неравенству*

$$\mu(n) \geqslant \prod_{\rho=0}^{n} (c_n^{\rho} + 1),$$

где G – класс граф-схем, у которых на любом пути нет двух кустов одной переменной.

Доказательство. Пусть $s^0(y_0)$ – сложность обобщенной канониче-

ской таблицы $K^0(y_0)$ функции $y_0 = f(x_1, x_2, ..., x_n)$, адекватной минимальной граф-схеме. Возьмем правильную дерево-схему с порядком следования переменных $x_1, x_2, ..., x_n$ и столбец функции, совпадающий со столбцом функции в $K^0(y_0)$. Тем самым определена некоторая функция $y_1 = f_1(x_1, x_2, ..., x_n)$.

Сложность $s(y_1)$ канонической таблицы $K(y_1)$ удовлетворяет неравенству $s(y_1) \leqslant s^0(y_0)$. Это следует из того, что если в i-м столбце таблицы $K^0(y_0)$ имеются одинаковые номера, то на тех же вершинах в $K(y_1)$ будут тоже одинаковые номера. Так как $s^0(y_1) \leqslant s(y_1)$, где $s^0(y_1)$ – сложность обобщенной канонической таблицы, адекватной минимальной граф-схеме для $y_1 = f_1(x_1, x_2, ..., x_n)$, то $s^0(y_1) \leqslant s(y_1) \leqslant s^0(y_0)$. Если $s^0(y_1) = s^0(y_0)$, то $s^0(y_1) = s(y_1)$. Это означает, что канонический метод синтеза доставляет минимальную граф-схему для $y_1 = f(x_1, x_2, ..., x_n)$ в классе G.

Пусть M^0 и M – две одинаково расположенные вершины последнего столбца в обобщенных канонических таблицах $K^0(y_0)$ и $K(y_1)$ соответственно. Пути, соединяющие начальные вершины деревьев с вершинами M^0, M, совпадают, поэтому количества единиц (нулей) в соответствующих строках исходных таблиц для y_0 и y_1 одинаковы. Следовательно, переход от y_0 к y_1 можно истолковать как некоторое преобразование строк в исходных таблицах для y_0 и y_1, при котором вес строки $\sum_1^n x_i$ остается неизменным.

Пусть N_ρ – множество из c_n^ρ строк, вес которых равен ρ. Отнесем к классу $(\alpha_0, \alpha_1, ..., \alpha_n)$, где $0 \leqslant \alpha_\rho \leqslant c_n^\rho$, все функции, которые на α_ρ строках из множества N_ρ принимают значения, равные 1. Из вышесказанного следует, что функции y_0, y_1 принадлежат одному классу. Таких классов будет ровно столько, сколько существует различных наборов $(\alpha_0, \alpha_1, ..., \alpha_n)$, т. е. $\prod_{\rho=0}^{n} (c_n^\rho + 1)$.

Для данной функции $y_0 = f(x_1, x_2, ..., x_n)$ построим указанным выше способом функцию y_1, затем по y_1 – функцию y_2 и т. д., т. е. построим конечную последовательность $y_0, y_1, ..., y_{\nu-1}, y_\nu$, такую, что $s^0(y_\nu) = s^0(y_{\nu-1})$. (Равенство обязательно осуществится, ибо сложность – натуральное число и $s^0(y_i) \leqslant s^0(y_{i-1})$.) Отсюда следует $s(y_\nu) = s^0(y_\nu)$, что означает минимальность граф-схемы функции y_ν, доставляемой каноническим методом. Так как все функции $y_0, y_1, ..., y_\nu$ принадлежат одному классу, то в каждом таком классе будет по крайней мере одна функция, для которой канонический метод доставляет минимальную граф-схему. Поэтому $\mu(n) \geqslant \prod_{\rho=0}^{n} (c_n^\rho + 1)$, что и требовалось доказать.

Следствие. *Канонический метод синтеза в классе G доставляет для симметрических функций минимальные граф-схемы.*

В самом деле, классы, определяемые наборами $(\alpha_0, \alpha_1, ..., \alpha_n)$, в которых $\alpha_\rho = 0$ или $\alpha_\rho = c_n^\rho$, содержат по одной функции, притом симметрической.

1.5. ПОЧТИ ЭЛЕМЕНТАРНЫЕ ГРАФ-СХЕМЫ

Граф-схему без контуров называют *почти элементарной.*

Элементарная граф-схема является частным случаем почти элементарной граф-схемы, когда ее внутренние вершины – только логические вершины. Остальные вершины в граф-схемах называют *арифметическими.*

На рис. 1.22 изображена почти элементарная граф-схема в алфавите $B = \{a, b, c\}$ от переменных x_1, x_2, x_3.

Любой путь, соединяющий вход и выход почти элементарной граф-схемы, определяет последовательность букв из алфавита B.

Вычисления по почти элементарной граф-схеме проводятся так же, как и в случае элементарных граф-схем.

Пусть заданы номер входа ν граф-схемы и набор значений $x_1 = \alpha_1$, $x_2 = \alpha_2$, ..., $x_n = \alpha_n$. Набор $(\alpha_1, \alpha_2, ..., \alpha_n)$ определяет путь от ν-го входа граф-схемы до некоторого ее выхода. На этом пути расположена определенная последовательность букв из алфавита B. Эту последовательность считаем результатом вычислений по почти элементарной граф-схеме для входа ν и набора $(\alpha_1, \alpha_2, ..., \alpha_n)$.

Таким образом, почти элементарная граф-схема с k выходами также определяет k функций. В отличие от элементарной граф-схемы каждое значение этих функций представляет собой последовательность букв из B.

Для вычисления по граф-схеме, приведенной на рис. 1.22, зададим $\nu = 1$ и набор (0, 1, 0), т. е. $x_1 = 0, x_2 = 1, x_3 = 0$. Вычисления начнем с первого входа. Согласно набору (0, 1, 0), исполнитель перемещается по пути к конечной вершине и прочитывает буквы из алфавита B в таком порядке: b, c, a, c. Результатом вычислений является выходная последовательность *bcac.* Итак, изображаемая почти элементарной граф-схемой функция для $\nu = 1$, $x_1 = 0$, $x_2 = 1$, $x_3 = 0$ равна последовательности *bcac.* Для $\nu = 1$ и набора (1, 0, 0) получим последовательность *bacac,* а для $\nu = 2$ и (1, 0, 0) – последовательность *aa*.

В дальнейшем будем рассматривать граф-схемы, у которых на выходах куста нет одинаковых букв. (В противном случае такую букву сотрем и запишем ее в новой вершине перед входом куста. Результат вычисления при этом не меняется.)

Канонический метод синтеза почти элементарных граф-схем так же, как и в случае синтеза элементарных граф-схем, заключается в выявлении классов

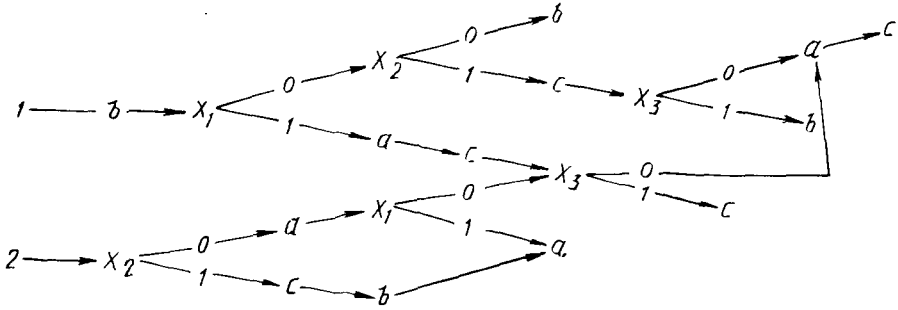


Рис. 1.22

эквивалентных вершин в исходной граф-схеме (чаще всего дереве) и последующем переходе к простой граф-схеме.

Метод синтеза почти элементарных граф-схем отличается от вышеизложенного метода синтеза элементарных граф-схем только некоторыми деталями. Он состоит также из двух этапов: на первом этапе получают каноническую таблицу по исходной граф-схеме, на втором по канонической таблице строят граф-схему.

Прежде чем нумеровать вершины исходной граф-схемы, следует в каждом кусте, на выходах которого записана одна и та же буква, стереть эту букву и записать ее перед входом куста.

Правила построения канонической таблицы для почти элементарной граф-схемы

1. В качестве номеров вершин заданной граф-схемы выбирают последовательные натуральные числа.
2. Номер проставляют между вершиной и стрелками, входящими в вершину дуг.
3. Нумерацию начинают с выходов граф-схемы.
4. Двум выходам почти элементарной граф-схемы присваивают одинаковые номера тогда и только тогда, когда они отмечены одинаковыми буквами из алфавита B.

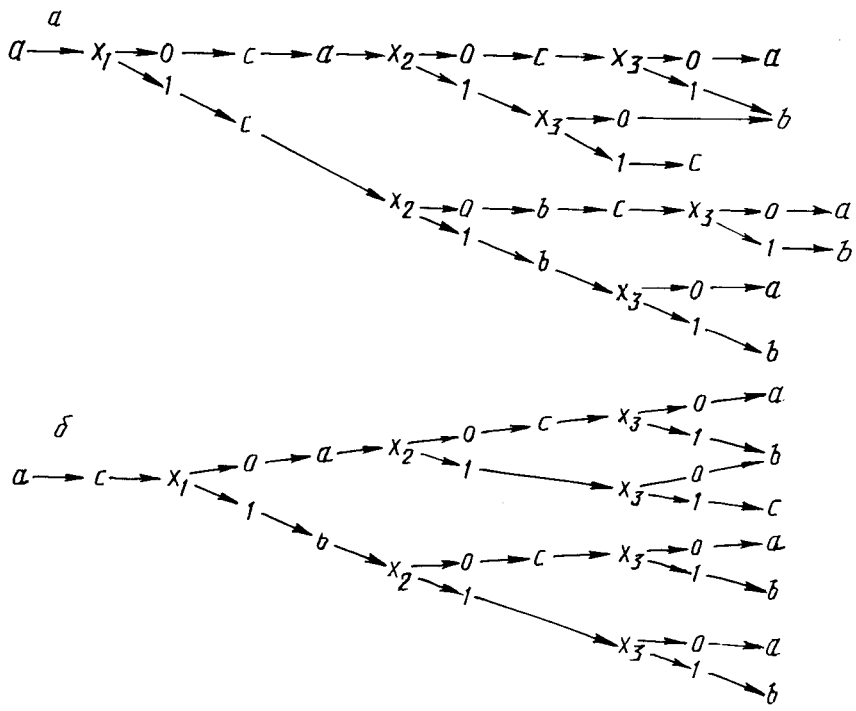


Рис. 1.23

5. Нумерация входа куста проводится по номерам его выходов.

6. Входам двух кустов присваивают одинаковые номера тогда и только тогда, когда они отмечены одной переменной и соответствующие выходы кустов также имеют одинаковые номера.

7. Арифметическим вершинам присваивают одинаковые номера тогда и только тогда, когда они отмечены одной буквой и конечные вершины их простых дуг имеют одинаковые номера.

Обычно конечные вершины граф-схемы не нумеруют, считая номерами записанные в них буквы.

Граф-схему с вершинами, пронумерованными согласно указанным правилам, назовем *канонической таблицей.*

Наиболее просто нумеруются правильные граф-схемы.

На рис. 1.23, *а* дана исходная правильная граф-схема, на рис. 1.23, *б* приведена она же после перезаписи одинаковых букв у выходов куста, а на рис. 1.24, *а* изображена ее каноническая таблица.

Правила перехода от канонической таблицы к почти элементарной граф-схеме

1. Анализ канонической таблицы проводится от ее входов к выходам.

2. Если несколько вершин имеют один и тот же номер, то отмечают одну вершину, а неотмеченные вершины соединяют дугами с отмеченной.

3. Дуги, исходящие из неотмеченных вершин, исключают.

4. Если при этом образовались дополнительные входы граф-схемы, то их исключают вместе с исходящими из них дугами (п. 4 повторяют до тех пор, пока имеются лишние входы).

5. Нуль-куст заменяют его выходом.

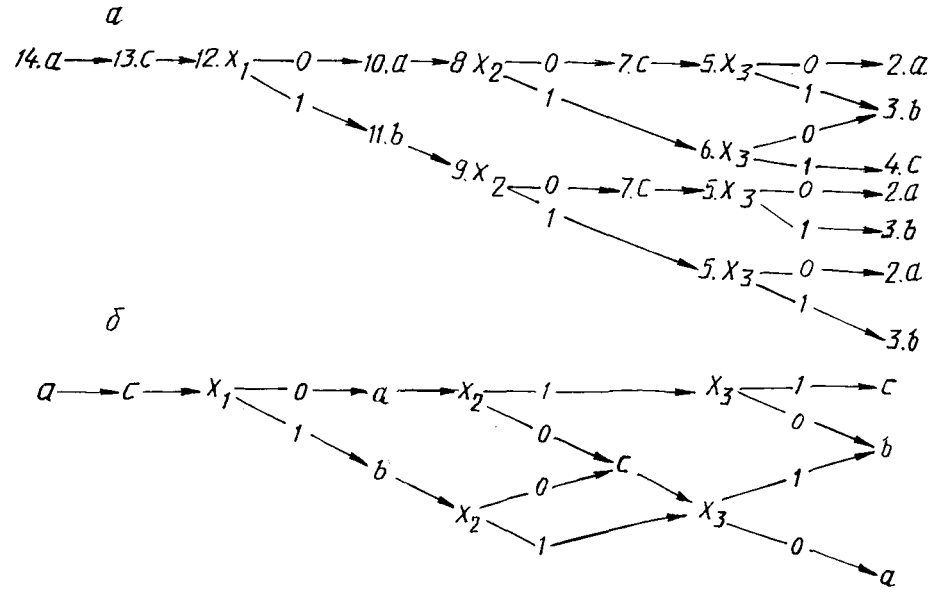


Рис. 1.24

На рис. 1.24, *б* изображена почти элементарная граф-схема, полученная по приведенной на рис. 1.24, *а* канонической таблице, согласно перечисленным правилам.

Пусть M – вершина почти элементарной граф-схемы. Обозначим граф-схему, определяемую вершиной M, через Γ_M, а изображаемую функцию – через φ_M. Отметим еще раз, что значениями φ_M будут последовательности букв из алфавита B, в частности из одной буквы. Так же как и в случае элементарных граф-схем, через функции φ_M определяются эквивалентные вершины, граф-схемы, а затем простые граф-схемы.

Пусть Γ_1 и Γ_2 – правильные почти элементарные граф-схемы с одинаковым порядком переменных и $\Gamma_1 \backsim \Gamma_2$. Перед входами начальных кустов граф-схем записаны соответственно две последовательности L_1 и L_2 букв алфавита B. Если $L_1 \neq L_2$, то одна последовательность является начальной частью другой. Предположим, что L_1 – часть L_2 и после L_1 следует буква a. Эта буква находится на всех путях в Γ_1 после L_1, ибо $\Gamma_1 \backsim \Gamma_2$. Пусть среди всех вершин с буквой a вершина M_1 соединяется с входом наибольшим числом дуг. Тогда на втором выходе M_2 куста тоже записана a. Получили противоречие. Итак, $L_1 = L_2$. Следовательно, входы начальных кустов эквивалентны. Аналогично доказывается, что арифметическая и логическая вершины не могут быть эквивалентны.

Набор значений $(a_1, a_2, ..., a_k)$ определяет в Γ_1 и Γ_2 две логические вершины M_1 и M_2, на путях к которым расположены последовательности букв L_1 и L_2. Аналогично доказывается, что $L_1 = L_2$. Отсюда $M_1 \backsim M_2$.

Используя этот результат, доказательство минимальности правильных граф-схем, полученных каноническим методом, проводят аналогично случаю элементарных граф-схем.

1.6. ГРАФ-СХЕМЫ С ОБРАТНЫМИ СВЯЗЯМИ

Граф-схемой с обратными связями от переменных $x_1, x_2, ..., x_n$ в алфавите B назовем почти элементарную граф-схему, удовлетворяющую следующим условиям:

1) алфавит B содержит еще "пустую" букву, обозначаемую через z, и букву s, которую назовем "останов";

2) в конечных вершинах граф-схемы записаны только буквы z или s, причем z – всегда с меткой;

3) для каждой группы вершин, отмеченных буквой z с одинаковой меткой, имеется только одна не конечная вершина с этой же меткой, но без буквы z;

4) других вершин с метками нет.

Буква z является только носителем метки; часто z не пишут, а просто заключают метки в кружки.

Конечную вершину граф-схемы без метки назовем *выходом граф-схемы с обратными связями.*

На рис. 1.25 изображена граф-схема с обратными связями. Вычисления по такой граф-схеме осуществляются так же, как по почти элементарной граф-

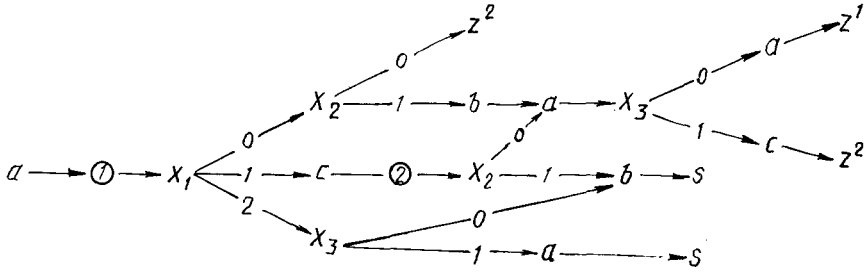


Рис. 1.25

схеме, но со следующим уточнением: если исполнитель обозревает выход граф-схемы, то вычисления прекращаются, в противном случае вычисления продолжаются, начиная с той же единственной не конечной вершины, в которой записана та же метка, что и в конечной вершине, но для очередного набора $x_1 = \beta_1$, $x_2 = \beta_2, \ldots, x_n = \beta_n$, если набор задан.

Результатом вычислений по граф-схеме с обратными связями будет конечная или бесконечная последовательность букв из алфавита B. Для примера проведем вычисления по граф-схеме, изображенной на рис. 1.25, для определенной входной последовательности наборов (x_1, x_2, x_3).

Пусть задана входная последовательность (1, 0, 1), (0, 0, 0), (2, 1, 0), (0, 1, 1). Первый член последовательности (1, 0, 1) доставит последовательность *acac*. Так как процесс завершается в конечной вершине с меткой *2*, дальнейшие вычисления начинают с внутренней вершины с меткой *2*. Следующий набор (0, 0, 0) доставляет последовательность *aa*, и вычисления продолжают с вершины с меткой *1*, т.е. с вершины куста x_1. (Заметим, что значение $x_1 = 0$ не понадобилось.) Набор (2, 1, 0) доставит последовательность *bs*, и вычисления завершатся, ибо исполнитель обозревает выход граф-схемы. Результатом вычислений по входной последовательности (1, 0, 1), (0, 0, 0), (2, 1, 0), (0, 1, 1) будет выходная последовательность *acacaabs*. Для входной последовательности (2, 0, 1), (1, 1, 0) выходной последовательностью является *aas*.

Таким образом, вычисления прекращаются либо при отсутствии очередного набора переменных, либо в выходе граф-схемы.

Вершины, обозначенные одинаковыми метками, считаются при вычислениях одной вершиной. Поэтому граф-схема с обратными связями является по существу граф-схемой с контурами, которая к тому же содержит дополнительную информацию, необходимую для вычисления по граф-схеме.

Для определения вычислительного процесса по граф-схеме с контурами или соединению таких граф-схем требуется задать еще подмножество вершин, называемых *опорными*. Подмножество обязательно содержит все вершины соединения в граф-схеме. При этом должно выполняться такое условие: если пронумеровать опорные вершины и затем их разъединить, исходная граф-схема преобразуется в граф-схему с обратными связями, в которой метками являются номера опорных вершин. Если исполнитель, обозревая опорную вершину, получает значение очередного набора переменных, то он перемещается к следующей опорной вершине и выписывает последовательность букв на этом

пути. В дальнейшем под граф-схемой будем подразумевать граф-схему с обратными связями.

Две вершины M и N граф-схемы называются *эквивалентными* ($M \backsim N$), если результаты определяемых ими вычислительных процессов совпадают для любой входной последовательности наборов.

Две граф-схемы Γ_1 и Γ_2 называются *эквивалентными* ($\Gamma_1 \backsim \Gamma_2$), если для каждого входа одной из них существует эквивалентный ему вход другой.

Из определения эквивалентности вершин следует: 1) $M \backsim M$; 2) если $M_1 \backsim M_2$, то $M_2 \backsim M_1$; 3) если $M_1 \backsim M_2$ и $M_2 \backsim M_3$, то $M_1 \backsim M_3$. Поэтому вершины граф-схемы с обратными связями распределяются по классам эквивалентных вершин.

Граф-схема является простой, если каждый класс ее эквивалентных вершин содержит только по одной вершине.

Синтез граф-схем заключается в том, чтобы по заданной граф-схеме (часто дерево-схеме) найти эквивалентную ей граф-схему с меньшим числом вершин.

Самый простой метод синтеза состоит в том, что граф-схему преобразовывают как почти элементарную граф-схему, считая букву z с данной меткой новой буквой.

Перейдем к методу синтеза, который доставляет правильные граф-схемы с наименьшим числом вершин для данного порядка переменных.

Синтез граф-схем с обратными связями состоит из двух этапов: на первом путем нумерации вершин исходной граф-схемы получают каноническую таблицу, на втором по канонической таблице строят простую граф-схему с обратными связями. В качестве номеров вершин выбирают обычно натуральные числа, но допустимы и любые другие символы.

Вначале следует стереть одинаковые буквы без меток у выходов куста и записать букву перед входом куста.

Правила построения канонической таблицы для граф-схемы с обратными связями

1. При нумерации вершин метки не учитываются.
2. Номерами конечных вершин являются сами буквы z и s.
3. Входам двух кустов одинаковые номера присваиваются только тогда, когда они отмечены одной переменной и соответствующие выходы кустов имеют одинаковые номера. Если номера выходов куста равны, то этот номер присваивается входу куста.
4. Двум арифметическим вершинам одинаковые номера присваиваются только тогда, когда они отмечены одной буквой и конечные вершины инцидентных им простых дуг имеют равные номера.
5. Конечной вершине с меткой присваивается новое значение, совпадающее со значением вершины, следующей за не конечной вершиной с такой же меткой.

После выполнения п. 5 могут образоваться *противоречивые* вершины. Это означает, что входы кустов имеют одинаковые номера, но есть соответствующие выходы кустов с разными номерами (аналогично для арифметических вершин).

6. Если противоречивых вершин нет, нумерация завершена. В противном случае следует перейти к выполнению п. 3, т. е. провести очередную нумерацию вершин граф-схемы по новым значениям конечных вершин.

Повторная нумерация упрощается тем, что новые номера присваивают только противоречивым вершинам. Поэтому, просматривая столбец вершин, нужно данную вершину сравнивать только с теми, которые имеют такой же прежний номер, что и данная.

Переход от канонической таблицы к граф-схеме с обратными связями осуществляется так же, как и в случае почти элементарной граф-схемы (см. § 1.5).

Пример 1.7. На рис. 1.26 изображена дерево-схема с обратными связями в алфавите $\{a, b, z, s\}$ от переменных x_1, x_2, x_3. Построить эквивалентную ей простую граф-схему.

Р е ш е н и е. Первая нумерация по дерево-схеме изображена на рис. 1.27, *а*. Перенесем новые номера в вершины с метками (см. рис. 1.27, *б*). Противоречивых вершин не оказалось. Следовательно, получена каноническая таблица (рис. 1.27, *б*). Построенная по ней граф-схема приведена на рис. 1.28.

Пусть несколько граф-схем с обратными связями от переменных x_1, x_2, ..., x_n соединены последовательно. Это означает, что все (или часть) выходы одних граф-схем соединены со входами других граф-схем. Вычисления по такой граф-схеме проводятся так же, как и в случае граф-схемы с обратными связями, только при переходе к каждой следующей граф-схеме используется очередной набор значений переменных. Поэтому в граф-схеме должны быть указаны соединения составляющих ее граф-схем.

На рис. 1.29 изображена граф-схема от одной переменной x в алфавите $\{a, b\}$. Она образована последовательным соединением пяти граф-схем. Очевидно, что множество опорных вершин состоит только из вершин соединения. Для входной последовательности $x = 1, x = 0, x = 0, x = 1$ имеем выходную последовательность *baab*.

Для определения эквивалентных вершин в граф-схеме с k вершинами соединений ее представляют в виде граф-схемы с обратными связями.

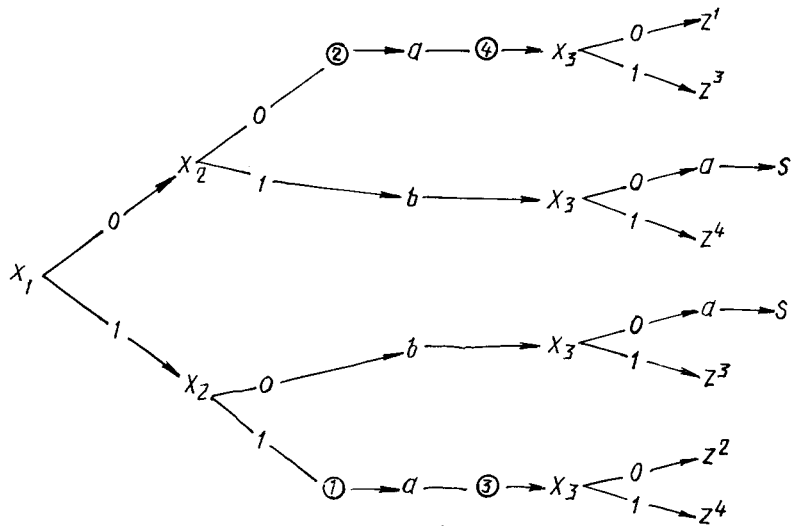


Рис. 1.26

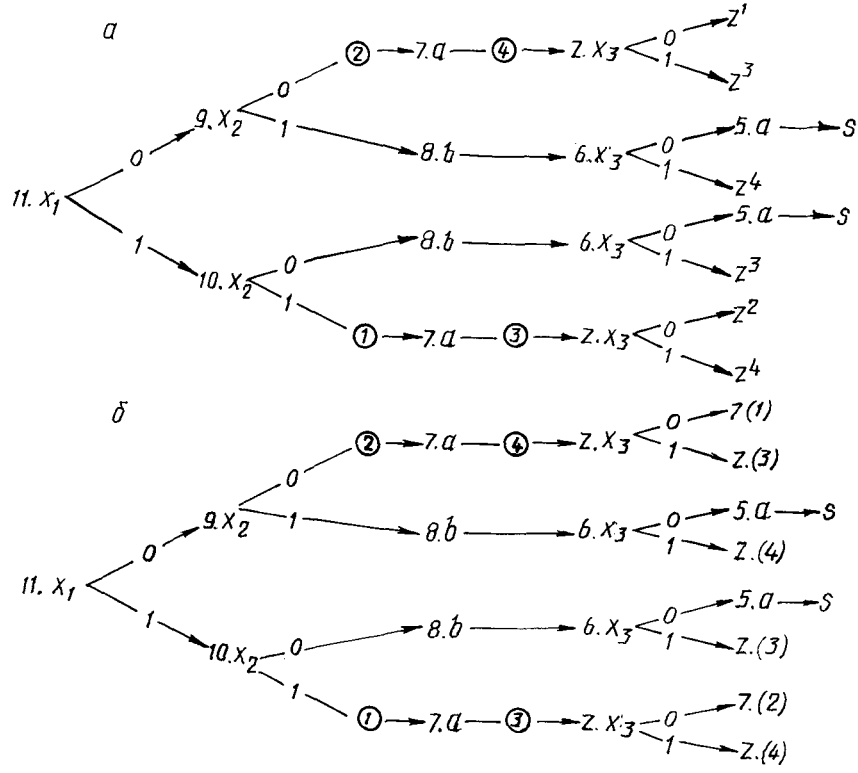


Рис. 1.27

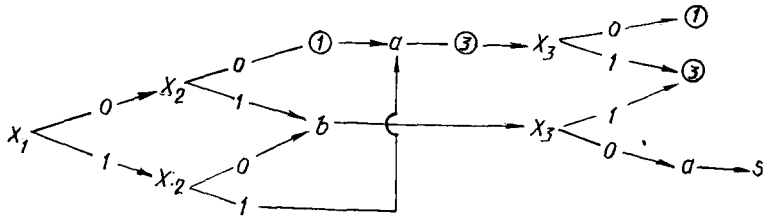


Рис. 1.28

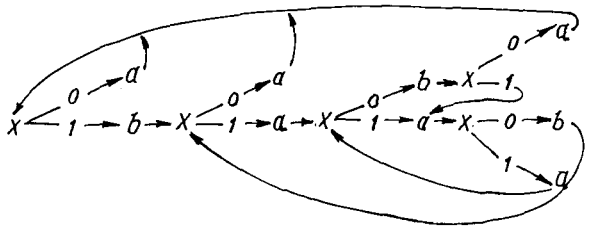


Рис. 1.29

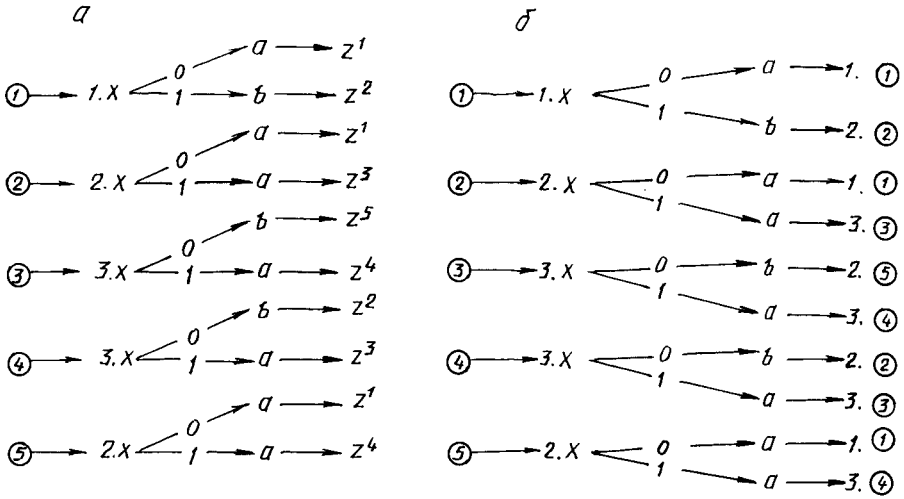


Рис. 1.30

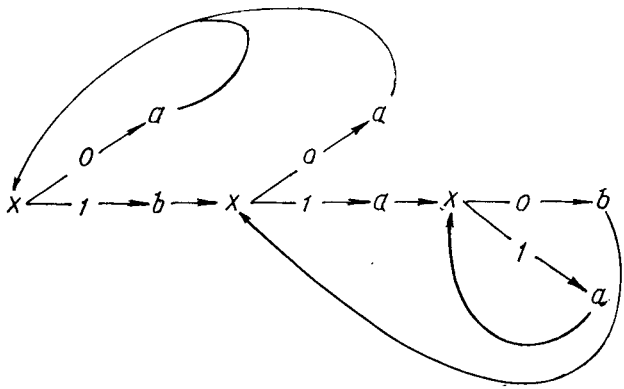


Рис. 1.31

Пример 1.8. Построить простую граф-схему, эквивалентную граф-схеме, изображенной на рис. 1.29.

Р е ш е н и е. В заданной граф-схеме имеются вершины соединений, поэтому изобразим ее в виде граф-схемы с пятью входами (рис. 1.30, *а*). Так как при нумерации метки не учитываются, то парам (a, b), (a, a), (b, a) присвоим номера 1, 2, 3. После корректировки номеров в конечных вершинах получим каноническую таблицу, поскольку противоречивых вершин нет (рис. 1.30, *б*). Как видно, имеются две пары эквивалентных входов. Последовательно соединим три граф-схемы согласно их номерам. Получим искомую граф-схему (рис. 1.31), эквивалентную граф-схеме, приведенной на рис. 1.29.

В основе канонического метода синтеза граф-схем с обратными связями лежит нумерация вершин исходной граф-схемы (чаще всего дерево-схемы) с целью выявления классов эквивалентных вершин. Перейдем к обоснованию канонического метода.

Две вершины M_1 и M_2 двух (или одной) граф-схем называют *v-эквивалентными*, если вычисления, начинающиеся с этих вершин, для произвольной

последовательности наборов $(a_1, a_2, ..., a_n)$ длиной $k \leqslant \nu$ доставляют одинаковые выходные последовательности букв алфавита B.

Вершины M_1 и M_2 называют *ν-отличимыми*, если существует хотя бы одна последовательность наборов переменных длиной ν, такая, что вычисления, начинающиеся с этих вершин, доставляют различные выходные последовательности букв алфавита B.

Если вершины M_1 и M_2 $(\nu + 1)$-эквивалентны, то они ν-эквивалентны; если вершины M_1 и M_2 ν-отличимы, то они $(\nu + 1)$-отличимы и, более того, не эквивалентны.

Отметим такие свойства эквивалентных и отличимых вершин:

1) если одноэквивалентные вершины M_1, M_2 при любом наборе переменных переводятся в ν-эквивалентные вершины, то M_1, M_2 – $(\nu + 1)$-эквивалентные вершины;

2) если вершины M_1, M_2 при некотором наборе переменных переводятся в ν-отличимые вершины, то M_1, M_2 – $(\nu + 1)$-отличимые вершины;

3) если вершины M_1 и M_2 – ν-эквивалентны для любого ν, то $M_1 \backsim M_2$.

Далее мы будем рассматривать правильные граф-схемы.

Обозначим через Γ^ν исходную граф-схему после ν-й нумерации. Так как при нумерации метки не учитываются, то Γ^ν считается почти элементарной граф-схемой.

Лемма 1.1. *После ν-й нумерации граф-схемы одинаковые номера будут присвоены ν-эквивалентным вершинам, разные номера – ν-отличимым вершинам.*

Доказательство. Воспользуемся индукцией по ν. При $\nu = 1$, т.е. для Γ^1, утверждение леммы очевидно (см. доказательство теоремы 1.1 в применении к почти элементраным граф-схемам). Предположим, что лемма справедлива для $\nu = m - 1$ и докажем, что она выполняется для $\nu = m$.

Пусть при нумерации граф-схемы Γ^{m-1} вершинам M_1, M_2 присвоены одинаковые номера. Следовательно, вершины M_1, M_2 эквивалентны в Γ^m, т. е. в исходной граф-схеме Γ они одноэквивалентны и при любом наборе переменных переходят в две конечные вершины соответственно N_1 и N_2 с равными номерами, полученными после $(m-1)$-й перенумерации. По предположению, N_1, N_2 являются $(m-1)$-эквивалентными вершинами. Отсюда следует, что M_1, M_2 – m-эквивалентные вершины.

Пусть при нумерации граф-схемы Γ^{m-1} вершинам M_1, M_2 присвоены различные номера. Значит, существует набор переменных, такой, что на двух соответствующих путях с началом в вершинах M_1 и M_2 в почти элементарной граф-схеме Γ^m будут различные последовательности букв из алфавита B или различные номера в конечных вершинах N_1, N_2. В первом случае M_1 и M_2 – одноотличимые вершины, во втором N_1, N_2, по предположению, – $(m-1)$-отличимые вершины). Следовательно, M_1, M_2 – m-отличимые вершины. Лемма доказана.

Обозначим через s_ν число различных номеров не конечных вершин с метками в Γ^ν. Согласно лемме 1.1, вершины с разными номерами в Γ^ν будут иметь различные номера и в $\Gamma^{\nu+1}$. Отсюда $s_{\nu+1} \geqslant s_\nu$. Равенство $s_{\nu+1} = s_\nu$ означает, что в Γ^ν и $\Gamma^{\nu+1}$ нумерация не конечных вершин с метками совпадает, следовательно, совпадает и нумерация конечных и не конечных вершин в $\Gamma^{\nu+1}$

Поэтому $\Gamma^{\nu+1}$ – каноническая таблица, и $\Gamma^n = \Gamma^{\nu+1}$ для всех $n > \nu$.

Лемма 1.2. *Если в канонической таблице разные номера присвоены вершинам с метками, то эти вершины $(k-1)$-отличимы, в другом случае они k-отличимы (k – число различных меток).*

Доказательство. Пусть нумерация проведена m раз и в итоге получена последовательность $s_1 < s_2 < \ldots < s_m \leqslant k$, причем $s_{m+1} = s_m$. Отсюда $s_1 + (m-1) \leqslant k$ или $m \leqslant k + 1 - s1$. Согласно лемме 1.1, в канонической таблице Γ^{m+1} разные номера присвоены $(m+1)$-отличимым вершинам. Вершины с метками будут m-отличимы, ибо уже после m-й нумерации получены все различные номера для этих вершин.

При $s_1 > 1$ имеем $m \leqslant k-1$. Если $s_1 = 1$, то очевидно, что $s_2 = s_1$ и Γ_1 – каноническая таблица. Лемма доказана.

Следствие. *Число нумераций, необходимое для построения канонической таблицы, не превосходит числа различных меток.*

Так как соединение нескольких граф-схем можно рассматривать как одну, лемма 1.2 верна и для вершин (в частности, для входов) нескольких граф-схем.

Теорема 1.3. *В канонической таблице равные номера присвоены только эквивалентным вершинам.*

Доказательство. Так как $\Gamma^n = \Gamma^{m+1}$ для любого $n > m$ (поскольку Γ^{m+1} – каноническая таблица), то вершины с одинаковыми номерами в Γ^{m+1} будут n-эквивалентны для любого n. Следовательно, они эквивалентны. Теорема доказана.

Так как при использовании канонического метода синтеза в граф-схеме остается по одному номеру из группы одинаковых номеров, построенная граф-схема будет простой.

Теорема 1.4. *Если две простые граф-схемы с одинаковым порядком переменных эквивалентны, то любой вершине одной граф-схемы соответствует эквивалентная ей вершина другой граф-схемы.*

Доказательство. Рассмотрим сначала одновходовые граф-схемы. Перед входами N_1, N_2 начальных кустов имеются соответственно две последовательности L_1 и L_2 букв алфавита B. Если $L_1 \neq L_2$, то одна последовательность является начальной частью другой. Предположим, что L_1 – часть L_2 и после L_1 (вершины N) следует буква a. Эта буква находится на всех путях в Γ_1 после L_1, ибо $\Gamma_1 \sim \Gamma_2$. Среди всех вершин с буквой a после L_1 найдем вершину M_1, которая соединяется с входом наибольшим числом дуг. Тогда на втором выходе M_2 куста тоже записана буква a. Так как $N \sim M_1$ и $N \sim M_2$, то $M_1 \sim M_2$. Получили противоречие, поскольку граф-схемы – простые. Следовательно, $L_1 = L_2$, и входы N_1, N_2 начальных кустов эквивалентны.

Набор значений $(a_1, a_2, \ldots, a_k)$ определяет в Γ_1 и Γ_2 две логические вершины N_1, N_2, на путях к которым записаны две последовательности L_1 и L_2. Аналогично доказывается, что $L_1 = L_2$ и $N_1 \sim N_2$. Очевидно, что арифметические вершины, предшествующие N_1 и N_2, также соответственно эквивалентны.

Если граф-схемы многовходовые, то для вершины M_1, определяемой входом N_1 и набором $(a_1, a_2, \ldots, a_k)$ в Γ_1, найдем в Γ_2 соответствующую вершину M_2, определяемую входом N_2 $(N_2 \sim N_1)$ и набором $(a_1, a_2, \ldots, a_k)$. Оче-

видно, что $M_1 \backsim M_2$. Теорема доказана.

С л е д с т в и е. *Канонический метод синтеза доставляет граф-схему с минимальным числом вершин в множестве правильных граф-схем с одинаковым порядком переменных.*

Простые граф-схемы от одной переменной всегда будут минимальными.

З а м е ч а н и е. Теоремы 1.3, 1.4 остаются справедливыми для граф-схем без буквы s, если в конечных вершинах, кроме z, возможны еще и другие буквы из алфавита B. Но последовательности L_1, L_2 букв из B равны при наличии дополнительного условия: последние буквы в L_1 и L_2 либо должны быть обе у выходов граф-схемы, либо нет.

1.7. РЕАЛИЗАЦИЯ ЭЛЕМЕНТАРНЫХ ГРАФ-СХЕМ

Напомним, что при реализации элементарной граф-схемы ее кусты одной переменной реализуются переключательными элементами одного реле, дуги – проводниками, а каждая буква алфавита реализуется исполнительным устройством, точнее, его пусковым элементом. В переключательной схеме исполнительные устройства обозначены теми же буквами.

Если по граф-схеме перемещается исполнитель алгоритма, то по контактной схеме "перемещается" ток. Но при изменении направления тока (если поменять местами полюса) работа электрической схемы не изменится. Однако куст граф-схемы $(c;\ x, a, b)$ в этом случае замещается элементом ψ_0 с тремя входами a, b, x и выходом c, реализующим функцию $c = a\bar{x} + bx$. Вход x в отличие от входов a, b называют *информационным*. В контактных схемах

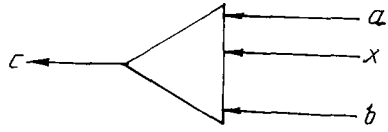


Рис. 1.32

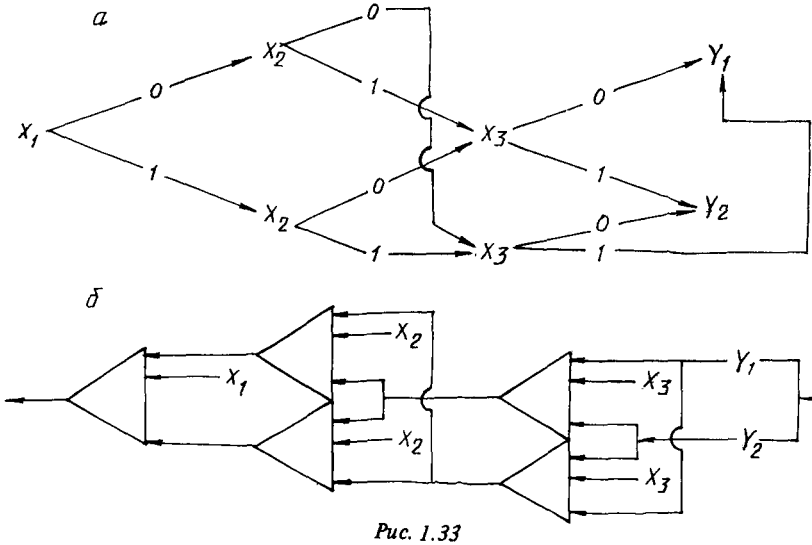


Рис. 1.33

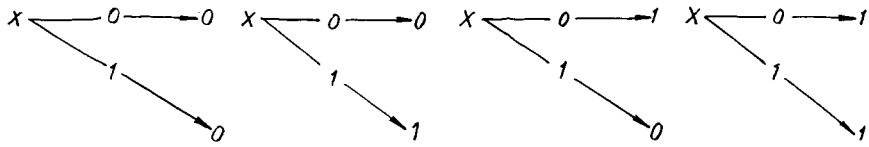


Рис. 1.34

функция $c = a\bar{x} + bx$ реализуется тем же переключателем с двумя контактами $\bar{x}$, x. На рис. 1.32 схематически изображен элемент $\psi_0 = a\bar{x} + bx$. Элемент ψ_0 может быть выполнен в виде контактного переключателя, электронного узла и т. п.

На рис. 1.33, *б* показана реализация элементарной граф-схемы, изображенной на рис. 1.33, *а*, на элементах ψ_0.

Интерес представляет синтез схем на других трехвходовых элементах $\psi(x; a, b)$. Можно, конечно, каждый куст канонической таблицы реализовать несколькими такими элементами, но это потребует большего количества элементов, чем синтез на $\psi_0(x; a, b)$.

Рассмотрим такой метод, при котором каждый куст модифицированной канонической таблицы также замещается одним элементом.

Пусть элемент $\psi(x; a, b)$ задан табл. 1.10. По условию, этим элементом в зависимости от значений входов a, b можно реализовать любую из четырех функций одной переменной x, или, иначе, любой из четырех кустов одной переменной (рис. 1.34). Чтобы элемент $\psi(x; a, b)$ удовлетворял таким условиям, в табл. 1.10 пары (y_{0v}, y_{1v}), соответствующие (a, b), должны быть различны и принимать все четыре значения: (0,0), (0,1), (1,0), (1,1). Следовательно, таких элементов будет 24 – по числу перестановок из четырех.

Таблица 1.10

a	b	x	y	a	b	x	y
0	0	0	y_{01}	1	0	0	y_{03}
0	0	1	y_{11}	1	0	1	y_{13}
0	1	0	y_{02}	1	1	0	y_{04}
0	1	1	y_{12}	1	1	1	y_{14}

Если считать одинаковыми элементы, различающиеся лишь обозначениями входов a, b, то элементов будет 12 (табл. 1.11).

Таблица 1.11

a	b	x	ψ_0	ψ_1	ψ_2	ψ_3	ψ_4	ψ_5	ψ_6	ψ_7	ψ_8	ψ_9	ψ_{10}	ψ_{11}
0	0	0	0	0	1	1	0	0	0	0	1	1	1	1
0	0	1	0	1	0	1	0	0	1	1	0	0	1	1
0	1	0	0	0	0	0	0	1	0	1	0	1	0	0
0	1	1	1	0	0	1	1	1	0	0	0	1	0	1
1	0	0	1	1	1	1	1	0	1	1	0	0	0	0
1	0	1	0	1	1	0	1	1	0	1	1	1	1	0
1	1	0	1	1	0	0	1	1	1	0	1	0	1	1
1	1	1	1	0	1	0	0	0	1	0	1	0	0	0

Чтобы реализовать, например, третий куст элементом ψ_1, следует положить $a = 1, b = 1$, ибо в этом случае $\psi_1(0; 1, 1) = 1, \psi_1(1; 1, 1) = 0$.

