При задании условий работы управляющего автомата в виде совокупности частных ЛСА, очевидно, в каждую из таких ЛСА могут входить одни и те же операторы и ЛУ. Поэтому когда вместо каждого символа частной ЛСА будут подставлены соответствующие выражения из операторов и ЛУ, ЛСА, описывающая общий алгоритм работы автомата, может оказаться не минимальной. При этом в такой ЛСА повторяются не только ЛУ, но и операторы. Однако описанными выше способами такое повторение операторов не может быть исключено.

Для минимизации ЛСА с повторяющимися операторами и ЛУ может быть использован метод объединения ЛСА [75]. Вначале этот метод рассмотрим для случая попарного объединения ЛСА. Произведем объединение двух частных ЛСА (3.13) и (3.14), описывающих два режима работы УА:

$$\mathfrak{A}_{\mathrm{I}} = \downarrow^{2} F_i p_1 \uparrow^{1} p_3 \uparrow^{2} \downarrow^{4} F_j p_4 \uparrow^{3} p_5 \uparrow^{4} p_2 \uparrow^{4} A_1 A_2 \omega \uparrow^{2} \downarrow^{3} A_3 \omega \uparrow^{2} \downarrow^{1}; \tag{3.13}$$

$$\mathfrak{A}_{\mathrm{II}} = \downarrow^{1} F_j p_4 \uparrow^{2} p_6 \uparrow^{1} \downarrow^{4} F_i p_1 \uparrow^{3} p_7 \uparrow^{4} p_2 \uparrow^{4} A_1 A_2 \omega \uparrow^{1} \downarrow^{3} A_3 \omega \uparrow^{1} \downarrow^{2}. \tag{3.14}$$

Общий алгоритм функционирования УА имеет вид

$$\mathfrak{A} = \downarrow^{2} r \uparrow^{1} \mathfrak{A}_{\mathrm{I}} \omega \uparrow^{2} \downarrow^{1} \mathfrak{A}_{\mathrm{II}} \omega \uparrow^{2}. \tag{3.15}$$

В зависимости от значения параметра r, задаваемого извне, можно настраивать управляющий автомат на выполнение $\mathfrak{A}_{\mathrm{I}}$ или $\mathfrak{A}_{\mathrm{II}}$.

Если подставить в ЛСА (3.1) выражения для $\mathfrak{A}_{\mathrm{I}}$, $\mathfrak{A}_{\mathrm{II}}$, то получим ЛСА с 21 членом (как и ранее, здесь не учитываем число тождественно-ложных условий).

Процесс построения объединенной ЛСА, в которой операторы не повторяются, состоит в следующем. По МСА двух частных ЛСА составляется МСА объединенной ЛСА, каждый элемент которой

$$a_{ij} = \bigvee_{l=1,2} a_{ij}^{l} \beta_i^{l}, \tag{3.16}$$

где β_i^{l} — определяющая функция, равная функции $f(r_1, \ldots, r_n)$, принимающей единичное значение на наборе значений переменных $r_1, \ldots, r_n$, который сопоставлен с ЛСА $\mathfrak{A}_l$. Если оператор A_i не входит в ЛСА $\mathfrak{A}_l$, то в функцию β_i^{l} соответствующий набор значений $r_1, \ldots, r_m$ войдет в качестве условного.

Если число наборов значений переменных $r_1, \ldots, r_n$ больше числа объединяемых ЛСА, то в определяющую функцию могут входить конъюнкции, которые принимают единичные значения на неиспользуемых наборах.

В нашем случае для всех операторов ЛСА $\mathfrak{A}_{\mathrm{I}}$ $\beta_i^{\mathrm{I}} = r$, а для всех операторов ЛСА $\mathfrak{A}_{\mathrm{II}}$ $\beta_i^{\mathrm{II}} = \bar{r}$, так как в $\mathfrak{A}_{\mathrm{I}}$ и $\mathfrak{A}_{\mathrm{II}}$ входят все операторы.

Составим предварительно МСА, равносильные ЛСА (3.13) и (3.14), в виде табл. 3.10 и 3.11:

Таблица 3.10

$\mathfrak{A} =$	A_1	A_2	A_3	F_i	F_j	$A_к$
A_0				1		
A_1		1				
A_2				1		
A_3				1		
F_i				$p_1\bar{p}_3$	$p_1 p_3$	$\bar{p}_1$
F_j	$p_2 p_4 p_5$		$\bar{p}_4$		$\bar{p}_2 p_4 p_5 \vee p_4 \bar{p}_5$	

Таблица 3.11

$$\mathfrak{A}_{II}=\begin{array}{c|ccccccc} & A_1 & A_2 & A_3 & & F_i & F_j & A_к \\ A_0 & & & & & & 1 & \\ A_1 & & 1 & & & & & \\ A_2 & & & & & & 1 & \\ A_3 & & & & & & 1 & \\ F_i & p_1p_2p_7 & & \bar{p}_1 & & p_1\bar{p}_2p_7 \vee p_1\bar{p}_7 & & \\ F_j & & & & & p_4p_6 & p_4\bar{p}_6 & \bar{p}_4 \end{array}$$

После этого по указанному выше правилу получим следующую МСА, равносильную объединенной ЛСА (табл. 3.12):

Таблица 3.12

$$\mathfrak{A}=\begin{array}{c|ccccc} & A_1 & A_2 & A_3 & F_i & F_j \\ A_0 & & & & r & \bar{r} \\ A_1 & & r\vee\bar{r}=1 & & & \\ A_2 & & & & r & \bar{r} \\ A_3 & & & & r & \bar{r} \\ F_i & p_1p_2p_7\bar{r}\vee\bar{p}_1r & & \bar{p}_1\bar{r} & p_1\bar{p}_3r\vee p_1\bar{p}_2p_7\bar{r}\vee p_1\bar{p}_7\bar{r} & p_1p_3r \\ F_j & p_2p_4p_5r\vee\bar{p}_4\bar{r} & & \bar{p}_4r & p_4p_6\bar{r} & \bar{p}_2p_4p_5r\vee p_4\bar{p}_5r\vee p_4\bar{p}_6\bar{r} \end{array}$$

От МСА (табл. 3.12) перейдем к системе формул перехода, после чего преобразуем ее в систему скобочных формул перехода, где объединим общие выражения:

$$\left\{\begin{array}{c} A_0 \to \downarrow^{\alpha} rF_i \vee \bar{r}F_j; \\ A_1 \to A_3; \\ A_2 \to rF_i \vee \bar{r}F_j = \omega \uparrow^{\alpha}; \\ A_3 \to rF_i \vee \bar{r}F_j = \omega \uparrow^{\alpha}; \\ F_i \to p_1p_2p_7\bar{r}A_1 \vee \bar{p}_1rA_1 \vee \bar{p}_1\bar{r}A_3 \vee p_1\bar{p}_3rF_i \vee \\ \vee p_1\bar{p}_2p_7\bar{r}F_i \vee p_1\bar{p}_7\bar{r}F_i \vee p_1p_3rF_j = p_1(r(p_3F_j \vee \bar{p}_3F_i) \vee \\ \vee \bar{r}(p_7(p_2A_1 \vee \bar{p}_2F_i) \vee \bar{p}_7F_i)) \vee \bar{p}_1(rA_1 \vee \bar{r}A_3) = \\ = p_1(r(p_3F_j \vee \bar{p}_3F_i) \vee \bar{r}(p_7 \downarrow^{\beta} (p_2A_1 \vee \bar{p}_2F_i) \vee p_7F_i)) \vee \\ \vee \bar{p}_1(rA_1 \vee \bar{r}A_3); \\ F_j \to p_2p_4p_5rA_1 \vee \bar{p}_4\bar{r}A_1 \vee \bar{p}_4rA_3 \vee p_4p_6\bar{r}F_i \vee \end{array}\right.$$