Пусть задана каноническая таблица функции $f(x_1, x_2, ..., x_n)$. Каждый куст таблицы реализует функцию $\psi_0(x; a, b) = a\bar{x} \vee bx$. Если заменить элементом $\psi(x; a, b)$ только начальный куст таблицы, модифицированная таблица будет реализовывать функцию, отличную от $f(x_1, x_2, ..., x_n)$.

Чтобы модифицированная каноническая таблица реализовывала функцию $f(x_1, x_2, ..., x_n)$, нужно изменить столбец y, т. е. значения, которые подаются на входы канонической таблицы. Выясним, как это сделать.

На входы a, b начального элемента $\psi(x; a, b)$ подаются одновременно те значения y_0, y_1 столбца y, которые соответствуют двум наборам: $(0, a_2, ..., a_n)$ и $(1, a_2, ..., a_n)$. Поэтому эти два значения y_0, y_1 следует заменить значениями a, b для элемента ψ_2 из табл. 1.11.

Рассмотрим для примера реализацию функции $f(x_1, x_2, x_3)$, определяемой табл. 1.12, переключательной схемой, в которой куст x_1 замещается элементом ψ_2, каждый куст x_2 — элементом ψ_6, каждый куст x_3 — элементом ψ_9.

Таблица 1.12

x_1	x_2	x_3	y	x_1	x_2	x_3	y
0	0	0	0	1	0	0	0
0	0	1	1	1	0	1	0
0	1	0	1	1	1	0	1
0	1	1	1	1	1	1	0

Две половинки столбца y, соответствующие $x_1 = 0, x_1 = 1$, расположим рядом. Каждую пару в строке заменим соответствующей парой a, b, определяемой функцией ψ_2:

$$\begin{array}{l|l} 00 & 01 \\ 10 & 00 \\ 11 & 10 \\ 10 & 00. \end{array}$$

Первый столбец относится к $x_1 = 0$, второй — к $x_1 = 1$. Таким образом, получим столбец значений входов в случае, когда только куст x_1 замещен элементом ψ_2, а остальные кусты замещены элементом ψ_0.

Теперь будем рассматривать в качестве исходных две канонические таблицы для $x_1 = 0$ и $x_1 = 1$. Построим для каждого из двух столбцов новые столбцы. Для элемента ψ_6 имеем:

$$\begin{array}{l|l} 0110 & 0010 \\ 0000 & 0101. \end{array}$$

Преобразовав строку для элемента ψ_9, получим необходимые значения входов модифицированной канонической таблицы:

$$00011001 \mid 11100010.$$

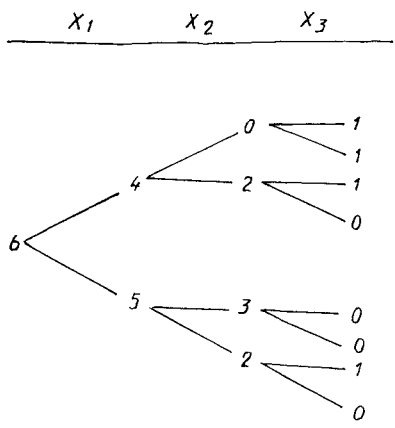


Рис. 1.35

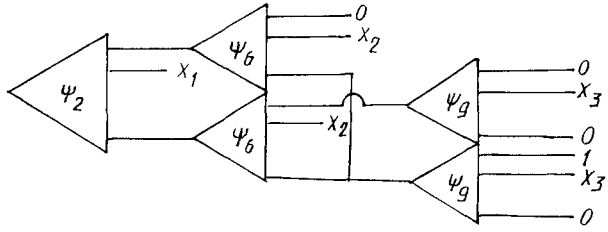


Рис. 1.36

Для найденного столбца построим модифицированную каноническую таблицу по известным правилам: разные пары получают разные номера, одинаковые пары – одинаковые номера. Кроме того, в столбце x_3 использовано такое свойство: $\psi_9(x;\ 1,1) = 0$.

На рис. 1.35 изображена каноническая таблица, построенная по табл. 1.12.

Каждый куст модифицированной канонической таблицы замещаем своим элементом. Логическая схема приведена на рис. 1.36.

Заметим, что процедуру определения значений входов модифицированной канонической таблицы можно упростить.

2. ЭФФЕКТИВНАЯ ВЫЧИСЛИМОСТЬ

2.1. ПОНЯТИЕ АЛГОРИТМА

Понятие алгоритма* принадлежит к числу основных понятий современной математики. Под *алгоритмом* понимают точное предписание, которое задает вычислительный процесс, начинающийся с произвольного исходного данного из некоторой совокупности возможных для этого процесса данных и направленный на получение полностью определяемого этими исходными данными результата.

Простейшими алгоритмами являются, например, известные из школы правила выполнения четырех арифметическох действий в десятичной системе счисления.

Алгоритм может быть задан таблицей с инструкцией по пользованию ею. Такие алгоритмы называют *табличными.* Алгоритмы, в которых основную роль играют арифметические действия, принято называть *численными.* Частным случаем численных алгоритмов являются алгоритмы, задаваемые формулами.

Рассмотрим примеры алгоритмов.

1. *Алгоритм Евклида для нахождения наибольшего общего делителя (НОД) двух натуральных чисел.* Приведем список указаний для этого алгоритма.

1. Сравнить заданные (полученные) два числа. Перейти к указанию 2.

2. Если эти числа равны, то каждое из них даст искомый результат. Процесс вычисления остановить. В противном случае перейти к указанию 3.

3. Вычесть из большего числа меньшее и оставить два числа: вычитаемое и разность. Перейти к указанию 4.

4. Перейти к указанию 1.

Исходными данными алгоритма является любая пара отличных от нуля натуральных чисел, результатом — одно отличное от нуля натуральное число.

Алгоритм Евклида является численным алгоритмом.

В дальнейшем в описании алгоритма будем опускать слова "перейти к указанию".

Обратимся снова к алгоритму Евклида. Если $a > b$, то, согласно п. 3, переходим от пары a, b к паре $a-b$, b. Если $a-b > b$, т. е. $a > 2b$, то переходим к

---

* Термин "алгоритм" происходит от имени узбекского математика аль-Хорезми, который в IX в. предложил правила арифметических действий. В средневековой Европе "алгоризмом" назывались десятичная позиционная система счисления и искусство счета в ней, поскольку именно благодаря трактату аль-Хорезми Европа познакомилась с позиционной системой. С XVIII в. вместо слова "алгоризм" стали употреблять "алгоризмус", в дальнейшем – "алгорифм". В последнее время повсеместно употребляют термин "алгоритм".

паре $a - 2b$, b, и так до тех пор, пока не получим разность $a - pb < b$. Следовательно, весь этот процесс можно заменить вычислением остатка от деления a на b. Поэтому алгоритм Евклида можно сформулировать в следующем виде.

1. Сравнить числа a и b. Пусть $a \geqslant b$.

2. Найти остаток r от деления a на b. Если $r = 0$, то перейти к п. 3, в противном случае – к п. 1, но уже для чисел $a = r$ и b.

3. НОД $= b$.

Теперь рассмотрим алгоритм Евклида для многочленов. Напомним, что НОД двух многочленов $f(x)$ и $g(x)$ называется такой многочлен $d(x)$, который, во-первых, делит $f(x)$ и $g(x)$ без остатка, и, во-вторых, обладает таким свойством, что всякий многочлен, делящий $f(x)$ и $g(x)$, делит и $d(x)$. НОД двух многочленов определяется однозначно с точностью до числового множителя.

Обозначим степень многочлена $f(x)$ через $c(f)$. Пусть $c(f) \geqslant c(g)$. Разделив $f(x)$ на $g(x)$, получим $f(x) = p(x)g(x) + r(x)$, где $c(r) < c(g)$ или $r(x) = 0$. Очевидно, что НОД $(f, g) =$ НОД (g, r).

Приведем описание алгоритма Евклида для многочленов $f(x)$ и $g(x)$.

1. Сравнить степени многочленов. Пусть $c(f) \geqslant c(g)$.

2. Найти остаток $r(x)$ от деления $f(x)$ на $g(x)$.

3. Если $r(x) = 0$, то НОД $(f, g) = g(x)$. В противном случае перейти к п. 2 для многочленов $g(x)$, $r(x)$.

2. *Алгоритм уточнения корня уравнения делением отрезка пополам.* Пусть неизвестный корень c уравнения $f(x) = 0$ отделен, т. е. найден отрезок $[a, b]$, которому принадлежит только этот корень. В качестве приближенного значения корня можно взять любую точку $x_0 \in [a, b]$. Очевидно, что погрешность $|c - x_0| < |b - a|$. Отсюда следует, что для уточнения корня с заданной точностью h нужно сужать отрезок $[a, b]$ до тех пор, пока его длина не станет меньше h.

Будем сужать отрезок, деля его пополам и выбирая ту половину, в которой находится корень. Признаком принадлежности корня отрезку является различие знаков функции $f(x)$ на концах этого отрезка.

Условимся обозначать конец отрезка буквой a, если $f(x)$ отрицательна на этом конце, и буквой b, если $f(x)$ положительна.

На отрезке $[a, b]$ имеется единственный корень уравнения $f(x) = 0$. Требуется найти приближенное значение x_0 корня с точностью h.

Приведем описание этого алгоритма.

1. Найти $c = (a + b)/2$, где точка c – середина отрезка $[a, b]$.

2. Вычислить $f(c)$.

3. Если $f(c) = 0$, то перейти к п. 8, в противном случае – к п. 4.

4. Вычислить $|b - c|$.

5. Если $|b - c| < h$, то перейти к п. 8, в противном случае – к п. 6.

6. Если $f(c) < 0$, то $a := c$; если $f(c) > 0$, то $b := c$.

7. Перейти к п. 1.

8. Положить $x_0 = c$.

9. Прекратить вычисления.

Замечание. В п. 6 использована операция $a := c$ ($b := c$), которая называется *операцией присваивания*. Запись $a := c$ означает, что следует вычислить c и полученное значение присвоить переменной a.

Кроме арифметических операций, вычислительный процесс содержит в п. 3, 5, 6 *операцию проверки условия:* в зависимости от выполнения ("Да") или невыполнения ("Нет") выбирается одно из двух возможных продолжений вычислений. Такой алгоритм называют *разветвляющимся.*

Операцию, результатом которой является число, будем называть *арифметической операцией* или *арифметическим оператором.* Операцию, которая может изменить последовательное выполнение указаний алгоритма, называют *операцией перехода* или *оператором перехода.*

Операторы перехода подразделяются на два типа: операторы *условного и безусловного переходов.* В операторах условного перехода всегда присутствует логическое условие типа "если..., то...". Ответом на вопрос, выполняется ли условие, является либо "Да", либо "Нет". Оператор условного перехода называют еще *логическим оператором.*

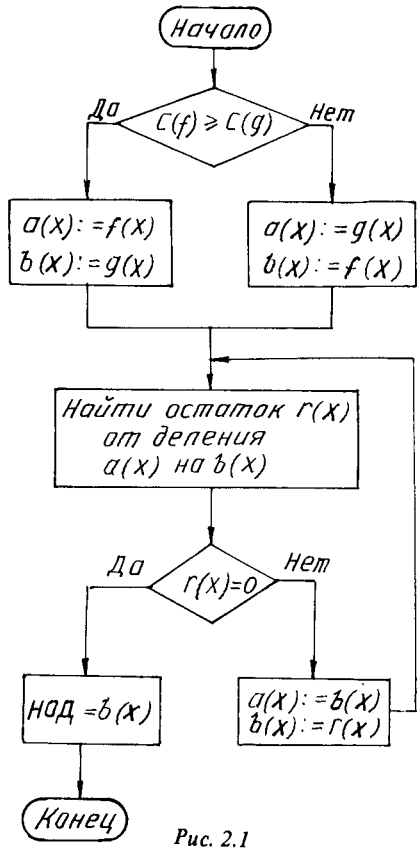


Рис. 2.1

В описании алгоритма п. 3, 5, 6 представляют собой операторы условного перехода, п. 7 алгоритма указывает на безусловный переход: п. 1–7 алгоритма образуют цикл, они повторяются до тех пор, пока не будет выполнено условие п. 3 или 5.

Если алгоритм содержит большое количество операторов условного и безусловного переходов, то сразу представить его в виде перечня указаний довольно трудно. Поэтому обычно сначала алгоритм изображают в виде схемы.

Под *схемой алгоритма* понимают рисунок, состоящий из блоков (изображений операторов) и линий со стрелками, которые указывают связи между блоками. Арифметический оператор заключается в прямоугольный блок, а логический оператор (условный переход) – в ромб. Из арифметического блока выходит одна стрелка, из логического – две: с надписями "Да" и "Нет". Начало или окончание выполнения алгоритма обозначают овалом.

На рис. 2.1, 2.2 изображены схемы алгоритма Евклида для многочленов и алгоритма уточнения корня уравнения делением отрезка пополам.

3. *Алгоритм построения симметрических слов.* Имеются буквы n типов: $a_1, a_2, ..., a_n$.

1. Написать слово P_1, состоящее из одной буквы первого типа, т. е. $P_1 = a_1$.

Пусть построено m-е слово P_m.

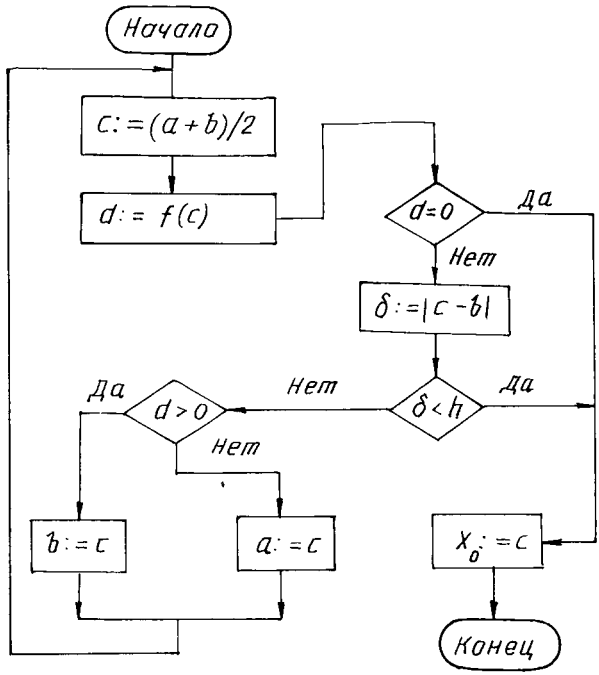


Рис. 2.2

2. Составить $P_{m+1} = P_m a_{m+1} P_m$, т. е. между двумя словами P_m вписать букву $(m+1)$-го типа: a_{m+1} .

3. После получения слова P_n процесс построения слов прекратить.

4. Искомое слово $P = P_n$.

На рис. 2.3 изображена схема описанного алгоритма.

Например, имеются четыре типа букв: *a*, *b*, *c*, *d* . Составить искомое слово с помощью приведенного алгоритма.

Согласно п. 1, имеем $P_1 = a$. На основании п. 2 получаем:

$$P_2 = aba\ ; \quad P_3 = abacaba\ ;$$
$$P_4 = abacabadabacaba\ .$$

Согласно п. 4, $P = abacabadabacaba$.

Исходными данными алгоритма является любая упорядоченная совокупность символов (букв).

Найдем длину l_n слова P_n , т. е. число букв в этом слове. Очевидно, что $l_{k+1} = 2l_k + 1$. Запишем последнее равенство в виде $l_{k+1} + 1 = 2(l_k + 1)$. Числа $l_k + 1$ образуют геометрическую прогрессию со знаменателем 2 и первым членом $l_1 + 1 = 2$. Поэтому $l_n + 1 = 2^n$, откуда $l_n = 2^n - 1$.

4. *Численный алгоритм определения типа буквы в симметрических сло-*

вах. Пусть задан номер m. Требуется найти тип буквы, порядковый номер которой в слове равен m.

1. Найти максимальную степень r числа 2, содержащегося в числе m, т.е. представить m в виде $m = 2^r(2t + 1)$, где второй множитель $2t + 1$ есть нечетное число.

2. Вычислить $s = r + 1$.

3. Тип буквы равен s.

Докажем истинность алгоритма. Очевидно, что при заданном m нужно рассматривать такие слова P_k, для которых $l_k \geqslant m$, т.е. $2^k - 1 \geqslant m$. При этом во всех словах P_k на m-м месте будет одна и та же буква. Поэтому достаточно найти букву в слове с наименьшим k.

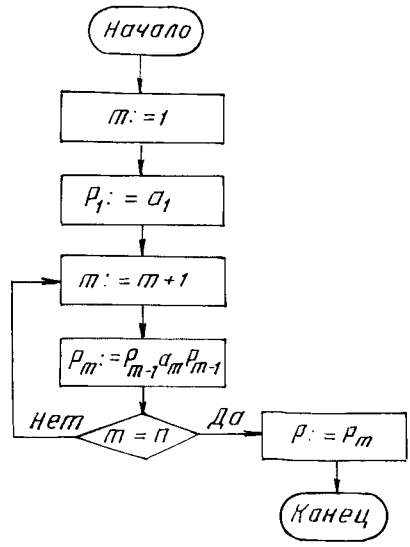


Рис. 2.3

Доказательство проведем по индукции. Пусть $m = 1$. Первой буквой во всех словах будет a_1, т.е. $s = 1$. Найдем s по алгоритму. Представим $m = 1$ формулой $1 = 2^0(2 \cdot 0 + 1)$. Как видим, $r = 0$, следовательно, $s = 1$. Итак, для $m = 1$ алгоритм выполняется.

Предположим, что алгоритм справедлив для $m \leqslant p - 1$, и докажем, что он выполняется для $m = p$.

Пусть $2^{k-1} - 1 < p \leqslant 2^k - 1$. Это означает, что искомая буква содержится в слове P_k и не содержится в слове P_{k-1}. Если $p = 2^{k-1}$, то искомая буква записана сразу за словом P_{k-1}, ибо его длина $l_{k-1} = 2^{k-1} - 1$. Согласно алгоритму построения слов, искомой буквой будет a_k. Так как $p = 2^{k-1}$, то $r = k - 1$ и $s = k$. Следовательно, численный алгоритм для такого p выполняется.

Если $p > 2^{k-1}$, то в слове P_k исключим начальное слово P_{k-1} и букву a_k. Получим, что в оставшейся части, совпадающей с P_{k-1}, на $(p - 2^{k-1})$-м месте находится искомая буква. Так как для слов, длина которых не превосходит $p - 1$, алгоритм по предположению выполняется, то имеет место представление $p - 2^{k-1} = 2^r(2t + 1)$. Тип искомой буквы $s = r + 1$. Но $p = 2^{k-1} + 2^r(2t + 1) = 2^r(2^{k-1-r} + 2t + 1) = 2^r(2t_1 + 1)$. Следовательно, r – максимальная степень числа 2 и для p, а значит, алгоритм для $m = p$ выполняется.

Из вышеизложенного следует: *буква a_1 в слове встречается на всех нечетных местах.*

С помощью последнего алгоритма по произвольному номеру m находят букву, поэтому он решает более общую задачу, чем алгоритм построения симметрических слов.

В частности, чтобы построить симметрическое слово с помощью численного алгоритма, следует задать номера 1, 2, ... и выписать для них в последовательности получаемые типы букв согласно алгоритму.

5. *Алгоритм построения связной пары симметрических слов.* Имеются буквы n типов с двумя признаками: a_1^0, a_1^1, a_2^0, a_2^1, ..., a_n^0, a_n^1 (признаки 0, 1 обозначены верхними индексами).

1. Составить первую пару слов: $P_1^0 = a_1^0$, $P_1^1 = a_1^1$.

Пусть построена m-я пара симметрических слов: P_m^0, P_m^1.

2. Построить слова $P_{m+1}^0 = P_m^1 a_{m+1}^0 P_m^1$, $P_{m+1}^1 = P_m^0 a_{m+1}^1 P_m^0$.

3. После получения P_n^0, P_n^1 построение связных пар симметрических слов прекратить.

4. Искомая пара: $P^0 = P_n^0$, $P^1 = P_n^1$.

Например, даны буквы, упорядоченные по трем типам и двум признакам: a^0, a^1, b^0, b^1, c^0, c^1. Построить пару симметрических слов. Имеем:

$$P_1^0 = a^0,\ P_2^0 = a^1 b^0 a^1,\ P_3^0 = a^0 b^1 a^0 c^0 a^0 b^1 a^0;$$

$$P_1^1 = a^1,\ P_2^1 = a^0 b^1 a^0,\ P_3^1 = a^1 b^0 a^1 c^1 a^1 b^0 a^1.$$

Ниже будет использована запись $a \equiv b (\mathrm{mod}\, c)$, означающая, что у натуральных чисел a, b одинаковые остатки от деления на c, т. е. $(a-b)$ делится на c. Запись $\delta :\equiv a (\mathrm{mod}\, c)$ означает, что δ присвоен остаток от деления числа a на c.

6. *Численный алгоритм определения буквы в паре симметрических слов.* Пусть задан номер m. Требуется в слове P_l^j, где $j = \{0, 1\}$, определить тип s и признак δ буквы с этим номером.

1. Представить $m = 2^r (2t + 1)$.

2. Найти $s = r + 1$.

3. Найти $\delta :\equiv s + l + j \pmod 2$.

4. Тип буквы равен s, признак $-\delta$.

Очевидно, что буквы данного типа s имеют в слове P_l^j равные признаки δ.

Докажем истинность алгоритма. Если убрать верхние индексы, слова P_l^0, P_l^1 обратятся в симметрическое слово P_l. Поэтому по определению типа буквы п. 1, 2 выполняются.

Для слов $P_1^0 = a_1^0$, $P_1^1 = a_1^1$ п. 3 алгоритма выполняется, ибо соответственно $\delta :\equiv 1 + 1 + 0$, $\delta :\equiv 1 + 1 + 1 \pmod 2$, т. е. $\delta = 0$, $\delta = 1$.

Пусть для слов P_ν^j при $\nu \leqslant k$ п. 3 выполняется. Докажем, что тогда он справедлив для $\nu = k + 1$.

В самом деле, $P_{k+1}^j = P_k^{\bar{j}} a_{k+1}^j P_k^{\bar{j}}$, где $\bar{j} = 0$, если $j = 1$, и наоборот.

Если $m = 2^k$, то на m-м месте в слове P_{k+1}^j будет буква a_{k+1}^j, т. е. $\delta = j$. Проверим выполнение п. 3: $\delta :\equiv (k + 1) + (k + 1) + j \pmod 2$, ибо $s = k + 1$, $l = k + 1$. Отсюда $\delta := j$, значит, п. 3 выполняется.

Если $m \neq 2^k$, то буква имеет тот же признак, что и в слове $P_k^{\bar{j}}$, ибо $P_k^{\bar{j}}$ входит в слово P_{k+1}^j. По предположению, для $P_k^{\bar{j}}$ п. 3 выполняется, и $\delta :\equiv s + k + \bar{j} \pmod 2$. Отсюда $\delta :\equiv s + (k + 1) + j \pmod 2$, что и требовалось доказать.

7. *Игра "Ханойская башня".*

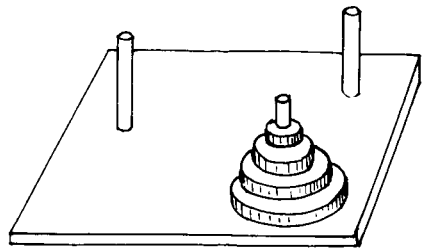


Рис. 2.4

Эту игру придумал французский математик Эдуард Люка. Она состоит в следующем. На доске расположены по кругу три колышка (рис. 2.4). На первый колышек нанизаны n колец (в виде пирамиды). Необходимо перенести пирамиду на другой колышек за наименьшее число ходов. За один ход разрешается переносить только одно кольцо, причем нельзя нанизывать большее кольцо на меньшее.

Найдем алгоритм игры, т. е. перечень указаний, следуя которым, будем выполнять требуемые ходы. Пронумеруем кольца, начиная с номера 1 для меньшего кольца. Ход по перемещению кольца a обозначим через a^j, где индекс $j = \{0, 1\}$ означает кратчайшее перемещение против $(j = 0)$ или по ходу $(j = 1)$ часовой стрелки. Перемещение пирамиды из k колец на ближайший стержень обозначим через P_k^j против хода часовой стрелки при $j = 0$ и по ходу часовой стрелки при $j = 1$. Под P_k^j будем понимать слово из букв, обозначающих ходы. Решением будет P_n^0 или P_n^1 (как условимся заранее). Так что индекс j известен для P_n^j.

Очевидно, что условие, которое должно выполняться, будет следующим: чтобы перенести в направлении j пирамиду из $k + 1$ колец, нужно переместить пирамиду из k колец в направлении $\bar{j}$. Освободившееся $(k + 1)$-е кольцо переместить в направлении j, а затем пирамиду из k колец перенести в направлении $\bar{j}$. Таким образом, $P_{k+1}^j = P_k^j a_{k+1}^j P_k^{\bar{j}}$, причем $P_1^j = a_1^j$. Следовательно, слова P_k^j удовлетворяют тому же алгоритму построения, что и пары симметрических слов.

Так как решением в случае n колец является слово P_n^j, можно воспользоваться следующим правилом: если требуется выполнить m-й ход, нужно переместить кольцо s в направлении δ, где $s = r + 1, \delta: \equiv s + n + j \pmod 2$, причем $m = 2^r(2t + 1)$, n – число заданных колец, j – заданное перемещение пирамиды. Число ходов $l_n = 2^n - 1$.

Очевидно, что *данное кольцо s всегда перемещается в одном направлении. Наименьшее кольцо $(s = 1)$ перемещается в направлении $\delta = j$, если число колец n нечетное, и в противоположном направлении $\delta = \bar{j}$, если число колец n четное.*

Так, например, в случае $n = 8$ потребуется $2^8 - 1 = 255$ ходов. Найдем 12-й ход при начальном $j = 0$. Представим число 12 в виде $12 = 2^2 \cdot 3$ и получим $r = 2$. Следовательно, $s = 2 + 1 = 3$. На 12-м ходу перемещают третье кольцо. Найдем направление перемещения: $\delta: \equiv 2 + 8 + 0 \pmod 2$, т. е. $\delta = 0$. На 12-м ходу третье кольцо перемещают против хода часовой стрелки.

Полученный алгоритм в определенном смысле избыточен, так как, пользуясь им, можно играть "вслепую". Но если при очередном ходе воспользоваться информацией о расположении верхних колец на стержнях, можно упростить алгоритм игры. Покажем это.

1. Первым ходом перемещают первое кольцо в направлении j при нечетном числе колец и в противоположном направлении при четном числе колец.

2. Направление первого хода фиксируют для первого кольца при любом ходе.

3. Перемещают первое кольцо через ход.

Пусть верхними кольцами на трех стержнях будут 1, a , b , причем $1 < a < b$.

4. Если при очередном ходе перемещают не 1, то остается только одна возможность: переместить меньшее кольцо a на большее кольцо b .

Сформулированный алгоритм игры с учетом информации о трех верхних кольцах можно использовать как алгоритм построения симметрических слов.

Создание алгоритма для определенного класса задач является достаточно сложным и требует высокой математической квалификации и изобретательности. Но когда такой алгоритм построен, процесс решения любой задачи данного класса может осуществить исполнитель, способный выполнить только элементарные операции, составляющие вычислительный процесс. Таким исполнителем может быть и машина.

Описанное ранее понятие алгоритма не является его определением, так как в свою очередь оно содержит нечеткие и весьма сложные понятия, такие, как "вычислительный процесс", "точное предписание". В результате уточнения понятия алгоритма или, вернее, понятия эффективной вычислимости, были разработаны три алгоритмические системы: машины Тьюринга (1936), нормальные алгоритмы Маркова (1947), рекурсивные функции (1934–1936).

В данной главе будут рассмотрены все три классические алгоритмические системы и, как вспомогательная, машина с неограниченными регистрами, впервые предложенная в 1963 г. Дж. Шепердсоном и Г.Стерджисом.

2.2. МАШИНА С НЕОГРАНИЧЕННЫМИ РЕГИСТРАМИ

Машина с неограниченными регистрами (МНР) состоит из исполнителя и бесконечного числа пронумерованных регистров R_1 , R_2 , ..., R_n , ... , каждый из которых в любой момент времени содержит некоторое натуральное число. Номер регистра называют его *адресом*. Число, содержащееся в R_n , обозначают a_n . Исполнитель (человек или устройство) может изменять содержимое регистров в ответ на некоторую команду. Алфавит операторов (команд) содержит только пять команд: $B = \{Z, S, T, P, St\}$, где Z, S – одноместные операторы (одноадресные команды); T, P – двухместные операторы (двухадресные команды). Характер операторов определен табл. 2.1.

Работа МНР задается граф-схемой в алфавите B . В каждой вершине куста граф-схемы записан оператор условного перехода P , а в арифметических вершинах – операторы Z , S , T и St , причем St – только в конечных вершинах граф-схемы. Все операторы, кроме St , снабжены адресами n , m .

Команда P(m, n) указывает исполнителю переходы в граф-схеме по соот-

Таблица 2.1

Тип команды	Команда	Действия исполнителя
Обнуление	$Z(n)$	Замена a_n на 0 ($a_n := 0$)
Прибавление единицы	$S(n)$	Прибавление 1 к a_n ($a_n := a_n + 1$)
Переадресация	$T(m,n)$	Замена a_n на a_m ($a_n := a_m$)
Условный переход	$P(m, n)$	Если $a_n = a_m$, то присвоение $P = 1$, иначе $P = 0$
Останов	St	Прекращение вычисления

ветствующей дуге, команды Z , S , T указывают характер преобразования содержимого регистра.

Последовательность чисел a_1 , a_2 , ..., записанных соответственно в регистрах R_1 , R_2 , ..., назовем *конфигурацией.*

Будем рассматривать начальные конфигурации, у которых только конечное число регистров содержит натуральные числа, отличные от нуля; остальные регистры – "пустые", т. е. их содержимое равно нулю.

По начальной конфигурации и заданной граф-схеме МНР производит вычисления. Это означает, что исполнитель, двигаясь по граф-схеме начиная от входа, последовательно выполняет действия, согласно операторам, записанным в вершинах граф-схемы. Вычисления прекращаются в вершине с оператором St . Заключительная конфигурация есть последовательность a_1 , a_2 , ... содержимого регистров R_1 , R_2 , ... на этом шаге.

Бывают вычисления, которые никогда не заканчиваются. Это означает, что при перемещении по граф-схеме исполнитель никогда не попадет в вершину с оператором St. В этом случае говорят, что вычисление расходится или что машина неприменима к начальной конфигурации.

Работа машины определяется граф-схемой. Поэтому говоря о конкретной машине, будем подразумевать граф-схему, которая определяет вычислительный процесс на этой машине.

Рабочей зоной машины называются все регистры, адреса которых записаны в операторах граф-схемы. Так как во время счета в граф-схеме адреса не изменяются, то рабочая зона для данной МНР фиксирована. Очевидно, что начальная, промежуточные и конечная конфигурации расположены в рабочей зоне. Регистры вне рабочей зоны заведомо пустые.

Если адреса регистров начальной конфигурации и операторов граф-схемы увеличить (уменьшить) на одно и то же число, то рабочая зона, а также конфигурации будут сдвинуты на это число.

Если имеется несколько машин, то, сдвинув соответственно их рабочие зоны, можно добиться, чтобы последние не имели общих регистров.

Обозначим через D_f область определения функции f . Говорят, что МНР вычисляет функцию $f(x_1, x_2, ..., x_n)$, если для всех a_1 , a_2 , ..., a_n , β вычисления сходятся тогда и только тогда, когда $(a_1, a_2, ..., a_n) \in D_f$ и в первом регистре заключительной конфигурации записано число $\beta = f(a_1, a_2, ..., a_n)$.

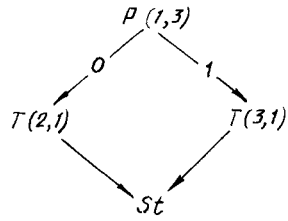


Рис. 2.5

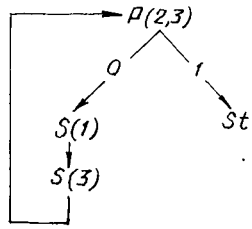


Рис. 2.6

Функция f называется *МНР-вычислимой*, если существует МНР, которая ее вычисляет.

Нуль-функция $O(x)$, функция следования $x + 1$ и функция выделения аргумента $U_i^n(x_1, x_2, ..., x_n) = x_i$ будут МНР-вычислимыми, так как они определяются операторами Z(1), S(1), T(i +1, 1).

На рис. 2.5 изображена граф-схема для вычисления функции

$$\operatorname{sgn}(x) = \begin{cases} 1, \text{ если } x \neq 0; \\ 0, \text{ если } x = 0. \end{cases}$$

Начальная конфигурация: $R_1 = x$, $R_2 = 1$, $R_3 = 0$. Регистры R_2, R_3 используются в вычислениях. Результат находится в R_1.

На рис. 2.6 изображена граф-схема для вычисления функции $x_1 + x_2$. Начальная конфигурация: $R_1 = x_1$, $R_2 = x_2$, $R_3 = 0$. Регистр R_3 используется в вычислениях. Результат (сумма) находится в R_1. Отметим, что функцию $x_1 + x_2$ от натуральных чисел вычисляют с помощью оператора S(x), т. е. функции следования $x + 1$. Это оказывается возможным потому, что функция $x_1 + x_2$ обладает таким свойством: $a + (b + 1) = (a + b) + 1$. Оно означает, что функция для следующего значения $b + 1$ выражается через функцию S(x) для уже известного значения $a + b$.

Определение функции $h(x, y)$ от натуральных чисел с помощью двух соотношений:

$$h(x, 0) = \varphi(x); \quad h(x, y + 1) = g(x, y, h(x, y)),$$

где функции φ, g известны, называют *определением* ее *через примитивную рекурсию*, а указанные соотношения – *уравнениями рекурсии*.

Часто функция $g(x, y, h(x, y))$ не зависит от y, т. е. $g(x, h(x, y))$. Если же она не зависит еще и от x, то уравнения рекурсии в этом случае запишутся в виде:

$$h(x, 0) = \varphi(x); \quad h(x, y + 1) = g(h(x, y)).$$

Сумма $h(x, y)$ чисел x и y, т. е. $x + y$, определяется такой примитивной рекурсией, где $\varphi(x) = x$; $g(z) = z + 1$.

Если рекурсия не зависит от x, то имеем уравнения:

$$h(0) = a; \quad h(y + 1) = g(y, h(y)),$$

где a – число.

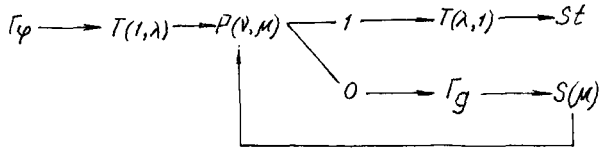


Рис. 2.7

Примитивная рекурсия непосредственно обобщается на функции многих переменных. В этом случае уравнения рекурсии запишутся в виде:

$$h(x_1, x_2, ..., x_n, 0) = \varphi(x_1, x_2, ..., x_n); \quad h(x_1, x_2, ..., x_n, y+1) =$$

$$= g(x_1, x_2, ..., x_n, y, h(x_1, x_2, ..., x_n, y)).$$

Определение через примитивную рекурсию есть по существу указание на способ ее вычисления.

Построим граф-схему для вычисления функции h. Пусть Γ_φ и Γ_g — граф-схемы для вычислимых функций φ и g. Сдвинем адреса в граф-схемах так, чтобы их рабочие зоны не пересекались. Регистры R_ν, R_λ предназначаются для записи соответственно y, φ, а вспомогательный регистр R_μ выполняет роль счетчика. В регистр R_1 записываем значение h.

На рис. 2.7 изображена граф-схема Γ_h, по которой вычисляют функцию h. В начальный момент $R_\mu = 0$. Как видим, примитивная рекурсия осуществляется в граф-схеме обратной связью, т. е. циклом.

Отсюда следует, что *если функции φ и g МНР-вычислимы, то определяемая через примитивную рекурсию функция тоже МНР-вычислима.*

Построим граф-схему для вычисления произведения $z_1 z_2$ натуральных чисел. Для этого воспользуемся уравнениями примитивной рекурсии $x \cdot 0 = 0$, $x(y+1) = xy + x$. Здесь $\varphi(x) = 0$, а $g(x, h(x, y)) = h(x, y) + x$.

При построении граф-схемы будем использовать граф-схему, приведенную на рис. 2.7. В нашем случае отсутствует граф-схема Γ_φ, а следовательно, и оператор $T(\lambda, 1)$, ибо можно сразу положить $R_1 = 0$.

На рис 2.8 изображена граф-схема умножения двух чисел, в которой используется оператор $C(1, 4)$ сложения двух чисел из регистров R_4, R_1 и записи их суммы в регистр R_4.

Начальная конфигурация: $R_1 = z_1$, $R_2 = z_2$, $R_3 = 0$, $R_4 = 0$. В регистре R_4 число из регистра R_1 суммируется столько раз, сколько число, содержащееся в R_2.

Чтобы получить граф-схему в операторах МНР, надо оператор $C(1, 4)$ представить в виде граф-схемы сложения (см. рис. 2.6). При этом следует изменить в ней некоторые адреса, так как суммируются числа из регистров R_4, R_1, а в качестве вспомогательного регистра взять R_5. В итоге получим граф-схему, изображенную на рис. 2.9. Начальная конфигурация: $R_1 = z_1$, $R_2 = z_2$, $R_3 = 0$, $R_4 = 0$, $R_5 = 0$. Регистры R_3, R_5 служат счетчиками. Построенная граф-схема содержит два цикла.

Пример 2.1. Построить граф-схему для вычисления функции $(x + y)^2$.

Р е ш е н и е. Пусть Γ_1 — граф-схема сложения (см. рис. 2.6). Положим $R_1 = x$, $R_2 = y$, $R_3 = 0$. В результате вычислений по Γ_1 получим конфигурацию $R_1 = x + y$, $R_2 =$

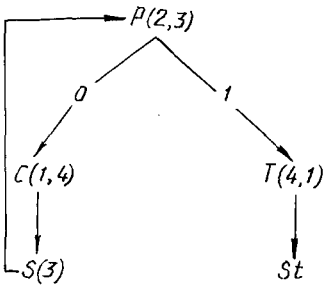


Рис. 2.8

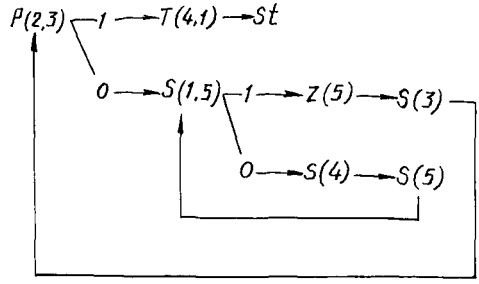


Рис. 2.9

$= y$, $R_3 = y$. В граф-схеме Γ_1 вместо оператора St поместим T (1,2) • Z(3) • St. По полученной граф-схеме Γ_1^0 найдем заключительную конфигурацию $R_1 = x + y$, $R_2 = x + y$, $R_3 = 0$. Положим $R_4 = 0$, $R_5 = 0$. Соединим граф-схему Γ_1^0 с граф-схемой Γ_2 для умножения чисел, т. е. вместо оператора St в Γ_1^0 поместим Γ_2 (см. рис. 2.9). В результате вычислений получим $R_1 = (x + y)^2$.

Вычисления в примере проводились по такой схеме: вначале нашли $z = (x + y)$, а затем $u = z^2$, т. е. вместо аргумента z в выражение для u подставили $(x + y)$.

В общем виде операция суперпозиции определяется следующим равенством:

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

где вместо аргументов φ_i подставлены функции $\varphi_1(x_1, x_2, ..., x_n)$, $\varphi_2(x_1, x_2, ..., x_n)$, ..., $\varphi_k(x_1, x_2, ..., x_n)$.

В примере 2.1 граф-схема для вычисления функции $(x + y)^2$ была получена как последовательное соединение граф-схем суммы и произведения, т. е. $\Gamma_1^0 \Gamma_2$. Граф-схему для суперпозиции функций ψ и φ_1 , φ_2 , ..., φ_k также получают последовательным соединением граф-схем. В самом деле, пусть рабочие зоны всех граф-схем Γ_ψ , Γ_{φ_1} , Γ_{φ_2} , ..., Γ_{φ_k} не пересекаются. В регистрах R_{ν_1} , R_{ν_2} , ... , R_{ν_k} записываются соответственно результаты вычислений Γ_{φ_1} , Γ_{φ_2} , ..., Γ_{φ_k}, в регистрах R_{μ_1} , R_{μ_2} , ..., R_{μ_k} содержатся аргументы φ_1 , φ_2 , ..., φ_k функции ψ .

На рис. 2.10 изображена граф-схема Γ_f для вычисления функции f .

Из вышеизложенного следуют два утверждения.

1. *Если функции* $\psi(\varphi_1, \varphi_2, ..., \varphi_k)$, $\varphi_1(x_1, x_2, ..., x_n)$, ..., $\varphi_2(x_1, x_2, ..., x_n)$, ..., $\varphi_k(x_1, x_2, ..., x_n)$ *МНР-вычислимы, то их суперпозиция* $f(x_1, x_2, ..., x_n) = \psi(\varphi_1, \varphi_2, ..., \varphi_k)$ *тоже МНР-вычислима.*

2. *Пусть* $\psi(x_1, x_2, ..., x_k)$ *– МНР-вычислимая функция и* x_{i_1} , x_{i_2} , ... , x_{i_k} *– последовательность из k переменных (возможно с повторением). Тогда* $f(x_1, x_2, ..., x_n) = \psi(x_{i_1}, x_{i_2}, ..., x_{i_k})$ *– тоже МНР-вычислимая функция, ибо* $f = \psi(\varphi_1, \varphi_2, ..., \varphi_k)$, *где* $\varphi_\nu = u_{i_\nu}^n(x_1, x_2, ..., x_n)$ *для всех* $\nu = \overline{1, k}$.

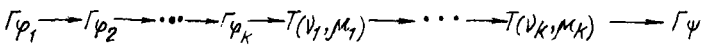


Рис. 2.10

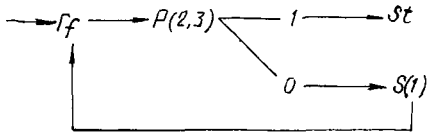


Рис. 2.11

Пример 2.2. Функция $f(y)$ определена для всех натуральных чисел. Найти наименьший натуральный корень уравнения $f(y) = 0$.

Р е ш е н и е. Схема решения уравнения $f(y) = 0$ такова. Последовательно вычисляем значения $f(0)$, $f(1)$, ... до тех пор, пока не получим $f(y_0) = 0$. Наименьший корень y_0. Если функция $f(y)$ не имеет корней, процесс поиска не ограничен.

Пусть граф-схема Γ_f вычисляет значение функции $f(y)$ и регистры R_2, R_1 используются для записи $f(y)$ и y соответственно. В регистр R_3 запишем нуль. Итак, начальная конфигурация: $R_1 = 0$, $R_2 = 0$, $R_3 = 0$.

На рис. 2.11 изображена граф-схема для определения наименьшего корня уравнения $f(y) = 0$.

Наименьший корень μ_y уравнения $f(x, y) = 0$ будет уже функцией $g(x)$. Условимся записывать эту функцию так: $g(x) = \mu_y(f(x, y) = 0)$. Вычисление $\mu_y(f(x, y) = 0)$ называют *операцией минимизации.* В общем случае операция минимизации $g(x_1, x_2, ..., x_n) = \mu_y(f(x_1, x_2, ..., x_n, y) = 0)$ означает определение наименьшего корня уравнения $f(x_1, x_2, ..., x_n, y) = 0$ при заданных $x_1, x_2, ..., x_n$. Граф-схема для общего случая будет такой же (см. рис. 2.11). Только следует предусмотреть в граф-схеме Γ_f сохранение значений $x_1, x_2, ..., x_n$ после каждого очередного вычисления функции $f(x_1, x_2, ..., x_n, y)$. Для этого нужно выделить n регистров для хранения $x_1, x_2, ..., x_n$ и после каждого вычисления функции f переписать эти значения в такие n регистров, которые участвуют в вычислениях f.

Очевидно, что *если* $f(x_1, x_2, ..., x_n, y)$ *МНР-вычислима, то функция* $\mu_y(f(x_1, x_2, ..., x_n, y) = 0)$ *тоже МНР-вычислима.*

Как было отмечено выше, рабочая зона данной МНР ограничена. Исходная информация принадлежит рабочей зоне, ибо вне ее регистры не используются. Отсюда следует, что МНР не решает задачи, у которых исходная информация потенциально не ограничена. К примеру, не может быть МНР, суммирующей произвольное число слагаемых.

2.3. РЕКУРСИВНЫЕ ФУНКЦИИ

Арифметической называют функцию, которая вместе со своими аргументами принимает только натуральные значения 0, 1, 2, ...

Базисными называются следующие арифметические функции:

1) функция $O(x)$, тождественно равная нулю;

2) функция следования $x + 1$;

3) для каждого $n \geqslant 1$ и $1 \leqslant i \leqslant n$ функция проекции U_i^n, определенная посредством формулы $U_i^n(x_1, x_2, ..., x_n) = x_i$.

Используя в качестве исходных базисные функции, можно с помощью небольшого числа операций строить все более и более сложные арифметические функции.