$$\vee \bar{p}_2 p_4 p_5 r F_j \vee p_4 \bar{p}_5 r F_j \vee p_4 \bar{p}_6 \bar{r} F_j =$$

$$= r(p_4(p_5(p_2 A_1 \vee \bar{p}_2 F_j) \vee \bar{p}_5 F_j) \vee \bar{p}_4 A_3) \vee \bar{r}(p_4(p_6 F_i \vee$$

$$\vee \bar{p}_6 F_j) \vee \bar{p}_4 A_1) = r(p_4(p_5 \omega \uparrow^{\beta} \vee \bar{p}_5 F_j) \vee$$

$$\vee \bar{p}_4 A_3) \vee \bar{r}(p_4(p_6 F_i \vee \bar{p}_6 F_j) \vee \bar{p} A_1).$$

Теперь нетрудно получить и объединенную ЛСА:

$$\mathfrak{A} = \downarrow^1 r \uparrow^2 \downarrow^6 F_i p_1 \uparrow^3 r \uparrow^4 p_3 \uparrow^6 \downarrow^2 F_j r \uparrow^7 p_4 \uparrow^8 p_5 \uparrow^2 \omega \uparrow^9 \downarrow^3 r \uparrow^8 \downarrow^5$$

$$A_1 A_2 \omega \uparrow^1 \downarrow^4 p_7 \uparrow^6 \downarrow^9 p_2 \uparrow^6 \omega \uparrow^6 \downarrow^7 p_4 \uparrow^5 p_6 \uparrow^2 \omega \uparrow^6 \downarrow^8 A_3 \omega \uparrow^1, \quad (3.17)$$

в которой имеется 17 членов вместо 21 в ЛСА (3.15).

Если в ЛСА (3.17) подставить значение $r=1$, то читатель может легко убедиться, что получим ЛСА, равносильную ЛСА (3.13), а при подстановке $r=0$ получим ЛСА, равносильную ЛСА (3.14).

Очевидно, при объединении L ЛСА вместо одной переменной потребуется n переменных $r_1, \ldots, r_n$, $2^n \geqslant L$, так как каждой из L ЛСА сопоставляется своя, отличная от других конъюнкция $R_l = r_1^{\sigma_1^l}, \ldots, r_n^{\sigma_n^l}$, называемая *определяющей конъюнкцией.*

Таким образом, объединенной ЛСА $\mathfrak{A}$ называют ЛСА, удовлетворяющую следующим условиям:

1) если оператор A_i входит хотя бы в одну из частных ЛСА $\mathfrak{A}_j$, то он обязательно входит в ЛСА $\mathfrak{A}$, причем только один раз (если в каждой из частных ЛСА нет повторений одинаковых операторов);

2) при подстановке набора значений переменных $\ldots, r_n$, на котором $R_j=1$, ЛСА $\mathfrak{A}$ превращается в ЛСА, равносильную частной ЛСА $\mathfrak{A}_j$.

Предполагая наличие только неповторяющихся операторов в каждой из частных ЛСА, получим, что каждый оператор в объединенную ЛСА входит только один раз. Однако в различных частных ЛСА после одного и того же оператора могут выполняться различные члены ЛСА. Поэтому каждый раз при выполнении оператора объединенной ЛСА необходимо знать в какой из частных ЛСА он в данный момент выполняется.

Проверка значений переменных $r_1, \ldots, r_n$, осуществляемая после выполнения оператора, входящего в различные частные ЛСА, позволяет выбрать тот член объединенной ЛСА, который должен выполняться в частной ЛСА, соответствующей данному набору значений переменных $r_1, \ldots, r_n$.

Нетрудно понять, что если оператор входит только в одну из L ЛСА, то

$$\beta_i^l = R_l \vee \frac{R_{i_1}}{0} \vee \ldots \vee \frac{R_{i_2 n}}{0} = 1.$$

При объединении L ЛСА определяющая функция может быть недоопределенной, поэтому из (3.16) следует, что элементы матрицы объединенной ЛСА могут быть недоопределены.

Получаемая при объединении L ЛСА матрица с такими недоопределенными элементами является недетерминированной МСА $\widehat{\mathfrak{A}}$.

Легко понять, что недетерминированная МСА переводится в МСА при подстановке вместо переменных $r_1, \ldots, r_n$ в β_i^l набора значений $\sigma_1^l, \ldots, \sigma_n^l$, на котором $R_l = r_1^{\sigma_1^l}, \ldots, r_n^{\sigma_n^l} = 1$.

Если с каждой из L объединяемых ЛСА сопоставлена лишь одна определяющая конъюнкция, а $L < 2^n$, то $2^n - L$ конъюнкций переменных $r_1, \ldots, r_n$ останутся неиспользуемыми. Для того чтобы объединенная ЛСА обладала свойством детерминированности, эти конъюнкции должны быть переведены в используемые. Например, их можно сопоставить с одной из ЛСА или потребовать, чтобы на соответствующих наборах значений переменных $r_1, \ldots, r_n$ выполнилась бы «пустая» ЛСА вида $\mathfrak{A} = A_k$. Однако такое доопределение приводит в большинстве случаев к неоптимальной ЛСА. Оказывается более целесообразным составить неиспользуемые ЛСА, которые могут быть и неравносильны ни одной из используемых частных ЛСА. Построение таких неиспользуемых ЛСА целесообразно производить не сразу, а в процессе получения объединенной ЛСА, т. е. здесь задача в каком-то смысле аналогична задаче синтеза контактной схемы или схемы из функциональных элементов по не полностью определенным булевым функциям.

Таким образом, как и при объединении двух ЛСА, в данном случае по частным ЛСА составляются МСА. Затем выявляются определяющие функции β_i^l, которые в данном случае могут быть и недоопределенными. После этого в соответствии с (3.16) построим недетерминированную МСА. По недетерминированной МСА получим недоопределенные формулы перехода.

В процессе вынесения за скобки переменных, как и в случае учета неиспользуемых наборов значений логических условий [39], недоопределенные формулы перехода переводим в определенные скобочные формулы перехода, причем доопределение формул перехода следует делать так, чтобы обеспечить максимальное удовлетворение требований к порядку вынесе-

ния переменных за скобки и выявлению общих выражений, при выполнении которых может быть построена ЛСА с минимальным (или близким к минимальному) числом членов.

После этого, как и ранее, переходим к объединенной ЛСА.

Рассмотрим пример объединения трех следующих ЛСА:

$$\mathfrak{A}_1 = \downarrow^2 p_1 \uparrow^1 A_1 p_2 \uparrow^2 A_2 \downarrow^3 A_3 p_3 \uparrow^3 A_4 \downarrow^1 A_к; \quad (3.18)$$

$$\mathfrak{A}_2 = p_2 \uparrow^1 A_2 \downarrow^2 A_3 p_3 \uparrow^2 A_4 p_1 \uparrow^3 \downarrow^1 A_1 \downarrow^3 A_к; \quad (3.19)$$

$$\mathfrak{A}_3 = A_1 p_2 \uparrow^1 A_3 A_4 \downarrow^1 A_к. \quad (3.20)$$

Требуется построить объединенную ЛСА; так как $L=3$, число дополнительных переменных равно двум (дополнительные логические переменные r_1 и r_2). Очевидно, при этом одна определяющая конъюнкция будет неиспользуемой.

Если не осуществлять объединение заданных ЛСА, а получить общий алгоритм функционирования в соответствии с ЛСА, например, вида

$$\mathfrak{A} = r_1 \uparrow^1 \mathfrak{A}_1 \omega \uparrow^2 \downarrow^1 r_2 \uparrow^3 \mathfrak{A}_2 \omega \uparrow^2 \downarrow^3 \mathfrak{A}_3 \downarrow^2 A_к, \quad (3.21)$$

то общее число членов будет равно 21.

Для объединения ЛСА (3.18)—(3.20) перейдем к МСА (табл. 3.13—3.15).

Таблица 3.13

$\mathfrak{A}_1$

	A_1	A_2	A_3	A_4	$A_к$
A_0	p_1				$\overline{p_1}$
A_1	$p_1\overline{p_2}$	p_2			$\overline{p_1}\,\overline{p_2}$
A_2			1		
A_3			$\overline{p_3}$	p_3	
A_4					1

Таблица 3.14

$\mathfrak{A}_2$

	A_1	A_2	A_3	A_4	$A_к$
A_0	$\overline{p_2}$	p_2			
A_1				1	
A_2			1		
A_3			$\overline{p_3}$	p_3	
A_4	p_1			$\overline{p_1}$	

Таблица 3.15

$\mathfrak{A}_3$

	A_1	A_3	A_4	$A_к$
A_0	1			
A_1		p_2		$\overline{p_2}$
A_3			1	
A_4				1

Теперь выберем определяющие конъюнкции R_1—R_3. Соседние определяющие конъюнкции должны быть приписаны тем парам ЛСА, для которых число совпадающих элементов МСА является максимальным. При этом упростится наибольшее число элементов объединенной МСА.

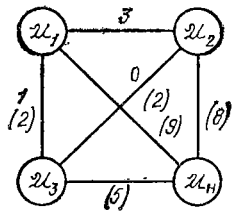

Рис. 3.2

В соответствии с этим условием приписывания определяющих конъюнкций подсчитываем число одинаковых элементов различных пар МСА. При этом будем подсчитывать число совпадающих элементов между каждой из всех четырех МСА.

Так как элементы неиспользуемой МСА не определены, то при сравнении с ней число совпадающих элементов берется равным числу элементов используемой МСА, не равных нулю.

Кроме того, заметим, что если из сравнения двух используемых МСА выяснится, что оператор A_i входит лишь в МСА $\mathfrak{A}_j$, а в МСА $\mathfrak{A}_l$ не содержится, то в число совпадающих элементов МСА $\mathfrak{A}_j$ и $\mathfrak{A}_l$ входят все ненулевые элементы строки A_i МСА $\mathfrak{A}_j$.

Для наглядности связи между МСА изобразим в виде графа, каждому ребру которого припишем цифру, указывающую число одинаковых элементов у МСА, приписанных вершинам, соединенным этим ребром. Для нашего примера такой отмеченный граф изображен на рис. 3.2, где $\mathfrak{A}_н$ — вершина, соответствующая неиспользуемой МСА. Тем парам МСА, которым соответствует наибольшее число совпадающих элементов, должны быть приписаны соседние определяющие конъюнкции. Надо заметить, что такой способ выявления определяющих конъюнкций не гарантирует построения объединенной ЛСА с минимальным числом членов. Однако он позволяет получить вполне приемлемые решения. Кроме того, надо иметь в виду, что наибольшее упрощение может быть получено в том случае, когда соседние определяющие конъюнкции приписываются тем парам МСА, у которых одинаковыми являются достаточно сложные элементы. Поэтому целесообразно в первую очередь приписывать таким парам МСА соседние определяющие конъюнкции, если даже у них число одинаковых элементов меньше, чем у других пар МСА.

В рассматриваемом примере таких сложных одинаковых элементов нет, поэтому выявление определяющих конъюнкций осуществлено в соответствии с графом, показанным на рис. 3.2, т. е.

$$R_1=r_1r_2;\ R_2=r_1\bar{r}_2;\ R_3=\bar{r}_1r_2;\ R_н=\bar{r}_1\bar{r}_2,$$

где $R_н$ — неиспользуемая определяющая конъюнкция.

Теперь получим определяющие функции β_i^l:

Это определяющие конъюнкции. Их получают методом парных кодов, т.е. тем алгоритмам у которых больше всего совпадающих ячеек в таблицах МСА присваем парные коды. Например тут у R_2 и R_3 коды парные.

$$\beta_0^1 = \beta_1^1 = \beta_3^1 = \beta_4^1 = r_1 r_2 \vee \frac{\bar{r}_1 \bar{r}_2}{0} = r_1 r_2;$$

$$\beta_2^1 = r_1 r_2 \vee \frac{\bar{r}_1 \bar{r}_2}{0} \vee \frac{\bar{r}_1 r_2}{0} = \frac{r_1}{1} r_2;$$

$$\beta_0^2 = \beta_1^2 = \beta_3^2 = \beta_4^2 = r_1 \bar{r}_2 \vee \frac{\bar{r}_1 \bar{r}_2}{0} = \frac{r_1}{1} \bar{r}_2;$$

$$\beta_2^2 = r_1 r_2 \vee \frac{\bar{r}_1 \bar{r}_2}{0} \vee \frac{\bar{r}_1 r_2}{0} = \frac{r_1}{1} r_2;$$

$$\beta_0^3 = \beta_1^3 = \beta_2^3 = \beta_4^3 = \bar{r}_1 r_2 \vee \frac{\bar{r}_1 \bar{r}_2}{0} = \bar{r}_1 \frac{r_2}{1}.$$

Строим недетерминированную МСА (табл. 3.16):

Таблица 3.16

	A_1	A_2	A_3	A_4	A_K
A_0	$r_1 r_2 p_1 \vee \frac{r_1}{1} \bar{r}_2 \bar{p}_2 \vee \bar{r}_1 \frac{r_2}{1}$	$\frac{r_1}{1} \bar{r}_2 p_2$			$r_1 r_2 \bar{p}_1$
A_1	$r_1 r_2 p_1 \bar{p}_2$	$r_1 r_2 p_2$	$\bar{r}_1 \frac{r_2}{1} p_2$		$r_1 r_2 \bar{p}_1 \bar{p}_2 \vee \frac{r_1}{1} \bar{r}_2 \vee \bar{r}_1 \frac{r_2}{1} \bar{p}_2$
$\hat{\mathfrak{A}} = A_2$			$\frac{r_1}{1} r_2 \vee \frac{r_1}{1} \bar{r}_2 = 1$		
A_3			$r_1 r_2 \bar{p}_3 \vee \frac{r_1}{1} \times \bar{r}_2 \bar{p}_2 = r_1 \bar{p}_3$	$r_1 r_2 p_3 \vee \frac{r_1}{1} \times \bar{r}_2 \bar{p}_3 \vee \bar{r}_1 \times \frac{r_2}{1} = r_1 p_3 \vee \bar{r}_1$	
A_4	$\frac{r_1}{1} \bar{r}_2 p_1$				$r_1 r_2 \vee \frac{r_1}{1} \bar{r}_2 \bar{p}_1 \vee \bar{r}_1 \frac{r_2}{1} = r_2 \vee \frac{r_1}{1} \bar{r}_2 \bar{p}_1$

Определяющие функции. Несколько случаев для их получения:

- Все блоки всех алгоритмов одинаковые – берём определющую конъюнкцию Алгоритма для которого ищем определющие функции и дизъюнктируем её с опр. Кон. Пустого алгоритма делённого на нуль.