Арифметическую функцию, которую можно получить из базисных функций, используя только операции суперпозиции и примитивной рекурсии, называют *примитивно рекурсивной функцией.*

Уточним операцию минимизации. Пусть $f(x_1, x_2, ..., x_n, y)$ – произвольная арифметическая функция. Построим функцию $g(x_1, x_2, ..., x_n)$ с помощью оператора минимизации:

$$g(x_1, x_2, ..., x_n) = \mu_y (f(x_1, x_2, ..., x_n, y) = 0).$$

Это означает:

1) для произвольного набора $(a_1, a_2, ..., a_n)$ необходимо составить уравнение $f(a_1, a_2, ..., a_n, y) = 0$;

2) если существует натуральное число y, являющееся решением этого уравнения, надо взять минимальное из решений и обозначить его через μ_y; если значения $f(a_1, a_2, ..., a_n, 0), ..., f(a_1, a_2, ..., a_n, \mu_y - 1)$ также определены, то полагают $g(a_1, a_2, ..., a_n) = \mu_y$;

3) в противном случае (когда либо уравнение не имеет решений, либо хотя бы одно из значений $f(a_1, a_2, ..., a_n, 0), ..., f(a_1, a_2, ..., \mu_y - 1)$ не определено) значение функции $g(a_1, a_2, ..., a_n)$ не определено.

Про функцию g говорят, что она получена из функции f с помощью операции минимизации. Используя эту операцию, получают и функции, обратные данным.

Арифметическую функцию, которую можно получить из базисных функций, используя только операции суперпозиции, примитивной рекурсии и минимизации, называют *частично рекурсивной функцией.*

Частично рекурсивные функции МНР-вычислимы. В самом деле, базисные функции $O(x)$, $x + 1$, U_i^n МНР-вычислимы. Операции примитивной рекурсии, суперпозиции и минимизации реализуются на МНР, т. е. существуют соответствующие граф-схемы (см. рис. 2.7, 2.10, 2.11).

Рассмотрим несколько наиболее важных примитивно рекурсивных функций.

1. Сумма двух натуральных чисел определяется так: $x + 0 = x$, $x + (y+1) = (x + y) + 1$. Следовательно, она определяется через примитивную рекурсию.

2. Арифметическая разность $x \dot{-} 1$ определяется через примитивную рекурсию: $0 \dot{-} 1 = 0$, $(x + 1) \dot{-} 1 = x$. Очевидно, что

$$x \dot{-} 1 = \begin{cases} 0, & \text{если } x = 0; \\ x-1, & \text{если } x \geqslant 1. \end{cases}$$

Определим теперь арифметическую разность $x \dot{-} y$ через примитивную рекурсию: $x \dot{-} 0 = x$, $x \dot{-} (y + 1) = (x \dot{-} y) \dot{-} 1$. Покажем, что

$$x \dot{-} y = \begin{cases} 0, & \text{если } x < y; \\ x-y, & \text{если } x \geqslant y. \end{cases} \qquad (2.1)$$

Для $y = 0$ имеем $x \dot{-} 0 = x$, следовательно, $x \dot{-} 0 = x - 0$ при $x \geqslant 0$.

Пусть равенство (2.1) выполняется для $y \leqslant k$. Покажем, что в этом случае оно справедливо и для $y = k + 1$. Если $x \geqslant k + 1$, то тем более $x > k$. Поэтому $x \dot{-} (k + 1) = (x \dot{-} k) \dot{-} 1 = (x - k) - 1 = x - (k + 1)$. Если $x < k + 1$, т. е. $x \leqslant k$, то $x \dot{-} (k + 1) = (x \dot{-} k) \dot{-} 1 = 0 \dot{-} 1 = 0$. Согласно принципу математической индукции, равенство выполняется.

3. Умножение натуральных чисел определяется через примитивную рекурсию: $x \cdot 0 = 0$, $x(y + 1) = xy + x$.

Вычислим несколько значений этой функции: $5 \cdot 1 = 5(0 + 1) = 5 \cdot 0 + 5 = 0 + 5 = 5$; $5 \cdot 2 = 5(1 + 1) = 5 \cdot 1 + 5 = 5 + 5 = 10$; $5 \cdot 3 = 5(2 + 1) = 5 \cdot 2 + 5 = 10 + 5 = 15$ и т. д.

4. Определим функцию $h(x)$ через рекурсию: $h(0) = 1$, $h(x + 1) = h(x) \cdot (x + 1)$.

Вычислим несколько значений этой функции: $h(1) = h(0+1) = h(0) \cdot 1 = 1 \cdot 1 = 1$; $h(2) = h(1 + 1) = h(1) \cdot 2 = 1 \cdot 2$ и т. д.

Функцию $h(x)$ называют *факториалом* и обозначают $x!$. Легко показать по индукции, что $x! = 1 \cdot 2 \cdots x$.

5. Определим функцию, обозначаемую x^y, через рекурсию: $x^0 = 1$, $x^{y+1} = x^y x$. Функцию x^y называют *степенной*.

Методом индукции легко доказать, что $x^y = x\ x \cdots x$, т. е. x^y равно произведению из y сомножителей по x.

6. Функция

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

примитивно рекурсивна, так как $\operatorname{sgn}(0) = 0$, $\operatorname{sgn}(x + 1) = 1$.

7. Функция

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

примитивно рекурсивна, так как $\overline{\operatorname{sgn}}(x) = 1 \dot{-} \operatorname{sgn}(x)$.

8. Функция

$$|x - y| = \begin{cases} x - y, & \text{если } x \geqslant y; \\ y - x, & \text{если } y > x \end{cases}$$

примитивно рекурсивна, поскольку $|x - y| = (x \dot{-} y) + (y \dot{-} x)$.

9. Функция

$$\max(x, y) = \begin{cases} x, & \text{если } x \geqslant y; \\ y, & \text{если } y > x \end{cases}$$

примитивно рекурсивна, так как $\max(x, y) = x + (y \dot{-} x)$.

10. Функция

$$\min(x, y) = \begin{cases} x, & \text{если } x \leqslant y; \\ y, & \text{если } y < x \end{cases}$$

примитивно рекурсивна, ибо $\min(x, y) = x \dot{-} (x \dot{-} y)$.

11. Пусть $y = qx + r$, где $0 \leqslant r < x$. Число q называют *частным* от деления y на x, число r – *остатком* от деления y на x и обозначают соответственно $qt(x, y)$, $rm(x, y)$.

Чтобы получить всюду определенные функции, положим $qt(0, y) = 0$, $rm(0, y) = 0$. Покажем, что $qt(x, y)$, $rm(x, y)$ – примитивно рекурсивные функции.

Очевидно, что если натуральное число z удовлетворяет неравенству $xz \leqslant y < x(z + 1)$, то $z = qt(x, y)$.

Для следующего значения $y + 1$ может выполняться неравенство с прежним значением z, тогда $qt(x, y + 1) = qt(x, y)$ либо $y + 1 = x(z + 1)$. В этом случае $qt(x, y + 1) = q(t, y) + 1$.

Таким образом,

$$qt(x, y + 1) = \begin{cases} qt(x, y), & \text{если } y + 1 < x(qt(x, y) + 1); \\ qt(x, y) + 1, & \text{если } y + 1 = x(qt(x, y) + 1). \end{cases}$$

Отсюда следует, что

$$qt(x, y + 1) = qt(x, y) + \overline{\operatorname{sgn}}(x(qt(x, y) + 1) \dot{-} (y + 1)).$$

Для остатка имеем $rm(x, y) = y \dot{-} xqt(x, y)$.

12. Определим для $a > 1$ функцию *арифметический логарифм* $\log_a x$ следующим образом: 1) $\log_a 0 = 0$; 2) если $x > 0$ и выполняется неравенство $a^z \leqslant x < a^{z+1}$, то $\log_a x = z$.

Ниже приведены первые 11 значений $\log_2 x$:

x	0	1	2	3	4	5	6	7	8	9	10
$\log_2 x$	0	0	1	1	2	2	2	2	3	3	3

Покажем, что $\log_a x$ – примитивно рекурсивная функция. Для данного значения x найдем $z = \log_a x$ из соотношения $a^z \leqslant x < a^{z+1}$. Для следующего значения $x + 1$ может выполняться прежнее неравенство, тогда $\log_a(x + 1) = \log_a x$ либо $x + 1 = a^{z+1}$. Во втором случае $\log_a(x+1) = \log_a x + 1$. Таким образом,

$$\log_a(x+1) = \begin{cases} \log_a x, & \text{если } x+1 < a^{\log_a x + 1}; \\ \log_a x + 1, & \text{если } x+1 = a^{\log_a x + 1}. \end{cases}$$

Отсюда следует $\log_a(x+1) = \log_a x + \overline{\operatorname{sgn}}(a^{\log_a x + 1} \dot{-} (x+1))$, что и требовалось доказать.

13. Теперь покажем, что функция, принимающая конечное число значений, примитивно рекурсивна. Очевидно, что

$$\overline{\operatorname{sgn}}|x - k| = \begin{cases} 1, & \text{если } x = k; \\ 0, & \text{если } x \neq k. \end{cases}$$

Поэтому

$$\prod_{\nu=1}^{n} \overline{\operatorname{sgn}}\,|x_\nu - k_\nu| = \begin{cases} 1, \text{ если } x_1 = k_1,\ x_2 = k_2, \ldots, x_n = k_n \\ 0, \text{ если хотя бы одно } x_\nu \neq k_\nu \end{cases};$$

Обозначим $h_{k_1, k_2, \ldots, k_n}(x_1, x_2, \ldots, x_n) = \prod_{\nu=1}^{n} \overline{\operatorname{sgn}}\,|x_\nu - k_\nu|$. Пусть

$$f(x_1, x_2, \ldots, x_n) = \begin{cases} \beta_i, \text{ если } x_1 = k_1^i,\ x_2 = k_2^i, \ldots, x_n = k_n^i,\ i = \overline{1,m} \\ 0 - \text{в остальных точках,} \end{cases};$$

тогда очевидно, что $f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{m} \beta_i h_{k_1^i, k_2^i, \ldots, k_n^i}$. Эта сумма есть примитивно рекурсивная функция.

2.4. МАШИНА ТЬЮРИНГА

Машина Тьюринга состоит из бесконечной в обе стороны *ленты*, разделенной на *ячейки*, и считывающей и записывающей *головки*. Головка в каждый данный момент обозревает некоторую ячейку ленты и выполняет следующие операции: 1) считывание содержимого обозреваемой ячейки; 2) запись в обозреваемую ячейку; 3) передвижение головки к смежной ячейке. Последовательное выполнение этих трех операций называется *тактом*.

В каждой ячейке ленты может быть записан лишь один символ из конечного алфавита $W = \{x_0, x_1, \ldots, x_n\}$. Символ x_0 будет обозначать пустой символ: $x_0 = \Lambda$. О ячейке, в которой записан пустой символ, будем говорить, что она пустая. Если все ячейки ленты пустые, то и сама лента пустая. К началу работы машины Тьюринга на ленте записана начальная информация, которая состоит из конечного числа непустых ячеек. В дальнейшем будет видно, что в процессе работы машины на ленте каждый раз имеется конечное число непустых ячеек.

Запись в ячейку очередного символа x_ν означает, что имеющийся в ячейке символ x_i стирается и на его место записывается x_ν. В частности, $x_i = x_\nu$.

Передвижение головки к смежной ячейке означает одну из трех возможностей: переместиться на одну ячейку вправо – П, переместиться на одну ячейку влево – Л, оставаться на месте – С.

Алфавит $\{x_0, x_1, \ldots, x_n\}$ является входным алфавитом машины. Головка находится на данном такте в одном из возможных состояний: $\{s_0, s_1, \ldots, s_m\}$. От обозреваемого символа $x(t)$ и состояния головки $s(t)$ на такте t зависят символ $y(t+1)$, печатаемый в обозреваемую ячейку, состояние $s(t+1)$ и перемещение $p(t+1)$ головки на следующем такте $t+1$. Иначе говоря, действия головки определяются соотношениями:

$$\begin{aligned} s(t+1) &= f_1(x(t),\ s(t)); \\ y(t+1) &= f_2(x(t),\ s(t)); \\ p(t+1) &= f_3(x(t),\ s(t)), \end{aligned} \qquad (2.2)$$

где t означает номер такта; $s(t)$ принимает значения из алфавита $\{s_0, s_1, ..., s_m\}$; $p(t)$ – из алфавита $\{\text{П}, \text{Л}, \text{С}\}$; y, x – из алфавита $\{x_0, x_1, ..., x_n\}$.

В момент $t = 0$ головка занимает начальное положение и обозревает содержимое определенной ячейки, а на ленте записана исходная информация. Головка приведена в начальное состояние.

Работа машины Тьюринга складывается из следующих один за другим тактов. Пусть в такте t $(t = 0, 1, 2, ...)$ головка находится напротив ячейки, в которой был записан символ x_i. Символ x_i поступает на вход головки. В зависимости от состояния $s(t)$ и входа $x(t) = x_i$ головка переходит в состояние $s(t+1)$ и вырабатывает значения $y(t+1)$, $p(t+1)$. Очередное значение $y(t+1)$ записывается головкой в ячейку. В зависимости от значения $p(t+1)$ головка перемещается вправо: $p(t+1) = \text{П}$, влево: $p(t+1) = \text{Л}$, остается на месте: $p(t+1) = \text{С}$.

Будем считать, что символ s_0 означает состояние покоя, т. е. при этом состоянии машина Тьюринга прекращает работу. Тогда для $s(t) = s_0$ и любого $x(t)$ имеем: $s(t+1) = s_0$, $p(t+1) = \text{С}$, $y(t+1) = x(t)$.

В зависимости от того, какая начальная информация была подана, возможны два случая:

1) машина Тьюринга после конечного числа тактов останавливается. В таком случае говорят, что она *применима* к начальной информации и перерабатывает ее в результирующую информацию, т. е. в ту, которая изображена на ленте после останова;

2) машина Тьюринга после конечного числа тактов не останавливается. Тогда говорят, что она *неприменима* к начальной информации.

Состояние головки называют также *состоянием машины Тьюринга.* Алфавит $\{s_0, s_1, ..., s_m, \text{П}, \text{Л}, \text{С}\}$ называют *внутренним,* а $\{x_0, x_1, ..., x_n\}$ – *внешним алфавитом машины.*

Рекуррентные формулы (2.2) удобно задавать таблицей для функций f_1, f_2, f_3 (например, табл. 2.2), которую называют *функциональной схемой.*

Так как s_0 является состоянием покоя, то его строка в таблице состоит из s_0, x_i и символа С. Для упрощения записи строки таблицы, содержащие s_0, мы будем опускать (см. табл. 2.3).

Таблица 2.2

s	x	s'	y	D
s_0	x_0	s_0	x_0	С
s_0	x_1	s_0	x_1	С
s_0	x_2	s_0	x_2	С
s_1	x_0	s_2	x_0	П
s_1	x_1	s_1	x_2	П
s_1	x_2	s_2	x_1	Л
s_2	x_0	s_1	x_1	П
s_2	x_1	s_2	x_2	Л
s_2	x_2	s_0	x_0	С

Таблица 2.3

s	x	s'	y	D
s_1	x_0	s_2	x_0	П
s_1	x_1	s_1	x_2	П
s_1	x_2	s_2	x_1	Л
s_2	x_0	s_1	x_1	П
s_2	x_1	s_2	x_2	Л
s_2	x_2	s_0	x_0	С

Если задана функциональная схема, то при каждой начальной информации и определенной начальной ячейке работа машины определена однозначно.

Пусть головка машины обозревает символ $x = a$, находясь в состоянии $s = s_i$. Тогда если в строке таблицы $s = s_i$, $x = a$ записано $s' = s_j$, $y = b$, $D = d_v$ (т. е. $f_1 = s_j$, $f_2 = b$, $f_3 = d_v$), головка напечатает символ b, осуществит движение d_v и перейдет в состояние s_j. Таким образом, машина осуществит переработку исходной записи на ленте (конфигурации) в соответствии с функциональной схемой, начиная с исходного состояния и начальной ячейки. Так как исполнитель имитирует машину Тьюринга, функциональную схему называют также *программой исполнителя.*

Приведем пример машины Тьюринга. Рассмотрим алгоритм сложения нескольких положительных чисел. Внешний алфавит $W = \{1, +, \Lambda\}$. На ленте в начальный момент записано слово, например:

		1	1	1	1	1	+	1	1	1	1	1	1		

что означает 5 + 6. В результате должна получиться сумма. В данном случае

		1	1	1	1	1	1	1	1	1	1	1		

Функциональная схема машины Тьюринга определяется табл. 2.4.

В начальный момент головка обозревает ячейку с самой левой единицей и машина настроена на состояние s_1. В первых пяти тактах, согласно табл. 2.4, головка перемещается вправо, пока не достигнет ячейки $x = +$. На шестом такте вместо + печатается 1, головка остается на месте и переходит в состоя-

Таблица 2.4

s	x	s'	y	D	s	x	s'	y	D
s_1	1	s_1	1	П	s_2	Λ	s_4	Λ	П
s_1	+	s_2	1	С	s_3	1	s_3	1	Л
s_1	Λ	s_3	Λ	Л	s_3	Λ	s_0	Λ	П
s_2	1	s_2	1	Л	s_4	1	s_1	Λ	П

ние s_2. В следующих пяти тактах головка перемещается влево, пока не достигнет пустой ячейки. На двенадцатом такте головка перемещается на одну ячейку вправо, а на тринадцатом стирает единицу, перемещается вправо до последней единицы, затем влево до первой единицы и останавливается.

Пусть машина Тьюринга задана функциональной схемой. Изобразим ее в виде граф-схемы от переменной p в алфавите:

$$B = \{(x_0, \text{П}), (x_0, \text{Л}), (x_0, \text{С}), ..., (x_n, \text{П}), (x_n, \text{Л}), (x_n, \text{С})\} .$$

Пару (x, D) из алфавита B назовем *арифметическим оператором*. Оператор (x, D) означает, что следует напечатать в обозреваемую ячейку букву x и либо переместить головку вправо или влево, либо оставить ее на месте при $D = \text{П}, D = \text{Л}, D = \text{С}$ соответственно.

Переменная p принимает значения $\{x_0, x_1, ..., x_n\}$. Букву p называют также *логическим оператором*, выполняющим считывание символа в обозреваемой ячейке. Переменная p является значением логического оператора.

Для того чтобы изобразить граф-схемой функциональную схему, надо на плоскости отметить m точек и возле каждой из них записать букву s_i, т. е. состояние машины. Из каждой точки s_i, за исключением s_0, следует провести $n + 1$ дугу, приписав каждой по одному значению x_j, причем дуга $p = x_j$ должна соединять вершину s_i с вершиной s_{ij}. В вершине s_{ij} надо записать соответствующий оператор (x_{ij}, d_{ij}).

Таким образом, начальная вершина s_i, дуга x_j, конечная вершина дуги s_{ij} и записанная в вершине пара (x_{ij}, d_{ij}) соответствуют строке $s_i x_j s_{ij} x_{ij} d_{ij}$ функциональной схемы. Буквы s_i, за исключением s_0 (состояние покоя или оператор St), можно стереть.

На рис. 2.12 изображена граф-схема, построенная по функциональной схеме табл. 2.4.

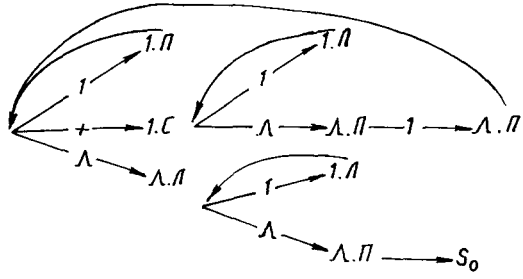


Рис. 2.12

Машину Тьюринга имитирует исполнитель, который в момент $t = 0$ обозревает как начальную ячейку, так и вход граф-схемы. В зависимости от обозреваемой буквы x_j исполнитель перемещается по соответствующей дуге x_j начального куста к следующей вершине. Затем он прочитывает записанный в вершине оператор (x_{ij}, d_{ij}), печатает в обозреваемой ячейке букву x_{ij} и либо перемещается по ленте на одну ячейку вправо при $d_{ij} = \text{П}$ или влево при $d_{ij} = \text{Л}$, либо остается на месте при $d_{ij} = \text{С}$.

Переход от граф-схемы к функциональной схеме очевиден. Для этого каждую вершину граф-схемы отмечают различными буквами s_i. Затем граф-схему записывают в виде таблицы, каждая строка которой соответствует определенной дуге.

В дальнейшем каждую машину Тьюринга будем задавать граф-схемой. Это обусловливается тем, что значительно проще проводить анализ граф-схемы и строить ее с учетом требуемых условий, чем функциональную схему. При этом можно эффективно использовать метод синтеза граф-схем, изложенный в гл.1. Это означает, что разработчик по условиям задачи строит граф-схему (схему алгоритма). Скорее всего, это будет дерево-схема, ибо для объединения хотя бы некоторых вершин требуется значительная изобретательность. Далее по известной уже методике нумеруют вершины исходной граф-схемы и объединяют вершины с одинаковыми номерами. Полученная граф-схема будет простой (согласно теореме 1.3). Но так как в граф-схеме для машины Тьюринга всего одна переменная, простая граф-схема будет минимальной.

При построении машин Тьюринга желательно использовать уже известные, ранее построенные, машины. При граф-схемном описании машин такой подход сводится к композиции граф-схем. *Композицией двух машин Тьюринга* назовем такую машину Тьюринга, граф-схема которой представляет собой композицию граф-схем исходных машин.

Суть использования композиций машин заключается в том, что построение машины Тьюринга проводят вначале на расширенном алфавите операторов, включив в него некоторые машины. Затем эти операторы (машины) замещаются своими граф-схемами.

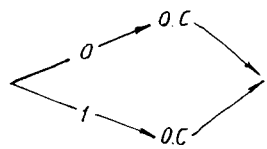
Без ограничения общности изложения будем рассматривать машины Тьюринга с внешним алфавитом $\{0, 1\}$, где 0 означает пустой символ. Записанная на ленте информация представляет собой слово (конфигурацию), состоящее из конечного числа единичных интервалов, разделенных нулевыми (пустыми) интервалами.

Условимся далее не указывать оператор x_j в конечной вершине дуги x_l, если $x_l = x_j$, т. е. считать, что содержимое ячейки не изменяется.

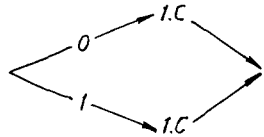
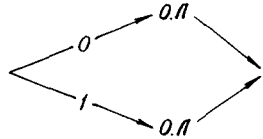
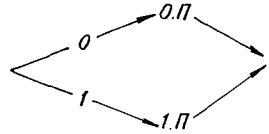
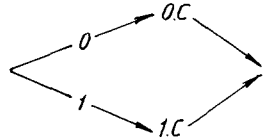
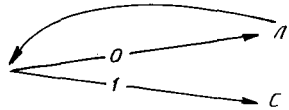
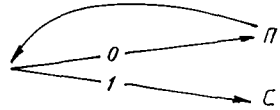
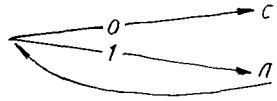
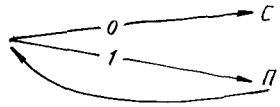
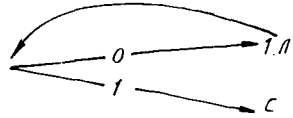
Слово, у которого единичные интервалы отделены одной пустой ячейкой, будем называть *основным кодом*.

В дальнейшем будет пользоваться алфавитом операторов, приведенным в табл. 2.5. Для удобства все операторы занумерованы и будут фигурировать под этими номерами в промежуточных построениях.

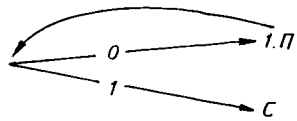
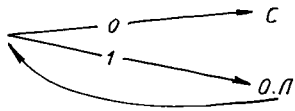
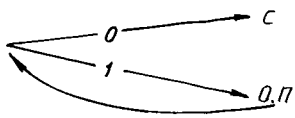
Таблица 2.5

Оператор	Содержание оператора	Граф-схема оператора
1	2	3
0	Запись нуля в обозреваемую ячейку	

Продолжение табл. 2.5

1	2	3
1	Запись единицы в обозреваемую ячейку	
2	Сдвиг головки на одну ячейку влево	
3	Сдвиг головки на одну ячейку вправо	
4	Головка остается на месте	
5	Прекращение работы машины	St
6	Сдвиг головки влево к первой отмеченной ячейке	
7	Сдвиг головки вправо к первой отмеченной ячейке	
8	Сдвиг головки влево к первой пустой ячейке	
9	Сдвиг головки вправо к первой пустой ячейке	
10	Запись единиц в ячейки слева до первой отмеченной ячейки	

Окончание табл. 2.5

1	2	3
11	Запись единиц в ячейки справа до первой отмеченной ячейки	
12	Стирание единиц в ячейках слева до первой пустой ячейки	
13	Стирание единиц в ячейках справа до первой пустой ячейки	

Операторы (машины) 6—13 изображаются граф-схемами с одним кустом. Все они построены с учетом их содержания. Для примера получим оператор под номером 6. По условию требуется построить машину Тьюринга, головка которой перемещается от обозреваемой пустой (нулевой) ячейки влево до первой отмеченной (единичной) ячейки. Головка обозревает пустую ячейку. Этому условию соответствует дуга $x = 0$. Головку следует переместить влево. Поэтому в конце дуги запишем оператор d = Л. Если головка снова обозревает пустую ячейку, то повторится начальная ситуация. В этом случае соединим конечную вершину дуги с начальной. Если ячейка оказалась отмеченной, машина прекращает работу. В конце второй дуги ($x = 1$) записываем оператор d = С. Так как содержимое ячейки не изменяется, то, по соглашению, в конце дуги буква x не записывается.

2.5. ПРИМЕРЫ ПОСТРОЕНИЯ МАШИН ТЬЮРИНГА

Последовательность ячеек, в которых записаны единицы, назовем *регистром*, если слева и справа она граничит с пустыми ячейками.

Если последовательность ячеек содержит $a + 1$ единицу, то говорят, что в регистре записано натуральное число a. Так что нуль изображается интервалом из одной единицы. "Пустых" регистров не бывает.

Набор чисел $(a_1, a_2, ..., a_k)$ условимся задавать на ленте основным кодом, интервалы которого содержат соответственно $(a_1 + 1), (a_2 + 1), ..., (a_k + 1)$ единиц. Иначе говоря, набор $(a_1, a_2, ..., a_k)$ запишем в k регистрах, разделенных одной нулевой (пустой) ячейкой. Пустые ячейки служат признаком границы регистра. Запись чисел в регистрах образует на ленте слово, ранее названное основным кодом.

Оператор 14(15). Построить машину Тьюринга, которая осуществляет перемещение головки, обозревающей ячейку основного кода, к левой единице основного кода.

Признаком начала основного кода является наличие более одной пустой ячейки слева от отмеченной ячейки. Поэтому головка движется влево, пока не

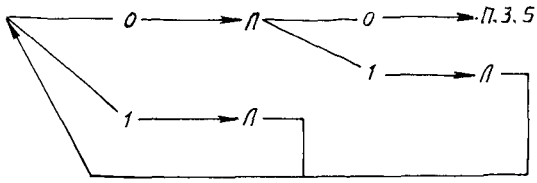


Рис. 2.13

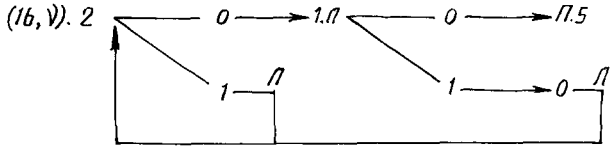


Рис. 2.14

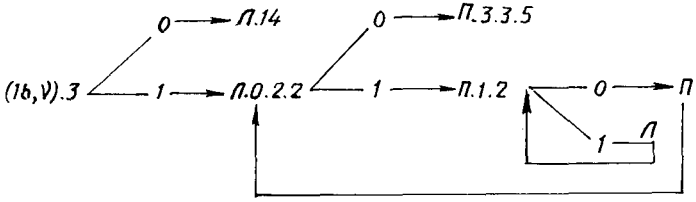


Рис. 2.15

обнаружит две соседние пустые ячейки, и затем перемещается вправо на две ячейки.

На рис. 2.13 изображена граф-схема. Присвоим построенной машине номер 14, а машине, осуществляющей перемещение из любой ячейки основного кода к его правому концу, – номер 15.

О п е р а т о р (16, ν) (*оператор обнаружения регистра*). Построить машину Тьюринга, перемещающую головку из начала первого регистра к началу ν-го регистра.

Головка перемещается вправо, пропуская единичные интервалы (регистры) и по одной $\nu - 1$ пустой ячейке. Поэтому граф-схемой будет последовательность операторов 9.3 ... 9.3.5, в которой пара 9.3 встречается $\nu - 1$ раз.

О п е р а т о р S(ν). Построить машину Тьюринга, осуществляющую прибавление единицы к содержимому ν-го регистра.

Прибавление единицы к числу a осуществляется путем записи еще одной единицы в соседнюю с единичным интервалом пустую ячейку. Но вначале следует обнаружить ν-й регистр (оператор 16, ν), затем слева в пустой ячейке записать единицу. Остальные левые регистры нужно сдвинуть влево на одну ячейку. Головка остановится в начале основного кода. Граф-схема изображена на рис. 2.14.

О п е р а т о р (17, ν). Построить машину Тьюринга, осуществляющую арифметическое вычитание единицы из содержимого ν-го регистра.

Пусть обнаружен ν-й регистр (оператор 16, ν). Следует проверить, выполняется ли равенство $a_\nu = 0$. Для этого надо переместить головку вправо (опе-

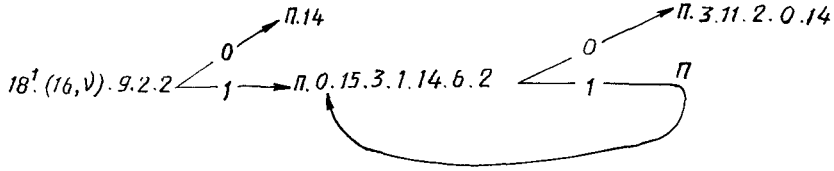


Рис. 2.16

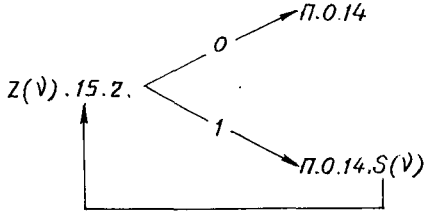


Рис. 2.17

ратор 3). Если обозреваемая ячейка пустая, то $a_\nu = 0$. Тогда возвращаем головку к началу основного кода (оператор 14). Если обозреваемая ячейка отмечена, то в ячейке, находящейся слева от нее, стираем единицу и все регистры, расположенные левее, перемещаем на одну ячейку вправо. Граф-схема приведена на рис. 2.15.

О п е р а т о р $Z(\nu)$. Построить машину Тьюринга, которая осуществляет обнуление ν-го регистра.

Чтобы записать нуль в ячейке, надо последовательно выполнить арифметическое вычитание единицы из содержимого ν-го регистра. Для этого нужно в граф-схеме, приведенной на рис. 2.15, стереть оператор 5 (St) и соединить вершину дугой с началом граф-схемы. Если в граф-схеме, изображенной на рис. 2.15, вместо оператора $(16, \nu)$ записать операторы 15.8.3, получим оператор обнуления последнего регистра (оператор $\mathbf{Z}$).

О п е р а т о р 18^1. Построить машину Тьюринга, оформляющую еще один (последний) регистр.

В этом случае головка должна через одну пустую ячейку после регистров записать единицу и возвратиться к началу первого регистра. Граф-схемой будет последовательность операторов 15.3.3.1.14.

О п е р а т о р 18^0. Построить машину Тьюринга, аннулирующую последний регистр.

Чтобы стереть все единицы последнего единичного интервала, следует переместить головку в правый конец основного кода (15), затем влево, стерев единицы, до первой пустой ячейки и вернуться к началу основного кода. Получим граф-схему вида 15.12.14.

О п е р а т о р $(19, \nu)$. Построить машину Тьюринга, записывающую в оформленный регистр содержимое ν-го регистра.

На рис. 2.16 изображена граф-схема оператора $(19, \nu)$.

О п е р а т о р $(20, \nu)$. Построить машину Тьюринга, записывающую в регистр R_ν содержимое последнего регистра и аннулирующую его.

Граф-схема оператора $(20, \nu)$ приведена на рис. 2.17.

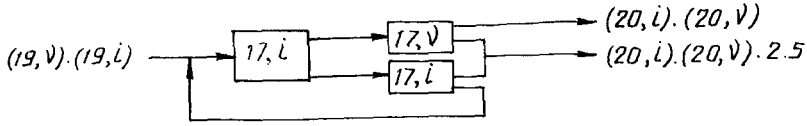


Рис. 2.18

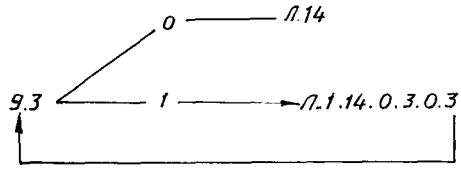


Рис. 2.19

Очевидно, что:

1) *оператор* $T(i, v)$, *выполняющий* $a_v := a_i$, *можно записать в виде граф-схемы*: $(19, i) . (20, v)$;

2) *оператор* $Q(i, v)$ *обмена содержимым между регистрами можно записать в виде граф-схемы*: $(19, i) . (19, v) . (20, i) . (20, v)$.

Оператор $P(i, v)$ (*оператор условного перехода*). Построить машину Тьюринга, сравнивающую содержимое регистров R_i и R_v. Если $a_i = a_v$, головка переходит к началу основного кода ($x = 1$, т. е. $P(i, v) = 1$), если $a_i \neq a_v$, головка переходит к ячейке, расположенной левее основного кода ($x = 0$, т. е. $P(i, v) = 0$).

На рис. 2.18 изображена граф-схема оператора $P(i, v)$. Верхние дуги блоков $17i$, $17v$ относятся к выходу 14, нижние – к выходу 5.

Отметим следующее: если в граф-схеме машины Тьюринга после перехода по дуге $x = 0$ выполняется некоторый оператор А, следует вначале выполнить сдвиг вправо на одну ячейку (оператор 3), т. е. 3.А, а после перехода по дуге $x = 1$ – оператор 4.А.

Оператор Сл. Построить машину Тьюринга, суммирующую любое число натуральных слагаемых, заданных основным кодом.

Сложение двух чисел сводится к замене разделяющей их пустой ячейки отмеченной ячейкой и аннулированию одной единицы в первом единичном интервале кода слагаемых. Если слагаемых больше, чем два, то действия повторяются. На рис. 2.19 изображена граф-схема искомой машины.

Класс задач, реализуемых на машинах Тьюринга, шире класса задач, реализуемых на МНР. В самом деле, в рассмотренных выше примерах были построены машины Тьюринга (операторы) $Z(n)$, $S(n)$, $P(i, v)$, $T(i, v)$, реализующие алфавит операторов МНР. Следовательно, граф-схема, полученная композицией граф-схем для данной МНР и операторов $Z(n)$, $S(n)$, $P(i, v)$, $T(i, v)$, является граф-схемой машины Тьюринга, решающей ту же задачу, что и данная МНР. В частности, выполнимы операции суперпозиции, минимизации и примитивной рекурсии.

Однако задача суммирования произвольного числа слагаемых, реализуемая на машине Тьюринга, неразрешима на МНР из-за ограниченности ее рабочей зоны.

2.6. ВЫЧИСЛИМЫЕ ФУНКЦИИ

Машина Тьюринга вычисляет функции $f(x_1, x_2, ..., x_n)$, если:

1) при $(\alpha_1, \alpha_2, ..., \alpha_n) \in D_f$ машина, будучи применена к основному коду для $(\alpha_1, \alpha_2, ..., \alpha_n)$ и находясь в начальном состоянии под его левой единицей, останавливается и выдает на ленте код для $\beta = f(\alpha_1, \alpha_2, ..., \alpha_n)$, при этом останов происходит под левой единицей кода для β;

2) при $(\alpha_1, \alpha_2, ..., \alpha_n) \in D_f$ машина, будучи применена к основному коду для $(\alpha_1, \alpha_2, ..., \alpha_n)$, не останавливается.

Функцию называют *вычислимой*, если существует машина Тьюринга, которая ее вычисляет.

Как отмечалось выше, класс задач, решаемых на машинах Тьюринга, шире класса задач, решаемых на МНР. Отсюда, в частности, следует, что класс МНР-вычислимых функций содержится в классе вычислимых функций.

Так как класс частично рекурсивных функций принадлежит классу МНР-вычислимых функций, он тем более принадлежит классу вычислимых функций.

Имеет место обратное утверждение.

Теорема 2.1. *Всякая вычислимая функция есть частично рекурсивная функция.*

Доказательство. Пусть машина Тьюринга вычисляет функцию $y = f(x_1, x_2, ..., x_k)$. Без ограничения общности считаем, что состояния s обозначаются натуральными числами 0, 1, ..., причем $s = 0$ означает оператор St, а буквы Л, П, С закодированы соответственно числами 2, 3, 4. Для упрощения доказательства считаем, что внешний алфавит состоит из двух букв: $\{0, 1\}$.

Пусть в данном такте головка считывает букву p. Тогда конфигурация на ленте имеет вид $...b_2 b_1 b_0 p c_0 c_1 c_2 ...$ Будем считать, что в ячейках ленты слева и справа от головки (цифры p) записаны цифры двоичного числа. Поэтому конфигурация на ленте определяется тремя натуральными числами (p, m, n), где p – цифра (0 или 1) в обозреваемой ячейке; $m = \sum_{i=0}^{\infty} b_i 2^i$; $n = \sum_{i=0}^{\infty} c_i 2^i$. Суммы здесь конечные, так как только конечное число цифр b_i и c_i отлично от нуля.

Полное состояние машины в каждый момент времени будет определяться четверкой натуральных чисел (s, p, m, n). В каждом такте производится преобразование набора (s, p, m, n) в новый набор (s', p', m', n'). Указанное преобразование полностью определяет машину Тьюринга. Найдем это преобразование.

Машина Тьюринга задана следующей функциональной схемой:

$$s' = Q(s, p), \quad p' = R(s, p), \quad d = D(s, p). \tag{2.3}$$

Пусть для определенности $d = 3$ (т. е. головка перемещается на ячейку вправо) и вместо p напечатано p'. Конфигурация на ленте имеет вид $...b_2 b_1 b_0 p' c_0 c_1 c_2$, и головка обозревает c_0. Левее головки записано двоичное число $m' = 2m + p'$, правее – $n' = [n/2]$. Если $d = 4$ (головка остается на

месте), то $m' = m$, $n' = n$. И, наконец, при $d = 2$ имеем $m' = [m/2]$, $n' = 2n + p'$.

Объединив в одну формулу все три случая, получим примитивно рекурсивные функции:

$$m' = (2m + p')\overline{\operatorname{sgn}}\,|d-3| + m\,\overline{\operatorname{sgn}}\,|d-4| + [m/2]\,\overline{\operatorname{sgn}}\,|d-2|;$$

$$n' = (2n + p')\,\overline{\operatorname{sgn}}\,|d-2| + n\,\overline{\operatorname{sgn}}\,|d-4| + [n/2]\,\overline{\operatorname{sgn}}\,|d-3|.$$

Подставив в правые части этих равенств вместо p', d функции $R(s,p)$, $D(s,p)$, получим:

$$m' = \psi_1(m, s, p); \quad n' = \psi_2(n, s, p). \tag{2.4}$$

Функции $R(s,p)$, $D(s,p)$ отличны от нуля в конечном числе пар (s,p), поэтому они примитивно рекурсивны. Следовательно, $\psi_1(m,s,p)$, $\psi_2(n,s,p)$ – тоже примитивно рекурсивные функции.

Обозначим через p^0 букву, обозреваемую в следующем такте. Очевидно, что $p^0 = c_0\,\overline{\operatorname{sgn}}\,|d-3| + p\,\overline{\operatorname{sgn}}\,|d-4| + b_0\,\overline{\operatorname{sgn}}\,|d-2|$. Но $c_0 = n - 2[n/2]$, $b_0 = m - [m/2]$, поэтому $p^0 = \psi_3(m,n,s,p)$ – примитивно рекурсивная функция.

На такте t значения s, m, n, p зависят от t и начальных значений $s = 1$, $n = n_0$, $m = 0, p = 0$, так как при $t = 0$ головку считаем обозревающей ячейку левее основного кода. Поэтому $m = \varphi_1(n_0, t)$, $n = \varphi_2(n_0, t)$, $s = \varphi_3(n_0, t)$, $p = \varphi_4(n_0, t)$, причем $\varphi_1(n_0, 0) = 0$, $\varphi_2(n_0, 0) = n_0$, $\varphi_3(n_0, 0) = 1$, $\varphi_4(n_0, 0) = 0$. Согласно равенствам (2.4), функции $\varphi_1, \varphi_2, \varphi_3, \varphi_4$ удовлетворяют соотношениям:

$$\varphi_1(n_0, t+1) = \psi_1(\varphi_1(n_0,t), \varphi_3(n_0,t), \varphi_4(n_0,t));$$

$$\varphi_2(n_0, t+1) = \psi_2(\varphi_2(n_0,t), \varphi_3(n_0,t), \varphi_4(n_0,t));$$

$$\varphi_3(n_0, t+1) = Q(\varphi_3(n_0,t), \varphi_4(n_0,t));$$

$$\varphi_4(n_0, t+1) = \psi_3(\varphi_1(n_0,t), \varphi_2(n_0,t), \varphi_3(n_0,t); \varphi_4(n_0,t)).$$

Эти соотношения являются естественным обобщением схемы примитивной рекурсии. Нетрудно показать, что данные функции могут быть получены из функций ψ_1, ψ_2, ψ_3, Q с помощью примитивной рекурсии и суперпозиции. Следовательно, функции $\varphi_1, \varphi_2, \varphi_3, \varphi_4$ примитивно рекурсивны.

После завершения вычислений на машине Тьюринга $s = 0, m = 0, p = 1$, поскольку головка обозревает левую ячейку интервала единиц.

Чтобы найти n, подставим в $\varphi_4(n_0, t)$ значение t_k такта завершения вычислений, которое определим из уравнения $\varphi_3(n_0, t) = 0$ (так как заключительное состояние $s = 0$). Тогда $t_k = \mu_t(\varphi_3(n_0, t) = 0)$. Следовательно, $t_k = t(n)$ – частично рекурсивная функция. Отсюда $n = \varphi_2(n_0, t_k) = F(n_0)$ – также частично рекурсивная функция от n_0.

Выразим n_0 через заданный набор чисел $(a_1, a_2, \ldots, a_k)$, т. е. в виде $n = \Phi(a_1, a_2, \ldots, a_k)$. Числа a_i представлены интервалами из $a_i + 1$ единиц.

Но перед α_j расположено $\beta_j = \sum_{\nu=1}^{j-1} (\alpha_\nu + 2)$ двоичных разрядов чисел α_ν и разделяющих их пустых ячеек. Поэтому в $n_0 = \sum_{i=0}^{\infty} c_i 2^i$ число α_j индуцирует слагаемое $2^{\beta_j - 1}(2^{\alpha_j + 1} - 1)$. Итак,

$$n_0 = \sum_{j=1}^{k} 2^{\beta_j - 1}(2^{\alpha_j + 1} - 1).$$

Следовательно, $n_0 = \Phi(\alpha_1, \alpha_2, ..., \alpha_k)$ – частично рекурсивная функция, поэтому и $n = F(\alpha_1, \alpha_2, ..., \alpha_k)$ – частично рекурсивная функция.

Так как $n = 2^{\alpha+1} - 1$ (как единичный интервал из $\alpha + 1$ единиц), то $\alpha = \log_2(n+1) - 1$ – частично рекурсивная функция: $\alpha = \theta(\alpha_1, \alpha_2, ..., \alpha_k)$. Теорема доказана.

Следствие. *Классы частично рекурсивных, МНР-вычислимых и вычислимых функций совпадают.*

В самом деле, класс частично рекурсивных функций принадлежит классу МНР-вычислимых функций (см. § 2.3), который в свою очередь входит в класс вычислимых функций. А последний класс, согласно теореме 2.1, принадлежит классу частично рекурсивных функций. Следовательно, все классы совпадают.

Любую задачу, решаемую машиной Тьюринга, можно трактовать как вычисление некоторой функции. Это очевидно, если входную информацию составляют слова ограниченной длины. Если же входная информация потенциально не ограничена, то ее можно считать одним натуральным числом, представимым в двоичной системе счисления, при условии, что внешний алфавит состоит из 0 и 1. Подобный подход был применен при доказательстве теоремы 2.1. Таким образом, любая задача, решаемая на машине Тьюринга, трактуется как вычисление частично рекурсивной функции одной переменной. Так как любую частично рекурсивную функцию можно вычислить на МНР, то и любую задачу, решаемую на машинах Тьюринга, можно решить на МНР. Выше было отмечено, что на МНР неразрешима уже такая простая задача, как сложение любого числа слагаемых. Поэтому точнее будет сказать, что частично рекурсивная функция содержится в классе функций, реализуемых на МНР, потому что на МНР реализуемы базисные функции и операторы, определяющие множество частично рекурсивных функций.

Что касается вычисления на МНР суммы, то, действительно, на МНР можно "суммировать" любое число слагаемых $a_1, a_2, ..., a_k$, если только все эти k слагаемых закодировать одним числом, записать его в регистр R_1 и обработать затем соответствующим алгоритмом.

Кодировать любое количество чисел можно различными способами. Например, $x = p_1^{a_1} p_2^{a_2} \cdots p_k^{a_k}$, где p_i – i-е простое число $(p_1 = 2, p_2 = 3, p_3 = 5, ...)$, или иначе: $x = 2^{a_1+1} + 2^{a_1+a_2+1} + ... + 2^{a_1+a_2+...+a_k+1}$

В любом случае кодирование предполагает действия с потенциально неограниченным числом элементов. Поэтому само кодирование на МНР невоз-

можно. Если же код x записать в регистр R_1, то дальнейшие операции по вычислению "суммы" на МНР возможны. В такой постановке имеем уже другую задачу. Нельзя сказать, что подобная процедура есть суммирование чисел, так как задаются не слагаемые, а их код, который, повторяем, на МНР не вычисляется.