- есть один «уникальный» блок – для него опр.ф. будет 1, т.к. мы дизъюнктируем опр.кон. алгоритма в котором есть этот «уникальный» блок со всеми опр.кон делёнными на нуль.

- данный блок есть в двух алгоритмах, но его нет в третьем – дизъюнктируем опр. кон. Алгоритмов в которых блок есть с опр.кон. пустого алгоритма дел-ым на нуль.

От недетерминированной МСА переходим к недоопределенным формулам перехода, которые затем преобразовываем:

$$A_0 \to r_1 r_2 p_1 A_1 \vee \frac{r_1}{1} \bar{r}_2 \bar{p}_2 A_1 \vee \bar{r}_1 \frac{r_2}{1} A_1 \vee \frac{r_1}{1} \bar{r}_2 p_2 A_2 \vee r_1 r_2 \bar{p}_1 A_к =$$

$$= r_1 (r_2 \downarrow^1 (p_1 A_1 \vee \bar{p}_1 A_к) \vee \bar{r}_2 (p_2 A_2 \vee \bar{p}_2 A_1)) \vee \bar{r}_1 A_1;$$

$$A_1 \to r_1 r_2 p_1 \bar{p}_2 A_1 \vee r_1 r_2 p_2 A_2 \vee \bar{r}_1 \frac{r_2}{1} p_2 A_3 \vee r_1 r_2 \bar{p}_1 \bar{p}_2 A_к \vee \frac{r_1}{1} \bar{r}_2 A_к \vee$$

$$\vee \bar{r}_1 \frac{r_2}{1} \bar{p}_2 A_к = r_1 (r_2 (p_2 A_1 \vee \bar{p}_2 (p_1 A_2 \vee \bar{p}_1 A_к) \vee \bar{r}_2 A_к)) \vee \bar{r}_1 (p_2 A_3 \vee \bar{p}_2 A_к) =$$

$$= r_1 (r_2 (p_2 A_2 \vee \bar{p}_2 \omega \uparrow^1) \vee \bar{r}_2 A_к) \vee \bar{r}_1 (p_2 A_3 \vee \bar{p}_2 A_к);$$

$$A_2 \to A_3;$$

$$A_3 \to r_1 \bar{p}_3 A_3 \vee r_1 p_3 A_4 \vee \bar{r}_1 A_4 = r_1 (\bar{p}_3 A_3 \vee p_3 A_4) \vee \bar{r}_1 A_4;$$

$$A_4 \to \frac{r_1}{1} \bar{r}_2 p_1 A_1 \vee r_2 A_к \vee \frac{r_1}{1} \bar{r}_2 \bar{p}_1 A_к = r_2 A_к \vee \bar{r}_2 (p_1 A_1 \vee \bar{p}_1 A_к) =$$

$$= r_2 A_к \vee \bar{r}_2 \omega \uparrow^1.$$

После этого составляем объединенную ЛСА:

$$\mathfrak{A} = r_1 \uparrow^1 r_2 \uparrow^2 \downarrow^6 p_1 \uparrow^7 \downarrow^1 A_1 r_1 \uparrow^4 r_2 \uparrow^7 p_2 \uparrow^6 \downarrow^3 A_2 \downarrow^5 A_3 r_1 \uparrow^8 p_3 \uparrow^5 \downarrow^8 A_4 r_2 \uparrow^6 \omega \uparrow^7 \downarrow^2 p_2 \uparrow^1 \omega \uparrow^3$$
$$\downarrow^4 p_2 \uparrow^7 \omega \uparrow^5 \downarrow^7 A_к, \qquad (3.22)$$

в которой содержится 16 членов вместо 21 в ЛСА (3.21).

В рассмотренном примере при определенных значениях дополнительных ЛУ выполняется одна из частных ЛСА, после чего процесс заканчивается.

Значения дополнительных логических условий r_1 и r_2 при этом должны быть заданы или каким-либо образом выбраны. Однако возможны случаи, когда частные ЛСА выполняются одна за другой и после выполнения всех ЛСА процесс повторяется. Тогда общий алгоритм функционирования автомата запишем в виде

$$\mathfrak{A} = \downarrow^1 \mathfrak{A}_1 \mathfrak{A}_2 \ldots \mathfrak{A}_L \omega \uparrow^1, \qquad (3.23)$$

при этом изменение значений дополнительных логических условий должно производиться в процессе выполнения общей ЛСА, а для формирования соответствующих их значений могут быть введены новые операторы $F_{\mathfrak{A}_1}$, $F_{\mathfrak{A}_2}$, ..., $F_{\mathfrak{A}_L}$, каждый из которых является одновременно оператором начала соответствующей ЛСА.

Таким образом, в каждой из частных ЛСА вместо оператора A_0 будет оператор $F_{\mathfrak{A}_i}$, который вместе с тем будет оператором конца ЛСА $\mathfrak{A}_{i-1}$. Теперь

$$\mathfrak{A} = \downarrow^1 F_{\mathfrak{A}_1} \mathfrak{A}_1 F_{\mathfrak{A}_2} \mathfrak{A}_2 \ldots F_{\mathfrak{A}_L} \mathfrak{A}_L \omega \uparrow^1. \qquad (3.24)$$

В ЛСА (3.24) порядок выполнения частных ЛСА единственный, однако может быть задан и некоторый алгоритм над частными ЛСА, так что допускается несколько различных последовательностей выполнения частных ЛСА. Тогда в общей ЛСА имеются внешние ЛУ, значения которых определяют эти последовательности. При объединении ЛСА эти условия входят в МСА объединенной ЛСА.

Хотя объединение частных ЛСА не всегда приводит к сокращению числа членов, но в большинстве практических случаев объединенная ЛСА содержит значительно меньшее число членов, чем общая ЛСА, в которой частные ЛСА не объединены.

Например, в [41] показано, что при объединении частных ЛСА, задающих алгоритм функционирования управляющего устройства одной из телефонных станций, может быть получена объединенная ЛСА с 76 членами вместо 202 членов в общей ЛСА, в которой частные ЛСА не объединены.

31 Определяющие функции. Процесс доопределения.

Опр.ф. -- в предыдущем вопросе выделены синим.

Как доопределять?

Вот дали тебе на объединение, допустим, три алгоритма. Ты уже построила по ним по отдельности МСА, по этим МСА в виде графа построила схему взаимосвязи между алгоритмами (т.е. где мы считаем у каких алгоритмов больше всего совпадающих ячеек), с помощью этого графа сделала определяющие конъюнкции, потом сделала определяющие функции и наконец построила объединённую МСА (далее ОМСА). Вот теперь и начинается магия. Процесс доопределения прост. Посмотри на таблицу 3.16 из предыдущего вопроса. Там есть записи типа: «r_1/1<и далее по тексту>», так вот, доопределение состоит в том, что мы выбираем, что оставить в записи – «r_1» или «1». Определённого алгоритма выбора нет, поэтому тут всё зависит от твоих действий. Главное выбирать так, чтобы в конечном итоге алгоритм оказался наиболее компактным. И если ты в одной строке выбираешь из, допустим, !r_2/1 (! – тут знак отрицания) 1, то желательно во всей ОМСА выбирать также, но следует помнить, что если в строке есть r_1, то должно быть и !r_1, если есть !r_2, то должно быть и r_2.