2.7. НОРМАЛЬНЫЙ АЛГОРИТМ МАРКОВА

Присоединим к множеству слов в данном алфавите W пустое слово (пустую последовательность букв) Λ.

Если α, β обозначают два слова, то $\alpha\beta$ обозначает соединение этих слов. Например, $\alpha = baab$, $\beta = babb$, тогда $\alpha\beta = baabbabb$. В частности, $a\Lambda = \Lambda a = a$. Справедливо также $(a_1 a_2)a_3 = a_1 (a_2 a_3)$.

Алфавит W_1 называется *расширением* алфавита W_2, если $W_2 \subseteq W_1$. Если алфавит W_1 есть расширение алфавита W_2, то всякое слово в алфавите W_2 является словом и в алфавите W_1.

Рассмотрим два слова (α и β) в некотором алфавите W. Если слово α является частью слова β, то говорят, что α *входит* в β или что имеется *вхождение* α в β. Например, слово $\alpha = ba$ входит в слово $\beta = abbab$. В общем случае слово α может входить в слово β несколько раз; так, слово aba входит в слово $bababaab$ два раза. Первое слева вхождение называют *первым вхождением*.

Два слова λ и μ в данном алфавите W, соединенных знаком $\rightarrow$ (или $\rightarrow\cdot$), т. е. $\lambda \rightarrow \mu$ (или $\lambda \rightarrow \cdot \mu$), называют *формулой подстановки* в алфавите W. Слово λ называют *левой*, слово μ – *правой частью формулы*. Например, $caab \rightarrow ba$ является формулой подстановки. Заметим, что знаки $\rightarrow$, $\rightarrow\cdot$ не являются буквами алфавита W, а каждое из слов λ или μ может быть пустым. Формула подстановки $\lambda \rightarrow \mu$ называется *простой*, а $\lambda \rightarrow \cdot \mu$ – *заключительной*.

Конечное число формул подстановки в алфавите W, заданных в определенном порядке:

$$\lambda_1 \rightarrow (\cdot)\,\mu_1;$$
$$\lambda_2 \rightarrow (\cdot)\,\mu_2;$$
$$\ldots\ldots\ldots$$
$$\lambda_k \rightarrow (\cdot)\,\mu_k,$$

где $\lambda \rightarrow (\cdot)\,\mu$ обозначает любую из формул подстановки, называется *схемой нормального алгоритма* и порождает следующий алгоритм в алфавите W.

Пусть дано некоторое слово α в алфавите W.

1. Обозреть последовательно левые части формул подстановок $\lambda_1, \lambda_2, \ldots, \lambda_k$ схемы алгоритма. Возможны два случая:

а) среди слов $\lambda_1, \lambda_2, \ldots, \lambda_k$ есть такие, которые входят в α;

б) ни одно из слов $\lambda_1, \lambda_2, \ldots, \lambda_k$ не входит в α. В случае "а" перейти к п. 2, в случае "б" – к п. 5.

2. Выбрать наименьшее целое число m, такое, что λ_m входит в α. Перейти к п. 3.

3. В слове α найти первое вхождение слова λ_m и заменить его словом μ_m. В результате получается слово α'. Если формула подстановки $\lambda_m \rightarrow (\cdot)\,\mu_m$ – заключительная, то перейти к п. 5, в противном случае – к п. 4.

4. Перейти к п. 1, но вместо слова α рассмотреть полученное слово α'.

5. Процесс преобразования слов прекратить. Полученное слово α' (в частности, $\alpha' = \alpha$) считать результатом применения нормального алгоритма к слову α.

Если процесс преобразования слова α не прекращается, то говорят, что алгоритм *неприменим* к слову α.

Пусть алфавит W и схема алгоритма имеют вид:

$$W = \{1, +\}: \quad \text{а) } 1 + \to + 1; \quad \text{б) } + 1 \to 1.$$

Преобразуем слово $\alpha = 111 + 1 + 11$. Последовательно получим слова:

111 + 1 + 11;
11 + 11 + 11;
1 + 111 + 11;
+ 1111 + 11;
+ 111 + 111;
+ 11 + 1111;
+ 1 + 11111;
++ 111111;
+ 111111;
111111.

Процесс оканчивается, потому что в последнем слове нет вхождений левых частей формул подстановки. Легко видеть, что алгоритм суммирует количество единиц, указанных в слове α.

Как мы убедились, нормальный алгоритм Маркова в алфавите W задает некоторое преобразование слов в этом алфавите. Произвольный алгоритм в алфавите отличается от нормального тем, что его список указаний либо содержит не только стандартные указания в виде формул подстановок, либо вообще их не содержит.

Два алгоритма в некотором алфавите называются *эквивалентными*, если области их применимости совпадают и оба они на этой области определяют одинаковое преобразование. Пока не удалось привести пример такого алгоритма в алфавите W, для которого нельзя было бы построить эквивалентный ему нормальный алгоритм. Часто сведение заданного алгоритма в алфавите W к эквивалентному ему нормальному алгоритму сопряжено со значительными трудностями.

Уточним действия исполнителя нормального алгоритма. Будем считать, что слово записано на ленте, разделенной на ячейки. В ячейке можно записать не более одной буквы алфавита. Исполнитель за один шаг обозревает ячейку, печатает букву или стирает ее (т. е. печатает пустой символ), переходит в соседнюю ячейку или остается на месте.

Итак, замещение вхождения λ правой частью μ подстановки представляет собой теперь более сложную операцию, ибо она может включать увеличение или уменьшение длины интервала (количества ячеек), занимаемого вхождением.

Будем говорить, что машина Тьюринга и нормальный алгоритм Маркова моделируют друг друга, если у них пошаговые преобразования любого слова совпадают.

2.8. ГРАФ-СХЕМА НОРМАЛЬНОГО АЛГОРИТМА

Оператором A в алфавите W называется всякое однозначное соответствие, сопоставляющее слова в алфавите W со словами в том же или другом алфавите. Совокупность всех слов, на которых оператор A определен, называется *областью определения оператора* A. Слово, которое ставится в соответствие слову α, будем записывать так: $A(\alpha)$ и говорить, что оператор A применяется к слову α. Примером оператора A в алфавите W может служить соответствие, которое каждое слово сопоставляет с этим же словом, только без первой буквы (оператор стирания первой буквы), например $A(aba) = ba$, $A(a) = \Lambda$, $A(\Lambda) = \Lambda$.

В нормальном алгоритме можно выделить два основных повторяющихся цикла.

1. Распознавание вхождения левой части данной i-й формулы подстановки $\lambda_i \to (\cdot)\, \mu_i$ в перерабатываемое слово α.

2. Запись в слове α вместо левой части λ_i формулы подстановки ее правой части μ_i.

Рассмотрим такое высказывание: "Левая часть λ_i i-й формулы подстановки входит в обозреваемое исполнителем слово". Это высказывание для данного слова α может быть истинно или ложно. Пусть P_i – логический оператор, такой, что $P_i(\alpha) = 1$, если это высказывание для α истинно, и $P_i(\alpha) = 0$, если оно ложно. Итак, если $P_i = 1$, то в слове α имеется вхождение λ_i, если $P_i = 0$, то такого вхождения нет.

Обозначим через C_i оператор замены в заданном слове α первого вхождения λ_i словом μ_i. Введем еще оператор C_0 – останов.

Логическую часть нормального алгоритма можно задать двоичной функцией $A = f(p_1, p_2, ..., p_k)$ с областью значений $C_0, C_1, ..., C_k$ (табл. 2.6). Прочерки в таблице означают любое из значений: 1 или 0, т. е. что значение A в строке не зависит от переменной.

Таблица 2.6

p_1	p_2	p_3	p_4	...	p_k	y
1	–	–	–	...	–	C_1
0	1	–	–	...	–	C_2
0	0	1	–	...	–	C_3
0	0	0	1	...	–	C_4
–	–	–	–	...	–	...
0	0	0	0	...	1	C_k
0	0	0	0	...	0	C_0

На рис. 2.20 изображена граф-схема двоичной функции $A = f(p_1, p_2, ..., p_k)$, которую назовем *граф-схемой нормального алгоритма.*

Пример 2.3. Имеем алфавит $W = \{1, +\}$ и схему алгоритма: а) $1+ \rightarrow +1$; б) $+1 \rightarrow 1$. Построить граф-схему алгоритма (сложения).

Р е ш е н и е. Обозначим через P_1 и P_2 такие логические операторы: P_1 – "в данном слове есть вхождение слова 1 + "; P_2 – "в данном слове есть вхождение слова + 1". Арифметические операторы C_1 и C_2 означают соответственно замену в слове α вхождения 1 + на + 1 и + 1 на 1. Схема алгоритма определяет табл. 2.7, по которой построим граф-схему (рис. 2.21).

Таблица 2.7

p_1	p_2	A
1	–	C_1
0	1	C_2
0	0	C_0

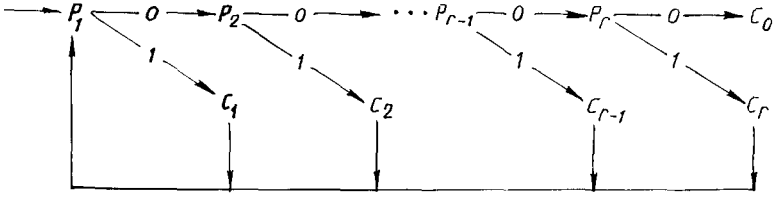


Рис. 2.20

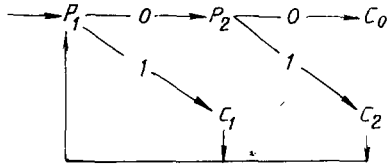


Рис. 2.21

Построенная граф-схема содержит операторы P_i, C_v, которые заданы нормальным алгоритмом.

Для операторов P_i, C_v построим соответствующие машины Тьюринга (граф-схемы), распознающие первые вхождения λ_i и записывающие вместо λ_i слово μ_i.

Без ограничения общности рассмотрим машины Тьюринга с внешним алфавитом $\{0, 1\}$. Буквы нормального алгоритма кодируются интервалами единиц, а слова – основными кодами. Например, если буквы $\{a, b, +\}$ закодировать соответственно 11, 111, 1111, то слово $a + b$ запишется так: 11011110111.

Покажем, как построить граф-схему V для машины Тьюринга, распознающей первое вхождение $\lambda = \beta_0\beta_1 \ldots \beta_t$ в слово α. Распознавание происходит в процессе побуквенного чтения слова α слева направо.

Буквы $\beta_i = \{0, 1\}$, причем нули служат знаками, разделяющими в слове λ интервалы единиц и букв. Слово λ ограничиваем нулями, т. е. $\beta_0 = 0$, $\beta_t = 0$.

Отмечаем столбец из t пронумерованных (начиная с единицы) точек. Точки являются входами кустов, составляющих искомую граф-схему. В начальный момент головка обозревает первую единицу слова α, поэтому входом граф-схемы будет вершина с номером 1. Вершина с номером k означает, что обнаружено совпадение начальной части слова λ из k букв, т.е. $\beta_0\beta_1 ... \beta_k$, со словом из последних k букв прочтенной части слова α.

Конечная вершина дуги $x = \gamma$ куста с номером k получит номер l, где l – наибольшая длина конечной части слова $\beta_0\beta_1 ... \beta_k\gamma$, которая совпадает с ее начальной частью. Если $\gamma = \beta_{k+1}$, то $l = k + 1$; если ни одна часть слова не совпадает с λ, то $l = 0$.

Равенство $l = t$ означает обнаружение вхождения λ в слово α. На рис.2.22 изображена граф-схема распознавания вхождения слова $\lambda = 010110$. Оператор Q означает возврат головки к началу слова α. Вершина Q является входом граф-схемы распознавания следующей подстановки.

Исполнитель с выхода куста с номером k переходит на вход куста с тем же номером k. При обнаружении вхождения головка останавливается у буквы β_t в слове α, т. е. обозревает пустую ячейку после интервала единиц.

После обнаружения вхождения следует вместо λ вписать слово $\mu = \mu_1\mu_2 ... \mu_m$. Если $m = t$, после оператора распознавания можно применить оператор записи $З(\mu) = 2\mu_m 2\mu_{m-1} ... 2\mu_1$ и оператор 14, который возвращает головку к началу полученного слова.

Следовательно, граф-схему V в вершине С последовательно соединяем с граф-схемой $З(\mu)$. 14.

Если $m > t$, вначале следует сдвинуть влево на $\nu = m - t$ ячеек ту часть слова α, которая находится левее головки. Если $m < t$, следует сдвинуть влево на $\nu = t - m$ ячеек ту часть слова α, которая расположена правее головки. В первом случае интервал ячеек для записи увеличим до m, во втором – уменьшим до m.

На рис. 2.23 изображена граф-схема D сдвига части слова α, расположенной левее головки на одну ячейку влево. Если требуется выполнить сдвиг на ν ячеек, то граф-схему D^ν получают, последовательно соединяя ν граф-схем D в вершинах 15.3, записанных вместо 5.

Итак, при $m > t$ применяют оператор $R = D^\nu$.15.3. $З(\mu)$. Операторы 15.3 включены потому, что $З(\mu)$ записывает слово μ справа налево. Аналогичное построение проводим для случая $t > m$.

Если в граф-схему нормального алгоритма вместо операторов P_i, C_ν подставить соответствующие граф-схемы, получим граф-схему для нормального алгоритма, но уже с операторами машины Тьюринга. Следовательно, каждый нормальный алгоритм моделируется машиной Тьюринга.

Справедливо обратное: машина Тьюринга моделируется нормальным алгоритмом в алфавите, состоящем из внешнего алфавита и алфавита состояний. В самом деле, машина Тьюринга задана функциональной схемой, строки которой имеют вид $s_i x_\nu s_{i\nu} x_{i\nu} d_{i\nu}$. Для каждой строки построим соответствующую подстановку:

$$s_i x_\nu \rightarrow s_{i\nu} x_{i\nu}, \quad \text{если } d_{i\nu} = С;$$

$$s_i x_\nu \rightarrow x_{i\nu} s_{i\nu}^{\nu}, \quad \text{если } d_{i\nu} = П;$$

$$x s_i x_\nu \rightarrow s_{i\nu} x x_{i\nu}, \quad \text{если } d_{i\nu} = Л,$$

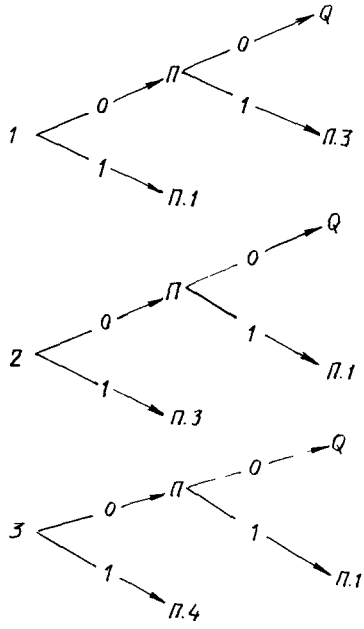


Рис. 2.22

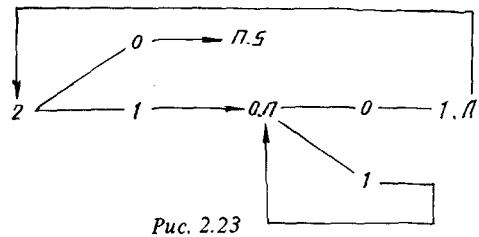


Рис. 2.23

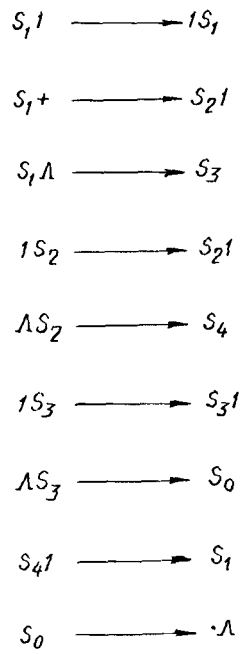


Рис. 2.24

где x – любая буква внешнего алфавита. Если $s = s_0$, подстановка будет заключительной: $s_0 \rightarrow \cdot\Lambda$.

Прежде чем применить нормальный алгоритм к начальной конфигурации в алфавите X, следует перед ней записать букву начального состояния. Так как на каждом шаге в конфигурации будет только одна буква из алфавита S, порядок подстановок в схеме алгоритма произвольный. На каждом шаге обработка слова нормальным алгоритмом совпадает с преобразованием машины Тьюринга.

Пример 2.4. По функциональной схеме (см. табл. 2.5) сложения положительных чисел построить соответствующий нормальный алгоритм.

Р е ш е н и е. Схема алгоритма будет содержать девять подстановок, из них одна – заключительная (рис. 2.24).

Как следствие из вышеизложенного, сформулируем утверждение. *Между множествами машин Тьюринга и нормальных алгоритмов можно установить взаимно однозначное соответствие, такое, что соответствующие машинам алгоритмы выполняют на каждом шаге одно и то же преобразование.* В частности, классы вычислимых функций совпадают.

3. АЛГОРИТМЫ И ПРОГРАММЫ

3.1. ОПРЕДЕЛЕНИЕ АЛГОРИТМА

В описании машины Тьюринга и МНР основным объектом является граф-схема. Нормальный алгоритм Маркова также можно задать с помощью граф-схемы. Таким образом, эффективно вычислимые процессы определяются через граф-схемы.

Граф-схема, вернее, схема алгоритма, является также основной частью описания алгоритмов, используемых на практике. Поэтому естественно полагать, что этот факт не является случайным свойством алгоритма. Мы примем это свойство за основополагающее в определении алгоритма. Таким образом, будет дано определение алгоритма, охватывающее классические алгоритмические системы и значительно приближенное к современной практике применения вычислительных и управляющих дискретных устройств.

Пусть даны алфавит $B = \{A_0, A_1, ..., A_m, P_1, ..., P_k\}$ и множество элементов ω, причем P_ν назовем *логическими*, A_i – *арифметическими операторами*, а A_0 – *оператором останова*.

Каждый оператор, кроме A_0, реализуется своей функцией с областью определения, принадлежащей множеству ω. Будем говорить, что оператор неприменим к элементу, если этот элемент не принадлежит области определения соответствующей функции. Значения арифметических операторов принадлежат множеству ω. Логические операторы принимают только два значения: 0 или 1. Каждому логическому оператору соответствует одна переменная, значения которой вычисляются в логическом операторе. Остальные переменные назовем *свободными*.

Имеется исполнитель, способный по предъявлению наименования оператора, т. е. буквы A_i или P_ν, и элемента $a \in \omega$ вычислить элемент $A_i(a)$ или число $P_\nu(a)$, сохранить их в своей памяти или прекратить вычисления, если a не принадлежит области определения оператора. При записи очередного значения предыдущая запись стирается.

Алгоритм от переменных $p_1, p_2, ..., p_n$ *в алфавите* $B = \{A_0, A_1, ..., A_m, P_1, ..., P_k\}$ *состоит из граф-схемы в алфавите B от переменных* $p_1, p_2, ..., p_n$, *в выходах которой записан оператор* A_0, *и исполнителя, действующего согласно инструкции.*

Множество ω назовем *областью задания алгоритма.*

Чтобы осуществить вычисления по алгоритму, следует задать определенный элемент $a_0 \in \omega$, т. е. записать a_0 в памяти исполнителя, а также записать значения свободных переменных.

Инструкция для исполнителя алгоритма

1. Обозреть начальную вершину граф-схемы.

2. Если в обозреваемой вершине M записан оператор С, перейти к п. 3, в противном случае – к п. 7.

3. Если оператор $C = A_0$, то перейти к п. 9, в противном случае – к п. 4.

4. Если оператор С неприменим к элементу a, содержащемуся в памяти, то перейти к п. 10, в противном случае – к п. 5.

5. Вычислить $C(a)$ и занести результат в память, т. е. выполнить a: $= A_i(a)$, если $C = A_i$, или $p_\nu := P_\nu(a)$, если $C = P_\nu$.

6. Из вершины M по единственной дуге переместиться к следующей вершине граф-схемы, перейти к п. 2.

7. Найти в памяти значение $p_\nu = \theta$, где p_ν – переменная в обозреваемой вершине M.

8. Из вершины M переместиться по дуге, соответствующей значению $p_\nu = \theta$, перейти к п. 2.

9. Вычисления прекратить.

10. Вычисления прекратить.

Результатом применения алгоритма к исходному элементу a_0 считаем элемент a, содержащийся в памяти исполнителя после прекращения вычислений по п. 9.

Если вычисления не прекращаются или прекращаются по п. 10, то считаем, что алгоритм неприменим к исходному элементу a_0.

Как видно из инструкции для исполнителя алгоритма, вершины с логическими операторами являются опорными, так как с них начинают вычисления по новым значениям переменных.

Если логический оператор P_ν соединен дугой с соответствующей ему переменной p_ν, то условимся исключать из граф-схемы эти дугу и переменную, совместив последнюю с оператором P_ν.

Пример 3.1. Множество ω состоит из всех слов алфавита $\{0, 1\}$. Алфавит операторов $B = \{A_0, A_1\}$. Оператор A_1 преобразует все буквы слова по правилу: $A_1(0) = 1$, $A_1(1) = 0$. На рис. 3.1, *а* изображена граф-схема алгоритма, применимого к любому слову, а на рис. 3.1, *б* – граф-схема алгоритма, неприменимого к любому слову.

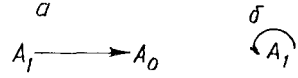


Рис. 3.1

Два алгоритма называются: 1) *эквивалентными (равносильными)*, если для любого элемента $a \in \omega$ они либо оба неприменимы, либо приводят к одинаковому результату; 2) *структурно эквивалентными*, если их граф-схемы эквивалентны.

Пример 3.2. Множество ω состоит из всех слов алфавита $\{0, 1\}$. Алфавит операторов $B = \{A_0, A_1, P\}$. Оператор Р определяется условием: $P(a) = 1$, если слово a начинается с единицы, и $P(a) = 0$ в противном случае; $A_1(0) = 1$, $A_1(1) = 0$.

Алгоритм, граф-схема которого приведена на рис. 3.2, *а*, применим к словам с начальной буквой 1 и неприменим к остальным словам. Алгоритм, граф-схема которого приведена на рис. 3.2, *б*, применим ко всем словам. Очевидно, что этот алгоритм эквива-

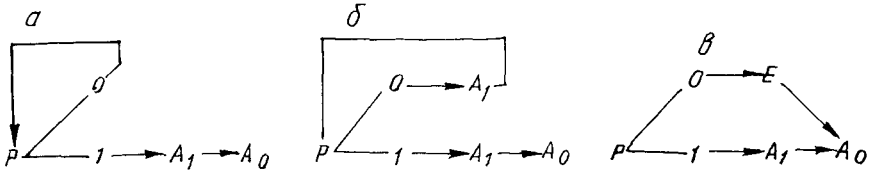


Рис. 3.2

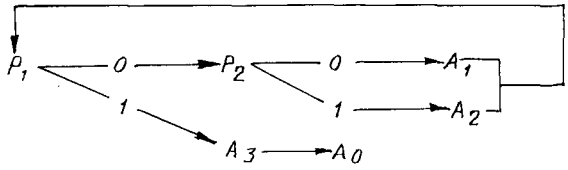


Рис. 3.3

лентен алгоритму с граф-схемой, изображенной на рис. 3.2, *в* , где E – единичный оператор, т. е. $E(a) = a$. Последний алгоритм задан уже в алфавите $B = \{A_0, A_1, E, P\}$. Эти алгоритмы не являются структурно эквивалентными.

Алфавит арифметических операторов не обязательно должен содержать только детерминированные операторы. Однако мы ограничимся рассмотрением алгоритмов с детерминированными операторами.

Рассмотрим более сложные примеры.

Пример 3.3. Построить алгоритм Евклида для определения НОД двух положительных натуральных чисел.

Р е ш е н и е. В § 2.1 были сформулированы три правила для получения НОД двух чисел. Применив эти правила, построим требуемую граф-схему.

Пусть a и b – заданные числа. На рис. 3.3 изображена граф-схема алгоритма Евклида, содержащего логические операторы

$$P_1(a, b) = \begin{cases} 1, \text{ если } a = b; \\ 0, \text{ если } a \neq b; \end{cases} \qquad P_2(a, b) = \begin{cases} 1, \text{ если } a \geqslant b; \\ 0, \text{ если } b > a \end{cases}$$

и арифметические операторы A_1, A_2, A_3, означающие соответственно: $b := b - a$; $a := a - b$; $d := a$. Через d обозначен искомый наибольший общий делитель.

Рассмотрим алгоритм Евклида, взяв областью его определения не натуральные числа, а множество отрезков.

Все операторы граф-схемы на множестве отрезков выполнимы. С помощью циркуля можно сравнивать два отрезка, складывать их, а также из большего вычитать меньший. Напомним, что если отрезок a сложить n раз, то полученный отрезок обозначают na и считают равным по определению произведению числа n на отрезок a . (Важно отметить, что речь идет об отрезках, а не их длине. Сложным понятием "длина отрезка" мы не пользуемся.)

Пусть имеются три отрезка: a , b , d . Если существуют два натуральных числа k , m , таких, что $a = kd$, $b = md$, то отрезки a, b называются *соизмеримыми*, а отрезок d – их *общей мерой*. Если к тому же k , m – взаимно простые числа, то d называют *наибольшей общей мерой*.

Лемма 3.1. *Если алгоритм Евклида перерабатывает отрезки a, b в отрезок d , то d является наибольшей общей мерой a и b .*

Доказательство. Продолжительность работы алгоритма Евклида оценим числом использований оператора P_1. Воспользуемся индукцией по n. Если $n = 1$, то $d = a$, $d = b$, и лемма справедлива. Пусть лемма верна для n. Докажем, что при этом предположении она справедлива для $n + 1$.

На первом шаге алгоритма, требующем одного выполнения оператора P_1, мы перешли от отрезков a, b к отрезкам $a_1 = a - b$, b (для определенности считаем $a > b$). Применение алгоритма к a_1, b потребует выполнения оператора P_1 n раз. Следовательно, есть наибольшая общая мера d, такая, что $a_1 = kd$, $b = md$ или $a - b = kd$. Отсюда $a = kd + md = (k + m)d$. Так как числа k, m – взаимно простые, то $k + m$, m – тоже взаимно простые числа. Лемма доказана.

Лемма 3.2. *Если отрезки a, b соизмеримы, то, применив к ним алгоритм Евклида, найдем их наибольшую общую меру.*

Доказательство. Пусть d – общая мера отрезков a и b и $a = kd$, $b = md$. После ν-го шага получим два отрезка: $a_\nu = k_\nu d$, $b_\nu = m_\nu d$. Так как наибольшее из двух натуральных чисел k_ν, m_ν с каждым шагом уменьшается, процесс должен прекратиться, т.е. алгоритм Евклида применим к соизмеримым отрезкам. Лемма доказана.

Из этих двух лемм следует, что *алгоритм Евклида применим только к соизмеримым отрезкам.*

Применим алгоритм Евклида к стороне AC и диагонали BC квадрата (рис. 3.4). Так как $BC > AC$, на первом шаге алгоритма перейдем к отрезкам AC и $CD = BC - AC$, отложив циркулем на диагонали отрезок $BD = AC$. Из неравенства треугольника $BC < AC + AB = 2 \cdot AC$ следует, что $BC - AC < AC$, т. е. $CD < AC$. Поэтому необходим очередной шаг алгоритма для поиска пары отрезков: $AC - CD$ и CD.

Найдем на рис. 3.4 отрезок $AC - CD$. Для этого из точки D проведем перпендикуляр к BD, который пересечет сторону квадрата AC в точке B', и соединим B' с B. Тогда $BA = BD$ и $\angle BAB' = \angle BDB'$ как прямые углы. Отсюда $\Delta AB'B = \Delta DB'B$, значит, $AB' = DB'$. Треугольник $B'CD$ – равнобедренный, поэтому $CD = DB'$, а значит, $CD = AB'$. Отсюда имеем $AC - CD = AC - AB' = CB'$. Итак, после второго шага получим отрезки CD и CB' $(CD \neq CB')$, т. е. сторону CD и диагональ CB' квадрата. Следовательно, вычисления по алгоритму никогда не прекратятся, иначе говоря, алгоритм Евклида неприменим к стороне и диагонали квадрата, т.е. они несоизмеримы.

С расширением алфавита операторов класс алгоритмов тоже расширяется, за исключением случаев, когда присоединяемые операторы представляют собой алгоритмы в исходном алфавите. Проиллюстрируем это примерами на геометрическое построение с помощью одной линейки. Объектами, которыми при этом оперируют, являются точки, отрезки, прямые. Но отрезок и прямую можно задать двумя точками. Поэтому любую конфигурацию из точек, отрезков и прямых можно описать совокупностью точек. Задачу на построение фигур, состоящих из точек, отрезков, прямых,

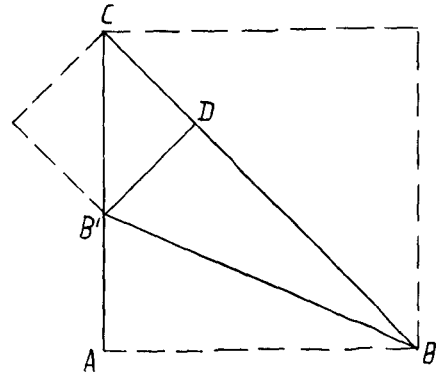


Рис. 3.4

всегда можно свести к задаче на построение совокупности точек.

Итак, будем считать, что задача на геометрическое построение состоит в том, чтобы по заданной совокупности точек построить, согласно условиям задачи, искомую совокупность точек. Поэтому в качестве области определения ω в задачах на построение будем брать множество конечных совокупностей точек плоскости.

Не вдаваясь в подробности, отметим только, что алфавит арифметических операторов содержит операторы: построения прямой, проходящей через две заданные точки; определения точки пересечения двух прямых; выбора произвольной точки на данном геометрическом объекте. Логические операторы проверяют, лежит ли точка на данной прямой, пересекаются ли прямые в данной точке.

Существуют задачи, неразрешимые в указанном алфавите операторов.

Покажем, что деление отрезка пополам невозможно с помощью одной только линейки. Пусть существует алгоритм построения середины данного отрезка с помощью линейки. В процессе построения к заданным точкам A, B добавляются точки C, D, ..., K, причем K является серединой отрезка AB.

Возьмем произвольную точку S, не лежащую в плоскости построения τ, и соединим ее со всеми точками A, B, C, D, ..., K. Получим пирамиду с вершиной S. Проведем плоскость τ' так, чтобы она рассекала пирамиду и не была параллельна заданному отрезку AB. Точки A, B, C, D, ..., K спроецируем с помощью точки S с плоскости τ на плоскость τ'. Получим точки A', B', C', D', ..., K'. Прямую l, проходящую через точки M, N в плоскости τ, спроецируем в прямую l', проходящую через соответствующие точки M', N' плоскости τ', а точку M пересечения двух прямых l_1, l_2 плоскости τ – в точку M' пересечения соответствующих прямых l_1', l_2' плоскости τ'. Поэтому логические операторы примут одинаковые значения на соответствующих элементах, а арифметические операторы доставят соответствующие элементы. Отсюда следует, что точки C', D', ..., K' можно рассматривать как точки, последовательно построенные в плоскости τ' из данных точек A', B' с помощью алгоритма деления отрезка пополам. Поэтому точка K' должна быть серединой отрезка $A'B'$. Но это невозможно, так как отрезок $A'B'$, согласно выбору плоскости τ', не параллелен отрезку AB, следовательно, K' не является серединой отрезка $A'B'$.

Если в алфавит операторов включить оператор построения окружности, задача деления отрезка пополам станет разрешимой, т. е. существует алгоритм решения этой задачи, и притом весьма простой.

Однако известно, что в расширенном алфавите операторов, т. е. в алфавите, обусловленном применением линейки и циркуля, неразрешимы задачи об удвоении куба, трисекции угла и квадратуре круга.

Алфавитным называют алгоритм, областью задания которого является множество всех слов в заданном алфавите. В отличие от алфавита операторов его называют *внешним алфавитом.* Внешний алфавит может быть конечным или бесконечным.

Значимость алфавитных алгоритмов обусловливается тем, что математические выкладки являются по существу преобразованиями математических текстов, состоящих из слов. Особенно это проявляется в сильно формализованных математических структурах.

В качестве примера алфавитных алгоритмов можно привести нормальный алгоритм Маркова, машину Тьюринга, МНР.

Пример 3.4. Внешний алфавит состоит из всех чисел и знака "+". Множество ω_0 содержит только упорядоченные последовательности чисел, разделенных знаками сложения. Построить алгоритм сложения для любого числа слагаемых в алфавите операторов $B = \{A_0, C, P\}$, где C осуществляет сложение двух первых чисел в слове; логический оператор P = 0, если слово состоит из более чем одного числа, и P = 1 в противном случае

Р е ш е н и е. На рис. 3.5 изображена граф-схема искомого алгоритма.

Опишем операцию над алгоритмами, называемую *композицией алгоритмов*, с помощью которой из одних алгоритмов можно составить другие алгоритмы.

Пусть в граф-схеме алгоритма α между концом M одной дуги и началом N следующей дуги записан оператор С. Вместо оператора С поместим граф-схему алгоритма β, для этого следует:

1) стереть оператор С;

2) отождествить (объединить) вершину M с входом алгоритма β;

3) отождествить вершину N со всеми или некоторыми вершинами A_0 алгоритма β, стерев при этом A_0.

В зависимости от выбранных для отождествления конечных вершин оператора A_0 алгоритма β построим различные граф-схемы.

Очевидно, что в результате композиции алгоритмов снова получают алгоритм.

Приведем наиболее важные типы композиции алгоритмов.

Последовательное соединение алгоритмов. В граф-схеме, изображенной на рис. 3.6, *а*, операторы A_1, A_2 замещают алгоритмами β_1, β_2. Для однозначности результата необходимо указать, какие выходы граф-схем β_1 следует отождествлять с входом A_2. Полученный алгоритм осуществляет последовательное выполнение алгоритмов β_1 и β_2.

Разветвление алгоритмов. В граф-схеме, приведенной на рис. 3.6, *б*, операторы A_1, A_2 замещают граф-схемами алгоритмов β_1, β_2. Полученный алгоритм осуществляет выполнение алгоритма β_1 или β_2 в зависимости от выполнения логического условия Р.

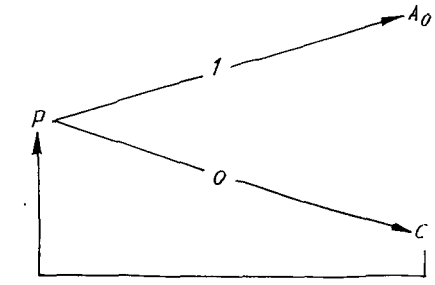


Рис. 3.5

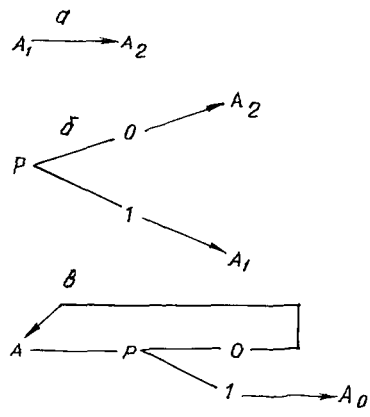


Рис. 3.6

Итерация алгоритма. В граф-схеме, изображенной на рис. 3.6, *в* , оператор А замещается алгоритмом β . Полученный алгоритм осуществляет выполнение алгоритма β до тех пор, пока не будет выполнено логическое условие Р.

3.2. НЕРАЗРЕШИМЫЕ ПРОБЛЕМЫ. УНИВЕРСАЛЬНЫЙ АЛГОРИТМ

Функцию $y = f(x)$ назовем *B-вычислимой,* если существует определяющий ее алгоритм в алфавите операторов B .

Если имеется в виду один алфавит B , то вместо "*B*-вычислимая" будем писать "вычислимая".

В данном конечном алфавите операторов B имеется конечное число граф-схем с количеством вершин, не превосходящим данного числа. Поэтому все граф-схемы в данном алфавите B образуют счетное множество. Следовательно, граф-схемы, а значит, и алгоритмы, можно пронумеровать. (Это справедливо и для счетного алфавита операторов.)

Теорема 3.1. *Если область задания алгоритмов бесконечна, то существуют невычислимые функции.*

Доказательство. Так как алгоритмов счетное множество, вычислимых функций тоже счетное множество, следовательно, их можно пронумеровать.

Пусть в последовательность $\varphi_1(x), \varphi_2(x), ..., \varphi_k(x), ...$ входят все функции. Выберем в ω счетное множество элементов $\epsilon_1, \epsilon_2, ..., \epsilon_k, ...$ Определим функцию:

$$f(x) = \begin{cases} \lambda, & \text{если } x \neq \epsilon_k \text{ при любом } k; \\ \delta_k, & \text{если } x = \epsilon_k \text{ и } \varphi_k(\epsilon_k) \text{ определено}; \\ \lambda, & \text{если } x = \epsilon_k \text{ и } \varphi_k(\epsilon_k) \text{ не определено}, \end{cases}$$

где λ , δ_k – элементы множества ω и $\delta_k \neq \varphi_k(\epsilon_k)$.

Очевидно, что $f(x) \neq \varphi_k(x)$ при любом k , следовательно, $f(x)$ не является *B*-вычислимой. Теорема доказана.

Если функции $z = f(y)$, $y = \varphi(x)$ вычислимы, то их суперпозиция $z = f(\varphi(x)) = \psi(x)$ – тоже вычислимая функция.

Алгоритм Γ_ψ , вычисляющий функцию $\psi(x)$, есть последовательное соединение $\Gamma_\varphi \Gamma_f$ алгоритмов для вычисления соответственно функций $\varphi(x)$ и $f(y)$.

Пусть между подмножеством κ алгоритмов в данном алфавите и подмножеством κ_0 элементов задано однозначное соответствие.

Если алгоритму α поставлен в соответствие элемент α^0 , то α^0 назовем *шифром алгоритма* α .

Алгоритм α либо применим к α^0 , либо неприменим к нему. В первом случае алгоритм α называют *самоприменимым,* во втором – *несамоприменимым.*

Проблему распознавания самоприменимости можно сформулировать так: по любому шифру определить, к какому типу относится соответствующий ему алгоритм.

Подмножество алгоритмов назовем *замкнутым,* если композиция алгоритмов принадлежит этому подмножеству.

Пусть κ – замкнутое подмножество алгоритмов в алфавите B и κ_0 – соответствующее множество шифров.

Теорема 3.2. *Если алгоритм α распознает самоприменимость алгоритмов из подмножества κ, то α не принадлежит κ.*

Доказательство. Предположим противное. Существует алгоритм $\alpha \in \kappa$, который любой самоприменимый шифр перерабатывает в σ_0, а всякий несамоприменимый шифр – в σ_1, где σ_0, σ_1 – два определенных элемента из множества ω.

По α построим такой алгоритм $\beta \in \kappa$, который несамоприменимый шифр преобразует по-прежнему в σ_1, а к самоприменимому шифру алгоритм β уже неприменим. (Этого можно достичь посредством композиции алгоритма α и алгоритма γ, граф-схема которого изображена на рис. 3.7, где $P(\sigma_0) = 0$, $P(\sigma_1) = 1$.)

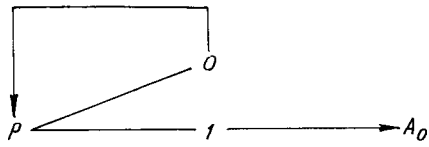


Рис. 3.7

Так как $\beta \in \kappa$, то шифр β^0 существует. Пусть алгоритм β самоприменим, т. е. элемент $\beta(\beta^0)$ определен. Но по построению β неприменим к β^0. Следовательно, получили противоречие.

Пусть теперь β – несамоприменимый алгоритм, т. е. $\beta(\beta^0)$ не определен. Но по построению $\beta(\beta^0) = \sigma_1$, т. е. опять получили противоречие. Теорема доказана.

Следствие. *Проблема распознавания самоприменимости алгоритмически неразрешима в классе всех алгоритмов исходного алфавита операторов.*

При доказательстве теоремы 3.2 предполагалось, что алгоритм γ, различающий буквы σ_0, σ_1, принадлежит κ, ибо σ_0, σ_1 различает исполнитель.

Таким образом, проблема распознавания самоприменимости алгоритмов может быть разрешима лишь для отдельных классов, принадлежащих множеству алгоритмов в данном алфавите, но при этом распознающий алгоритм для каждого класса не содержится в этом классе, если алгоритм γ принадлежит алфавиту B. Это относится и к другим проблемам, связанным с распознаванием самоприменимости алгоритма.

Весьма желательно иметь такой алгоритм α, который распознавал бы по данному алгоритму β и исходному элементу a_0, применим ли алгоритм β к элементу a_0. В построении такого алгоритма α заключается проблема останова.

Предположим, что алгоритм α существует. Пусть β – произвольный алгоритм. Тогда для β и $a_0 = \beta^0$ алгоритм α решал бы проблему распознавания самоприменимости. Полученное противоречие и доказывает неразрешимость проблемы останова в исходном алфавите операторов.

Для отдельных классов множества алгоритмов в данном алфавите проблема останова разрешима, но алгоритм распознавания не принадлежит самому классу.

Перейдем к рассмотрению универсального алгоритма. Пусть задана граф-схема. Запишем в начальных вершинах каждой простой дуги переменную p_0 и положим $p_0 = 0$. Пронумеруем все вершины граф-схемы. Обозначим через θ значения дуги (0 или 1), а через ν, r, s – соответственно номера операторов, вершин и переменных.

Дуга граф-схемы определяется парой (r, θ) – номером вершины r и значением θ дуги. Сама дуга определяет ν, r', s – номера оператора ν, конечной вершины r' дуги и ее переменной s соответственно, т. е. дуга характеризуется набором чисел (r, θ, ν, r', s).

Каждое число t набора (r, θ, ν, r', s) закодируем последовательностью из $t + 1$ единиц. Числа в наборе разделим нулем. Например, набор (3, 0, 2, 4, 1) получит код 1111010111011111011. Выпишем в строку все коды дуг, отделив каждый код двумя нулями. Такую строку назовем шифром алгоритма. Его можно считать натуральным числом в двоичной системе счисления или в какой-либо другой, с основанием $q \geqslant 2$. В результате каждый алгоритм a получит номер $N = a^0$, по которому он однозначно восстанавливается. Более того, укажем теперь алгоритм, который по номеру исходного алгоритма перерабатывает заданный элемент $a_0 \in \omega$ так же, как сам исходный алгоритм. Естественно назвать такой алгоритм *универсальным* для системы алгоритмов, определяемой данным алфавитом операторов.

Память универсального алгоритма содержит ячейки $r, \theta, p_0, p_1, \ldots, p_n, a$ для записи соответственно номера вершины, значения дуги, переменных и обрабатываемого элемента. В универсальном алгоритме имеется также память для шифра.

Пусть в память занесены $r, \theta, p_0, p_1, \ldots, p_n, a$ (в начале работы r равен номеру входа граф-схемы).

Исполнитель универсального алгоритма, просматривая последовательно шифр, находит в шифре пару чисел r, θ, прочитывает следующее за ними в шифре число ν, к элементу a применяет оператор, номер которого ν, и заносит результат либо в одну из ячеек p_i, либо в ячейку a. Затем он прочитывает следующие в шифре числа r', s и заносит r' в ячейку памяти r, а p_s – в ячейку θ. После этого исполнитель возвращается к началу шифра и повторяет процедуру.

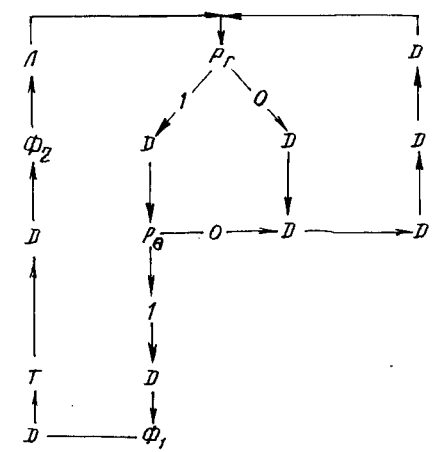


Рис. 3.8

На рис. 3.8 изображена граф-схема универсального алгоритма. Алфавит универсального алгоритма содержит, кроме операторов данной системы алгоритмов, еще операторы: D – перемещение исполнителя по шифру вправо на одно число t, т.е. перемещение к очередному числу после одного или двух нулей; Л – перемещение к началу шифра; Т – запись в памяти обозреваемого числа $r := r'$; P_r и P_θ – сравнения чисел, где $P_r = 1$, если

$t = r$, и $P_r = 0$, если $t \neq r$; $P_\theta = 1$, если $t = \theta$, и $P_\theta = 0$, если $t \neq \theta$; Φ_1 – выполнение оператора A_v по его номеру $t = v$; Φ_2 – присвоение $\theta := p_s$ по номеру s.

Для любого алгоритма α можно построить счетное множество шифров. Действительно, пусть алгоритм α определяется граф-схемой Γ_0. Возьмем произвольную граф-схему Γ в том же алфавите операторов, что и Γ_0, и будем пару Γ_0, Γ считать одной граф-схемой Γ_1. Если алгоритм β определяется граф-схемой Γ_1 с входом, совпадающим с входом граф-схемы Γ_0, то очевидно, что β совпадает с α, ибо в процессе вычисления по алгоритму β граф-схема Γ не участвует.

Итак, существует счетное множество граф-схем, определяющих один и тот же алгоритм. Поэтому одному алгоритму соответствует счетное множество шифров, а значит, и номеров.