32 Алгоритмически неразрешимые проблемы. Примеры.

Алгоритмическая проблема — это проблема, в которой требуется найти единый метод (алгоритм) для решения бесконечной серии однотипных единичных задач. Такие проблемы называют также массовыми проблемами. Они возникали и решались в различных областях математики на протяжении всей ее истории. Примеры таких проблем рассматривались ранее.

Математики в начале XX в. столкнулись с тем, что для некоторых массовых проблем не удается подобрать общий алгоритм для их решения. В связи с этим возникла необходимость дать точное определение самому понятию алгоритма.

Алгоритмически неразрешимые проблемы

Алан Тьюринг высказал предположение (известное как Тезис Чёрча — Тьюринга), что любой алгоритм в интуитивном смысле этого слова может быть представлен эквивалентной машиной Тьюринга. Уточнение представления о вычислимости на основе понятия машины Тьюринга (и других эквивалентных ей понятий) открыло возможности для строгого доказательства алгоритмической неразрешимости различных массовых проблем (то есть проблем о нахождении единого метода решения некоторого класса задач, условия которых могут варьироваться в известных пределах). Простейшим примером алгоритмически неразрешимой массовой проблемы является так называемая проблема применимости алгоритма (называемая также проблемой остановки). Она состоит в следующем: требуется найти общий метод, который позволял бы для произвольной машины Тьюринга (заданной посредством своей программы) и произвольного начального состояния ленты этой машины определить, завершится ли работа машины за конечное число шагов, или же будет продолжаться неограниченно долго.

В течение первого десятилетия истории теории алгоритмов неразрешимые массовые проблемы были обнаружены лишь внутри самой этой теории (сюда относится описанная выше проблема применимости), а также внутри математической логики (проблема выво-димости в классическом исчислении предикатов). Поэтому считалось, что теория алгоритмов представляет собой обочину математики, не имеющую значения для таких её классических разделов, как алгебра или анализ. Положение изменилось после того, как А. А. Марков и Э. Л. Пост в 1947 году установили алгоритмическую неразрешимость известной в алгебре проблемы равенства для конечнопорождённых и конечноопределённых полугрупп (т. н. проблемы Туэ). Впоследствии была установлена алгоритмическая неразрешимость и многих других «чисто математических» массовых проблем. Одним из наиболее известных результатов в этой области является доказанная Ю. В. Матиясевичем алгоритмическая неразрешимость десятой проблемы Гильберта.

Алгоритмически неразрешимые проблемы

За время своего существования человечество придумало множество алгоритмов для решения разнообразных практических и научных проблем. Зададимся вопросом – а существуют ли какие-нибудь проблемы, для которых невозможно придумать алгоритмы их решения?

Утверждение о существовании алгоритмически неразрешимых проблем является весьма сильным – мы констатируем, что мы не только сейчас на знаем соответствующего алгоритма, но мы не можем принципиально никогда его найти.

Успехи математики к концу XIX века привели к формированию мнения, которое выразил Д. Гильберт – «в математике не может быть неразрешимых проблем», в связи с этим формулировка проблем Гильбертом на конгрессе 1900 года в Париже была руководством к действию, констатацией отсутствия решений в данный момент.

Первой фундаментальной теоретической работой, связанной с доказательством алгоритмической неразрешимости, была работа Курта Гёделя – его известная теорема о не-полноте символических логик. Это была строго формулированная математическая проблема, для которой не существует решающего ее алгоритма. Усилиями различных исследователей список алгоритмически неразрешимых проблем был значительно расширен.

Сегодня принято при доказательстве алгоритмической неразрешимости некоторой задачи сводить ее к ставшей классической задаче – «задаче останова».

Имеет место быть следующая теорема:

Теорема 3.1. Не существует алгоритма (машины Тьюринга), позволяющего по описанию произвольного алгоритма и его исходных данных (и алгоритм и данные заданы 2 символами на ленте машины Тьюринга) определить, останавливается ли этот алгоритм на этих данных или работает бесконечно.

Таким образом, фундаментально алгоритмическая неразрешимость связана с бесконечностью выполняемых алгоритмом действий, т.е. невозможностью предсказать, что для любых исходных данных решение будет получено за конечное количество шагов.

Тем не менее, можно попытаться сформулировать причины, ведущие к алгоритмической неразрешимости, эти причины достаточно условны, так как все они сводимы к проблеме останова, однако такой подход позволяет более глубоко понять природу алгоритмической неразрешимости:

а) Отсутствие общего метода решения задачи

Проблема 1: Распределение девяток в записи числа π;

Определим функцию f(n) = i, где n – количество девяток подряд в десятичной записи числа π, а i – номер самой левой девятки из n девяток подряд: π=3,141592… f(1) = 5.

Задача состоит в вычислении функции f(n) для произвольно заданного n.

Поскольку число π является иррациональным и трансцендентным, то мы не знаем никакой информации о распределении девяток (равно как и любых других цифр) в десятичной записи числа π. Вычисление f(n) связано с вычислением последующих цифр в разложении π, до тех пор, пока мы не обнаружим n девяток подряд, однако у нас нет общего метода вычисления f(n), поэтому для некоторых n вычисления могут продолжаться бесконечно – мы даже не знаем в принципе (по природе числа π) существует ли решение для всех n.

Проблема 2: Вычисление совершенных чисел;

Совершенные числа – это числа, которые равны сумме своих делителей, например: 28 = 1+2+4+7+14.

Определим функцию S(n) = n-ое по счёту совершенное число и поставим задачу вычисления S(n) по произвольно заданному n. Нет общего метода вычисления совершенных чисел, мы даже не знаем, множество совершенных чисел конечно или счетно, поэтому наш алгоритм должен перебирать все числа подряд, проверяя их на совершенность. Отсутствие общего метода решения не позволяет ответить на вопрос о останове алгоритма. Если мы проверили M чисел при поиске n-ого совершенного числа – означает ли это, что его вообще не существует?

Проблема 3: Десятая проблема Гильберта;

Пусть задан многочлен n-ой степени с целыми коэффициентами – P, существует ли алгоритм, который определяет, имеет ли уравнение P=0 решение в целых числах?

Ю.В. Матиясевич показал, что такого алгоритма не существует, т.е. отсутствует общий метод определения целых корней уравнения P=0 по его целочисленным коэффициентам.

б) Информационная неопределенность задачи

Проблема 4: Позиционирование машины Поста на последний помеченный ящик;

Пусть на ленте машины Поста заданы наборы помеченных ящиков (кортежи) произвольной длины с произвольными расстояниями между кортежами и головка находится у самого левого помеченного ящика. Задача состоит установке головки на самый правый помеченный ящик последнего кортежа.