Если операторы определяют систему алфавитных алгоритмов, то шифр можно закодировать словами внешнего алфавита. Для этого выберем две различные буквы внешнего алфавита, которые будем считать "нулем" и "единицей". Число t запишем в виде последовательности из $t + 1$ "единиц" и т. д. Таким образом, указан способ построения для алфавитного алгоритма α соответствующего ему слова α^0, т. е. шифра из области определения ω алгоритма. Для указанного соответствия можно сформулировать проблему распознавания самоприменимости алфавитных алгоритмов. Доказательство неразрешимости этой проблемы следует из теоремы 3.3.

Пусть среди операторов, определяющих данную систему алфавитных алгоритмов, есть операторы D, Л, Т (это имеет место, например, для машин Тьюринга, нормальных алгоритмов Маркова). В этом случае универсальный алгоритм системы принадлежит самой системе алгоритмов.

Назовем *тактом алгоритма* перемещение исполнителя по дуге граф-схемы и выполнение им оператора конечной вершины дуги.

Обозначим через $T_\alpha(x)$ необходимое число тактов для вычисления $\alpha(x)$. Рассмотрим алгоритмы в алфавите операторов B, принимающих только два значения: v_0 и v_1.

Пусть $f(x)$ – числовая функция и α^0 – шифр алгоритма α. Если $T_\alpha(\alpha^0) \leqslant f(\alpha^0)$ и $\alpha(\alpha^0) = v_0$, то говорят, что алгоритм α *f-самоприменим*. Алгоритм β *распознает f-самоприменимость*, если $\beta(\alpha^0) = v_1$, когда алгоритм α является f-самоприменимым, и $\beta(\alpha^0) = v_0$ в противном случае.

Теорема 3.3. *Если в исходном алфавите операторов существует алгоритм β, распознающий f-самоприменимость, то $T_\beta(\beta^0) > f(\beta^0)$, где β^0 – шифр алгоритма β.*

Доказательство. Предположим, что $\beta(\beta^0) = v_1$. Это означает, что алгоритм β f-самоприменим, что противоречит f-самоприменимости. Предположим теперь, что $\beta(\beta^0) = v_0$, т. е. алгоритм β f-несамоприменим. Случай $T_\beta(\beta^0) \leqslant f(\beta^0)$, $\beta(\beta^0) = v_0$ невозможен, ибо β не является f-самоприменимым. Следовательно, $T_\beta(\beta^0) > f(\beta^0)$. Теорема доказана.

Рассмотрим теперь класс алфавитных алгоритмов, алфавит операторов которого содержит также счетчик $\lambda := \lambda + 1$, инвертор И ($И(v_0) = v_1$, $И(v_1) = v_0$) и оператор P^0 сравнения натуральных чисел, т. е. слов, записанных буквами "нуль" и "единица".

Если в граф-схеме универсального алгоритма на дуге от оператора D к

оператору Φ_1 поместить счетчик (при начальном $\lambda = 0$), универсальный алгоритм будет вычислять еще и число тактов λ.

Если к счетчику последовательно подсоединить логический оператор

$$P^0 = \begin{cases} 0, \text{ если } \lambda = f + 1; \\ 1, \text{если } \lambda \neq f + 1 \end{cases}$$

и дугу $p^0 = 1$ направить к входу граф-схемы Φ_1, а вторую дугу $p^0 = 0$ – к оператору записи $a: = v_1$ и затем к A_0, то полученная граф-схема определит алгоритм Y_1, содержащий еще и обязательный останов после $\lambda = f$ тактов с результатом $a = v_0$.

Пусть $f(x)$ – вычислимая функция, значения которой – натуральные числа, т. е. слова из "нулей" и "единиц", и F – ее определяющий алгоритм. Тогда последовательное соединение $\beta = FY_1И$ есть алгоритм распознавания f-самоприменимости.

Алгоритм β принадлежит исходной системе алгоритмов, ибо ей принадлежит алгоритм Y_1.

Согласно теореме 3.3, имеем $T_\beta(\beta^0) > f(\beta^0)$, где β^0 – шифр алгоритма β. Но для алгоритма β можно указать счетное множество шифров. Поэтому неравенство $T_\beta(x) > f(x)$ выполняется для счетного множества чисел β_0. Следовательно, для числа тактов $T_a(x)$ нельзя указать в качестве границы вычислимую функцию $f(x)$, ибо имеется хотя бы один алгоритм, например алгоритм β, f-распознаваемости, для которого неравенство $T_\beta(x) \leqslant f(x)$ не выполняется на счетном множестве элементов x.

3.3. ПРИМЕР АЛФАВИТНОГО АЛГОРИТМА

Обозначим через ω множество всех слов в алфавите $T = \{+, -, \times, :, (,), a\}$, где a – любое число, например: 3,4+)×5,87:); (5 + 7) : 9 и т. д. Выделим подмножество ω_0 слов, называемых *формулами.*

Приведем *определение формулы.*

1. Любое число есть формула.

2. Если A и B – формулы, то $(A + B)$, $(A - B)$, $(A \times B)$, $(A : B)$ – тоже формулы.

3. Других формул, кроме определяемых п. 1, 2, нет.

Формулы $(a + b)$, $(a - b)$, $(a \times b)$, $(a : b)$, где a, b – числа, называют *элементарными.*

Если формула не является числом, она содержит хотя бы одну элементарную формулу. Определение формулы является по существу инструкцией по ее построению.

Слово $(((5{,}3 - 7) \times (7{,}87 : 0{,}25)) + (-7))$ есть формула, так как, согласно п. 1, числа 5,3; 7; 7,87; 0,25; (–7) – формулы; на основании п. 2 элементарными формулами будут (5,3–7), (7,87:0,25), а также слово $((5{,}3-7) \times (7{,}87:0{,}25))$.

И, наконец, согласно п. 2, заданное слово есть формула.

Рассмотрим оператор S, определенный только на множестве элементарных формул, кроме формулы вида $(a : 0)$. Каждой элементарной формуле оператор S ставит в соответствие число, при этом буквы +, –, ×, : считаются знаками арифметических операций. Например, S (5,3 + 7) = 12,3; S (5 × 3)=15.

Определим оператор (алгоритм) С на множестве слов:

1) исполнитель обозревает заданное слово $\lambda \in \omega$ и находит первое вхождение элементарной формулы;

2) применяет к элементарной формуле оператор S ;

3) стирает элементарную формулу и вместо нее записывает полученное число;

4) если в слове λ нет элементарных формул, то С неприменим к слову λ .

Так, например, после применения оператора С к слову $((3{,}5 + (4{,}8 \times 2)) - (7{,}03 - 4))$ получим слово $((3{,}5 + 9{,}6) - (7{,}03 - 4))$. К слову $(8 + (9{,}6 +) - 3$ оператор С неприменим, ибо нет вхождения элементарных формул.

Определим алфавитный алгоритм α с алфавитом операторов $\{P, C, A_0\}$, где логический оператор

$$P(\lambda) = \begin{cases} 1, \text{ если слово } \lambda \text{ состоит из одного числа;} \\ 0, \text{ если слово } \lambda \text{ состоит не только из одного числа.} \end{cases}$$

Граф-схема алгоритма α дана на рис. 3.5. Применим алгоритм α к формуле $((3{,}5 + (4{,}8 \times 2)) - (7{,}03 - 4))$. Так как P = 0, то, применив С, найдем формулу $((3{,}5 + 9{,}6) - (7{,}03 - 4))$, для которой P = 0. Очередное применение С доставит формулу $(13{,}1 - (7{,}03 - 4))$. Далее получим P = 0 и $(13{,}1 - 3{,}03)$; снова имеем P = 0 и формулу 9,8. Так как теперь P = 1, это число есть результат применения алгоритма α . Очевидно, что алгоритм α применим только к формулам и вычисляет определяемые ими числа.

В алгоритме α для очередного выполнения оператора С не нужно возвращаться к началу формулы. Ведь если исполнитель обозревает записанное им число, то левее этого числа не будет элементарных формул, так как оно записано вместо первого вхождения элементарной формулы. Очередное вхождение может либо содержать записанное число, либо быть расположенным правее этого числа. Например, после первого применения оператора С к формуле $((8 + ((7 + 3)(8 - 1))) - 15)$ получаем формулу $((8 + (10 \times (8 - 1))) - 15)$ и обозреваем число 10. Второй раз применяем оператор С к этой формуле, но уже правее числа 10. В результате имеем $((8 + (10 \times 7)) - 15)$. Третье применение С осуществляется на месте, т. е. в формуле с обозреваемым числом 7. Получаем $((8 + 70) - 10)$. Очередное применение С также осуществляется на месте. Имеем $(78 - 10)$. И, наконец, последнее применение С тоже осуществляется в формуле записанного ранее числа 78. В результате получаем 68.

Выясним вид формулы, которая обрабатывается исполнителем алгоритма α, не перемещающимся вправо по формуле после обнаружения первой элементарной формулы. Формулу с таким свойством назовем *простейшей*.

Пусть $(a * b)$ – первая элементарная формула и $a * b = A$, где $*$ – любой из четырех арифметических символов. После подстановки вместо $a * b$ числа A снова должна образоваться элементарная формула, ибо, по условию, очередное вычисление осуществляется на месте, т. е. с участием числа A . Такой формулой может быть либо $(A * c)$, либо $(c * A)$, где c – некоторое число. Отсюда следует строение простейших формул.

1. Слово $(a * b)$, где a, b – числа, есть простейшая формула.

2. Если A – простейшая формула и c – число, то $(A * c)$, $(c * A)$ – тоже простейшие формулы.

Легко доказать по индукции, что отличительная черта простейших формул заключается в том, что все левые скобки предшествуют всем правым скобкам, или, иначе, в наличии только одной элементарной формулы, например $(((a_5 \times (a_4 - ((a_1 + a_2) \times a_3))) - a_6) : a_7)$.

В настоящее время широкое распространение получили микрокалькуляторы. Нажатием клавиш в микрокалькулятор вводятся числа и операции. Но микрокалькулятор производит вычисления либо над одним числом (вычисление элементарных функций), либо над двумя числами (выполнение арифметических операций).

При последовательном введении в микрокалькулятор двух чисел a, b первое число a перемещается из регистра R_2 в регистр R_1, второе число b остается в регистре R_2. Число из R_2 фиксируется также на индикаторе. Результат вычисления элементарной формулы $a * b = A$ остается в R_2, но при введении очередного числа c само A перемещается в R_1. Поэтому на микрокалькуляторе можно производить вычисления формул вида $(A * c)$, где A – формула такого же вида.

Определим простейшие *левые формулы.*

1. Элементарная формула $(a * b)$ есть простейшая левая формула.

2. Если A – простейшая левая формула, то $(A * c)$, где c – число, – тоже простейшая левая формула.

Очевидно, что на микрокалькуляторе можно производить *цепочные вычисления*, т.е. вычисления без записи промежуточных результатов,только для простейших левых формул.

Отличительной чертой простейших левых формул является то, что все левые скобки расположены в начале формулы.

Если в микрокалькуляторе имеется оператор обмена содержимым регистров R_1 и R_2, возможны также вычисления формул вида $(c * A)$. Отсюда следует, что на таком микрокалькуляторе возможны цепочные вычисления только для простейших формул.

3.4. АДРЕСНЫЕ АЛГОРИТМЫ

При изучении общих свойств алгоритмов не учитывают природу элементов области их задания. Поэтому, согласно определению алгоритма, все операторы являются функциями одной переменной. Даже в алфавитных алгоритмах значениями функции (оператора) и ее переменной считаются слова произвольной длины.

Однако из-за требования "простоты" алфавитный оператор преобразует обычно не все слово, а только определенную его часть. Например, в нормальных алгоритмах Маркова преобразуется только та часть слова, которая указана в формуле подстановки; в машинах Тьюринга преобразуется обозреваемая буква.

В прикладных алгоритмах элементы области определения операторов бывают упорядоченными последовательностями переменных, но каждый конкретный оператор зависит от определенной части последовательности, т. е. является функцией определенных переменных.

Пусть теперь алфавит операторов содержит не только одновходовые операторы. В этом случае входы, а также выходы операторов необходимо идентифицировать, т. е. присвоить им и соответствующим переменным какие-то наименования. При этом одинаковые переменные получают одинаковые наименования.

Пусть имеется конечная совокупность *простых наименований,* или *идентификаторов.* Каждый идентификатор обозначается символом. Единственное свойство простых идентификаторов состоит во взаимной различимости в написании и, желательно, произношении (например, x , 75 , b , три, вес, дом, задача и т. п.). С каждым наименованием задается его область значений.

Сложное наименование представляет собой упорядоченный набор простых наименований. Составляющие этот набор простые идентификаторы, кроме первого, называются *индексами;* их областями значений являются целые числа. Сложное наименование называют *массивом*: одномерным, двумерным и так далее, в зависимости от числа индексов (например, a_{iv} , x_j , температура i и т. п.).

Следует отметить, что сложный идентификатор x_i всегда обслуживается *индексным оператором* I , который вычисляет очередное значение индекса i . Обычно индексный оператор выбирают таким: $i:=i+1$.

Простой или сложный идентификатор, в котором индексам присвоены целые значения, назовем *конкретным идентификатором* (z , код, a_{35} , y_7 . x_{-3}), остальные – *общими идентификаторами.*

Два оператора считаются равными, если они определяются одной функцией и соответствующие переменные обозначены одинаковыми идентификаторами. Например, операторы $y = a + x$, $z = a + c$ различны.

Иногда вход и выход оператора имеют одно и то же наименование. В подобных случаях вместо знака "=" (равенства) будем употреблять знак присваивания ":=", например $a:= f(x , a , b)$. Заметим, что у выхода логического оператора и соответствующей ему переменной один идентификатор.

Алгоритм, все переменные которого имеют наименования, назовем *символьным.*

Упорядочим все конкретные идентификаторы данного алгоритма. Для этого выпишем в строку все простые идентификаторы, затем идентификаторы с индексом 1, с индексом 2 и т. д. (Ради простоты изложения считаем, что все сложные идентификаторы – одноиндексные, ибо остальные сводятся к таковым.) Последовательность конкретных идентификаторов назовем *рабочей зоной алгоритма.*

Если присвоить идентификаторам рабочей зоны определенные значения, получим последовательность, которую назовем *конфигурацией.*

Те конкретные идентификаторы, которые обозначают только входы операторов, составляют *постоянную информацию,* поскольку она не изменяется в процессе счета. Некоторые идентификаторы выделены, и их совокупность называют *индикатором.* После прекращения счета значения идентификаторов индикатора считаются, согласно символьному алгоритму, *результатом счета.* Этот результат счета называют еще *выходной информацией.*

Чтобы осуществить вычисления по алгоритму, следует задать начальную конфигурацию или, иначе, *входную информацию,* которая содержит как часть постоянную информацию. Уточним инструкцию для исполнителя.

Инструкция для исполнителя символьных алгоритмов

1. Обозреть начальную вершину граф-схемы алгоритма.

2. Если в обозреваемой вершине записан оператор, то перейти к п. 3, в противном случае – к п. 13.

3. Если в обозреваемой вершине записан оператор A_0, то перейти к п. 14, в противном случае – к п. 4.

4. Если обозреваемый оператор содержит переменные, обозначенные общими идентификаторами, то перейти к п. 5, иначе – к п. 8.

5. Выбрать переменную с общим идентификатором.

6. В конфигурации найти значение индекса этого общего идентификатора.

7. Присвоить это значение соответствующему индексу общего идентификатора. (Получен конкретный идентификатор.) Перейти к п. 4.

8. (Переменные обозреваемого оператора обозначены конкретными идентификаторами.) В конфигурации найти значения идентификаторов и приписать их соответствующим идентификаторам обозреваемого оператора.

9. Если оператор не определен для полученных значений, то перейти к п. 16. В противном случае – к п. 10.

10. Вычислить значение обозреваемого оператора.

11. В конфигурации присвоить полученное значение (конкретному) идентификатору.

12. По единственной дуге переместиться к следующей вершине; перейти к п. 2.

13. Найти в конфигурации значение идентификатора логической переменной и переместиться к следующей вершине по дуге, соответствующей этому значению; перейти к п. 2.

14. Прекратить вычисления.

15. Прочесть результат вычислений на индикаторе.

16. Прекратить вычисления. Алгоритм неприменим к входной информации.

Если вычисления не прекращаются, то говорят, что алгоритм *неприменим к входной информации.*

Граф-схему, в которой вместо каждого обозначения оператора дано его описание, назовем *схемой алгоритма.* Описание оператора называют *блоком.* Вместо 0 или 1 на дугах пишут "Нет", "Да".

Пример 3.5. Построить символьный алгоритм сложения для любого числа слагаемых при условии, что алфавит операторов содержит операторы сравнения и сложения двух чисел.

Р е ш е н и е. Так как $y = \sum_{i=1}^{m-1} x_i$, то возьмем $y_1 = x_1$, $y_{i+1} = y_i + x_{i+1}$. Простые идентификаторы – i, y, сложный – x_i. Входная информация: $i = 1, y = 0, x_1 = a_1, \ldots, x_{m-1} = a_{m-1}$. Схема алгоритма изображена на рис. 3.9. Она содержит блок сложения $y := y + x_i$, индексный блок $i := i + 1$ и блок логического оператора $i = m$. Индикатором является идентификатор y.

Рассмотрим действия исполнителя. Он обозревает логический блок, находит в конфигурации $I = 1$, сравнивает 1 с m. Пусть $1 \neq m$, тогда исполнитель перемещается по дуге "Нет" к арифметическому блоку с оператором $y + x_i$, где y – простой идентификатор; x_i – сложный. Исполнитель находит в конфигурации $i = 1$, затем $x_1 = a_1$ и присваивает x_i значение a_1. Далее он находит в конфигурации $y \approx 0$, присваивает это значение пере-

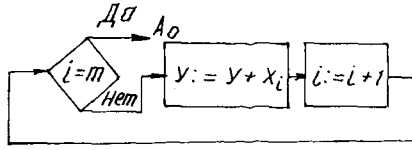


Рис. 3.9

менной y, вычисляет сумму $a_1 + 0 = a_1$, выполняет $y := a_1$ и заносит в конфигурацию $y = a_1$, стирая при этом $y = 0$. По единственной дуге исполнитель перемещается к блоку от переменной i, затем находит в конфигурации $i = 1$, вычисляет $1 + 1 = 2$ и выполняет $i := 2$. Затем исполнитель заносит в конфигурацию новое значение $i = 2$ вместо прежнего $i = 1$ и возвращается к логическому блоку. Далее он находит в конфигурации $i = 2$. Пусть $2 \neq m$, тогда исполнитель перемещается по дуге "Нет" к арифметическому блоку и т.д. При останове результат считывается у идентификатора y.

При композиции двух алгоритмов случается, что одинаковые наименования входят в разные алгоритмы и обозначают различные переменные. Поэтому прежде следует в одном из алгоритмов переименовать эти переменные.

В процессе счета после вычисления значения очередного оператора некоторому идентификатору будет присвоено новое значение. Можно сказать, что на каждом шаге вычислительного процесса, т. е. при вычислении в данном операторе, происходит преобразование конфигурации, а сам алгоритм преобразовывает входную конфигурацию в конечную. Таким образом, множество конфигураций и составляет область задания алгоритма.

Пример 3.6. Построить символьный алгоритм для машины Тьюринга в алфавите $\{0, 1\}$.

Р е ш е н и е. Оператором считывания содержимого ячейки является логический оператор

$$P = \begin{cases} 0, \text{ если } x_i \neq 1; \\ 1, \text{ если } x_i = 1. \end{cases}$$

Простыми идентификаторами будут i, d, p, сложным – x_i. Идентификаторы p, x_i принимают значения 0 или 1, $d = \{-1, 0, 1\}$, а i "пробегает" все целые числа: ..., –2, –1, 0, 1, 2, ...

Начальная конфигурация имеет вид $i = 0$, $x_0 = \tau_0$, ..., $x_k = \tau_k$. Остальные $x_i = 0$ при $i < 0$ или $i > k$. Идентификатор $i = 0$ соответствует ячейке, обозреваемой головкой в начальный момент.

Пусть в граф-схеме машины Тьюринга имеется куст (*рис. 3.10, а*). В схеме символьного алгоритма куст замещается схемой алгоритма (рис. 3.10, *б*).

Исполнитель обозревает логический блок со сложным идентификатором $x_i = 1$. Согласно инструкции, исполнитель находит в конфигурации значение $i = e$ и получает конкретный идентификатор x_e. Затем он находит в конфигурации $x_e = \tau$ и перемещается по дуге $p = \tau$. Далее выполняет $x_e := \delta_\tau$, $i := i + d_\tau$, заносит эти значения в конфигурацию, обозревает очередной логический блок и т. д.

Алфавит операторов символьного алгоритма для машины Тьюринга содержит минимум возможных операторов: оператор *присваивания* и оператор *сложения* целых чисел с $d = \{-1, 0, 1\}$.

Алгоритм, в котором каждый идентификатор – натуральное число, назовем *адресным*.

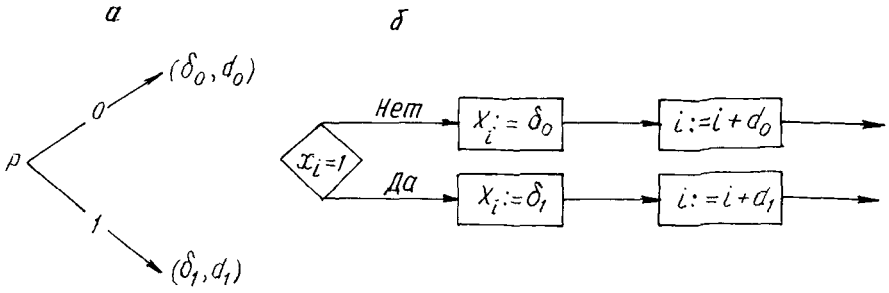


Рис. 3.10

Память для исполнителя адресных алгоритмов можно реализовать в виде бесконечной в одну сторону ленты, разделенной на ячейки, снабженные последовательными номерами. Эти номера называют *адресами ячейки.* Присваивание конкретному идентификатору n значения a ($n := a$) интерпретируется как запись a в ячейку по адресу n, т. е. с номером n. Примером адресного алгоритма может служить МНР.

Построение алгоритма проще проводить в содержательных идентификаторах, ибо человеку присуще неформальное мышление, но в ЭВМ реализуется адресный алгоритм, так как в машинах информация содержится в адресных ячейках. Поэтому рассмотрим переход от символьного алгоритма к адресному. При этом переходе все идентификаторы получают свои адреса (номера). По такому адресу при реализации алгоритма на вычислительной машине будут содержаться в ячейке памяти значения идентификатора.

В операторах идентификаторы заменяются адресами, т. е. символьный оператор $u = f(x, y, ..., z)$ записывают в виде $(u) := f((x), (y), ..., (z))$, где $(u), (x), (y), ..., (z)$ означают адреса ячеек, которые предназначены для $u, x, y, ..., z$.

Пусть, например, идентификаторы x, y_i, $3/7$, z получили соответственно адреса 5, 7, 2, 10. Оператор $z = (3/7)x + y_i$ запишем в виде адресного оператора $(10) := (2)(5) + (7)$. Чтобы выполнить этот оператор, следует извлечь содержимое ячеек 2, 5, 7, произвести над ними указанные действия и результат переслать в ячейку с адресом 10.

Рассмотрим, как осуществляется замена сложного идентификатора. Индексный оператор $I(i)$, вычисляющий очередное значение индекса i переменной x_i, будет замещен адресным оператором $V(v)$, который вычисляет адрес v ячейки очередного значения x_i.

Пусть j — адрес (номер), полученный идентификатором i. В отличие от других идентификаторов в ячейке j содержится не значение i, а очередной адрес v. Поэтому запишем $j := V(v)$. Это означает, что в ячейку с адресом j пересылается значение, вычисляемое в адресном операторе V по значению v, содержащемуся в ячейке j.

Пусть сложный идентификатор x_j получил адрес κ. Если оператор зависит от x_j, в ячейку κ нужно заслать число, содержащееся в ячейке, адрес ν которой находится в ячейке j. Для выполнения такой пересылки необходимо включить в алфавит операторов адресных алгоритмов сложный оператор переадре-

сации $T^0(j, \kappa)$. В случае, когда x_i – значение оператора, число, содержащееся в ячейке κ, нужно переслать в ячейку, адрес v которой находится в ячейке j. Для такой пересылки необходимо иметь сложный оператор переадресации $T^1(\kappa, j)$.

Таким образом, можно сделать следующее заключение. Чтобы перейти от символьного алгоритма к адресному, нужно всем идентификаторам, в том числе и числам, присвоить номера (адреса), индексные операторы заменить адресными, а номер j индекса i присвоить его адресу v.

Оператор присваивания $A := B$ замещается оператором переадресации $T(m, n)$, где n – адрес A; m – адрес B. Обращение к x_j замещается сложным оператором переадресации $T^0(j, \kappa)$ или $T^1(\kappa, j)$, где j – адрес v; κ – адрес сложного идентификатора x_i.

Пример 3.7. Построить адресный алгоритм сложения для любого числа слагаемых.

Р е ш е н и е. В примере 3.5 был построен символьный алгоритм сложения для любого числа слагаемых (см. рис. 3.9). Поэтому преобразуем только схему алгоритма. Для этого индексный оператор $i := i + 1$ заменяем адресным оператором $v := v + 1$. Идентификаторам присваиваем адреса. Одно из возможных распределений адресов указано в табл. 3.1. Число слагаемых равно $m - 1$. Вычисления прекратятся при адресе $v = m + 5$, поэтому в пятой ячейке помещаем число $m + 5$.

Таблица 3.1

Адрес	1	2	3	4	5	6	7	...	$5+(m-1)$
Идентификатор	y	v	x_i	1	$m+5$	a_1	a_2	...	a_{m-1}

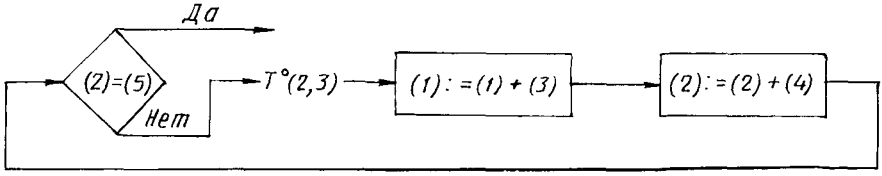


Рис. 3.11

Во второй ячейке находится адрес очередного x_i. В третью ячейку пересылаем очередное x_i. Операторы $y := y + x_i$, $i = m$, замещаем адресными операторами (1) := (1) + (3), (2) = (5) (заметим, что (1) := (1) + (3) означает, что в оператор сложения двух чисел пересылается содержимое первой и третьей ячеек, а результат засылается в первую ячейку). Аналогичное замещение выполняем и для остальных адресных операторов. Выбор x_i осуществляется оператором T^0 (2, 3). Схема алгоритма приведена на рис. 3.11. Индикатором является первая ячейка. Начальная конфигурация (входная информация) указана в табл. 3.2.

Таблица 3.2

Адрес	1	2	3	4	5	6	7	...	$5+(m-1)$
Содержимое	0	6	0	1	$m+5$	a_1	a_2	...	a_{m-1}

Обычно устанавливают следующее (наиболее простое) соответствие между индексом i и адресом v: $v = i + b$, где b – некоторое число.

Иногда наибольшие значения различных индексов заранее неизвестны, и поэтому нельзя указать отдельные зоны в памяти для соответствующих адресов. В этом случае можно выбрать такое соответствие между индексами и адресами: $v = ai + b$, где a – постоянная, равная числу индексов, b – некоторое число (для каждого индекса свое), причем разность двух таких чисел не должна делиться на a. (Если индекс i принимает и отрицательные значения, то b выбирается таким, чтобы $v > 0$.) При этом условии не будет общих ячеек (адресов v) для различных индексов. Например, в алгоритме имеются два индекса i_1, i_2, точнее, два сложных идентификатора x_{i_1}, y_{i_2}. Так как $a = 2$, то $v = 2i + b$. Для адресов имеем: $v_1 = 2i_1 + b_1$, $v_2 = 2i_2 + b_2$, где b_1, b_2 – любые числа, такие, что $b_1 - b_2$ не делится на 2. Для одного индекса адреса будут четные, для другого – нечетные, например $v_1 = 2i_1 + 7$, $v_2 = 2i_2 + 24$.

Пример 3.8. Построить адресный алгоритм для машины Тьюринга.

Р е ш е н и е. Достаточно указать блок адресного алгоритма, соответствующий фрагменту символьного алгоритма, изображенного на рис. 3.10, *б*.

Индексный оператор имеет вид $i := i + d$. Но взять адресный оператор такого вида нельзя, потому что i может быть любым целым числом, а адрес v принимает только неотрицательные значения. Поэтому неотрицательным значениям i поставим в соответствие нечетные v, а отрицательным i – четные v по формуле

$$v = \psi(i) = \begin{cases} 2i + 1, \text{ если } i \geqslant 0; \\ -2i, \text{ если } i < 0. \end{cases}$$

Теперь по $\varphi(i) = i + d$ и $\psi(i)$ следует найти адресный оператор $v := \psi(\varphi(\psi^{-1}(v)))$. Формальные преобразования здесь затруднительны. Проще найти оператор, учитывая смысл функций φ и ψ.

Если $i \geqslant 0$, $i + d \geqslant 0$ или $i < 0$, $i + d < 0$, то из $\psi(i)$ следует, что $v := v + 2d$ для нечетных v и $v := v - 2d$ для четных v. Пусть $\epsilon(v) = -1$, если v – четное число, и $\epsilon(v) = 1$, если v – нечетное число. Объединим две формулы для v в одну: $v := v + \epsilon \cdot 2d$. Обозначим $c = v + \epsilon \cdot 2d$. Вычислим c для $i = 0$, $d = -1$. Так как $i = 0$, $v = \psi(0) = 1$, $\epsilon(1) = 1$, то $c = 1 + 2(-1) = -1$. Очередное v, согласно формуле для $\psi(i)$, в случае $i = -1$ будет равно 2, т. е. $v \neq c$. Аналогично найдем, что для $i = -1$, $i + d = 0$ также $v \neq c$ и $c = 0$, $v = 1$. Следовательно,

$$v = \begin{cases} 2, \text{ если } c = -1; \\ 1, \text{ если } c = 0; \\ c, \text{ если } c > 0. \end{cases}$$

Четность v меняется только при $c = 0$. Поэтому

$$\epsilon := \begin{cases} \epsilon, \text{ если } c \neq 0; \\ -\epsilon, \text{ если } c \neq 0. \end{cases}$$

Для формулы, вычисляющей очередной адрес v, укажем символьный алгоритм (рис. 3.12).

Теперь следует оператор $i := i + d$ в блоке, приведенном на рис. 3.10, *б*, заменить схемой алгоритма, изображенной на рис. 3.12, и полученную символьную схему преобра-

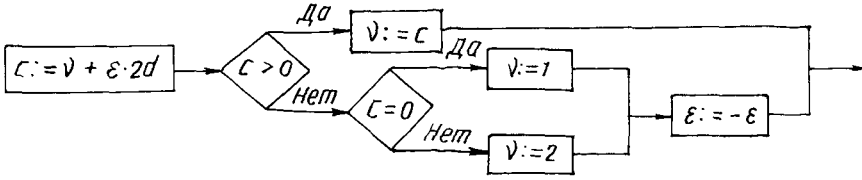


Рис. 3.12

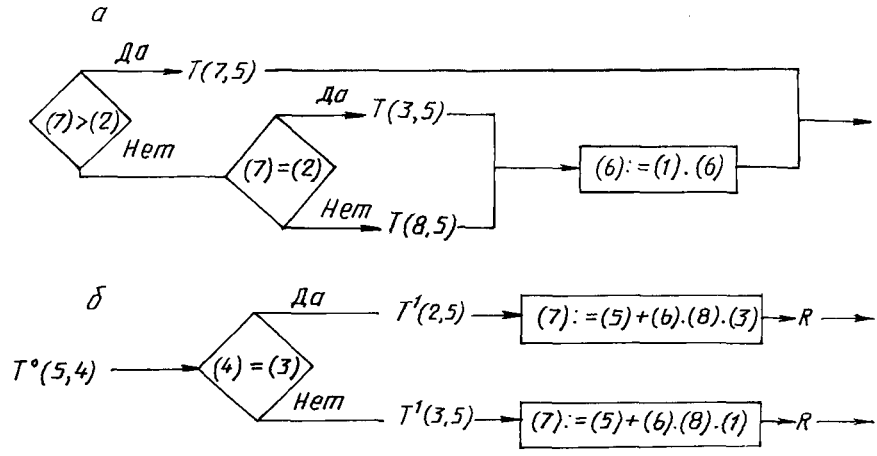


Рис. 3.13

зовать во фрагмент адресного алгоритма. Эта операция сводится к замене идентификаторов и чисел адресами. В табл. 3.3 указано распределение адресов для идентификаторов и чисел.

Таблица 3.3

Адрес	1	2	3	4	5	6	7	8
Идентификатор	−1	0	1	x_i	v	ϵ	c	2

На рис. 3.13, *а* изображен адресный блок R, соответствующий символьному блоку, приведенному на рис. 3.12, но без первого оператора. На рис. 3.13, *б* изображена искомая схема, соответствующая схеме, данной на рис. 3.10. Для определенности здесь $\delta_0 = 1$, $\delta_1 = 0$, $d_0 = -1$, $d_1 = 1$. Начальная конфигурация дана в табл. 3.4. Входной информацией заполнено k ячеек с адресами от 9 до $8 + k$.

Таблица 3.4

Адрес	1	2	3	4	5	6	7	8	9	10	...	$8+k$
Содержимое	−1	0	1	–	9	0	–	2	a_1	a_2	...	a_k

Выше были рассмотрены такие адресные алгоритмы, у которых сложные идентификаторы являются только одноиндексными. Если идентификатор

многоиндексный: $x_{i_1, i_2, \ldots, i_k}$, всегда можно перейти к одноиндексному идентификатору, взяв в качестве индекса j пеановскую функцию $j = \pi_k(i_1, i_2, \ldots, i_k)$, которая определяется следующим образом:

$$\pi_2(i_1, i_2) = i_1 + ((i_1 + i_2)(i_1 + i_2 + 1))/2;$$

$$\pi_k(i_1, \ldots, i_{k-1}, i_k) = \pi_2(\pi_{k-1}(i_1, i_2, \ldots, i_{k-1}), i_k).$$

Нетрудно доказать, что при этом осуществляется инъективное соответствие между наборами $(i_1, i_2, \ldots, i_k)$ и π_k.

3.5. РАЗРАБОТКА АЛГОРИТМОВ

После построения алгоритма решения задачи, точнее, класса однотипных задач, всегда возникают вопросы: правильно ли построен алгоритм, верно ли решает он задачу? Для их решения следует проверить правильность алгоритма. Естественная и наиболее простая проверка – *тестирование,* которое и применяется на практике. Суть тестирования заключается в том, что, имея некоторую совокупность правильно решенных задач, решают их затем с помощью алгоритма и убеждаются в его истинности. Если имеются расхождения, то алгоритм соответственно исправляют. Такое тестирование только повышает уверенность в истинности алгоритма, но не может служить ее доказательством (это напоминает попытки доказать теорему методом неполной математической индукции).

Можно ли строго математически доказать истинность алгоритма? При такой общей постановке вопроса следует категорически ответить: "Истинность алгоритма доказать нельзя". Дело в том, что задание на построение алгоритма решения действительно важной для практики задачи определяется не формально. Дедуктивное доказательство истинности какого-нибудь предложения может быть проведено только в рамках самой математики. Если понятие алгоритма есть объект математический, то условие задачи содержит нематематические объекты.

Прежде чем построить алгоритм, задачу формализуют, т. е. строят для нее математическую модель. Но в таком случае истинность алгоритма проверяется не для поставленной задачи, а для ее модели, что далеко не одно и то же. Математическая модель и алгоритм являются математическими объектами, и поэтому существует объективная возможность доказательства соответствия построенного алгоритма модели. Так как число различных моделей потенциально неограничено, указать основные положения доказательства истинности алгоритма невозможно.

Сделаем следующий вывод: доказывать истинность алгоритма можно относительно некоторого формального описания задачи, предшествующего самому алгоритму. В общей теории алгоритмов понятие алгоритма является первичным, ибо нет понятия, которое предшествовало бы ему и по существу являлось его описанием. Поэтому в рамках общей теории возможен вопрос об истинности данного алгоритма относительно другого алгоритма, в истинности которого нет никакого сомнения и построение которого не требует сложных умозаключений. Таким первоначальным алгоритмом решения поставленной

задачи является, по нашему мнению, дерево-схема или близкая к ней граф-схема. В частности, это может быть дерево, заданное в виде *таблицы*.

Если построение граф-схемы проведено каноническим методом по исходной дерево- или граф-схеме, то доказательства истинности построенного алгоритма не требуется, ибо, как было доказано, построенная граф-схема эквивалентна исходной.

Процесс построения алгоритма состоит в определении по условиям задачи алфавита операторов и последующем синтезе граф-схемы в алфавите операторов. На начальном этапе операторы не обязательно должны совпадать с операторами данной ЭВМ или алгоритмического языка. Они могут быть более элементарными задачами, подлежащими решению, или известными уже алгоритмами. Выделение алфавита операторов проводится после анализа условия поставленной задачи и выполняется совместно с некоторой "прикидкой" ее решения. Окончательный выбор алфавита операторов происходит в процессе построения исходной граф-схемы.

Логические операторы отображают не только закономерность алгоритмизируемого процесса, но также условия, вытекающие из его математического описания. Логические операторы следует упорядочить. Часто относительный порядок логических операторов продиктован условиями задачи. Если возможны варианты, следует зафиксировать одну определенную последовательность операторов.

Каноническим методом синтеза получают правильные простые граф-схемы. Суть метода, как отмечалось в гл. 1, заключается в обнаружении эквивалентных вершин граф-схемы и последующем объединении. Поэтому алгоритм, получаемый каноническим методом, будет структурно эквивалентен исходному алгоритму.

Подчеркнем еще раз, что каноническим методом синтеза получают алгоритм с уже заданной структурой, которую автор алгоритма определил посредством построенной им начальной граф-схемы.

Проще всего в качестве исходной граф-схемы строить дерево-схему, последовательно выбирая дуги с операторами. При этом проводится анализ для каждой полученной вершины. Если условия, следующие из задачи, в данной построенной вершине M_1 и уже имеющейся вершине M_2 одни и те же, то с вершины M_1 построение не продолжают и вершины M_1 и M_2 обозначают одной меткой. Исходная граф-схема должна быть адекватна задаче, ибо никакие последующие методы преобразования граф-схемы не исправят допущенные ошибки.

Любую схему алгоритма можно записать в виде таблицы. В самом деле, если дана схема алгоритма, ее можно представить в виде граф-схемы. При наличии в граф-схеме контуров их разъединяют и проставляют в вершинах "разреза" две одинаковые метки. В результате получают граф-схему без контуров, но с метками. Затем все общие вершины разъединяют и получают дерево. Если на какой-нибудь ветви встречаются одинаковые логические операторы P_ν, то одному из них присваивают другой индекс. Полученное дерево записывают в виде таблицы.

Рассмотрим построение таблиц для алгоритмов.

Если в процессе счета значения логических переменных не изменяются, то на любой ветви дерева будет не более одного куста каждой переменной. По-

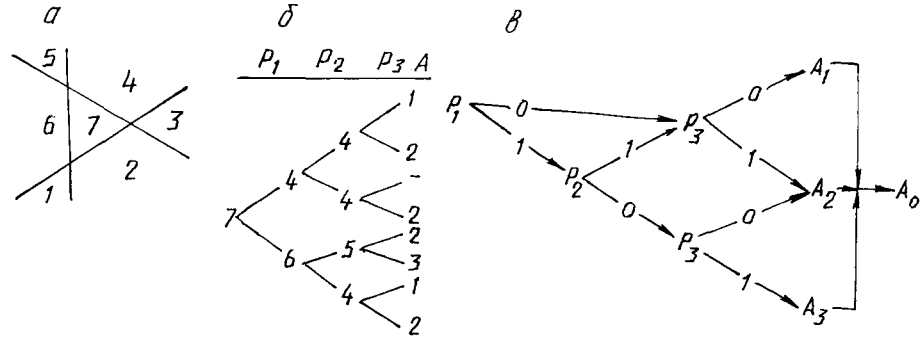


Рис. 3.14

этому в дереве имеется столько путей, сколько существует различных наборов значений переменных. Следовательно, дерево можно задать таблицей.

Пример 3.9. Функция от двух переменных $z = f(x, y)$ вычисляется в зависимости от положения точки $M(x, y)$ по трем различным формулам: 1) $z = x^2 + y$; 2) $z = x^2 + xy$; 3) $z = x^2 - y^2$. Плоскость разбита на семь частей полуплоскостями (рис. 3.14, *а*):

$$2y - x + 3 \geqslant 0; \quad y + x - 2 \geqslant 0; \quad x - 1 \geqslant 0;$$
$$2y - x + 3 < 0; \quad y + x - 2 < 0; \quad x - 1 < 0.$$

Функция вычисляется: по первой формуле в 1-й и 5-й частях, по второй в 2, 3, 4, 6-й частях, по третьей формуле в 7-й части. Построить алгоритм, вычисляющий функцию $z = f(x, y)$.

Р е ш е н и е. Введем арифметические операторы A_1, A_2, A_3, вычисляющие функцию соответственно по первой, второй и третьей формулам и логические операторы P_1, P_2, P_3, где

$$P_1 = \begin{cases} 0, \text{ если } 2y - x + 3 < 0; \\ 1, \text{ если } 2y - x + 3 \geqslant 0; \end{cases} \qquad P_2 = \begin{cases} 0, \text{ если } y + x - 2 < 0; \\ 1, \text{ если } y + x - 2 \geqslant 0, \end{cases}$$

$$P_3 = \begin{cases} 0, \text{ если } x - 1 < 0; \\ 1, \text{ если } x - 1 \geqslant 0. \end{cases}$$

В табл. 3.5 будут четыре столбца: p_1, p_2, p_3, A. Добавим еще столбец N – столбец номеров частей плоскости. Столбцы p_1, p_2, p_3 заполним заранее так, чтобы таблица имела стандартный вид. Столбец N заполним в соответствии со значениями наборов переменных (p_1, p_2, p_3). Столбец A заполним по столбцу N согласно условиям задачи. Проставим индексы i операторов A_i. Выпишем столбец A и построим определяемую им каноническую таблицу (рис. 3.14, *б*). Доопределим A в третьей строке таблицы так, чтобы получить минимальное число различных номеров. На следующем этапе получим графсхему из четырех кустов (рис. 3.14, *в*).

Таблица 3.5

N	p_1	p_2	p_3	A	N	p_1	p_2	p_3	A
1	0	0	0	1	6	1	0	0	2
2	0	0	1	2	7	1	0	1	3
–	0	1	0	–	5	1	1	0	1
3	0	1	1	2	4	1	1	1	2

Несколько сложнее проводится построение таблицы в случае, когда наблюдается возврат вычислений к операторам предыдущих столбцов таблицы. Рассмотрим построение таких таблиц.

Пусть после анализа задачи, кроме арифметических, были выделены логические операторы P_1, P_2, ..., P_n, расположенные в указанном порядке. Заметим, что различные P_i могут обозначать одинаковые операторы, но расположенные в разных местах последовательности операторов, реализующей вычислительный процесс. Такое обозначение следует из условия и способа решения поставленной задачи.

Таблица от n переменных содержит $2n + 1$ столбцов: N_0, P_1, N_1, P_2, ..., P_n, N_n и 2^n строк. Столбцы P_1, P_2, ..., P_n заполняются заранее, согласно правилам получения таблицы стандартного вида. Такая "заготовка" облегчает заполнение столбцов $N_0, N_1, ..., N_n$.

В столбце N_0 записываются операторы (если они есть), которые выполняются перед оператором P_1.

Пусть построены столбцы N_0, N_1, ..., N_{k-1} и в столбце N_k уже записаны значения до j-й строки. По условию задачи проводится анализ j-й строки для значений $p_1, p_2, ..., p_k$. При этом возможны три случая.

1. Если выполнения оператора не требуется, в столбце N_k ставят прочерк.

2. Если вычисления прекращаются сразу или после выполнения операторов, то в столбце N_k последним записывают A_0.

3. Если последующие вычисления совпадают с вычислениями, которые начинаются с оператора С (переменной x), уже имеющимися в таблице, то в столбце N_k записывают букву Z с меткой и эту же метку записывают перед С (переменной x).

Итак, строки завершаются либо A_0, либо Z.

В остальных строках с теми же значениями $p_1, p_2, ..., p_k$ проставляют точки, что означает совпадение значений в столбце N_k для этих строк.

Представление алгоритма таблицей или, что то же, дерево-схемой гораздо проще непосредственного построения на интуитивном уровне схемы алгоритма, после которого требуется отладка (исключение ошибок), значительно превышающая по времени само построение.

Построение граф-схемы по таблице формализовано в виде канонического метода, который был изложен в гл. 1.

Пример 3.10. Построить алгоритмы обнаружения первого вхождения в бесконечном слове в алфавите $\{0, 1\}$:

1) слова $\mu =$ 01011;

2) одного из слов: $\mu_1 =$ 0010, $\mu_2 =$ 0100, $\mu_3 =$ 1010, $\mu_4 =$ 1110;

3) одного из слов: $\mu_1 =$ 0010, $\mu_2 =$ 0110, $\mu_3 =$ 1010, $\mu_4 =$ 1110.

Отметим, что исполнитель может перемещаться вправо вдоль заданного слова и распознавать обозреваемую им букву.

Р е ш е н и е. 1. Составим алфавит операторов $\{A_0, D, P_1, P_2, P_3, P_4, P_5\}$, где A_0 – останов; D – перемещение вдоль слова на одну букву вправо; P_i – логический оператор, который для i-й буквы x вхождения μ принимает значение $P_i(x) = x$, причем все P_i по содержанию совпадают.

Составим табл. 3.6 от двоичных переменных p_1, p_2, p_3, p_4, p_5 и арифметических операторов A_0, D. В таблице будет 32 строки. Но, так как для 16 строк с $p_1 = 1$ будет один и тот же результат, оставим вместо них только одну строку.

Таблица 3.6

N_0	p_1	N_1	p_2	N_2	p_3	N_3	p_4	N_4	p_5	N_5
	0	②D	0	Z^2						
	0	·	0	·						
	0	·	0	·						
		·		·						
	0	·	0	·						
	0	·	0	·						
	0	·	0	·						
	0	·	0	·						
①	0	·	0	·						
	0	·	1	D	0	③D^2	0	Z^2		
	0	·	1	·	0	·	0	·		
	0	·	1	·	0	·	1	D	0	Z^3
	0	·	1	·	0	·	1	·	1	A_0
	·	·	·	·						
	0	·	1	·	1	DZ^1				
	0	·	1	·	1					
	0	·	1	·	1					
	0	·	1	·	1					
	1	DZ^1								

Таблица 3.7

p_1	N_1	p_2	N_2	p_3	N_3	p_4	N_4
0	D	0	①D	0	Z^1		
0	·	0	·	0	·		
0	·	0	·	1	D	0	A_0
0	·	0	·	1	·	1	Z^2
0	·	1	D	0	D	0	A_0
0	·	1		0	·	1	Z^4
0	·	1	·	1	Z^2		
0	·	1	·	1	·		
1	D	0	③D	0	Z^1		
1	·	0	·	0	·		
1	·	0	·	1	④D	0	A_0
1	·	0	·	1	·	1	Z^2
1	·	1	②D	0	Z^3		
1	·	1	·	0	·		
1	·	1	·	1	D	0	A_0
1	·	1	·	1	·	1	Z^2

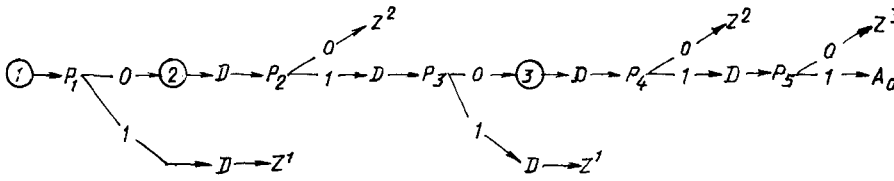


Рис. 3.15

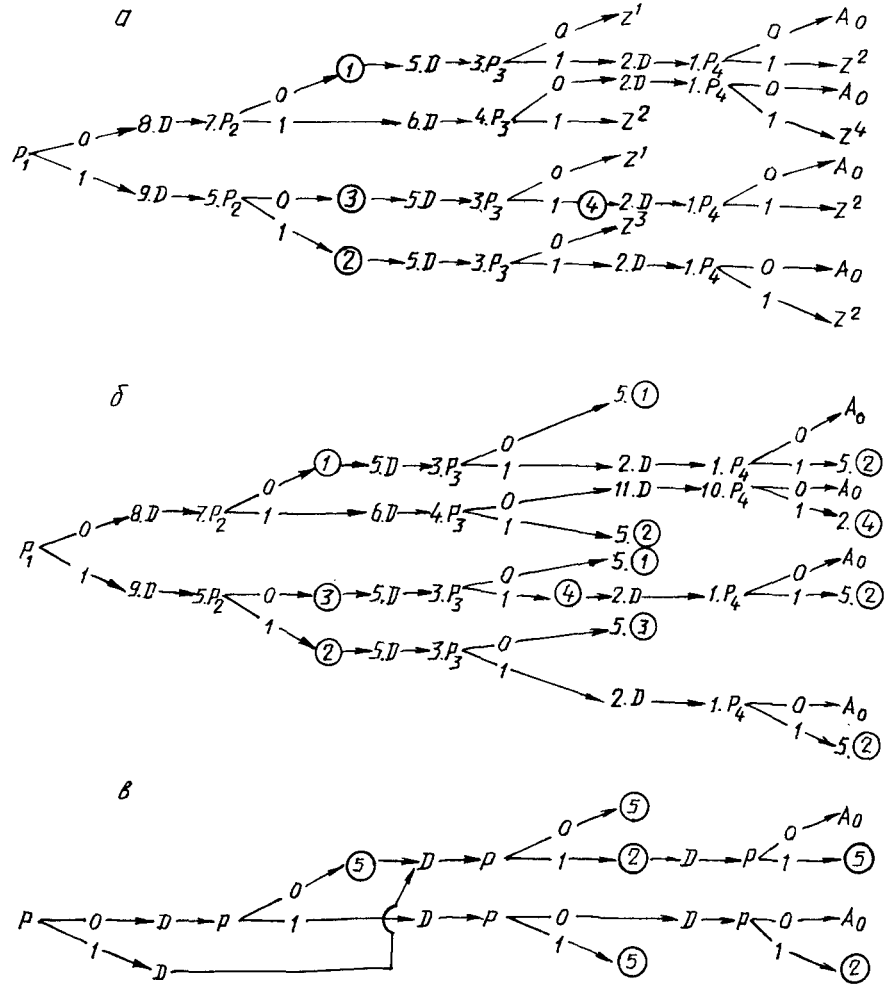


Рис. 3.16

После чтения буквы всегда следует перемещение. Поэтому в столбцах N_i будет записан оператор D. В строке с $p_1 = 1$ после D исполнитель возвращается к оператору P_1, ибо поиск вхождения μ начинается заново для бесконечного слова. Так как имеется возврат к оператору P_1, то после оператора D записываем Z с меткой *1*, а саму метку *1* записываем в столбце N_0. В строке с $p_1 = 0, p_2 = 0$ по тем же соображениям счет возвращается к оператору D столбца N_1. Поэтому в первой строке записываем метку *2*. В остальных строках с $p_1 = 0, p_2 = 0$ проставляем по точке. Запись в остальных столбцах для этих строк не требуется. В оставшихся строках для $p_3 = 1$ после D счет возвращается к оператору P_1. Поэтому после D в столбце N_3 записываем Z с меткой *1*. Продолжаем заполнять таблицу для строк с $p_3 = 0$. При $p_4 = 0$ вычисления возвращаются к оператору D в столбце N_1. Поэтому Z присваиваем метку *2* и продолжаем заполнение строк для $p_4 = 1$. При $p_5 = 1$ обнаруживаем вхождение μ. Записываем A_0 в столбце N_5. При $p_5 = 0$ вычисления возвращаются к D в столбце N_3, ибо поиск вхождения μ начинается после 010. Поэтому перед соответствующим оператором D записываем новую метку *3*, а в столбце N_5 записываем Z^3.

По таблице строим дерево (рис. 3.15). Далее нумерацию не проводим, так как очевидно, что здесь упрощения не будет.

Напомним, что в граф-схеме алгоритма операторы или Z с одинаковыми метками отождествляются, что приводит к образованию циклов.

2. По условию задачи с помощью аналогичных рассуждений составляем табл. 3.7. Например, в первой строке при $p_1 = 0, p_2 = 0, p_3 = 0$ проверка начинается со слова, у которого первые две буквы – нули. Поэтому в столбцы N_2 и N_3 заносится метка *1*.

По табл. 3.7 строим дерево с первоначальной нумерацией, при которой метки не учитываются (рис. 3.16, *а*). После переноса номеров 5 и 2 в конечные вершины соответственно 1, 2, 3 и 4 с метками образовываются противоречивые номера двух пар $(A_0, 5)$ и $(A_0, 2)$. После перенумерации противоречивых номеров не оказалось. Следовательно, получена искомая каноническая таблица (рис. 3.16, *б*). Соответствующая граф-схема изображена на рис. 3.16, *в*. Она содержит по шесть операторов распознавания буквы и операторов перемещения.

3. По условию задачи составляем табл. 3.8.

По табл. 3.8 строим дерево с первоначальной нумерацией, при которой метки не учитываются (рис. 3.17, *а*). После переноса номеров 4 и 2 в конечные вершины соответственно с метками *1, 2* и *4, 3* противоречивых номеров не оказалось. Искомая граф-схема изображена на рис. 3.17, *б*.

Конечно, при анализе условия третьей задачи можно заметить, что останов обусловливается здесь только наличием одного слова 10, и, использовав это свойство, непосредственно получить простую граф-схему. Но важно то, что канонический метод формально выявляет это свойство. Также и во второй задаче формально выявлена логика задачи, и тем самым получена граф-схема с шестью операторами. Вряд ли непосредственный анализ условия задачи привел бы к подобной логике и ее использованию при построении схемы алгоритма.

Замечание. Равные операторы у выходов куста можно стереть и записать один из них у входа куста только в том случае, если этот оператор коммутирует с логическим оператором входа куста.

В последнее время в практике программирования получили распространение структурированные граф-схемы. Рассмотрим синтез таких граф-схем.

Вершину слияния всех конечных вершин куста (рис. 3.18), где A, B – операторы или граф-схемы, назовем *нулевой*.

Граф-схему, у которой нет вершин слияния, отличных от нулевых, назовем *простой структурированной*.

Таблица 3.8

p_1	N_1	p_2	N_2	p_3	N_3	p_4	N_4
0	D	0	①D	0	Z^1		
0	·	0	·	0	·		
0	·	0	·	1	D	0	A_0
0	·	0	·	1	·	1	Z^4
0	·	1	D	0	Z^2		
0	·	1	·	0	·		
0	·	1	·	1	④D	0	A_0
0	·	1	·	1	·	1	Z^3
1	D	0	②D	0	Z^1		
1	·	0	·	0	·		
1	·	0	·	1	D	0	A_0
1	·	0	·	1	·	1	Z^4
1	·	1	D	0	Z^2		
1	·	1	·	0	·		
1	·	1	·	1	③D	0	A_0
1	·	1	·	1	·	1	Z^3

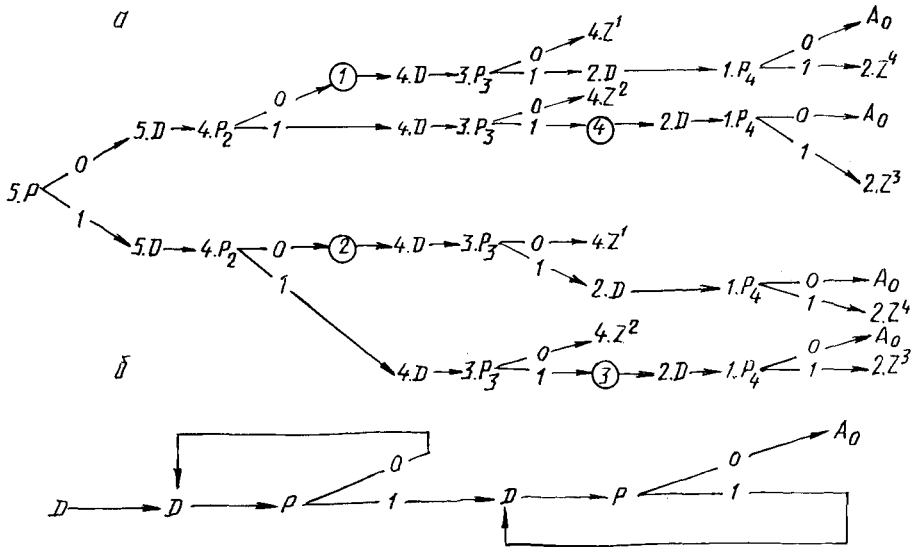


Рис. 3.17

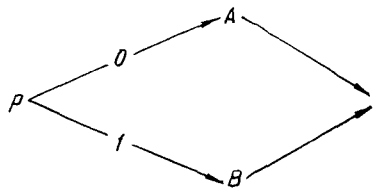


Рис. 3.18

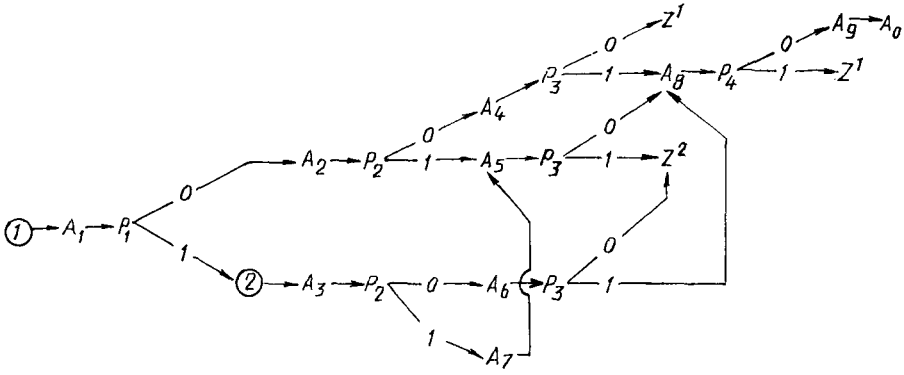


Рис. 3.19

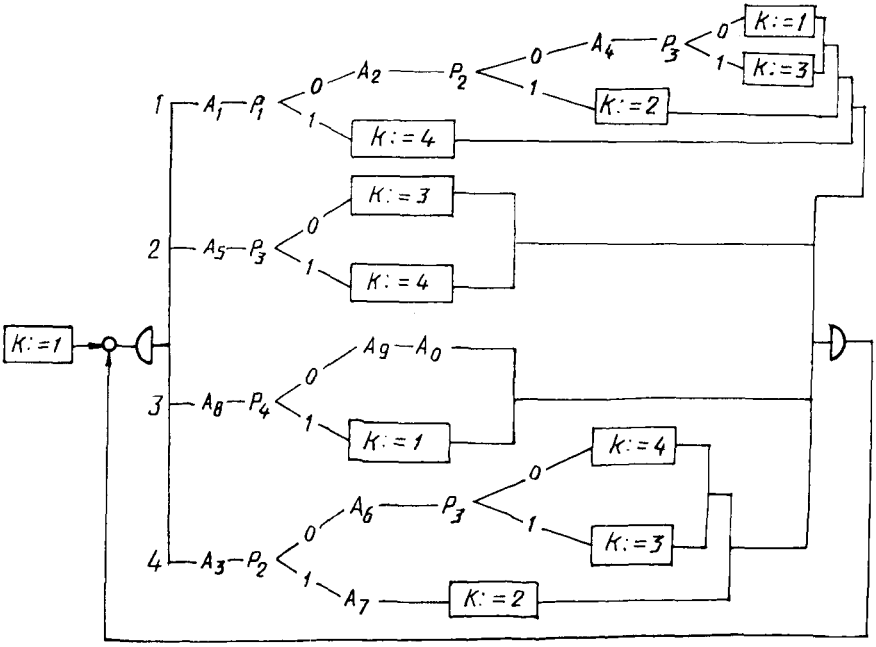


Рис. 3.20

Пусть имеется N пронумерованных от 1 до N простых структурированных граф-схем, у которых все или часть предшествующих A_0 арифметических операторов являются операторами переадресации вида $k := m$, где $1 \leqslant m \leqslant N$. Совокупность N таких простых структурированных граф-схем, играющих роль подпрограмм, назовем *структурированной граф-схемой (программой)*.

Любую граф-схему можно представить в виде структурированной граф-схемы. Для этого достаточно продублировать ненулевые вершины слияния, что приведет к увеличению объема граф-схемы.

Другой вариант представления состоит в использовании операторов переадресации $k := m$, где m – номер составляющей граф-схемы.

Правила построения структурированной программы по граф-схеме

1. Приписать номер 1 оператору начальной вершины граф-схемы.

2. Последовательно пронумеровать, начиная с цифры 2, все ненулевые вершины слияния. (Полученные номера будут значениями параметра k. Каждая из пронумерованных вершин является началом некоторой граф-схемы, конечными вершинами которой являются либо оператор A_0, либо пронумерованные вершины.)

3. Изобразить эти N граф-схем в порядке возрастания их номеров, записав в конечной вершине с номером m оператор $k := m$.

4. Входы граф-схем с указанными номерами объединить вертикальной линией. Выходы также объединить вертикальной линией.

В результате получают структурированную граф-схему, ибо в граф-схемах (подпрограммах) нет ненулевых вершин слияния.

Согласно этим правилам, на граф-схеме, изображенной на рис. 3.19, построена структурированная граф-схема (рис. 3.20).

Отметим, что при построении структурированной граф-схемы возможна ситуация, когда часть объединенных вершин дублируется.

3.6. ПРОГРАММЫ И ПРОГРАММИРОВАНИЕ

Прежде чем реализовать алгоритм на ЭВМ, следует представить его в виде программы.

Перейдем к рассмотрению вопросов программирования. Пусть дана некоторая граф-схема. Изобразим ее так, чтобы вершины ее располагались в одной строке и первой слева была бы начальная вершина граф-схемы. Такую граф-схему назовем *программой*, а сам процесс перехода от граф-схемы к программе – *программированием*.

Будем говорить, что программа задана на алгоритмическом или машинном языке, если алфавит операторов состоит из операторов языка или машинных команд.

Программа без дополнительных условий не представляет никакого интереса, потому что она по сравнению с граф-схемой не содержит дополнительной информации. Изучать программу независимо от граф-схем, считая ее первичным понятием, не имеет смысла, ибо по сравнению с граф-схемой в программе связи между операторами просматриваются с трудом. Еще в большей степени это относится к построению программы независимо от граф-схемы.

Значительный интерес представляют программы с различными условиями. Рассмотрим следующее условие.

I. Если в программе нелогическая вершина соединена дугой со следующей непосредственно за ней справа вершиной, то эта дуга аннулируется.

Более важным является такое условие.

II. Если в программе вершина соединена дугой со следующей непосредственно за ней справа вершиной, то эта дуга аннулируется.

Иначе говоря, условие I мы распространили на логические вершины.

Все оставшиеся дуги называются *операторами безусловного перехода,* а дуги куста – *операторами условного перехода.*

Если программа содержит, кроме начальной, еще k вершин, то будет $k!$ различных программ, определяемых граф-схемой. Возникает задача построения программы с наименьшим числом операторов безусловного и условного переходов.

Вершину граф-схемы, в которую входит более одной дуги, называют *вершиной слияния.* В вершину слияния программы может входить на одну дугу меньше, чем в граф-схему, ибо две вершины одной из этих дуг можно записать в программе рядом, исключив при этом дугу.

Дугу, началом которой является логическая вершина, назовем *Л-дугой*; дугу, конечной вершиной которой служит вершина слияния, – *С-дугой*; остальные дуги – *простыми дугами.*

В программе содержатся по крайней мере дуги, входящие во вход граф-схемы; дуги каждой вершины слияния, кроме одной; при условии I – обе *Л*-дуги каждого куста, а при условии II – по одной *Л*-дуге. Далее будем рассматривать программирование при выполнении условия II .

Чтобы нагляднее представить процесс построения программы, разобьем его на два этапа. На первом этапе белые дуги граф-схемы окрашиваются, например, в красный или зеленый цвет. Буквы красной дуги в программе записываются рядом, и красные дуги, естественно, аннулируются. Остаются только зеленые дуги. Учитывая это, можно сформулировать следующие необходимые требования к окраске дуг программы: если одна дуга куста красная, то вторая должна быть зеленой; в вершину слияния может входить не более одной красной дуги; дуги, входящие в начальную вершину граф-схемы, должны быть зеленого цвета.

Теорема 3.4. *Множество красных дуг граф-схемы состоит из непересекающихся максимальных путей.*

Доказательство. Пусть вершина B принадлежит красной дуге. Она не может быть общей для трех красных дуг, так как две из них либо сходятся к вершине B , либо выходят из нее. В первом случае B – вершина слияния, во втором – логическая вершина. Но у таких вершин не может быть более одной красной дуги, т. е. получили противоречие.

Аналогично доказывается, что если B – общая вершина двух красных дуг, то они имеют одинаковые направления. Но тогда B – вершина одного максимального пути. Теорема доказана.

Опишем *процедуру перехода от граф-схемы алгоритма к его программе.*

1. Окрасить в зеленый цвет дуги, входящие в начальную вершину граф-схемы.

2. Окрасить в красный цвет простые дуги.

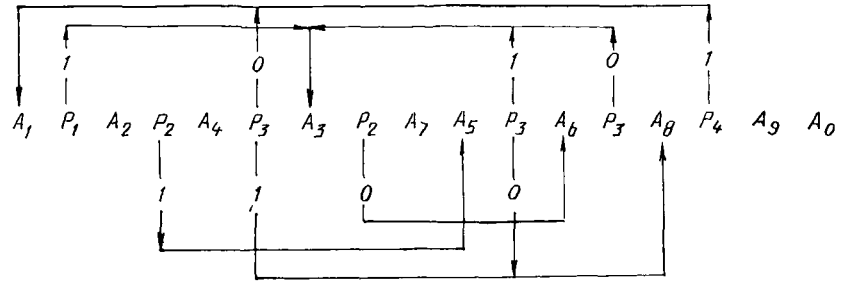


Рис. 3.21

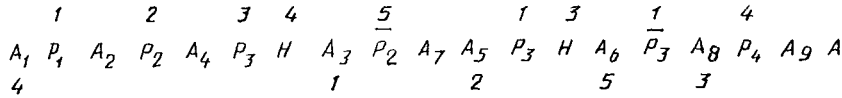


Рис. 3.22

3. Окрасить в красный цвет только одну *С*-дугу каждой вершины слияния. Остальные *С*-дуги окрасить в зеленый цвет. (При этом окраску проводить так, чтобы зеленых кустов было возможно меньше.)

4. В кусте с хотя бы одной белой дугой окрасить дуги в разные цвета.

5. Записать в строку операторы максимальных красных путей, взяв в качестве первого путь с оператором входа. Сохранить при записи только зеленые дуги.

Пример 3.11. Написать программу, определяемую граф-схемой, приведенной на рис. 3.18.

Р е ш е н и е. После окраски дуг граф-схемы образуются три максимальных красных пути. Расположим операторы путей в строку. Получим одну из возможных программ (рис. 3.21).

При написании программы часто требуется, чтобы в ней отсутствовали нулевые *Л*-дуги. Это условие следует учитывать уже при выполнении п. 3, 4.

При наличии зеленой нулевой *Л*-дуги оператор P заменяют его отрицанием $\overline{P}$. Если имеется зеленый куст, то посередине нулевой дуги следует отметить вершину, записать в ней букву H (*буферный оператор*) и перекрасить часть дуги до H в красный цвет.

Для большей наглядности дуги в программе не изображают. Вместо них расставляют метки: сверху буквы – метка начала дуги, снизу – метка конца дуги.

На рис. 3.22 изображена программа без нулевых *Л*-дуг, составленная по граф-схеме, приведенной на рис. 3.19.

Уточним окраску дуг граф-схем с целью оптимизации программы по числу операторов безусловного и условного переходов.

Вначале опишем процедуру окраски *Л*-дуг граф-схемы.

1. Окрасить в зеленый цвет *Л*-дуги, которые входят в начальную вершину граф-схемы.

2. Окрасить в красный цвет каждую *Л*-дугу, которая не входит в вершину слияния. Вторую дугу куста окрасить в зеленый цвет.

3. Если есть белая *Л*-дуга, которая входит в вершину слияния вместе с красной дугой, окрасить ее в зеленый цвет. Если есть белая *Л*-дуга, которая входит в вершину слияния в отсутствие красной дуги, окрасить ее в красный цвет. Вторую дугу куста окрасить в зеленый цвет.

Все *Л*-дуги образуют некоторый *Л*-граф, в котором две дуги двоичного куста считаем одним неориентированным ребром, т. е. логические вершины не считаются вершинами этого *Л*-графа. Ребра *Л*-графа – либо зеленые, либо двухцветные (половина ребра зеленая, половина красная).

Если *Л*-граф – несвязный, рассматриваем его связные компоненты. Так как последние не имеют общих вершин, то окраска каждой отдельной компоненты не ограничивает окраску других компонент. Поэтому максимальное множество красных дуг *Л*-графа состоит из суммы максимальных множеств красных дуг отдельных компонент.

Вершину графа назовем *особой*, если все входящие в нее дуги зеленого цвета.

Для получения наибольшего числа красных дуг включим в процедуру окраски *Л*-дуг граф-схемы еще один пункт.

4. Если есть простая цепь двухцветных ребер, которая соединяет особую вершину A и вершину B зеленого ребра, то у всех двухцветных ребер простой цепи поменять цвета *Л*-дуг. У зеленого ребра *Л*-дугу вершины B окрасить в красный цвет.

Теорема 3.5. *Окраска, проводимая согласно п. 1–4, доставляет максимальное число красных Л-дуг.*

Доказательство. Пусть в связной компоненте имеются особая вершина A и зеленое ребро, инцидентное вершине B. Найдем простую цепь $AD_1D_2...D_kB$. Обе *Л*-дуги последнего ребра D_kB – зеленые, у остальных ребер первая дуга зеленая, вторая – красная. (В противном случае одну из вершин D_i взяли бы в качестве B.) Окрасим цепь согласно п. 4 процедуры окраски. Этим мы исключим особую вершину и зеленое ребро, а в остальных вершинах цепи сохраним по красной дуге. Повторением п. 4 в компоненте исключают или особые вершины, или зеленые ребра.

Пусть в отдельной компоненте нет зеленых ребер. Следовательно, в ней нет зеленых двоичных кустов, и число красных *Л*-дуг максимально. Предположим теперь, что в компоненте нет особых вершин. Это означает, что в любую вершину компоненты входит одна красная *Л*-дуга, т. е. максимальное число красных *Л*-дуг. Теорема доказана.

Присоединив к каждой компоненте все белые *С*-дуги, которые входят в ее вершины, получим *ЛС*-граф. Белые *С*-дуги одной компоненты не ограничивают окраску дуг других компонент, потому что они не входят в вершины других компонент. Следовательно, максимальное множество красных дуг *ЛС*-графа состоит из суммы максимальных множеств красных дуг каждой компоненты.

Сформулируем п. 5, 6 процедуры окраски белых дуг *ЛС*-графа.

5. Если в вершину вместе с красной *Л*-дугой входят белые *С*-дуги, то окрасить их в зеленый цвет. В противном случае одну *С*-дугу окрасить в красный, остальные *С*-дуги – в зеленый цвет.

В результате получим четыре типа вершин:

1) особая вершина, когда в нее входят только зеленые *Л*-дуги;

2) в вершину входит красная *Л*-дуга и не входят *С*-дуги;

3) в вершину входят красная *Л*-дуга и зеленые *С*-дуги;

4) в вершину входит красная *С*-дуга.

6. Если есть простая цепь из двухцветных ребер, которая соединяет вершины *A*, *B* соответственно первого и третьего типов, то в каждом ребре цепи поменять цвета составляющих ее *Л*-дуг и окрасить в красный цвет одну *С*-дугу, инцидентную вершине *B* .

Лемма 3.3. *При окраске дуг, проводимой согласно п. 1–6, в компоненте ЛС-графа не будет одновременно вершин первого и третьего типов.*

Доказательство. Пусть в компоненте имеются вершины *A*, *B* первого и третьего типов соответственно.

Так как в компоненте нет зеленых ребер (поскольку есть особая вершина *A*), то цвета половинок ребер в простой цепи $AD_1D_2 \ldots D_kB$ чередуются и последняя дуга (половинка ребра D_kB) – красная. Уточним окраску согласно п. 6, исключив таким образом первый и третий типы вершин *A*, *B*, не изменяя при этом типы вершин D_1, D_2 , ..., D_k . Общее, т. е. максимальное, число красных *Л*-дуг не изменилось. Вершина третьего типа при этом преобразовалась в вершину четвертого типа. Таким образом, с помощью подобной окраски в компоненте можно исключить либо первый, либо третий тип вершины. Лемма доказана.

Мы не рассмотрели еще вершины, которые не принадлежат *ЛС*-графу, т.е. в которые входят только белые дуги. Окраска граф-схемы завершается операциями 7, 8.

7. Одну белую дугу вершины слияния окрасить в красный цвет, остальные дуги – в зеленый.

8. Оставшиеся простые дуги окрасить в красный цвет.

Рассмотренные п. 1–8 составляют *инструкцию по окраске дуг граф-схемы.*

Теорема 3.6. *Программа, построенная согласно инструкции по окраске дуг граф-схемы, содержит минимальное число операторов как безусловного, так и условного перехода.*

Доказательство. Пусть в компоненте *ЛС*-графа нет вершин первого типа, т. е. могут быть вершины только второго, третьего и четвертого типов. Это значит, что в каждую вершину входит красная дуга, т. е. их число максимально.

Предположим теперь, что в компоненте есть вершина первого типа. Следовательно, в компоненте могут быть еще вершины второго и четвертого типов, что обеспечивает максимальное число красных *С*-дуг. Так как в компоненте также максимальное число красных *Л*-дуг, то в каждой компоненте, а следовательно, и в *ЛС*-графе, общее число красных дуг максимально. Остальные дуги граф-схемы окрашены согласно п. 7, 8, при этом к их числу добавляется максимальное число красных дуг. Следовательно, в граф-схеме минимальное число как зеленых дуг, так и зеленых двоичных кустов. Это значит, что построенная граф-схема содержит минимальное число операторов как безусловного, так и условного перехода. Теорема доказана.

Поставим теперь условие, чтобы число дуг (зеленых), направленных в программе справа налево, было наименьшим. Наличие одной части таких дуг обусловлено существованием циклов, а наличие другой части – расположением красных путей, поэтому следует выбрать оптимальное их расположение.

Граф, который получают стягиванием каждого максимального красного пути граф-схемы в вершину, назовем *основным*. Начальной вершиной основного графа считаем вершину, соответствующую начальной вершине граф-схемы. Контур в основном графе свидетельствует о наличии в программе дуги, направленной справа налево.

Если мы пронумеруем вершины основного графа, то тем самым укажем порядок расположения частей в программе.

Пусть в основном графе нет контуров. Номера ρ для его вершин определим следующим образом:

1) $\rho(M_0) = 1$, где M_0 – начальная вершина;

2) $\rho(M) = \max\{\rho(M_1), \rho(M_2), ..., \rho(M_k)\} + 1$, где $M_1M, M_2M, ..., M_kM$ – дуги, входящие в вершину M.

Дуги программы в этом случае будут направлены вправо, кроме тех дуг конечных вершин, которые обеспечивают образование обратных связей.

Исследуем теперь вопрос об образовании контуров основного графа.

Лемма 3.4. *Если в результате стягивания некоторых дуг граф-схемы образовался контур, то среди стягиваемых дуг обязательно имеется Л-дуга.*

Доказательство. Стягивание нескольких дуг осуществляется последовательным стягиванием каждой из этих дуг. Пусть при стягивании дуги AB образовался контур. Значит, направление AB противоположно направлению контура, и дуга AC образует с AB двоичный куст. Следовательно, стягиваемая дуга AB есть *Л*-дуга, а вторая *Л*-дуга (AC) куста принадлежит контуру. Лемма доказана.

Следствие. *Основной граф при условии программирования I не содержит контуров.*

Исследуем возможность образования контуров не для исходной граф-схемы, а для граф-схемы из двоичных кустов, полученной стягиванием всех не *Л*-дуг. Если контур может образоваться в исходной граф-схеме, он образуется и в полученной граф-схеме. Ведь стягивание дуг контура только видоизменяет его (иногда до петли). Исчезнуть контур не может, ибо, как следует из леммы 3.4, он обязательно содержит *Л*-дугу.

Теорема 3.7. *В граф-схеме, состоящей только из двоичных кустов, можно в каждом кусте отметить такую дугу, что при различных вариантах стягивания отмеченных дуг контур не образуется.*

Доказательство. Пронумеруем логические вершины. Пусть у $k-1$ первых двоичных кустов выбраны такие *Л*-дуги, что их стягивание образует граф G_{k-1} без контуров.

Рассмотрим логическую вершину M_k и две ее *Л*-дуги M_kA, M_kB. Предположим, что при стягивании каждой дуги образуются контуры. Каждый из контуров проходит через вторую *Л*-дугу (согласно лемме 3.4). Следовательно, граф G_{k-1} имеет два пути: от A к B и от B к A, т. е. в G_{k-1} есть контур, что противоречит условию о G_{k-1}; поэтому двух контуров быть не может. Таким образом, из двух дуг M_kA, M_kB можно отметить такую (часто обе), что ее стягивание вместе с уже ранее отмеченными $k-1$ дугами не образует контур. Теорема доказана.

Заметим, что при стягивании любых $m\,(m < k)$ отмеченных *Л*-дуг контур также не образуется. Действительно, если бы образовался контур, при стяги-

вании остальных $k - m$ отмеченных дуг он бы сохранился, так как состоит из отмеченных дуг.

При раскраске граф-схемы с целью получения программы с наименьшим числом дуг, направленных справа налево, следует в красный цвет окрашивать только отмеченные *Л*-дуги. При этом может не быть достигнут минимум числа зеленых дуг и зеленых кустов. Поэтому рассмотрим другое построение такой программы.

После окраски граф-схемы согласно п. 1–8 инструкции поместим буферный оператор H в середину каждой красной *Л*-дуги *MA*, которая, согласно теореме 3.7, не была отмечена. Вторую половину (дугу *HA*) окрасим в зеленый цвет. Так как H записан в простой вершине, стягивание красной *Л*-дуги *MH* не приведет к образованию контура.

3.7. РЕАЛИЗАЦИЯ АЛГОРИТМОВ

Выполнение алгоритма осуществляется перемещением исполнителя по граф-схеме и вычислением в операторах. Поэтому техническая реализация алгоритма сводится к моделированию его граф-схемы переключательной схемой, в которой каждый оператор (буква алфавита) замещен исполнительным устройством, реализующим оператор.

Переключательной схемой можно реализовать только граф-схемы без контуров, ибо нельзя осуществить "движение" тока по контуру. Следует еще учитывать, что исполнительные устройства не включаются в цепь последовательно. Поэтому только элементарные граф-схемы реализуются переключательными схемами. Напомним, что при этом каждый куст граф-схемы моделируется переключательным узлом, а буквы у выходов – соответствующими исполнительными элементами.

Покажем, что и в общем случае моделирование граф-схемы сводится к моделированию элементарных граф-схем. Но вначале рассмотрим реализацию последовательного соединения элементарных граф-схем.

Пример 3.12. Реализовать граф-схему, изображенную на рис. 3.23, *а* .

Р е ш е н и е. Присваиваем входу номер 1, а вершине соединения – номер 2 и представляем заданную граф-схему в виде эквивалентной граф-схемы с обратными связями (рис. 3.23, *б*). Остальным выходам присваиваем номер 1, ибо очередные вычисления должны начинаться с входа граф-схемы.

Обратные связи реализуются специальными устройствами. При контактной реализации такими устройствами являются электромагнитные реле. Для реализации обратных связей необходимо номера $M = \{1, 2\}$ закодировать числами в двоичной системе счисления так, как указано в табл. 3.9.

q	M
0	1
1	2

По табл. 3.9 строим элементарную граф-схему Γ_q и соединяем каждый ее выход M с входом M граф-схемы, изображенной на рис. 3.23, *б* . Получаем элементарную граф-схему Γ_c от переменных q , y , x , у выходов которой, кроме букв a , b , c , записаны коды Q (рис. 3.23, *в*).

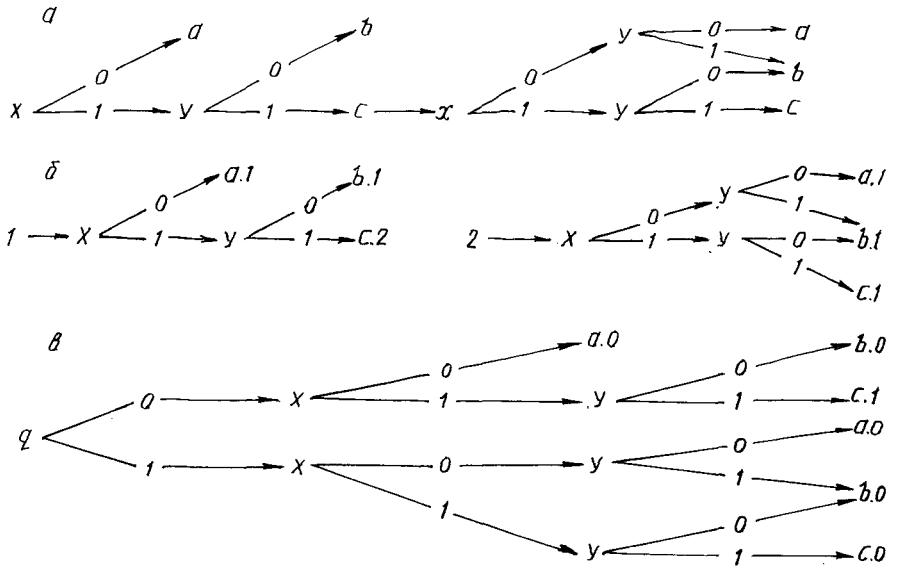


Рис. 3.23

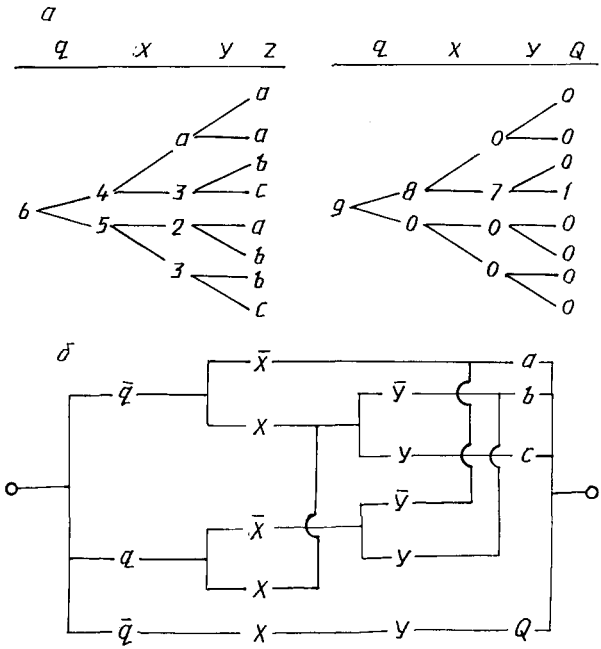


Рис. 3.24

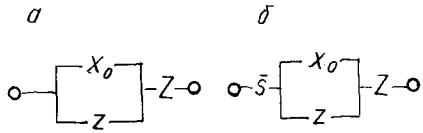


Рис. 3.25

Граф-схема Γ_c определяет две функции: z и Q. Функция z принимает значения a, b, c, а функция Q – значения 0, 1. По Γ_c можно составить таблицу для функций z, Q, а затем синтезировать две граф-схемы. Составлять таблицу не обязательно. Выпишем из Γ_c два столбца выходов и построим сразу канонические таблицы (рис. 3.24, *а*). Контактная реализация показана на рис. 3.24, *б*. Как видим, второй набор значений x, y в случае, если первый набор $x = 1$, $y = 1$, реализует цепь у второго входа ($q = 1$). В остальных случаях цепь будет реализована у первого входа.

При реализации алгоритма могут быть поставлены дополнительные технические условия, например временные, для исполнительных устройств. Считаем, что эти условия реализуются в самих исполнительных устройствах, а не в переключательной схеме.

Если работа переключательной схемы начинается с нажатия пусковой кнопки x_0, то параллельно схеме подключают схему, приведенную на рис. 3.25, *а*, а последовательно схеме – контакт z.

Если при технической реализации ставится еще и условие отключения электрической схемы после окончания работы по алгоритму, то в граф-схеме алгоритма должен быть указан оператор A_0. Он реализуется реле S, контакт которого $\bar{s}$ включается последовательно с пусковой схемой (рис.3.25, *б*).

Рассмотрим теперь техническую реализацию алгоритма. Если граф-схема содержит контуры, следует перейти к граф-схеме с обратными связями. Для этого находят опорные вершины и "разрезают" по ним контур. Вместо одной вершины получают две: конечную и внутреннюю. Обе вершины нумеруют одним номером (меткой). Исполнитель алгоритма из конечной вершины с номером возвращается во внутреннюю с тем же номером. Таким образом, будет осуществлен исходный алгоритм.

В полученной граф-схеме с обратными связями выделяют составляющие ее элементарные граф-схемы. Для этого вначале нумеруют вход граф-схемы и все буквы, кроме букв конечных вершин. Каждая пронумерованная вершина определяет элементарную граф-схему, образованную всеми исходящими из вершины отрезками путей. Отличительной особенностью таких отрезков является то, что они содержат только одну букву в конце.

В полученной системе Γ^* элементарных граф-схем входы и выходы пронумерованы, за исключением выходов A_0. В зависимости от технических условий выходу A_0 можно присвоить новый или уже имеющийся номер.

Некоторые элементарные граф-схемы бывают определены для различных наборов переменных (ветвей), поэтому такие граф-схемы можно совместить в одну, сохранив только один номер входа. При этом на выходах граф-схем другие номера входа заменяются оставшимися номерами. Каждому номеру присваивают двоичные коды: $(q_1, q_2, ..., q_k)$ для входов и $(Q_1, Q_2, ..., Q_k)$ для выходов. Нулевой код присваивается входу исходной граф-схемы. Коды $(q_1, q_2, ..., q_k)$ и соответствующие номера записывают в таблицу, по кото-

рой строят граф-схему Γ_q, и последовательно соединяют ее с системой элементарных граф-схем Γ^*. В полученной граф-схеме Γ_c номера выходов заменяют соответствующими кодами $(Q_1, Q_2, ..., Q_k)$. По граф-схеме строят $k+1$ каноническую таблицу для функций Y, Q_1, Q_2, ..., Q_k, которые реализуют переключательной схемой.

Пример 3.13. Реализовать контактной схемой алгоритм, представленный граф-схемой, изображенной на рис. 3.26, *а*.

Р е ш е н и е. В граф-схеме имеются два контура. В общей вершине слияния выполняем "разрез" и присваиваем номер 1 трем полученным вершинам: одной внутренней и

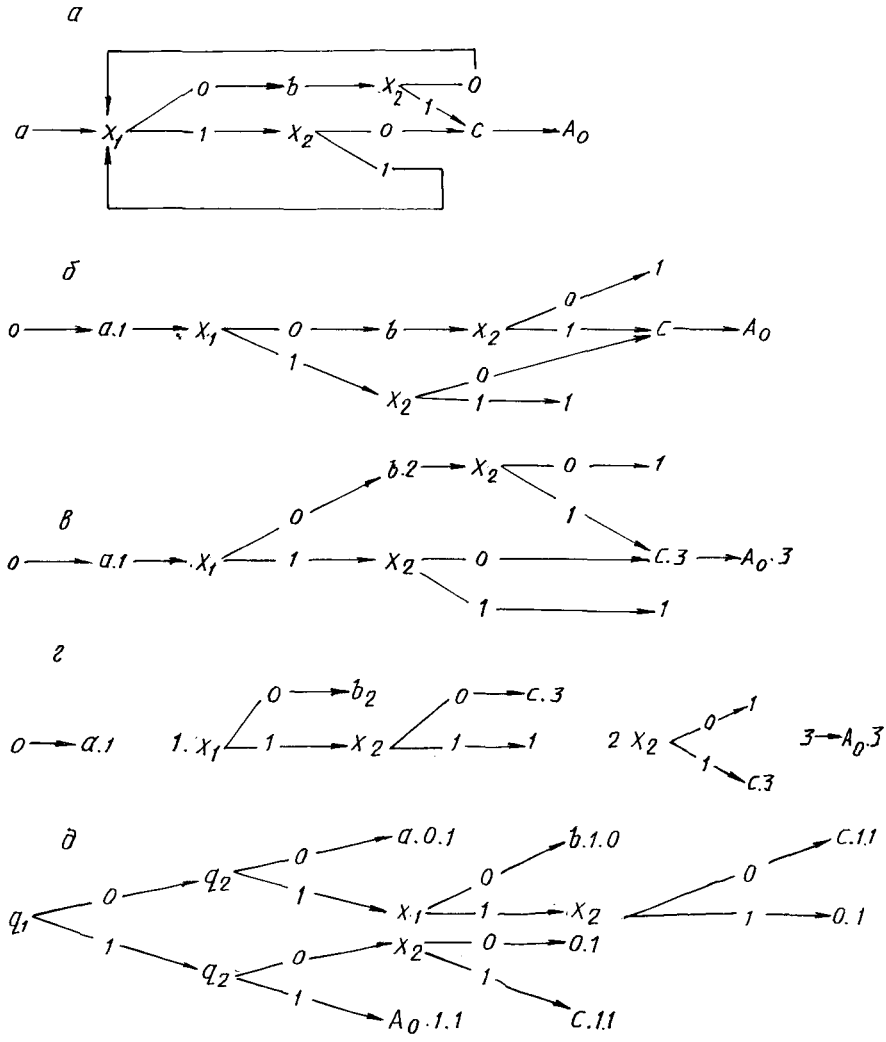


Рис. 3.26

двум конечным (рис. 3.26, *б*). В полученной граф-схеме с обратными связями нумеруем вход и буквы (рис. 3.26, *в*); исходя из требований устойчивости, после окончания вычислений по переключательной схеме присваиваем номер входа 3 выходу A_0. Согласно номерам входов, получаем четыре элементарные граф-схемы (рис. 3.26, *г*). Составляем таблицу двоичных кодов (табл. 3.10) и соответствующую ей граф-схему Γ_q последовательно соединяем с четырьмя граф-схемами (рис. 3.26, *д*).

Таблица 3.10

q_1	q_2	M
0	0	0
0	1	1
1	0	2
1	1	3

Оставив в конечных вершинах значение только одной из функций Y, Q_1, Q_2, получим три граф-схемы. Произведя сквозную нумерацию, построим три канонические таблицы, а затем переключательные схемы. Для краткости составим по граф-схеме, приведенной на рис. 3.26, *д*, табл. 3.11, а затем по ней канонические таблицы (рис. 3.27, *а*) и контактную схему (рис. 3.27, *б*). Для надежности работы контактной схемы прочерки в столбце Y заменим нулями. Схема включается кнопкой x_0.