Попытка построения алгоритма, решающего эту задачу приводит к необходимости ответа на вопрос – когда после обнаружения конца кортежа мы сдвинулись вправо по пустым ящикам на M позиций и не обнаружили начало следующего кортежа – больше на лен-те кортежей нет или они есть где-то правее? Информационная неопределенность задачи состоит в отсутствии информации либо о количестве кортежей на ленте, либо о максимальном расстоянии между кортежами – при наличии такой информации (при разрешении информационной неопределенности) задача становится алгоритмически разрешимой.

в) Логическая неразрешимость (в смысле теоремы Гёделя о неполноте)

Проблема 5: Проблема «останова» (см. теорема 3.1);

Проблема 6: Проблема эквивалентности алгоритмов;

По двум произвольным заданным алгоритмам (например, по двум машинам Тьюринга) определить, будут ли они выдавать одинаковые выходные результаты на любых исходных данных.

Проблема 7: Проблема тотальности;

По произвольному заданному алгоритму определить, будет ли он останавливаться на всех возможных наборах исходных данных. Другая формулировка этой задачи – является ли частичный алгоритм P всюду определённым?

3. Проблема соответствий Поста над алфавитом ∑

В качестве более подробного примера алгоритмически неразрешимой задачи рассмотрим проблему соответствий Поста (Э. Пост, 1943 г.). Мы выделили эту задачу, поскольку на первый взгляд она выглядит достаточно «алгоритмизуемой», однако она сводима к проблеме останова и является алгоритмически неразрешимой.

Постановка задачи:

Пусть дан алфавит : $\sum:|\sum| >= 2$ (для односимвольного алфавита задача имеет решение) и дано конечное множество пар из $\sum^{+}x\sum^{+}$, т.е. пары непустых цепочек произвольного языка над алфавитом : , ……, .

Проблема: Выяснить существует ли конечная последовательность этих пар, не обязательно различных, такая что цепочка, составленная из левых подцепочек, совпадает с последовательностью правых подцепочек – такая последовательность называется решаю-щей.

В качестве примера рассмотрим ∑ = {a,b}

1. Входные цепочки: (abbb, b), (a, aab), (ba, b)

Решающая последовательность для этой задачи имеет вид:

(a,aab) (a,aab) (ba,b) (abbb,b), так как : a a ba abbb ≡ aab aab b b

2. Входные цепочки: (ab,aba), (aba,baa), (baa,aa)

Данная задача вообще не имеет решения, так как нельзя начинать с пары (aba,baa) или (baa,aa), поскольку не совпадают начальные символы подцепочек, но если начинать с цепочки (ab,aba), то в последующем не будет совпадать общее количество символов «a», т.к. в других двух парах количество символов «a» одинаково.

В общем случае мы можем построить частичный алгоритм, основанный на идее упорядоченной генерации возможных последовательностей цепочек (отметим, что мы имеем счетное множество таких последовательностей) с проверкой выполнения условий задачи. Если последовательность является решающей – то мы получаем результативный ответ за конечное количество шагов. Поскольку общий метод определения отсутствия решающей последовательности не может быть указан, т.к. задача сводима к проблеме «останова» и, следовательно, является алгоритмически неразрешимой, то при отсутствии решающей последовательности алгоритм порождает бесконечный цикл.

В теории алгоритмов такого рода проблемы, для которых может быть предложен частичный алгоритм их решения, частичный в том смысле, что он возможно, но не обязательно, за конечное количество шагов находит решение проблемы, называются частично разрешимыми проблемами.

В частности, проблема останова так же является частично разрешимой проблемой, а проблемы эквивалентности и тотальности не являются таковыми.

33 Трудноразрешимые проблемы. Примеры

Под задачей понимается некоторый вопрос, на который нужно найти (вычислить) ответ. Задачи бывают массовыми (общими) и частными (индивидуальными). Общая задача определяется: • списком параметров – свободных переменных, конкретные значения которых неопределенны; • формулировкой условий – свойств, которыми должен обладать ответ (решение задачи). Массовые задачи часто называют алгоритмическими проблемами. Частная задача получается из массовой, если всем параметрам массовой задачи придать конкретные значения – здесь исходные данные. В общем случаи для любой алгоритмически разрешимой задачи существует несколько разрешающих алгоритмов. С практической точки зрения важно не просто существование разрешающих алгоритмов, а поиск среди них наиболее простого и наименее трудоемкого. Под сложностью задачи принято понимать минимальную из сложностей алгоритмов, решающих эту задачу. Компьютерные науки пока не накопили достаточно сведений для того, чтобы задача минимизации сложности алгоритма была поставлена математически строго. Поэтому без ответов остаются такие вопросы, связанные с нижней оценкой сложности алгоритмов, а значит, и сложности задач: • Существует ли для заданной задачи алгоритм минимальной сложности? • Как убедиться, что найденный алгоритм действительно минимальный или, напротив, не минимальный? • Можно ли для рассматриваемой задачи доказать, что никакой разрешающий ее алгоритм не может быть проще найденной нижней оценки сложности? При разработке алгоритмов можно наблюдать, что для некоторых задач можно построить алгоритм полиномиальной сложности. Такие задачи называют полиномиальными. Полиноминально разрешимые задачи можно успешно решать на компьютере и даже в тех случаях, когда они имеют большую размерность. Для других задач не удается найти полиномиальный алгоритм. поэтому их называют трудноразрешимыми. К классу трудноразрешимых задач относится большое число задач алгебры, математической логики, теории графов, теории автоматов и других разделов дискретной математики. В большинстве своем это так называемые переборные задачи.Переборная задача характеризуется экспоненциальным множеством вариантов, среди которых нужно найти решение, и может быть решена алгоритмом полного перебора. Переборный алгоритм имеет экспоненциальную сложность и может хорошо работать только для небольших размеров задачи. С ростом размера задачи число вариантов быстро растет, и задача становится практически неразрешимой методом перебора. Многие из переборных задач являются экспоненциальными по постановке. Экспоненциальные по постановке задачи не представляют особого интереса для теории алгоритмов, поскольку для них, очевидно, невозможно получить алгоритм полиномиальной сложности. Возникает вопрос: если известно, что некоторая задача алгоритмически разрешима, то неудача в разработке для нее полиномиального разрешающего алгоритма является следствием неумения конкретного разработчика или следствием каких-то свойств самой задачи? Ответ на этот вопрос дает классическая теория алгоритмов, которая классифицирует задачи по сложности. При этом классифицируются лишь распознавательные задачи – задачи, имеющие распознавательную форму. В распознавательной форме суть задачи сводится к распознаванию некоторого свойства, а ее решение – один из двух ответов: «да» или «нет». С точки зрения математической логики задаче распознавания свойства соответствует предикат P(x), где x – множество фактических значений входных переменных задачи. Существуют задачи, которые изначально имеют распознавательную форму. Например,

являются ли два графа изоморфными? Другой пример – задача о выполнимости булевой функции, которая является исторически первой распознавательной задачей, глубоко исследованной в теории алгоритмов. Многие задачи, которые в исходной постановке представлены в иной форме (к ним относятся задачи дискретной оптимизации), довольно просто приводятся к распознавательной форме. Например, если требуется найти хроматическое число графа, то формулируют вопрос: верно ли, что заданный граф является k – раскрашиваемый, где k – заданное целое положительное число. Между тем, имеются задачи, которые нельзя привести к распознавательной форме. Это. В первую очередь, конструктивные задачи – задачи на построение объектов дискретной математики, обладающих заданными свойствами: генерация всех подмножеств конечного множества; генерация всех n! Различных перестановок; построение плоской укладки графа; построение остовного дерева графа и т. п. Такие задачи могут быть как трудноразрешимыми, так и полиноминально разрешимыми. Они пока не попадают под существующую в теории алгоритмов классификацию.