Таблица 3.11

q_1	q_2	x_1	x_2	Y	Q_1	Q_2
0	0	0	0	*a*	0	1
0	0	0	1	*a*	0	1
0	0	1	0	*a*	0	1
0	0	1	1	*a*	0	1
0	1	0	0	*b*	1	0
0	1	0	1	*b*	1	0
0	1	1	0	*c*	1	1
0	1	1	1	–	0	1
1	0	0	0	–	0	1
1	0	0	1	*c*	1	1
1	0	1	0	–	0	1
1	0	1	1	*c*	1	1
1	1	0	0	A_0	1	1
1	1	0	1	A_0	1	1
1	1	1	0	A_0	1	1
1	1	1	1	A_0	1	1

Пример 3.14. Кнопка x может находиться в двух состояниях: нажата ($x = 1$) и не нажата ($x = 0$). Интервал времени, в течение которого кнопка сохраняет свое состояние, назовем *тактом*. Построить контактную схему, которая обеспечивает горение лампочки $Л$ только при нечетном числе нажатий кнопки.

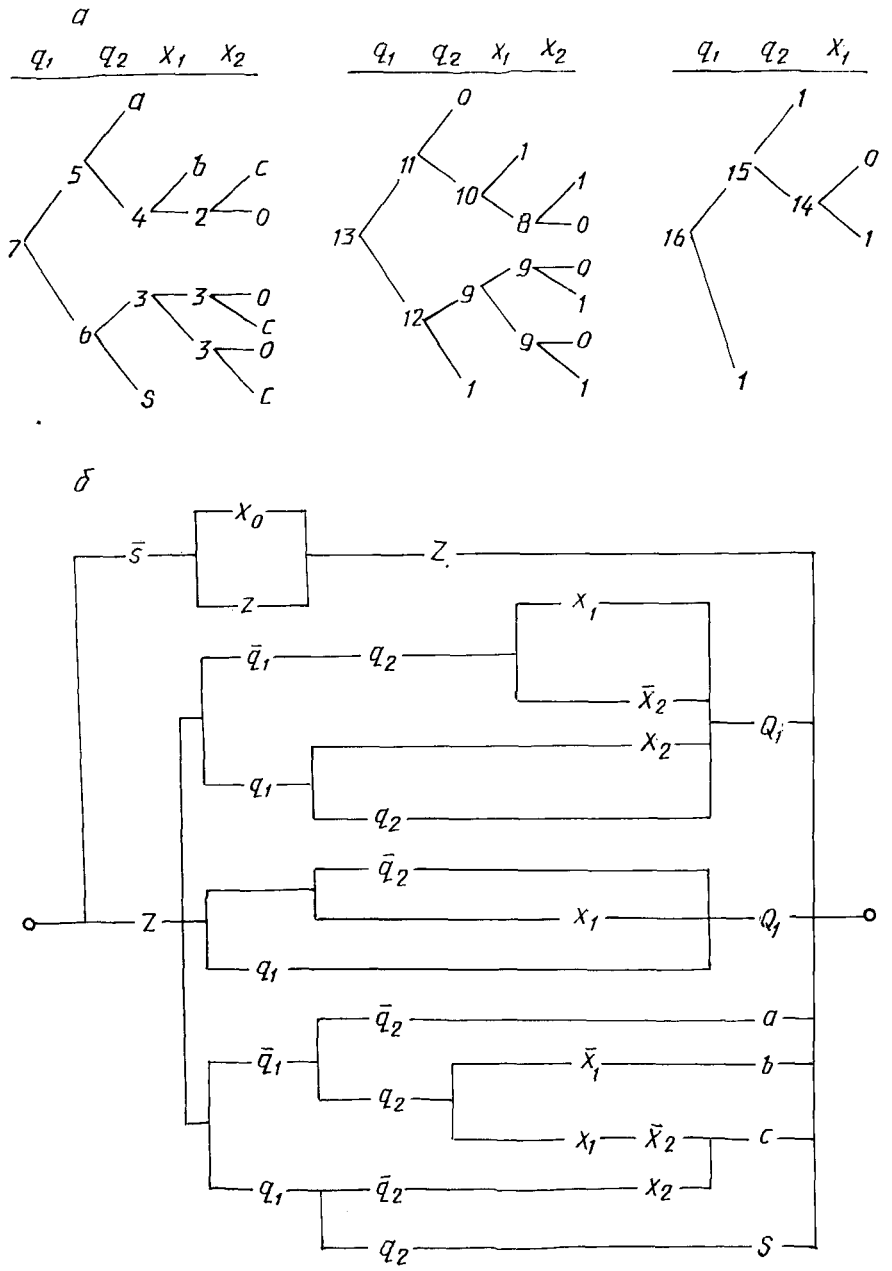

а
q_1 q_2 x_1 x_2
q_1 q_2 x_1 x_2
q_1 q_2 x_1
б
x_0
$\bar{s}$
z
Z
x_1
$\bar{q}_1$
q_2
$\bar{x}_2$
Q_i
x_2
q_1
q_2
$\bar{q}_2$
Z
x_1
Q_1
q_1
$\bar{q}_2$
a
$\bar{q}_1$
$\bar{x}_1$
b
q_2
x_1
$\bar{x}_2$
c
q_1
$\bar{q}_2$
x_2
q_2
S

Рис. 3.27

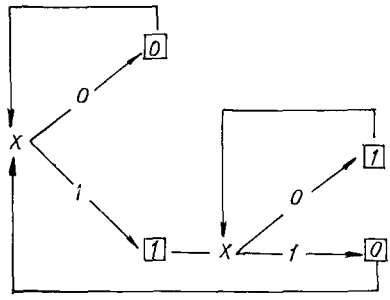


Рис. 3.28

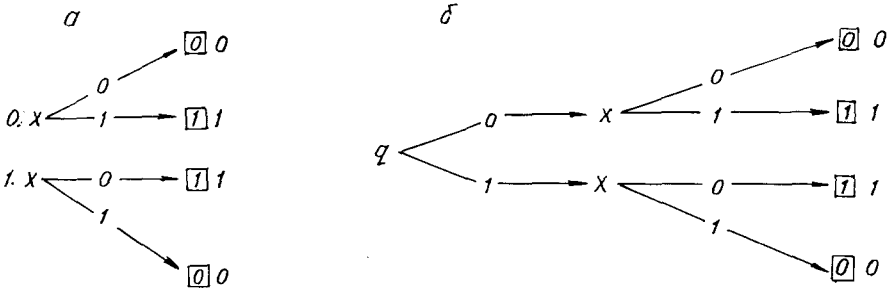


Рис. 3.29

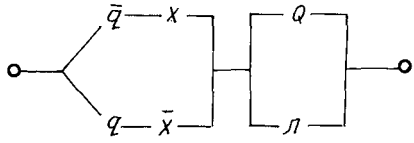


Рис. 3.30

Р е ш е н и е. Рассмотрим случай, когда такты имеют одинаковую продолжительность. Например, если такт равен 1 мин и кнопка нажата в течение 5 мин, это значит, что кнопка была нажата в течение пяти тактов. Если лампочка горит, считаем $Л = 1$, иначе $Л = 0$.

На рис. 3.28 приведена граф-схема алгоритма работы контактной схемы. В квадратах указаны значения $Л$. В граф-схеме три контура. Две последовательно соединенные элементарные граф-схемы содержат по одному кусту. Обозначим вход нулем, а вершину соединения – единицей и присвоим эти номера выходам, связанным с ними обратными связями (рис. 3.29, *а*).

Граф-схему $Г_q$ последовательно соединим с двумя входами *0* и *1* (рис. 3.29, *б*). Так как $Q = Л$, то контактную схему строим для граф-схемы Q и параллельно подключаем лампочку $Л$. Требуемая контактная схема приведена на рис. 3.30.

Пока нажата кнопка, лампочка горит на нечетных тактах. В самом деле, пусть в первом такте $x = 1$. Получим $Q = 1, Л = 1$. Но тогда $q = 1, \bar{q} = 0$, а кнопка еще нажата ($x = 1$). Получим $Q = 0$, затем снова $Q = 1$ и т. д.

При указанной реализации, когда предполагается равенство тактов, требуется синхронизация всех элементов схемы. На их техническом обеспечении останавливаться не будем.

Рассмотрим более важный случай, когда тактом считается интервал времени, в течение которого значения переменных $x_1, x_2, ..., x_n$ не изменяются. Этому условию схема, приведенная на рис. 3.30, не удовлетворяет.

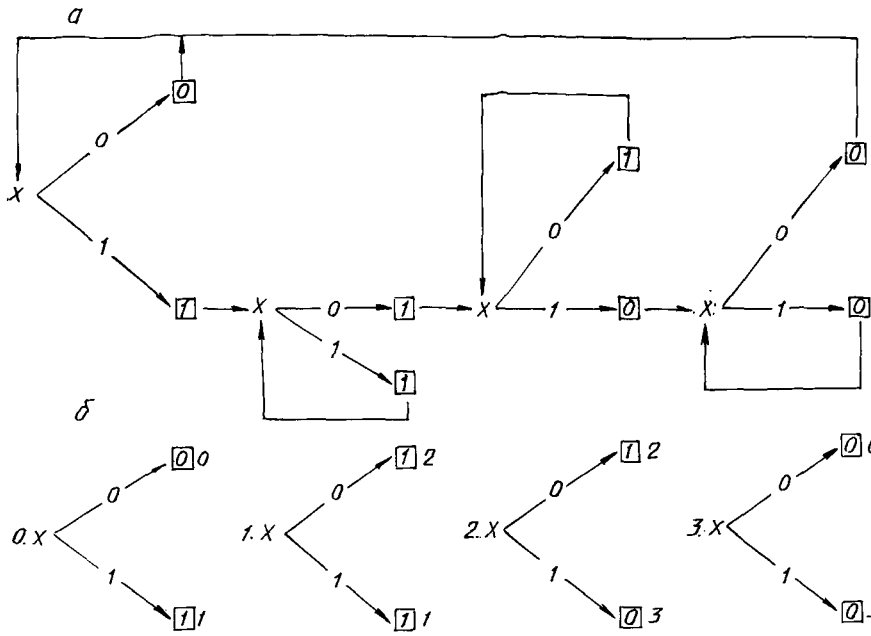


Рис. 3.31

Условие о тактах характеризует работу переключательной схемы, и поэтому его следует предусматривать уже при построении граф-схемы алгоритма, вводя обратную связь.

Пусть исполнитель алгоритма находится в начальной вершине A граф-схемы и такт определяется значениями $x_1 = \alpha_1$, $x_2 = \alpha_2$, ..., $x_n = \alpha_n$, т. е. при этих значениях исполнитель попадает в A. Путь, определяемый набором $(\alpha_1, \alpha_2, ..., \alpha_n)$, соединяет вершину A с конечной вершиной B граф-схемы. Чтобы выполнялось условие о тактах, нужно вершину B соединить дугой (обратной связью) с вершиной A.

Учитывая изложенное, построим граф-схему алгоритма работы контактной схемы из примера 3.14.

На рис. 3.31, *а* изображена граф-схема алгоритма, а на рис. 3.31, *б* – составляющие ее элементарные граф-схемы.

Закодируем номера q (Q) двоичными наборами: 0 = (0, 0), 1 = (0, 1), 2 = (1, 0), 3 = (1, 1). Получим табл. 3.12.

Таблица 3.12

q_1	q_2	x	Y	Q_1	Q_2	q_1	q_2	x	Y	Q_1	Q_2
0	0	0	0	0	0	1	0	0	1	1	0
0	0	1	1	0	1	1	0	1	0	1	1
0	1	0	1	1	0	1	1	0	0	0	0
0	1	1	1	0	1	1	1	1	0	1	1

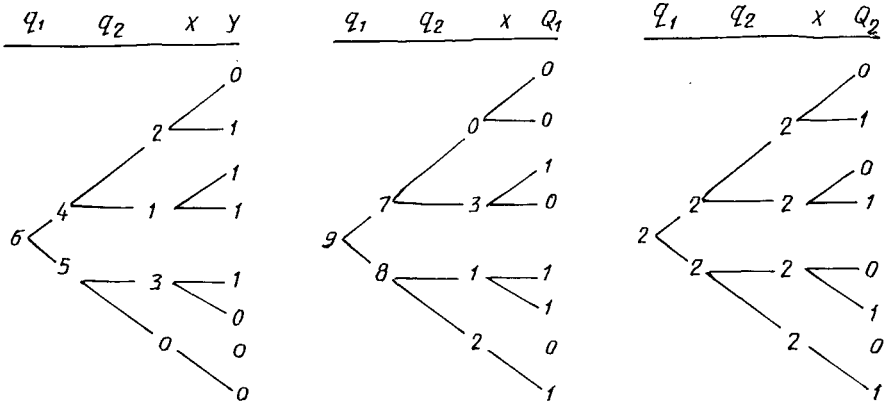


Рис. 3.32

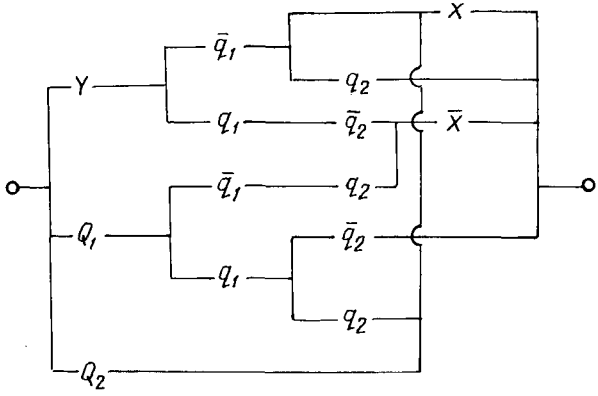


Рис. 3.33

На рис. 3.32 изображены три канонические таблицы с общей нумерацией. Искомая контактная схема дана на рис. 3.33.

Таким образом, техническая реализация алгоритма содержит переключательную схему управления и процессоры для реализации операторов.

Конечно, бессмысленно конструировать устройства для алгоритмов временного пользования. Кроме того, экономически нецелесообразно проводить техническую реализацию всех встречающихся на практике алгоритмов. Поэтому возникает необходимость в одном универсальном аппарате, на котором можно было бы реализовать любой алгоритм. Таким аппаратом является универсальная машина (УМ), реализующая универсальный алгоритм. Как было показано, универсальный алгоритм по шифру данного алгоритма преобразовывает входную информацию как данный алгоритм.

В УМ должны быть процессоры, реализующие операции перемещения вдоль шифра, сравнения и запоминания чисел, а также операцию выбора требуемого оператора. Эти операции осуществляются операторами D, Л, Φ_1, Φ_2, P_r, P_θ, Т (см. § 3.2). Перечисленные процессоры составляют в УМ блок

управления. В УМ имеются рабочие процессоры (арифметические и логические), моделирующие операторы из алфавита операторов.

Приведем описание одного из возможных подходов к осуществлению УМ. Память блока управления УМ представляет собой замкнутую ленту, разделенную на ячейки. Номер ячейки является ее постоянным адресом.

Ячейка состоит из четырех частей: N, ν, b, s. Числа N, ν, b, s заданы в двоичной системе счисления. Каждый разряд чисел образует вдоль ленты "дорожку", состоящую из нулей и единиц, которые моделируются, например, различной напряженностью магнитного поля ленты. Каждая "дорожка" обслуживается считывающей головкой. Все головки подразделяются на четыре группы: $\Gamma_N, \Gamma_\nu, \Gamma_b, \Gamma_s$.

В блоке управления имеются еще отдельные ячейки памяти, не принадлежащие ленте. Эти ячейки назовем регистрами. Имеются регистры R_N, R_M, а также регистры $R_1, R_2, ..., R_n$ для переменных $p_1, p_2, ..., p_n$.

Каналы головок Γ_N, Γ_b подключены соответственно к регистрам R_N, R_M. Процессор Π_0 реализует граф-схему сравнения чисел, которые записаны в регистрах R_N, R_M. При равенстве чисел срабатывает выход z процессора Π_0. Процессор Π_1 реализует оператор Φ_1. В зависимости от номера ν срабатывает требуемый выход процессора Π_1, включающий соответствующий рабочий процессор. Число ν считывается с ленты головками Γ_ν. Процессор Π_2 реализует оператор Φ_2.

Набор (r, θ, ν, r', s) шифра записывается в ячейку (M, ν, r', s), где $M = 2r + \theta$. Число M получают путем приписывания $\theta = \{0, 1\}$ к числу r.

Пусть в регистре R_M содержится число M, а на ленте имеется ячейка (M, ν, r', s). Головки Γ_N при движении ленты передают в регистр R_N текущий адрес N, который сравнивается с M в процессоре Π_0. При $N = M$ выход z процессора срабатывает и сигнал $z = 1$ включает головки Γ_ν, Γ_b, Γ_s, которые пересылают в Π_1, R_M, Π_2 числа ν, $b = 2r'$, s. Далее срабатывает процессор Π_1. Он включает рабочий процессор под номером ν. Результат заносится в регистр R_a или, если процессор логический, в регистр R_i. Затем срабатывает процессор Π_2 и из регистра R_s в младший разряд регистра R_M пересылается значение θ_s. После этого УМ готова к обработке очередных ячеек ленты.

Каждая УМ определяется своими рабочими процессорами. Блоки управления могут различаться размерами ленты и разрядностью регистров.

При контактной реализации регистр моделируется набором реле. Одно реле соответствует одному двоичному разряду. Процессоры Π_0, Π_1 и Π_2 осуществляются в виде контактных схем.

В гл. 1 были построены элементарные граф-схемы сравнения, сложения и вычитания натуральных чисел.

На рис. 3.34 изображены схемы алгоритмов умножения $(ab = c)$ и деления $(a : b = c)$ натуральных чисел.

В алгоритмах используются операторы сравнения чисел, сложения, вычитания и присваивания. Поэтому сами операции умножения и деления также выполняются с помощью переключательных схем. Отсюда следует, что алгоритмы, составленные из арифметических операций и операции сравнения чисел, моделируются переключательными схемами (иначе говоря, дискретными устройствами). Эти устройства оперируют натуральными числами. Но операции

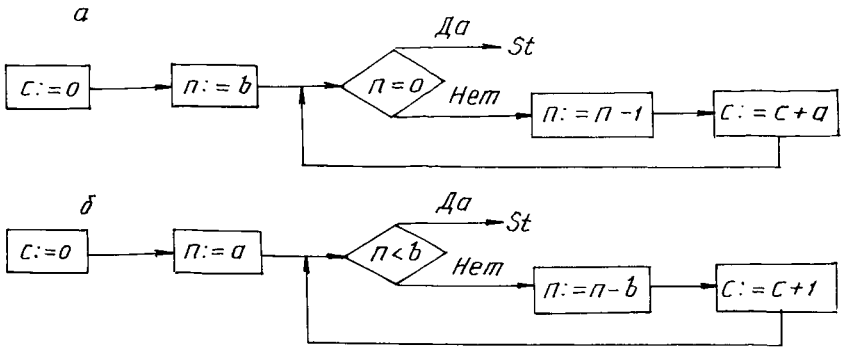


Рис. 3.34

сложения, вычитания и сравнения десятичных дробей отличаются от тех же операций над целыми числами только тем, что следует зафиксировать запятую, отделяющую целую часть числа от дробной. Поэтому принято считать, что n_1 первых разрядов изображают целую часть числа, а остальные n_2 разрядов — его дробную часть. Поэтому указанные устройства не требуют значительного усложнения для того, чтобы оперировать десятичными дробями. Таким образом, вычислительные машины представляют собой реализацию универсального алгоритма в алфавите операторов сложения, вычитания и сравнения двух десятичных дробей с определенным числом разрядов целой и дробной частей.

Известно, что всякую непрерывную функцию можно с любой степенью точности выразить формулой, составленной из арифметических операций. Поэтому любую непрерывную или частично непрерывную функцию можно реализовать с требуемой точностью дискретным устройством. Таким образом, дискретные устройства как средства реализации алгоритма являются универсальными.

4. ПОСТРОЕНИЕ АЛГОРИТМОВ ПО НЕПОЛНОЙ ИНФОРМАЦИИ

4.1. ЗАДАЧА РАСПОЗНАВАНИЯ ОБРАЗОВ

Особенность сложных процессов заключается в том, что полное их описание требует большого объема информации, значительная часть которой часто неизвестна. Поэтому возникают задачи построения алгоритмов, моделирующих процесс, на основании доступной части информации. Указанные задачи объединены в общую проблему обучения машин, или, иначе, проблему искусственного интеллекта. Проблема построения алгоритмов по неполной информации находится в начальной стадии изучения.

Решение многих так называемых интеллектуальных задач требует умения классифицировать различные исходные данные. К таким задачам относятся диагностика заболеваний, прогноз погоды, прогноз залежей полезных ископаемых, распознавание речи и т. д. Общим для всех этих задач является то, что в каждой из них имеется свое множество объектов, которые распределяются по классам. Задача распознавания образов заключается в том, чтобы при предъявлении некоторого объекта указать класс, которому он принадлежит. Например, при медицинской диагностике объект представляет собой набор симптомов, а классом является заболевание.

В роли классификатора может выступать не только человек, но и машина. Остается добавить самое существенное. Предполагается, что устройство машины не предусматривает изначальной возможности правильной классификации, что машину этому следует научить, поэтому под *классификационной задачей* понимают задачу обучения машины классифицированию.

Типичной является задача обучения распознаванию рукописных знаков. Нельзя указать четкие правила, на основании которых одну рукописную букву отличают от другой. В этом случае обучающемуся показывают рукописные знаки и сообщают, какие это буквы, т. е. к каким классам данные знаки относятся. В результате обучения у человека вырабатываются навыки правильной классификации. Таким образом, задача обучения машин классифицированию является задачей моделирования творческого процесса: выработки нужных правил, т. е. понятий.

Существуют различные методы обучения машин распознаванию образов. Их обоснование невозможно в рамках самой математики. Только практика может дать оценку каждому методу.

Обоснуем применение граф-схемного подхода для распознавания образов.

Различают два типа изображений: чувственные и абстрактные. К первым относятся зрительные, вкусовые и тому подобные изображения. Абстрактные изображения представляют собой наборы признаков, а абстрактные образы – это классы абстрактных изображений.

В процессе эволюции у человека сформировались специальные органы для

распознавания чувственных образов. Для распознавания абстрактных образов таких органов не существует. Обращение к абстрактным изображениям началось уже при достаточно высоком уровне сознания. Для распознавания абстрактных образов человек пользуется универсальным методом – дедукцией. В каждом конкретном случае она сводится по существу к анализу некоторой цепочки признаков. Отсюда заключаем, что задача обучения распознаванию определенных абстрактных образов состоит в построении граф-схемы данного алгоритма распознавания.

Отметим, что при машинном распознавании чувственные изображения также представляют в виде некоторого набора признаков, т. е. в виде абстрактного изображения. Но в алгоритме распознавания можно использовать характерные особенности чувственных образов. Существующие методы распознавания образов явно или неявно используют особенности зрительного восприятия. Это выражается хотя бы в том, что наборы признаков, как считается, образуют некоторое метрическое или топологическое пространство.

Пусть наборы $(p_1, p_2, ..., p_n)$ составляют множество определенных абстрактных изображений. Так как обычно признак имеет несколько значений, их можно закодировать натуральными числами, поэтому заранее считаем, что $0 \leqslant p_i \leqslant g_i - 1$, где g_i – значность признака p_i. Названия образов также несущественны, поэтому и образы считаем натуральными числами. Итак, задание множества абстрактных изображений с указанием, какому образу принадлежит данное изображение, равносильно заданию логической функции $y = f(p_1, p_2, ..., p_n)$. Например, при граф-схемном методе дифференциальной диагностики разных форм бронхиальной астмы было использовано $n = 77$ признаков. Большинство признаков имело два значения: 1 (его наличие), 0 (отсутствие).

При обучении распознаванию образов известны некоторые m изображений и их принадлежность образу. Это означает, что логическая функция $y = f(p_1, p_2, ..., p_n)$ задана только для последовательности из m наборов, которую называют *тренировочной последовательностью.* Проблема распознавания образов состоит в том, чтобы по тренировочной последовательности построить алгоритм, определяющий значение y для любого набора из области определения функции. Подобную задачу можно назвать *интерполяционной задачей для логических функций.*

4.2. ОБУЧЕНИЕ В КЛАССЕ ЭЛЕМЕНТАРНЫХ АЛГОРИТМОВ

Арифметический оператор назовем *буквенным,* если определяемая им функция постоянна. Значением оператора является некоторая буква (или число), которая может служить обозначением оператора.

Алгоритм, граф-схема которого элементарна и содержит только буквенные операторы, назовем *элементарным.*

В элементарном алгоритме вычисления производятся только по граф-схеме для заданных значений переменных $p_1, p_2, ..., p_n$, без переработки информации.

Очевидно, что элементарный алгоритм определяет некоторую логическую функцию $y = f(p_1, p_2, ..., p_n)$, значения которой записаны в конечных вершинах элементарной граф-схемы.

Пусть $G(n)$ – семейство логических функций $f(p_1, p_2, ..., p_n)$. Обозначим набор $(p_1, p_2, ..., p_n)$ через x, т. е. $x = (p_1, p_2, ..., p_n)$.

Как было отмечено выше, интерполяционная задача состоит в построении по тренировочной последовательности элементарного алгоритма, т. е. его граф-схемы. Но тренировочная последовательность включает не все наборы, ибо в противном случае не существовала бы задача обучения (так как функция полностью задана). Поэтому возникает задача построения алгоритма по неполной информации, т. е. по неполной таблице функции $f(x)$. Так как неполная таблица определяет не одну граф-схему (функцию), необходимо указать дополнительное условие, придерживаясь которого, можно получить определенную граф-схему.

В множестве абстрактных изображений, для которых никакие свойства или отношения не выполняются, возможно единственное оптимизирующее условие: по тренировочной последовательности построить элементарную граф-схему минимальной сложности. Если рассматриваются правильные граф-схемы, следует построить правильную граф-схему минимальной сложности. Но такое построение обязательно содержит перебор различных вариантов, что практически неосуществимо. Поэтому рассмотрим алгоритм построения граф-схемы, основанный на каноническом методе, и обоснуем его. Ради простоты изложения алгоритм построения элементарных алгоритмов рассматривается для двоичных переменных.

Таблицу для n переменных назовем таблицей n-го порядка. Построение граф-схемы происходит в три этапа. На первом этапе по заданной таблице порядка n строят $n-1$ таблиц порядков $n-1, n-2, ..., 1$ и соответствующие им кусты, на втором из кустов компонуют каноническую таблицу, на третьем по канонической таблице получают граф-схему. Порядок следования строк в таблице не играет роли. Пусть двоичные переменные следуют в таком порядке: $p_1, p_2, ..., p_n$. Две строки $\alpha_1, \alpha_2, ..., \alpha_i$ и $\beta_1, \beta_2, ..., \beta_i$ называются *соседними по переменной* p_i, если $\alpha_i \neq \beta_i$, а все остальные компоненты соответственно равны между собой. Пара соседних строк образует полный комплект.

Для первой строки таблицы n-го порядка (заданной таблицы функции) находим соседнюю по последней переменной p_n. Например, для первой строки (1001) табл. 4.1 соседней является шестая (1000).

Из номеров (a, b) столбца y, отвечающих полному комплекту, построим куст $(c; p_n; a, b)$. Для этого следует выписать номера комплекта сверху вниз

Таблица 4.1

p_1	p_2	p_3	p_4	y_4	Куст
1	0	0	1	2	
0	1	0	0	1	(3; p_4; 1, 2)
1	0	1	0	2	
1	1	0	0	1	
0	0	1	0	2	(4; p_4; 2, 1)
1	0	0	0	1	
0	0	1	1	1	

в порядке возрастания значения p_n. Кусты нумеруем и записываем в дополнительном столбце. Приведенным выше первой и шестой строкам таблицы отвечает, например, куст $(3;\ p_4\ ;\ 1, 2)$. Если в таблице для первой строки соседней нет, значит, функция не определена для соседней строки. Тогда примем следующее условие: *доопределим для соседней строки такое же значение функции y, как и для первой.* В результате получим нуль-куст.

Закончив построение куста для первого комплекта строк, занесем значения переменных $p_1, p_2, ..., p_{n-1}$ этого комплекта в первую строку следующей таблицы $(n-1)$-го порядка. Номер c куста или значения y_n в случае нуль-куста запишем в той же строке столбца функции y_{n-1} таблицы $(n-1)$-го порядка. Затем перейдем к ближайшей, не рассмотренной еще строке таблицы n-го порядка и повторим весь цикл заново. При нумерации кустов в пределах данного столбца будем придерживаться известных правил построения канонической таблицы: кусты с различными комплектами номеров обозначим разными номерами, с равными комплектами – одинаковыми номерами, комплекты из равных номеров – тем же номером.

Построение таблицы $(n-1)$-го порядка заканчивается после того, как проанализированы все строки таблицы n-го порядка. Полученные кусты назовем *кустами переменной* p_n (см. табл. 4.2–4.4 и соответствующие им кусты, полученные по алгоритму построения). Затем, считая первичной таблицу

Таблица 4.2

p_1	p_2	p_3	y_3	Куст
1	0	0	3	
0	1	0	1	
1	0	1	2	$(5;\ p_3;\ 3, 2)$
1	1	0	1	
0	0	1	4	

Таблица 4.3

p_1	p_2	y_2	Куст
1	0	5	
0	1	1	$(6;\ p_2;\ 5, 1)$
1	1	1	
0	0	4	$(7;\ p_2;\ 4, 1)$

Таблица 4.4

p_1	y_1	Куст
1	6	
0	7	$(8;\ p_1;\ 7, 6)$

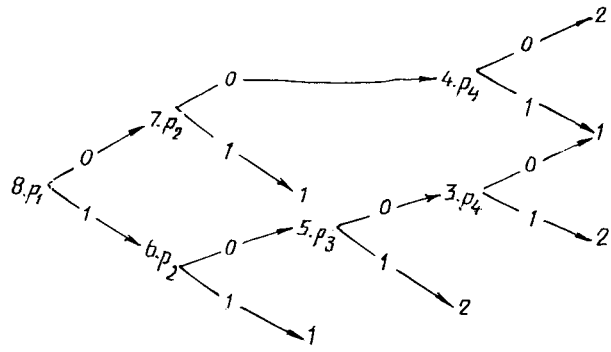


Рис. 4.1

$(n-1)$-го порядка, описанным выше способом строим кусты переменной p_{n-1} и таблицу $(n-2)$-го порядка. Процесс продолжается до тех пор, пока не будет построен куст переменной p_1. При этом равные комплекты номеров для таблиц различных порядков следует обозначать разными номерами, за исключением комплектов одинаковых номеров.

Для построения граф-схемы в вершинах кустов укажем переменную, дуги кустов обозначим ее значениями и кусты состыкуем так, чтобы выходная вершина одного куста с номером c объединялась с входной вершиной другого куста с тем же номером c. Построение проведем от кустов таблицы 1-го порядка к кустам таблиц n-го порядка. На рис. 4.1 изображена граф-схема для табл. 4.1, построенная по алгоритму.

Пусть функция $y = f(p_1, p_2, ..., p_n)$ определяется правильной граф-схемой сложности k (число кустов в граф-схеме равно k).

Теорема 4.1. *Для двоичной функции $y = f(p_1, p_2, ..., p_n)$ существует такая тренировочная последовательность длиной $l < (k-1)^2$, что построенная по ней согласно алгоритму граф-схема изображает требуемую функцию $y = f(p_1, p_2, ..., p_n)$.*

Доказательство. Так как правильную граф-схему можно получить из канонической таблицы "склеиванием" вершин с одинаковыми номерами, доказательство будем проводить для канонических таблиц.

Пусть тренировочная последовательность содержит такие значения $f(x_1)$, $f(x_2), ..., f(x_m)$, что при нумерации вершин различные номера обязательно получат те вершины канонической таблицы до i-го столбца, которые и должны иметь различные номера.

В i-м столбце содержится k_i различных номеров (очевидно, что $\sum_{i=0}^{n-1} k_i = k$). Они образуют k_i кустов с $2k_i$ выходными вершинами в $(i+1)$-м столбце, в котором имеется k_{i+1} разных номеров. Следовательно, $2k_i$ вершин распределяются по k_{i+1} группам из γ_ν вершин в ν-й группе. Очевидно, что $\sum_{\nu=1}^{k_{i+1}} \gamma_\nu = 2k_i$.

Вершины каждой группы нужно отличать от вершин других групп. Чтобы отличить вершину ν-й группы от $2k_i - \gamma_\nu$ вершин остальных групп, следует для первой вершины взять не более $k_{i+1} - 1$ значений функции, а для $2k_i - \gamma_\nu$

вершин – по одному значению. Поэтому, чтобы отличить вершины ν-й группы, можно взять не более $\gamma_\nu (k_{\nu+1} - 1) + (2k_i - \gamma_\nu)$ значений. Для отличия вершин всех k_{i+1} групп $(i+1)$-го столбца потребуется не более

$$\frac{1}{2} \sum_{\nu=1}^{k_{i+1}} \left\{ \gamma_\nu (k_{i+1} - 1) + (2k_i - \gamma_\nu) \right\} = \frac{1}{2} \left\{ 2k_i (k_{i+1} - 1) + \right.$$

$$\left. + 2k_{i+1} k_i - 2k_i \right\} = 2k_i (k_{i+1} - 1)$$

значений функции.

Для всей канонической таблицы получим

$$\sum_{i=1}^{n-1} 2k_i (k_{i+1} - 1) \leqslant \left(\sum_{i=1}^{n} k_i \right)^2 - 2 \sum_{i=1}^{n} k_i = k^2 - 2k .$$

Итак, число требуемых значений функции $m < (k-1)^2$. Теорема доказана.

Способ нумерации вершин согласно алгоритму при указанном выборе значений функции удовлетворяет условию минимальности правильной граф-схемы. Поэтому оценка длины тренировочной последовательности $l < (k-1)^2$ справедлива также и для метода, доставляющего минимальные правильные граф-схемы.

Теорема 4.1 служит некоторым обоснованием интерполяции логических функций с использованием предложенного алгоритма и является аналогом теорем о сходимости метода интерполяции в математическом анализе.

В задачах распознавания образов обычно оперируют большим числом признаков. Эти признаки безусловно взаимозависимы. В этом можно убедиться, подсчитав информативность одного признака относительно другого (например, при диагностике заболеваний). Но тогда наличие в граф-схеме отдельных, а не разветвляющихся путей для последних переменных таблицы является не особенным случаем, а правилом. Может быть, этим фактом и объясняется эффективность простого правила доопределения, по которому получают нулькуст.

В граф-схему, построенную в случае большого числа признаков, обычно входят не все признаки. Поскольку они несут определенную информацию, можно использовать их и строить по ним граф-схему. Такие соображения приводят к методу, называемому *консилиумом*. Опишем его.

1. По тренировочной таблице T строят граф-схему Γ_1.

2. Если в граф-схеме Γ_1 не все переменные использованы, то: а) из тренировочной таблицы вычеркивают вошедшие в граф-схему переменные и получают новую тренировочную таблицу T_1; б) по таблице T_1 строят новую граф-схему Γ_2, если эта таблица непротиворечива.

3. Для таблицы T_1 и граф-схемы Γ_2 проводят аналогичные построения согласно п. 1, 2.

4. В случае получения противоречивой таблицы или использования всех переменных построение граф-схемы прекращают.

Значения функции $f(p_1, p_2, ..., p_n)$ определяют следующим образом: для данного набора переменных $(p_1, p_2, ..., p_n)$ находят k значений по k построенным граф-схемам. Из этих значений выбирают такое, которое получено

более чем в половине граф-схем. В противном случае f по данному набору не определяется.

Рассмотрим систему учитель–ученик, определяемую семейством двоичных функций $G(n)$. Процесс обучения заключается в следующем.

Учитель сообщает ученику тренировочную последовательность для некоторой функции $f(p_1, p_2, ..., p_n) \in G_n$. При этом он пользуется некоторым правилом выбора определенной тренировочной последовательности. Назовем такие правила *методикой учителя.*

Ученик по тренировочной последовательности строит граф-схему, используя при этом некоторые условия построения. Назовем эти условия *методикой ученика.*

Методики учителя и ученика взаимозависимы: они должны обеспечить для тренировочной последовательности любой функции $f(p_1, p_2, ..., p_n)$ реализацию алгоритма ее вычисления. Последнее означает, что для определенного способа построения граф-схемы не полностью заданной арифметической функции, которым пользуется ученик, учитель в свою очередь подбирает такую тренировочную последовательность, которая для указанного способа однозначно определяет граф-схему искомой функции. Такие согласованные тренировочные последовательности будем называть *представительными.*

Если методика учителя заключается в том, что он задает полную тренировочную последовательность, т. е. все значения функции, то ученик должен уметь строить граф-схемы для полностью определенных функций. Интерес представляют, однако, системы учитель–ученик, которые обрабатывают тренировочные последовательности значительно меньшей длины. К ним, в частности, относятся системы, в которых методика построения граф-схем определяется предложенным ранее алгоритмом. Как было показано, для таких систем существует представительная тренировочная последовательность, длина которой не превосходит $(k-1)^2$.

Уже отмечалось, что методики ученика и учителя в системе учитель–ученик должны быть согласованы. Но, вообще говоря, методика учителя неизвестна. Ведь в роли учителя не всегда выступает человек, проводящий сознательное, целенаправленное обучение. Поэтому работоспособность любой модели должна быть экспериментально проверена.

Приведем другое истолкование предложенного метода интерполяции двоичной функции.

В пространстве R_n двоичных наборов (точек) $x = (p_1, p_2, ..., p_n)$ определим расстояние $\rho(x, y)$ для точек $x = (p_1, p_2, ..., p_n)$, $y = (q_1, q_2, ..., q_n)$ следующей формулой:

$$\rho(x, y) = \sum_{i=1}^{n} |p_i - q_i| 2^{i-1}.$$

Легко проверить выполнение основных свойств расстояния:

$$\rho(x, x) = 0; \quad \rho(x, y) = \rho(y, x); \quad \rho(x, y) \leqslant \rho(x, z) + \rho(z, y).$$

Расстояние $\rho(x, y)$ есть двоичное число, k-й разряд которого $a_{k-1} = |p_k - q_k|$, поэтому $0 \leqslant \rho \leqslant 2^n - 1$.

Лемма 4.1. *Для заданной точки* $x_0 = (p_1^0, p_2^0, ..., p_n^0)$ *и заданного натурального числа* $\rho_0 \leqslant 2^n - 1$ *существует только одна точка* y_0, *такая, что* $\rho(x_0, y_0) = \rho_0$.

Доказательство. Представим ρ_0 в двоичной системе счисления: $\rho_0 = \sum_{k=1}^{n} \alpha_k 2^{k-1}$. Так как $\alpha_{k-1} = |p_k^0 - q_k^0|$ и $\alpha_{k-1} = \{0, 1\}$, то $q_k^0 = p_k^0$, если $\alpha_{k-1} = 0$, и $q_k^0 = \overline{p}_k^0$, если $\alpha_{k-1} = 1$. Лемма доказана.

Пример 4.1. Пусть $x_0 = (1, 0, 0, 0)$ и $\rho_0 = 13$. Определить y_0 таким образом, чтобы $\rho(x_0, y_0) = 13$.

Решение. Представим ρ_0 в двоичной системе счисления: $13 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$. Следовательно, $q_4 = \overline{p}_4$, $q_3 = \overline{p}_3$, $q_2 = p_2$, $q_1 = \overline{p}_1$, т. е. $q_1 = 0, q_2 = 0$, $q_3 = 1$, $q_4 = 1$ или $y_0 = (0, 0, 1, 1)$.

В некоторой области $D \in R_n$ задана функция $y = f(p_1, p_2, ..., p_n)$. Требуется доопределить функцию на все пространство R_n.

Укажем процедуру доопределения (интерполяции) двоичной функции, т. е. определим в точке $y_0 \overline{\in} D$ функцию $f(p_1, p_2, ..., p_n)$.

1. Найти среди точек области D точку x_0, ближайшую к данной точке y_0. Согласно лемме 4.1, точка x_0 – единственная.

2. Положить $f(y_0) = f(x_0)$.

Теорема 4.2. *Предлагаемый алгоритм доопределения совпадает с графсхемным методом доопределения.*

Доказательство. Проведем доказательство по индукции относительно числа переменных n. Для $n = 1$ теорема верна. Пусть теорема справедлива для $n = m$. Покажем, что тогда она верна и для $n = m + 1$.

Пусть требуется доопределить функцию f в точке $x_0 = (p_1^0, p_2^0, ..., p_m^0, p_{m+1}^0)$. Для определенности возьмем $p_{m+1}^0 = 0$. Если область D содержит хотя бы одну точку z с $p_{m+1} = 0$, то ближайшая к x_0 точка y_0 имеет координату $p_{m+1} = 0$. В противном случае расстояние $\rho(x_0, y_0) \geqslant 2^m$, в то время как $\rho(x_0, z) < 2^m$. Следовательно, доопределение в этом случае осуществляется с помощью известных значений функции в области $D_0 \subset D$, т. е. точек из D, для которых $p_{m+1} = 0$. Но это означает, что рассматривается функция $f_0(p_1, p_2, ..., p_m) = f(p_1, p_2, ..., p_m, 0)$ от m переменных. Для таких функций по предположению теорема верна.

Рассмотрим теперь случай, когда область D не содержит точек с $p_{m+1} = 0$. Иначе говоря, в части канонической таблицы для $p_{m+1} = 0$ в конечных вершинах проставлены прочерки.

Ближайшая к точке x_0 точка y_0 будет иметь координату $p_{m+1} = 1$. Поэтому $\rho(x_0, y_0) = 2^m + \sum_{i=1}^{m} |p_i^0 - q_i^0| 2^{i-1}$. Доопределим $f(x_0) = f(y_0)$. Точку $z_0 = (p_1^0, p_2^0, ..., p_m^0, 1)$ назовем *соответствующей* точке x_0. Если $z_0 \in D$, то $y_0 = z_0$, значит, $f(x_0) = f(z_0)$. Если $z_0 \overline{\in} D$, то точка y_0 – ближайшая к z_0. Поэтому доопределим $f(z_0) = f(y_0)$. Таким образом, $f(x_0) = f(z_0)$. Полу-

чили, что во всех точках x с $p_{m+1} = 0$ функция $f(x)$ равна функции $f(z)$, где точка z отличается от точки x только координатой p_{m+1}.

Если в части канонической таблицы, определяемой вершиной $p_{m+1} = 0$, функция не определена, то эта вершина, согласно граф-схемному методу доопределения, получает такой же номер, что и вершина $p_{m+1} = 1$. Это означает, что $f(x) = f(z)$, где z – соответствующая x точка. Теорема доказана.

Если определить расстояние с весами, меньшими 2^l, то не будет выполняться лемма 4.1. Потребуются еще дополнительные условия для однозначного доопределения функции.

Как видим, координаты точки $x = (p_1, p_2, ..., p_n)$ из-за различия весов неравноправны при интерполировании функций. Это было ясно уже при граф-схемном доопределении, когда последние переменные в таблице функции при больших n не входили в граф-схему. Начальные переменные всегда оставались в граф-схеме. Поэтому при использовании граф-схемного метода доопределения функции следует переменные располагать в таблице функции в порядке убывания их информативности.

4.3. ИНФОРМАТИВНОСТЬ ПРИЗНАКОВ

Определение значения функции $y = f(p_1, p_2, ..., p_n)$ можно истолковать как эксперимент. *Нулевым* назовем эксперимент по определению значения функции только для множества всех значений y. (Представим себе, что все значения y записаны на отдельных, помещенных в урну карточках и из урны случайным образом извлекается одна карточка.)

Пусть $P(\alpha)$ – вероятность извлечения $y = \alpha$. Энтропия нулевого эксперимента $\mathcal{Э}_0 = -\sum_{\alpha} P(\alpha) \log P(\alpha)$.

Назовем i-м экспериментом определение значения y при заданном значении p_i.

Пусть $p_i = \beta$. В этом случае y рассматривается не для всех наборов переменных, а только для тех, в которых $p_i = \beta$. Относительно $p_i = \beta$ условная энтропия

$$\mathcal{Э}_i(\beta) = -\sum_{\alpha} P(\alpha/\beta) \log P(\alpha/\beta),$$

где $P(\alpha/\beta)$ – вероятность того, что $y = \alpha$ при $p_i = \beta$.

Возьмем среднюю энтропию $\mathcal{Э}_i = \sum_{\beta} P(\beta)\, \mathcal{Э}_i(\beta)$, где $P(\beta)$ – вероятность значения $p_i = \beta$. Итак,

$$\mathcal{Э}_i = -\sum_{\beta} \sum_{\alpha} P(\beta) P(\alpha/\beta) \log P(\alpha/\beta).$$

Так как $P(\beta) P(\alpha/\beta) = P(\alpha, \beta)$ – вероятность того, что $y = \alpha$, $p_i = \beta$, то

$$\mathcal{Э}_i = -\sum_{\beta} \sum_{\alpha} P(\alpha, \beta) \log P(\alpha/\beta) \quad \text{или} \quad \mathcal{Э}_i = -\sum_{\beta} \sum_{\alpha} P(\alpha,\beta) \log \frac{P(\alpha,\beta)}{P(\beta)}.$$

Разность $I_i = \mathcal{Э}_0 - \mathcal{Э}_i$ определяет количество информации, заключенной в i-м эксперименте.

Информацию I_i будем считать информативностью признака p_i для $y = f(p_1, ..., p_i, ..., p_n)$.

Информативность I_v признака p_v обладает следующими свойствами:

1) $0 \leqslant I_v \leqslant \mathcal{Э}_0$;

2) $I_v = 0$ тогда и только тогда, когда величины y и p_v взаимно независимы;

3) $I_v = \mathcal{Э}_0$ тогда и только тогда, когда y функционально зависит от p_v, т. е. $y = \psi(p_v)$.

Докажем эти свойства. Отметим, что если $\sum_{i=1}^{n} a_i = \sum_{i=1}^{n} b_i$ и $a_i \geqslant 0$, $b_i \geqslant 0$, то

$$\sum_{i=1}^{n} a_i \log a_i \geqslant \sum_{i=1}^{n} a_i \log b_i , \tag{4.1}$$

причем равенство достигается только при $a_i = b_i$. При доказательстве будет использовано неравенство (4.1).

1. Проведем преобразование

$$I_i = \mathcal{Э}_0 - \mathcal{Э}_i = \sum_{\beta} \sum_{\alpha} P(\alpha, \beta) \left(\log \frac{P(\alpha, \beta)}{P(\beta)} - \log P(\alpha)\right) .$$

Отсюда

$$I_i = \sum_{\beta} \sum_{\alpha} P(\alpha, \beta) \log P(\alpha,\beta) - \sum_{\beta} \sum_{\alpha} P(\alpha, \beta) \log P(\alpha) P(\beta) . \tag{4.2}$$

Так как $\sum_{\beta} \sum_{\alpha} P(\alpha, \beta) = 1$, $\sum_{\beta} \sum_{\alpha} P(\alpha) P(\beta) = 1$, то на основании неравенства (4.1) имеем $I_i \geqslant 0$.

2. Пусть y и p_i взаимно независимы. Это значит, что $P(\alpha, \beta) = P(\alpha) P(\beta)$. Подставив $P(\alpha)P(\beta)$ в выражение (4.2), получим $I_v = 0$. Обратное утверждение также следует из выражения (4.2), ибо значение $I_v = 0$, согласно неравенству (4.1), достигается только при $P(\alpha, \beta) = P(\alpha) P(\beta)$.

3. Функциональная зависимость y от p_v означает, что условная вероятность $P(\alpha/\beta)$ принимает только два значения: 0 или 1, поэтому при функциональной зависимости в каждом члене суммы $\mathcal{Э}_i = -\sum_{\beta} \sum_{\alpha} P(\alpha, \beta) \log P(\alpha/\beta)$, либо $\log P(\alpha/\beta) = 0$, либо $P(\alpha, \beta) = P(\beta) P(\alpha/\beta) = 0$, отсюда $\mathcal{Э}_i = 0$. Пусть теперь $\mathcal{Э}_i = 0$. Так как слагаемые в $\mathcal{Э}_i$ одного знака, то из $\mathcal{Э}_i = 0$ следует, что каждое слагаемое $P(\alpha, \beta) \log P(\alpha, \beta) = 0$, поэтому либо $\log P(\alpha/\beta) = 0$, либо $P(\alpha, \beta) = 0$, т. е. либо $P(\alpha/\beta) = 1$, либо $P(\alpha/\beta) = 0$.

Информативность двух или нескольких признаков определяется аналогично, при этом данные признаки (переменные) следует рассматривать как один признак (переменную).

Интервалом постоянства переменной назовем множество всех наборов с равными значениями данной переменной. Будем говорить, что переменная v покрывает переменную u, если интервалы постоянства u содержатся в интервалах постоянства v.

Теорема 4.3. *Если переменная v покрывает переменную u в области определения функции y, то $I_u \geqslant I_v$.*

Доказательство. Имеем $I_v = \mathcal{Э}_0 - \mathcal{Э}_v$, $I_u = \mathcal{Э}_0 - \mathcal{Э}_u$. Следует показать, что $\mathcal{Э}_u \leqslant \mathcal{Э}_v$.

Обозначим через δ значения u, покрываемые интервалом $v = \beta$. Очевидно, что $\sum\limits_{\delta} P(\delta) = P(\beta)$, $\sum\limits_{\delta} P(\alpha, \delta) = P(\alpha, \beta)$. Рассмотрим разность

$$P(\alpha, \beta) \log \frac{P(\alpha, \beta)}{P(\beta)} - \sum_{\delta} P(\alpha, \delta) \log \frac{P(\alpha, \delta)}{P(\delta)} =$$

$$= \sum_{\delta} P(\alpha, \delta) \log \frac{P(\alpha, \beta)}{P(\beta)} - \sum_{\delta} P(\alpha, \delta) \log \frac{P(\alpha, \delta)}{P(\delta)} =$$

$$= P(\alpha, \beta) \sum_{\delta} \frac{P(\alpha, \delta)}{P(\alpha, \beta)} \log \frac{P(\alpha, \beta)}{P(\alpha, \delta)} \frac{P(\delta)}{P(\beta)} =$$

$$= P(\alpha, \beta) \sum_{\delta} \frac{P(\alpha, \delta)}{P(\alpha, \beta)} \log \frac{P(\alpha, \beta)}{P(\alpha, \delta)} + P(\alpha, \beta) \sum_{\delta} \frac{P(\alpha, \delta)}{P(\alpha, \beta)} \log \frac{P(\delta)}{P(\beta)} =$$

$$= P(\alpha, \beta) \sum_{\delta} \frac{P(\alpha, \delta)}{P(\alpha, \beta)} \log \frac{P(\delta)}{P(\beta)} - P(\alpha, \beta) \sum_{\delta} \frac{P(\alpha, \delta)}{P(\alpha, \beta)} \log \frac{P(\alpha, \delta)}{P(\alpha, \beta)}.$$

Согласно неравенству (4.1), имеем

$$P(\alpha, \beta) \log \frac{P(\alpha, \beta)}{P(\beta)} - \sum_{\delta} P(\alpha, \delta) \log \frac{P(\alpha, \delta)}{P(\delta)} \leqslant 0.$$

Оценим разность

$$Э_u - Э_v = \sum_{\alpha} \sum_{\beta} P(\alpha, \beta) \log \frac{P(\alpha, \beta)}{P(\beta)} - \sum_{\alpha} \sum_{\beta} \sum_{\delta} P(\alpha, \delta) \log \frac{P(\alpha, \delta)}{P(\delta)} =$$

$$= \sum_{\alpha} \sum_{\beta} \left(P(\alpha, \beta) \log \frac{P(\alpha, \beta)}{P(\beta)} - \sum_{\delta} P(\alpha, \delta) \log \frac{P(\alpha, \delta)}{P(\delta)} \right) \leqslant 0.$$

Итак, $Э_u \leqslant Э_v$, следовательно, $I_u \geqslant I_v$. Теорема доказана.

Следствие 1. *Набор признаков не менее информативен, чем любая его часть.*

В самом деле, обозначим набор и его часть через переменные u и v соответственно. Очевидно, что v покрывает u. Отсюда следует, что $I_u \geqslant I_v$.

Следствие 2. *Зависимые признаки можно не учитывать.*

Действительно, пусть имеем признаки p, u, v и $v = \varphi(u)$. Набор (p, u, v) покрывает пару (p, u). Отсюда $I_{puv} \leqslant I_{pu}$. Но, согласно следствию 1, $I_{puv} \geqslant I_{pu}$, поэтому $I_{puv} = I_{pu}$. Очевидно, что рассуждения справедливы и тогда, когда p, u, v сами являются наборами признаков.

Если есть группа зависимых признаков, следует выделить такую наименьшую ее часть, чтобы остальные признаки группы зависели от них, и оставить только выделенную часть признаков.

Пусть в таблице функции $y = f(p_1, p_2, ..., p_n)$ число строк равно m; в столбце y число $y = \alpha$ повторяется $t(\alpha)$ раз; в столбце p значение $p = \beta$ встречается $k(\beta)$ раз; в двух столбцах y, p пара $y = \alpha$, $p = \beta$ встречается

$s(\alpha, \beta)$ раз. Тогда, полагая $p(\alpha) = t(\alpha)/m$, $p(\beta) = k(\beta)/m$, $p(\alpha, \beta) = s(\alpha,\beta)/m$, получаем:

$$I_p = \log m + \frac{1}{m}\left(\sum_{\alpha}\sum_{\beta} s(\alpha, \beta)\log s(\alpha, \beta) - \sum_{\beta} k(\beta)\log k(\beta) - \right.$$
$$\left. - \sum_{\alpha} t(\alpha)\log t(\alpha)\right).$$

Пример 4.2. Вычислить информативность признаков табл. 4.5.

Таблица 4.5

p_1	p_2	p_3	p_4	p_5	p_6	y
1	1	1	0	1	0	7
1	1	1	1	1	1	7
1	0	0	0	0	0	4
0	0	1	1	0	1	4
0	0	1	1	1	0	7
0	1	1	0	1	1	4
0	1	1	1	1	0	4
1	0	1	0	1	1	7

Р е ш е н и е. Имеем $m = 8$, $t(4) = 4$, $t(7) = 4$. Для p_1 получаем $k(0) = 4$, $k(1) = 4$, $s(1, 7) = 3$, $s(1, 4) = 1$, $s(0, 4) = 3$, $s(0, 7) = 1$. Находим

$$I_1 = \log 8 + \frac{1}{8}((3\log 3 + 1\log 1 + 3\log 3 + 1\log 1) - (4\log 4 +$$
$$+ 4\log 4) - (4\log 4 + 4\log 4)),$$

откуда $I_1 = \frac{3}{4}\log 3 - 1$.

Для p_2 имеем $k(0) = 4$, $k(1) = 4$, $s(1, 7) = 2$, $s(0, 4) = 2$, $s(1, 4) = 2$, $s(0, 7) = 2$. Находим $I_2 = 0$.

Для признаков получаем следующие информативности: $I_1 = \frac{3}{4}\log 3 - 1$; $I_2 = 0$; $I_3 = 2 + \frac{1}{8}(3\log 3 - 7\log 7)$; $I_4 = 0$; $I_5 = \frac{5}{4} - \frac{3}{4}\log 3$; $I_6 = 0$, а также $\mathcal{Э}_0 = 1$. Наибольшая информативность может быть равна $\mathcal{Э}_0$.

Вычислим еще $I_{3,5}$. Для пары p_3, p_5 имеем: $k(1, 1) = 6$; $k(0, 0) = 1$; $k(1, 0) = 1$; $s(1, 1, 7) = 4$; $s(0, 0, 4) = 1$; $s(1, 0, 4) = 1$; $s(1, 1, 4) = 2$, откуда

$$I_{3,5} = 3 + \frac{1}{8}((4\log 4 + 2\log 2) - 6\log 6 - (4\log 4 + 4\log 4))$$

или $I_{3,5} = \frac{3}{2} - \frac{3}{4}\log 3$.

Аналогично найдем, что $I_{2,4} = 0$, и т. д.

Следует отметить, что информативность каждого признака в отдельности еще не характеризует его вклад в совокупности. Например, для табл. 4.5 имеем: $I_1 > 0$; $I_3 > 0$; $I_5 > 0$; $I_2 = I_4 = I_6 = 0$. Более того, $I_{3,5} > I_3$; $I_{3,5} > I_5$, а $I_{2,4} = 0$. Казалось бы, вклад p_3, p_5 весомее, чем p_2, p_4, однако $I_{1,3,5} =$

$= \frac{5}{4} - \frac{3}{8}\log 3$, $I_{2,4,6} = 1$. Получили, что $I_{2,4,6}$ принимает максимальное значение и признаки p_2, p_4, p_6 полностью определяют функцию, а признаки p_1, p_3, p_5 однозначно не определяют функцию. Поэтому следует с осторожностью относиться к отбору признаков в задачах распознавания образов и без анализа не отбрасывать малоинформативные признаки.

Безразмерную величину $\theta_p = I_p/\mathcal{Э}_0 = (\mathcal{Э}_0 - \mathcal{Э}_p)/\mathcal{Э}_0$ называют *коэффициентом информативности* признака p. Очевидно, что если $I_{p_1} > I_{p_2}$, то $\theta_{p_1} > \theta_{p_2}$. Поэтому θ_p характеризует признак так же, как I_p. Но вместе с тем коэффициент информативности позволяет сравнивать влияние признаков, в том числе одного признака, на различные функции (например, при диагностике различных заболеваний сравнивать важность какого-нибудь признака в каждом из них).

Перечислим свойства коэффициента информативности θ, непосредственно следующие из соответствующих свойств информативности признаков:

1) $0 \leqslant \theta \leqslant 1$;

2) $\theta_p = 0$ тогда и только тогда, когда величины p и y независимы;

3) $\theta_p = 1$ тогда и только тогда, когда y функционально зависит от p;

4) если переменная v покрывает переменную u в области определения функции y, то $\theta_u \geqslant \theta_v$;

5) коэффициент информативности набора признаков не меньше коэффициента информативности каждого из признаков набора в отдельности.

Наличие указанных свойств может служить обоснованием выбора коэффициента информативности в качестве оценки влияния одной величины на другую.

4.4. ОБУЧЕНИЕ В КЛАССЕ БУКВЕННЫХ АЛГОРИТМОВ

Алгоритм, содержащий только буквенные операторы, назовем *буквенным*. В этом случае вычисление осуществляется только по граф-схеме, без преобразования в операторах.

Пусть Γ – буквенный алгоритм от переменных $p_1, p_2, ..., p_n$. Очевидно, что буквенный алгоритм не содержит логических операторов. Значения переменных $p_1, p_2, ..., p_n$ задаются. Обозначим набор $(p_1, p_2, ..., p_n)$ через x и рассмотрим алгоритм Γ только относительно переменной x. Одна граф-схема от переменных $p_1, p_2, ..., p_n$ интерпретируется как куст переменной x, в конечных вершинах которого записаны последовательности букв, относящиеся к соответствующим ветвям граф-схемы. Будем считать такую последовательность новой буквой. Обозначим идентификатором y буквы полученной граф-схемы, состоящей из кустов переменной x, все вершины которой опорные, т.е. с метками.

В результате вычислений по заданной входной последовательности $x_1, x_2, ..., x_v$ получим выходную последовательность $y_1, y_2, ..., y_k$, где $k \leqslant v$ и $k = v$ только в случае прекращения вычисления оператором A_0.

Значения y_k являются результатами переработки алгоритмом Γ входной последовательности $x_1, x_2, ..., x_v$.

Буквенные алгоритмы, к которым относятся и элементарные, являются примерами наиболее простых алгоритмов. Этим и определяется их практическое значение.

Сформулируем теперь экстраполяционную задачу для арифметических функций $y = F(u)$, которая затем будет истолкована как задача обучения распознаванию закономерностей в последовательностях.

Пусть известны значения некоторой арифметической функции $y = F(u)$ только на отрезке $0 \leqslant u \leqslant N$. Требуется доопределить функцию, т. е. для любого натурального u указать значение $F(u)$. Далее будет показано, что задание вычислимой функции на отрезке $0 \leqslant u \leqslant N$ при достаточно большом N однозначно определяет ее для всех натуральных чисел.

Заменим входную последовательность $x_1, x_2, ..., x_m$ ее числовым эквивалентом $u = x_m + gx_{m-1} + ... + g^{m-1}x_1$, где g – значность переменной x, являющаяся основанием системы счисления для u.

Поставим в соответствие числу u, т. е. входной последовательности x_1, x_2, ..., x_m, последний член y_k выходной последовательности $y_1, y_2, ..., y_k$. Вычисление по буквенному алгоритму можно при определенных условиях интерпретировать как вычисление арифметической функции $y = F(u)$.

Теорема 4.4. *Буквенный алгоритм тогда и только тогда определяет арифметическую функцию $y = F(u)$, когда при $x = 0$ начальная вершина граф-схемы переходит в эквивалентную ей.*

Доказательство. Число u не изменится, если приписать нуль левее старшего разряда. Поэтому при $x = 0$ начальная вершина должна перейти в эквивалентную ей вершину. (В случае простой граф-схемы вершина должна перейти в себя.) Теорема доказана.

Ниже рассматриваются буквенные алгоритмы с простыми граф-схемами.

Если в конечных вершинах граф-схемы (рис. 4.2, *а*) убрать обратные связи, а взамен продолжить наращивание кустов, исходная граф-схема преобразу-

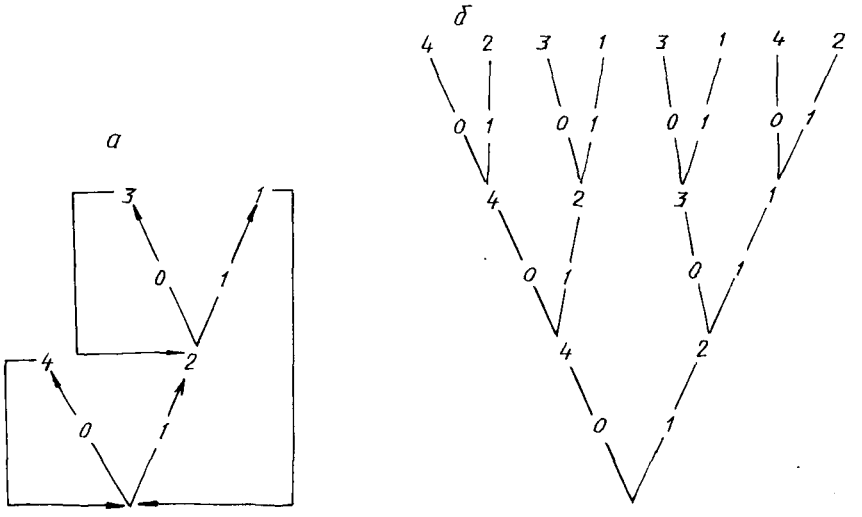


Рис. 4.2

ется в дерево, вообще говоря, бесконечное (рис. 4.2, *б*). Каждая вершина дерева определяется входной последовательностью $x_1, x_2, ..., x_m$ или ее числовым эквивалентом u. Равные числа u соответствуют, согласно теореме 4.4, эквивалентным вершинам.

В m-м ярусе дерева порядковый номер вершины, начиная с нуля, совпадает с ее числовым эквивалентом u (см. рис. 4.2, *б*). Поэтому все дерево буквенного алгоритма можно задать одним бесконечным ярусом, первые g^m вершин которого составляют m-й ярус дерева.

Семейство G функций $F(u)$, реализуемых буквенными алгоритмами, представляет собой сумму: $G = \sum_{k=1}^{\infty} G(k)$, где $G(k)$ – класс функций, определяемых простыми граф-схемами с числом вершин не больше k.

Теорема 4.5. *Если* $F_1(u) \in G(k)$ *и* $F_2(u) \in G(k)$, *то из равенства* $F_1(u) = F_2(u)$ *на отрезке* $0 \leqslant u \leqslant g^{2k-1}$, *где* g *– основание системы счисления* u, *следует* $F_1(u) = F_2(u)$ *для всех целых* $u \geqslant 0$.

Доказательство. Когда u принимает все значения на отрезке $0 \leqslant u \leqslant g^{2k-1}$, соответствующие ей последовательности образуют все множества последовательностей $x_1, x_2, ..., x_{2k-1}$ длиной $2k-1$. Поэтому равенство $F_1(u) = F_2(u)$ на отрезке $0 \leqslant u \leqslant g^{2k-1}$ означает равенство вычисляемых последовательностей длиной $2k-1$ для граф-схемы с числом вершин не более k. Отсюда следует, что граф-схемы эквивалентны (см. лемму 1.2). Последнее означает, что функции, определяемые этими граф-схемами, равны: $F_1(u) = F_2(u)$. Теорема доказана.

Являясь тривиальным следствием леммы 1.2, теорема 4.5 тем не менее проясняет задачу экстраполяции в классе арифметических функций, определяемых буквенными алгоритмами, и тем самым задачу экстраполяции в целом.

Допустим, что экстраполирование проводится в классе арифметических функций, определяемых буквенными алгоритмами. Задача поиска искомой функции $F(u)$ по тренировочной последовательности $F(0), F(1), ..., F(t)$ сводится к построению по этой последовательности подходящей граф-схемы. Это означает, что функция, реализуемая такой граф-схемой, должна на отрезке $0 \leqslant u \leqslant t$ совпадать с тренировочной последовательностью. Если бы этому условию удовлетворяла только одна простая граф-схема, она была бы решением экстраполяционной задачи. Поэтому добавим еще условие: *из всех граф-схем, реализующих тренировочную последовательность, следует выбрать граф-схему с наименьшим числом кустов.*

Теорема 4.6. *Пусть для* $F(u) \in G(k)$ *известны первые* t *значений* $F(0), F(1), ..., F(t-1)$. *Построенная по этой тренировочной последовательности граф-схема с наименьшим числом кустов реализует при* $t \geqslant g^{2k-1}$ *функцию* $F(u)$.

Доказательство. Построенная минимальная граф-схема имеет $s \leqslant k$ кустов. Поэтому реализуемая ею функция $F_1(u) \in G(s) \in G(k)$. Так как $F(u) \in G(k)$ и $F_1(u) \in G(k)$, то по теореме 4.5 при $t \geqslant g^{2k-1}$ имеем $F_1(u) = F(u)$. Теорема доказана.

Экстраполирование можно интерпретировать как обучение. Рассмотрим систему ученик–учитель. Ученик умеет по тренировочной последовательности

строить граф-схему согласно вышеприведенному условию. Учитель владеет методикой, заключающейся в том, что он последовательно сообщает ученику значения $F(0), F(1), F(2), \ldots$ После очередного задания $F(t)$, т. е. задания последовательности $F(0), F(1), \ldots, F(t)$, ученик строит граф-схему.

В дереве буквенного алгоритма, начальная вершина которого переходит в себя при $x = 0$, каждый ярус совпадает с начальной частью следующего яруса. Следовательно, вершины двух соседних ярусов с одинаковым порядковым номером u в ярусе будут эквивалентны, и поэтому при использовании канонического метода синтеза должны иметь один и тот же номер.

Приведем алгоритм экстраполирования арифметической функции, определяемой буквенным алгоритмом.

1. Записать в строку тренировочную последовательность $F(0), F(1), \ldots, F(t)$.

2. Выполнить $a := 1$.

3. Разбить строку на комплекты по g^a чисел (напомним, что g – основание системы счисления аргумента u).

4. Пронумеровать комплекты, придерживаясь следующего правила: разные комплекты пронумеровать разными номерами s, одинаковые – равными номерами s.

5. Считая каждый номер s входной вершиной куста, построить g исходящих дуг.

6. Номера s в такой же последовательности записать в выходные вершины кустов.

7. Если двум равным номерам входных вершин соответствуют разные комплекты номеров выходных вершин кустов, то перейти к п. 8, в противном случае – к п. 9.

8. Выполнить $a := a + 1$, перейти к п. 3.

9. Если для всех номеров входных вершин указаны номера выходов кустов, то перейти к п. 11; в противном случае – к п. 10.

10. Удлинить тренировочную последовательность, перейти к п. 1.

11. Записать в выходах кустов тренировочную последовательность.

12. Прекратить вычисления.

Полученные кусты образуют граф-схему искомого алгоритма.

Замечание. При $a = 1$ можно п. 4 считать также п. 11.

Пример 4.3. Для $F(u)$, где переменная u задана в двоичной системе счисления, дана тренировочная последовательность 111011011110001011. Выявить заключенную в ней закономерность.

Решение. Строим граф-схему (рис. 4.3, *а*) для $a = 1$. Полученная граф-схема полностью не определена, ибо для вершины $s = 3$ не указаны переходы. Требуется удлинить тренировочную последовательность. Граф-схема, построенная по удлиненной последовательности для $a = 1$ (рис. 4.3, *б*), полностью определена.

На рис. 4.4 изображена искомая граф-схема. Закономерность, по которой получена тренировочная последовательность, выражается словесно следующим образом: если в двоичной записи u имеется четное число пар 11, то $F(u) = 1$; в противном случае $F(u) = 0$. Например, $F(11011) = 1$, $F(0111101) = 0$.

Пример 4.4. Построить буквенный алгоритм, вычисляющий остатки от деления на 3 в двоичной системе счисления.

Решение. Строим алгоритм для последовательности остатков (рис. 4.5, *а*). Для вершины c не указаны переходы, т. е. алгоритм еще не построен, поэтому удлиняем тре-

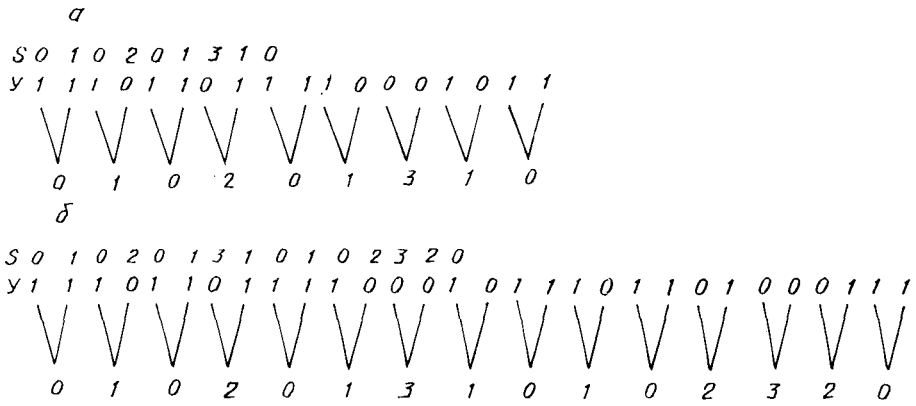


Рис. 4.3

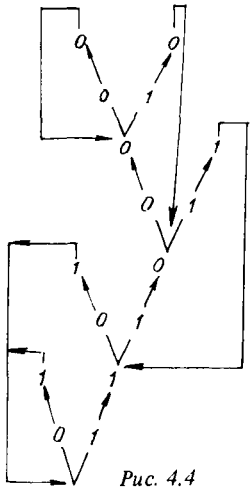


Рис. 4.4

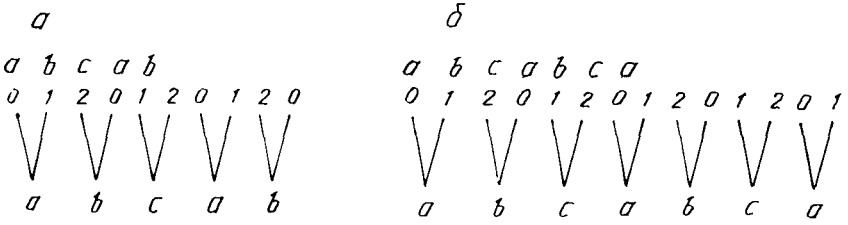


Рис. 4.5

нировочную последовательность до 14 чисел и строим алгоритм (рис. 4.5, *б*), граф-схема которого изображена на рис. 4.6. Как видим, достаточно было 12 членов тренировочной последовательности, чтобы построить граф-схему.

Длина l тренировочной последовательности остатков должна быть кратна m и g. Кусты такой последовательности затем периодически повторяются. Естественно брать в качестве длины наименьшее общее кратное: $l = \text{НОК}(m, g)$. Известно, что $\text{НОК}(m, g) = (m \cdot g) : \text{НОД}(m, g)$.

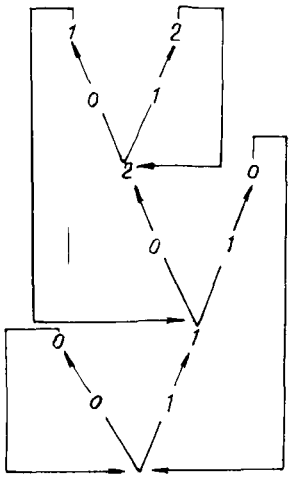


Рис. 4.6

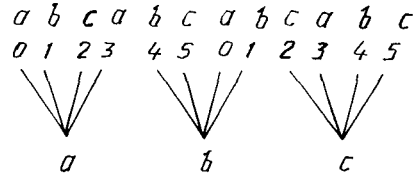


Рис. 4.7

Таким образом, для получения алгоритма вычисления остатков от деления на m в системе счисления с основанием g следует взять тренировочную последовательность длиной $l = (m \cdot g) : d$, где d – НОД (m, g). Число различных вершин равно $l : g = m : d$.

На рис. 4.7 изображена граф-схема для вычисления остатков от деления на 6 в четверичной системе счисления. Так как НОД $(6, 4) = 2$, то $l = (6 \cdot 4) : 2 = 12$.

Пример 4.5. По заданным суммам двух слагаемых найти алгоритм сложения в двоичной системе счисления.

Р е ш е н и е. Вначале следует сформулировать задачу в терминах дискретной экстраполяции. Пару слагаемых x, y будем задавать в виде одного числа u. Число u представляем в четверичной системе счисления. Каждый разряд u состоит из пары одноименных разрядов x, y, причем разряды в u расположены в обратном порядке, т. е. младшие разряды x, y образуют старший разряд в u и т. д. Например, если $x = 1100$, $y = 1001$, то $u = 01001011$.

Возьмем $0 \leqslant x \leqslant 7$ и $0 \leqslant y \leqslant 7$, следовательно, $0 \leqslant u \leqslant 63$. Построим для $v = F(u)$ тренировочную последовательность, считая v старшим разрядом (исключая последний перенос, если он есть) суммы $x + y$. Это означает, что при одноразрядных слагаемых значение $F(u)$ равно первому разряду u_0 суммы $x + y$, при двухразрядных слагаемых $F(u) = u_1$ – второму разряду суммы, при трехразрядных слагаемых $F(u) = u_2$ – третьему разряду суммы $x + y$. Например, для $u = 27$ имеем $u = 011011$. Следовательно, $x = 110$, $y = 101$, а сумма $110+101 = 1011$, поэтому $F(27) = 0$. Найдем $F(5)$. В двоичной системе $5 = 101$. Отсюда $x = 00$, $y = 11$, а сумма $00 + 11 = 11$. Итак, $F(5) = 1$.

В табл. 4.6 записаны значения функции $v = F(u)$.

Построим граф-схему в четверичной системе счисления для указанной в табл. 4.6 тренировочной последовательности длиной 64.

Выпишем столбец v и разобьем его на комплекты по четыре числа в каждом, которые объединим в кусты. Так как разных комплектов только два: (0, 1, 1, 0) и (1, 0, 0, 1), то будет два различных номера: a и b.

После переноса всех номеров в верхнюю строку (рис. 4.8) найдем переходы для вершин. В результате получим граф-схему (рис. 4.9), которая определяет алгоритм сложения двух чисел в двоичной системе счисления.

Таблица 4.6

u	v	u	v	u	v	u	v
0	0	16	0	32	0	48	0
1	1	17	1	33	1	49	1
2	1	18	1	34	1	50	1
3	0	19	0	35	0	51	0
4	0	20	0	36	0	52	1
5	1	21	1	37	1	53	0
6	1	22	1	38	1	54	0
7	0	23	0	39	0	55	1
8	0	24	0	40	0	56	1
9	1	25	1	41	1	57	0
10	1	26	1	42	1	58	0
11	0	27	0	43	0	59	1
12	1	28	1	44	1	60	1
13	0	29	0	45	0	61	0
14	0	30	0	46	0	62	0
15	1	31	1	47	1	63	1

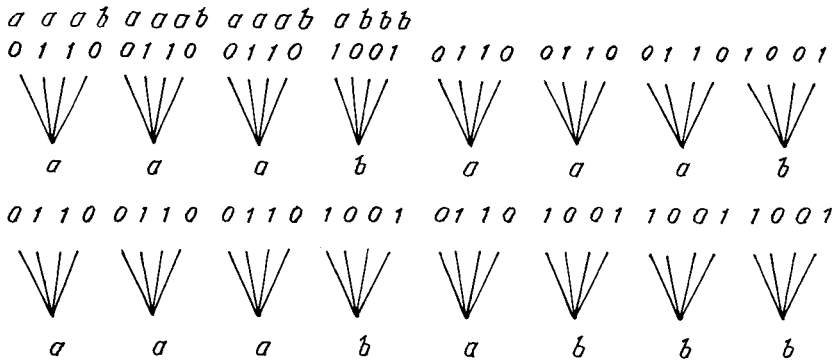


Рис. 4.8

В примерах 4.2–4.5 построение граф-схемы алгоритма по заданной тренировочной последовательности завершалось при $\alpha = 1$. Рассмотрим пример, в котором $\alpha > 1$.

Пример 4.6. Задана последовательность из первых 64 значений арифметической функции $y = F(u)$: 2,1,2,1,1,2,2,1,2,1,2,1,2,1,1,2,2,1,2,1,2,1,2,1,2,1,1,2,2,1,2,1,2,1,2,1,2,1,1,2,2,1,2,1,2,1,2,1,2,1,1,2,2,1,2,1,2,1,2,1,1,2,2,1,2,1,2,1,2,1,2,1. Найти алгоритм вычисления $F(u)$ для любого натурального числа u, заданного в двоичной системе счисления.

Р е ш е н и е. Согласно п. 1–4 алгоритма экстраполяции, запишем в строку последовательность, разобьем ее на комплекты по 2 числа: 21, 21, 12, 21, ибо $\alpha = 1$, и пронумеруем их. Согласно п. 5, 6 алгоритма, построим кусты, у выходов которых запишем полученную последовательность номеров a, a, b, a (рис. 4.10, a).

Уже для $\alpha = 1$ замечаем, что выполняется п. 7 алгоритма: двум номерам a соответствуют разные комплекты (a, a) и (b, a). Поэтому полагаем $\alpha = 2$ и берем комплекты из

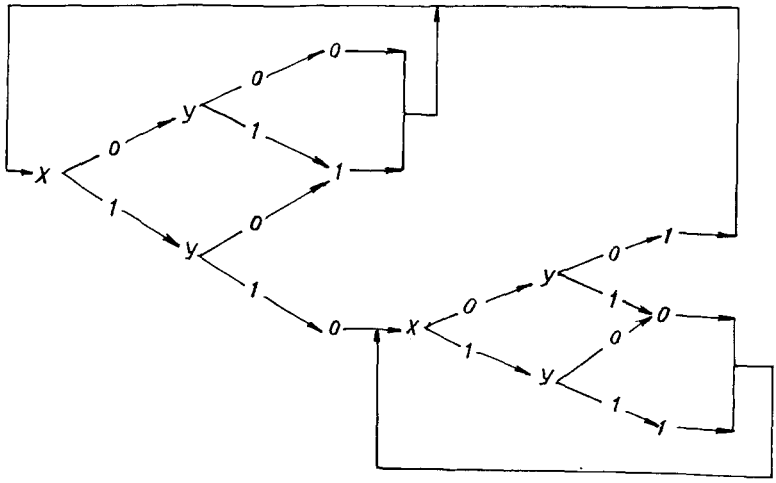


Рис. 4.9

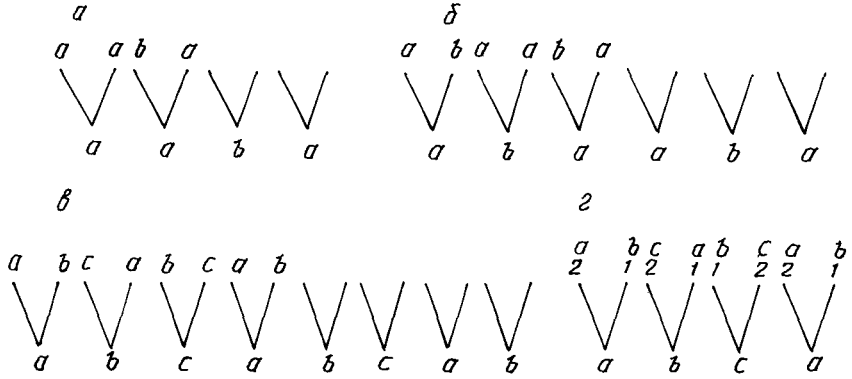


Рис. 4.10

$2^2 = 4$ чисел: 2121, 1221, 2121, 2121, 1221, 2121, ... Нумерация их доставляет последовательность *a, b, a, a, b, a,* ... Согласно п. 5, 6 алгоритма, строим кусты (рис. 4.10, *б*). Для кустов выполняется п. 7: двум номерам *a* соответствуют разные комплекты (*a, b*), (*b, a*).

Полагаем $a = 3$ и разбиваем последовательность на комплекты из $2^3 = 8$ чисел: 21211221, 21212121, 12212121, 21211221, 21212121, 12212121, 21211221, 21212121,... В соответствии с п. 4 получаем последовательность номеров *a, b, c, a, b, c, a, b,* ...

Согласно п. 5, 6 алгоритма экстраполяции, строим кусты (рис. 4.10, *в*). Как видим, п. 7 не выполняется, п. 9 выполняется. Согласно п. 11, записываем последовательность номеров у выходов кустов (рис. 4.10, *г*). Построение завершено. На рис. 4.11 изображена граф-схема искомого алгоритма.

Как отмечалось ранее, если в конечных вершинах с метками продолжить построение соответствующих кустов, образуется потенциально бесконечное дерево.

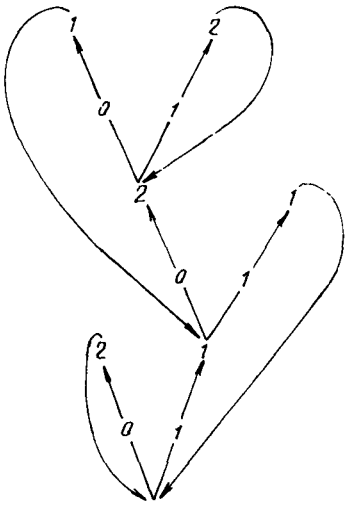


Рис. 4.11

Пусть в граф-схеме k вершин. Пронумеруем их и сохраним в дереве такую же нумерацию. Если номера некоторого яруса уже имеются в предыдущих ярусах, номера следующих ярусов также содержатся в них.

Предположим, что в отличие от первых $\nu - 1$ ярусов, в ν ярусах уже содержатся все k номеров. Значит, в каждом из этих ярусов имеется хотя бы один номер, отличный от номеров предыдущих ярусов. Следовательно, $\nu \leqslant k$. Итак, в первых k ярусах дерева обязательно содержатся все k номеров вершин граф-схемы.

Теорема 4.7. *Пользуясь алгоритмом экстраполирования, получают граф-схему искомого буквенного алгоритма.*

Доказательство. Пусть по тренировочной последовательности построена граф-схема при данном α. Она будем минимальной, ибо число различных номеров (кустов) не больше числа различных комплектов длиной g^{α}. Удлинение тренировочной последовательности не меняет нумерацию, если уже получена граф-схема. Согласно теореме 4.6, обучение осуществилось.

Покажем теперь, что процесс построения всегда конечен. Пусть простая граф-схема буквенного алгоритма, которым определяется арифметическая функция, содержит k кустов. В дереве из k ярусов для арифметической функции находятся все неэквивалентные вершины. Это значит, что в верхнем ярусе длиной g^k дерева, или, иначе, для $0 \leqslant u \leqslant g^k - 1$, уже имеются все неэквивалентные вершины, т. е. все вершины искомой граф-схемы. Но неэквивалентные вершины будут $(k-1)$-отличимы (см. следствие из леммы 1.2). Поэтому деревья из $(k-1)$-го яруса, корни которых находятся в вершинах $0 \leqslant u \leqslant g^k - 1$, различают все неэквивалентные вершины и определяются комплектами из g^{k-1} чисел. Следовательно, длина тренировочной последовательности $l \leqslant g^{2k-1}$ и $\alpha \leqslant k - 1$. Теорема доказана.

Задача построения алгоритмов по тренировочной последовательности была рассмотрена в классе буквенных алгоритмов, начальная вершина которых инвариантна относительно значения $x = 0$. Задачу построения алгоритма по результату его работы в классе буквенных алгоритмов следует формулировать следующим образом: заданы все входные и соответствующие им выходные последовательности длиной $l \leqslant k$. Требуется найти буквенный алгоритм, который заданные входные последовательности перерабатывает в соответствующие выходные последовательности.

Поставленная задача сводится к задаче синтеза для алгоритмов с инвариантной относительно $x = 0$ начальной вершиной. Для этого вместо искомого алгоритма Γ строят алгоритм Γ_1, граф-схема которого образована последовательным соединением куста в вершине $x = 1$ с входом граф-схемы Γ. Дугу

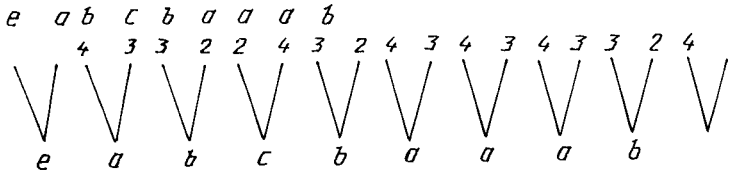


Рис. 4.12

$x = 0$ куста направляют к начальной вершине куста. В граф-схеме Γ_1 начальный куст аннулируют и получают граф-схему искомого алгоритма.

Входная последовательность $x_1, x_2, ..., x_m$ для алгоритма Γ порождает входную последовательность $g, x_1, x_2, ..., x_m$ для алгоритма Γ_1, поэтому числовой эквивалент $u = x_m + gx_{m-1} + ... + g^{m-1}x_1 + g^m$. Этим упорядочивают вершины граф-схемы искомого алгоритма и задание их в тренировочной последовательности. Например, вершина граф-схемы Γ, определяемая входной последовательностью $x_1 = 1, x_2 = 0, x_3 = 2$ при $g = 3$, в тренировочной последовательности имеет порядковый номер $u = 2 + 3 \cdot 0 + 3^2 \cdot 1 + 3^3 = 38$. В тренировочной последовательности первые g значений не определены, так как $u > g - 1$.

Пример 4.7. Построить буквенный алгоритм по тренировочной последовательности 4,3,3,2,2,4,3,2,4,3,4,3,4,3,3,2,4 от переменной x значности $g = 2$.

Р е ш е н и е. Выписываем тренировочную последовательность, считая $u = 2$ для первого ее члена, и нумеруем указанным способом кусты (рис. 4.12). Так как переходы для вершин первой строки определены, то задана граф-схема искомого алгоритма (рис. 4.13).

Сформулируем экстраполяционную задачу в классе алгоритмов заданного алфавита операторов.

Пусть область определения операторов есть счетное множество $\{a_i\}$ пронумерованных элементов. Искомый алгоритм A задан тренировочной последовательностью $a_1 = A(a_1), a_2 = A(a_2), ..., a_s = A(a_s)$ длиной s. Требуется построить алгоритм, значения которого совпадают с тренировочной последовательностью на первых s элементах a_i области. Покажем, что при достаточно большом s существует только один алгоритм, удовлетворяющий поставленному условию.

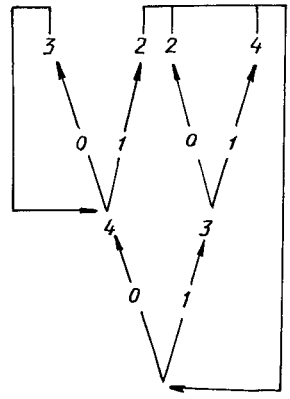


Рис. 4.13

Обозначим через $N(k)$ число таких граф-схем в заданном алфавите, которые содержат не более k вершин, а через $G(k)$ — соответствующий класс алгоритмов.

Для двух неэквивалентных алгоритмов $A_1 \in G(k), A_2 \in G(k)$ можно указать элемент a_j, который их отличает, т.е. $A_1(a_j) \neq A_2(a_j)$. Так как число элементов a_j не более $N^2(k)$, выберем максимальный номер $t(k)$ для этих элементов.

На последовательности $a_1, a_2, \ldots, a_s$ при $s \geqslant t(k)$ любые неэквивалентные алгоритмы из $G(k)$ будут отличимы. Действительно, построенная по тренировочной последовательности минимальная граф-схема будет содержать $r \leqslant k$ вершин, т. е. соответствующий алгоритм A принадлежит классу $G(r)$ и тем более классу $G(k)$. Следовательно, минимальная граф-схема определяет искомый алгоритм экстраполяционной задачи.

Вышеизложенное можно интерпретировать следующим образом: закономерность, присущая бесконечной последовательности, может быть обнаружена уже в некоторой ее начальной части (принцип индукции).

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. *Абрамов С.А.* Математические построения и программирование. – М.: Наука, 1978. – 191 с.
2. *Блох А.Ш.* Граф-схемы и их применение. – Мн.: Выш. шк., 1975. – 302 с.
3. *Катленд Н.* Вычислимость. Введение в теорию рекурсивных функций. – М.: Мир, 1983. – 256 с.
4. *Криницкий Н.А.* Алгоритмы вокруг нас. – М.: Наука, 1984. – 224 с.
5. *Мальцев А.И.* Алгоритмы и рекурсивные функции. – М.: Наука, 1965. – 391 с.
6. *Манин Ю.И.* Вычислимое и невычислимое. – М.: Сов. радио, 1980. – 126 с.
7. *Марков А.А., Нагорный Н.М.* Теория алгоритмов. – М.: Наука, 1984. – 432 с.
8. *Минский М.* Вычисления и автоматы. – М.: Мир, 1971. – 364 с.
9. *Орлов В.А.* Граф-схемы алгоритмов распознавания. – М.: Наука, 1982. – 116 с.
10. *Танаев В.С., Поварич М.П.* Синтез граф-схем алгоритмов выбора решений. – Мн.: Наука и техника, 1974. – 111 с.
11. *Трахтенброт Б.А.* Алгоритмы и вычислительные автоматы. – М.: Сов. радио, 1974. – 200 с.
12. *Роджерс Х.* Теория рекурсивных функций и эффективная вычислимость. – М.: Мир, 1972. – 624 с.
13. *Успенский В.А.* Лекции о вычислимых функциях. – М.: Физматгиз, 1960. – 492 с.

ОГЛАВЛЕНИЕ

Абрам Шлемович Блох

ГРАФ-СХЕМЫ И АЛГОРИТМЫ

Зав. редакцией Е.В. Сукач. Редактор М.С. Молчанова. Мл. редактор В.М. Кушилевич. Худож. редактор Ю.С. Сергачев. Техн. редактор Л.И. Счисленок. Корректоры Р.К. Логинова, И.И. Ганелес. Оператор А.И. Маль.

ИБ № 2483

Подписано в печать 31.08.87г. АТ 14901. Формат 60x90 1/16. Бумага офсет. Офсет. печать. Гарнитура Пресс-Роман. Усл. печ. л. 9. Усл. кр.-отт. 9,375. Уч.-изд.л. 10,41.Тираж 8200 экз. Зак. 6045. Цена 40 к.

Издательство "Вышэйшая школа" Государственного комитета БССР по делам издательств, полиграфии и книжной торговли. 220048, Минск, проспект Машерова, 11.

Типография "Победа". 222310, Молодечно, ул. Тавлая, 11.

Отпечатано с оригинала-макета, подготовленного в издательстве "Вышэйшая школа".