Классификация задач по сложности

Задачи, как и алгоритмы принято классифицировать по сложности. Множество всех распознавательных задач, для которых существует полиномиальный разрешающий алгоритм, образуют класс P. Ясно, что распознавательные трудноразрешимые задачи не принадлежат классу P. Класс NP – это множество распознавательных задач, которые могут быть разрешены за полиномиальное время на недетерминированной машине Тьюринга (НМТ). Оракул предлагает решения, которые после проверки верификатором приобретают «юридическую» силу. Таким образом, задачи класса NP являются «полиноминально проверяемыми». Например, в задаче коммивояжера оракул предлагает некоторую перестановку всех вершин графа, а верификатор проверяет, образует ли эта перестановка гамильтонов цикл графа. Ясно, что такую проверку можно выполнить с полиномиальной сложностью – надо лишь проверить смежность соседних вершин. Построить одну перестановку вершин тоже можно с полиномиальной сложностью. Оба шага решения задачи полиномиальные, поэтому задача коммивояжера принадлежит классу NP. Трудноразрешимой ее делает факториальное число повторений этих шагов. Следует отметить, что такую двухшаговую процедуру поиска решения можно применить к любой распознавательной задаче полиномиальной сложности.

7.9. Трудноразрешимые задачи

В разделе 7.6 рассмотрено понятие алгоритмически неразрешимой проблемы. Для многих практически важных задач такая проблема, как правило, не актуальна, так как известно, что они являются алгоритмически разрешимыми.

При решении задачи на компьютере исходные данные и результат решения всегда представлены с некоторой (конечной) точностью, так как компьютер работает с машинным словом, имеющим конечную длину. Таким образом, мы имеем конечное число машинных слов, среди которых находится и слово, представляющее результат решения задачи. В таких случаях всегда можно решить задачу, используя так называемый *тривиальный алгоритм* — полный перебор возможных значений машинного слова. Однако подобный подход к решению задач практически неприемлем уже для относительно небольших размеров машинного слова. Время решения реальных задач может оказаться настолько большим, что они практически неразрешимы.

При разработке алгоритмов можно наблюдать, что для некоторых задач удается построить алгоритмы, удовлетворяющие практическим потребностям, и успешно решить их на компьютере. Для других задач не удается разработать алгоритм более эффективный, чем алгоритм полного перебора. Все такие задачи называют *трудноразрешимыми*.

Можно привести многочисленные примеры практически важных задач, которые являются трудноразрешимыми. Это *задача коммивояжера*, в которой требуется найти гамильтонов цикл в реберно-взвешенном графе такой, что суммарная стоимость циклических ребер является наименьшей. Это задача поиска дизъюнктивной нормальной формы (ДНФ) булевой функции, в записи которой используется наименьшее число литералов (переменных и их отрицаний), известная как *задача построения минимальной ДНФ* булевой функции, и многие другие.

Возникает вопрос: является ли неудача в разработке алгоритмов для решения некоторых задач следствием неумения конкретного разработчика или следствием каких-то свойств задачи?

Ответ на поставленный вопрос пытается дать теория вычислительной сложности.

Чтобы ответить на этот вопрос, необходимо каким-то образом классифицировать существующие задачи.

Прежде всего определим форму, в которой должна быть представлена задача. В теории вычислительной сложности принято рассматривать задачи, сформулированные в *распознавательной форме*. При представлении задачи в распознавательной форме формулируется вопрос, на который может быть один из двух ответов: «да» или «нет».

Имеются задачи, которые изначально имеют распознавательную форму. Например: является ли число Z простым? Являются ли два графа G_1 и G_2 изоморфными? Другой пример — задача Выполнимость (Satisfiability — SAT), которая исторически является первой распознавательной задачей, сформулированной американским математиком С. Куком. В этой задаче представлена некоторая булева функция f в конъюнктивной нормальной форме и задается вопрос: существует ли набор значений переменных функции f, на котором она принимает значение 1 (истина)? Еще пример: существует ли набор из n ребер данного графа, который образует гамильтонов цикл, то есть цикл, при обходе ребер которого каждая вершина графа посещается только один раз?

Задачи, которые в исходной постановке представлены в иной, не распознавательной форме, приводят к такой форме. Например, если требуется найти наибольшее независимое множество вершин графа, то формулируют вопрос: существует ли для данного графа $G = (V, E)$ и положительного целого числа $K \le Card(V)$ независимое множество мощности не менее K?

7.10. Класс NP

Определим класс задач, рассматриваемых в теории вычислительной сложности.

Ранее уже отмечалось, что существующие алгоритмы делятся на два класса: полиномиальные и экспоненциальные. Алгоритмы, время работы которых оценивается некоторым полиномом от размерности задачи, принято считать «хорошими». В то же время алгоритмы с экспоненциальным временем работы считаются «плохими».

Множество всех задач, для которых существует полиномиальный решающий алгоритм, обозначим через P.

Выясним, как можно охарактеризовать задачи, для которых в настоящее время не известен полиномиальный разрешающий алгоритм.

Ясно, что время работы алгоритма для нахождения, например, всех подмножеств данного n-множества, всех покрывающих деревьев данного n-графа или всех его максимальных независимых множеств всегда есть экспоненциальная функция от размерности задачи n (числа вершин графа), то есть такой алгоритм изначально предопределен быть «плохим». Можно полагать, что в этом случае сама постановка задачи влечет экспоненциальность времени работы алгоритма. Такие задачи называют *экспоненциальными по постановке*. Очевидно, что такие задачи неинтересны для нас, поскольку очевидно, что получить хороший (полиномиально-временной) алгоритм для них невозможно. Поэтому такие задачи целесообразно исключить из дальнейшего рассмотрения. Исключим также из рассмотрения задачи, для которых не существует никакого алгоритма решения (например, задачу о проблеме остановки — см. раздел 7.6).

Проблема остановки машины Тьюринга

В теории сложности выделяют такие задачи, в которых время построения решения есть полиномиальная функция от размерности за-

дачи. Кроме того, вводится требование, чтобы за полиномиальное время можно было проверить правильность полученного решения.

Для характеризации задач, решаемых на компьютере, вводят понятие *недетерминированной* машины Тьюринга. Рассмотренную ранее в разделе 7.4 машину Тьюринга называют *детерминированной* (*ДМТ*).

Недетерминированная машина Тьюринга (НМТ) содержит все те же устройства, что и ДМТ. НМТ также содержит дополнительное устройство (О), которое иногда называют *оракулом* (*прорицателем*) — см. рис. 7.4. Оракул с помощью универсальной головки У2 читает исходные данные задачи, записанные на ленте, стирает их, «угадывает» решение и записывает это решение на ленту. Затем машина работает как обычная (детерминированная) машина Тьюринга — это процесс *верификации (проверки) решения*. Универсальная головка УГ читает решение, записанное на ленте, и проверяет его. Если решение правильное, то на ленте записывается единица (ответ «да»), если нет — то нуль.

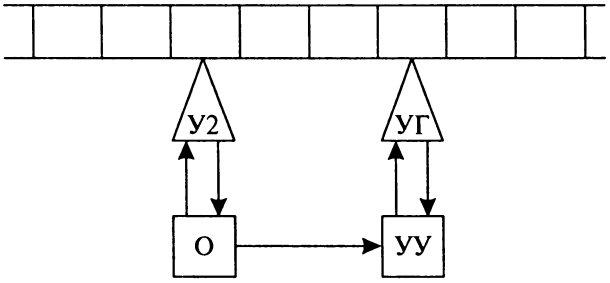

Рис. 7.4. Недетерминированная машина Тьюринга

Множество всех задач, разрешимых на недетерминированной машине Тьюринга за полиномиальное время, образует *класс NP*.

Существует другая интерпретация работы недетерминированной машины Тьюринга. Согласно этой интерпретации, основное отличие работы НМТ от работы детерминированной машины Тьюринга состоит только в возможности одновременной проверки правильности всех альтернатив. На каждом этапе вычислений НМТ строит множество промежуточных решений, и дальнейшая работа НМТ производится только с теми ранее полученными промежуточными решениями, которые удовлетворяют условиям задачи.

Однако для практического применения удобно использовать альтернативное определение класса NP. Будем говорить, что задача Z принадлежит классу NP, если:

1) задача может быть задана конечным числом символов N;

2) решение задачи также может быть представлено конечным числом M символов, где M есть полиномиальная функция от N : $M = f(N)$;

3) время проверки полученного решения t есть некоторая полиномиальная функция от N : $t = t(N)$.

Большинство практически важных задач принадлежат к классу NP. Например, вышеупомянутая задача поиска гамильтонова цикла принадлежит классу NP, так как исходная задача задается конечным графом, например матрицей смежности этого графа. Такая матрица содержит $n \times n$ элементов, где n — число вершин графа. Решение задачи — совокупность из n ребер графа. За полиномиальное время можно убедиться, что предъявленная совокупность из n ребер действительно образует гамильтонов цикл.

Можно заключить, что задачи класса P принадлежат классу NP, то есть имеет место соотношение $P \subset NP$.

Конечно, желательно всегда иметь только «хорошие» алгоритмы. Однако для многих практически важных задач, таких как задача нахождения наибольшего независимого множества вершин графа, задача поиска гамильтонова цикла и многих других, пока не удается разработать полиномиальный решающий алгоритм.

Поэтому возникает вопрос: класс P совпадает ли с классом NP или является его строгим подмножеством? Этот вопрос поставлен уже более тридцати лет назад и до сих пор не нашел своего решения. Ответ на эту проблему имеет важное практическое значение.

Интересно, что математический институт Клэя (США) (Clay Mathematics Institute) объявил приз в размере одного миллиона долларов за решение этой проблемы (см. http://www.claymath.org/prize_problems/).

7.11. NP-полные задачи

Для некоторых задач класса NP было обнаружено удивительное свойство. Оказалось, что некоторые из них универсальны в том смысле, что построение полиномиального алгоритма для любой такой задачи влечет за собой возможность построения такого же алгоритма для всех задач класса NP. Такие задачи называют *NP-полными*.

Чтобы понять суть этого свойства, необходимо определить некоторые понятия.

Прежде всего заметим, что всякий алгоритм способен решать не одну конкретную задачу, а определенный класс задач. Например, алгоритм Прима способен найти наименьшее по весу покрывающее дерево (каркас) для любого конечного графа. Такое множество конкретных, или индивидуальных, задач (instances) называют массовой задачей (decision problem).

Пусть имеются две массовые задачи Z_1 и Z_2 из NP. Будем говорить, что массовая задача Z_1 *полиномиально сводится* к массовой задаче Z_2, если:

1) для любой индивидуальной задачи из Z_1 можно за полиномиальное время построить соответствующую ей индивидуальную задачу из Z_2;

2) решение построенной индивидуальной задачи из Z_2 за полиномиальное время может быть преобразовано в решение соответствующей индивидуальной задачи из Z_1.

Заметим, что когда говорится о полиномиальном времени построения индивидуальной задачи из Z_2 или о преобразовании ее решения в решение индивидуальной задачи из Z_1 также за полиномиальное время, то подразумевается возможность построения соответствующего полиномиального алгоритма для такого преобразования.

Процесс сведения одной задачи класса NP в другую задачу этого класса не всегда простой и может потребовать известной изобретательности. В следующем разделе рассмотрим пример сведения задачи поиска минимальной ДНФ к задаче разбиения графа на минимальное число клик.

Теперь определим понятие NP-полной задачи.

Массовую задачу Z называют *NP-полной*, если любая задача из этого класса полиномиально сводится к решению задачи Z. Таким образом, теперь ясно, что разработка полиномиального алгоритма для решения любой NP-полной задачи практически означает возможность построения такого алгоритма для любой задачи класса NP.

Исторически первой NP-полной задачей является задача Выполнимость (Satisfiability, или сокращенно SAT). Она формулируется следующим образом.

Литералом назовем любую из булевых переменных $x_1, x_2, \ldots, x_n$, взятую с отрицанием или без отрицания.

Дизъюнкцию различных литералов называют *дизъюнктом (clause)*.

Конъюнкция дизъюнктов определяет некоторую булеву функцию. Иначе такое представление булевой функции называют ее *конъюнктивной нормальной формой*. Требуется найти такие значения булевых переменных, для которых построенное выражение истинно (если оно существует).

В настоящее время найдены сотни NP-полных задач. Имеются списки таких задач (см., например,[7.3]).

Из изложенного выше ясно, что если необходимо разработать решающий алгоритм для некоторой задачи и эта задача является NP-полной, то результат такой разработки, скорее всего, будет отрицательным. Если предложенной задачи нет в известных списках NP-полных задач, то, чтобы доказать NP-полноту новой задачи, необходимо установить, что такая задача принадлежит классу NP, и свести решение любой известной NP-полной задачи к решению предложенной задачи.

В настоящее время для решения NP-полных задач используют в основном различного рода эвристические алгоритмы, подобные алгоритму MIN. С другой стороны, если разработка решающего алгоритма для задачи из класса P может рассматриваться как рутинная работа, то поиск решающего алгоритма для решения NP-полной задачи или ее подкласса требует высокой математической подготовки.

7.12. Пример полиномиального сведения

7.12.1. Постановка задачи

Как указывалось в главе, когда говорят о минимизации булевых функций в классе ДНФ, то подразумевают одну из двух задач: задачу построения *кратчайшей ДНФ* и собственно *минимальную ДНФ*. Кратчайшая ДНФ имеет наименьшее число дизъюнктивных членов (простых импликант) по сравнению с другими ДНФ данной функции, а минимальная — наименьшее суммарное число литералов (то есть переменных функции с отрицанием или без), используемых при записи ДНФ.

Пусть имеется булева функция $f(x_1, \ldots, x_m)$, заданная множеством V всех ее единичных наборов, то есть таких значений переменных, на которых функция f принимает единичные (истинностные) значения.

Напомним, что любую конъюнкцию k ($k \le m$) переменных функции f

$$U = x_{i_1}^{\sigma_{i_1}} \wedge x_{i_2}^{\sigma_{i_2}} \wedge \cdots \wedge x_{i_k}^{\sigma_{i_k}},$$

взятых с отрицанием или без отрицания, называют *элементарной*. Здесь $x_i^{\sigma_i} = x_i$, если $\sigma_i = 1$, и $x_i^{\sigma_i} = \bar{x}_i$, если $\sigma_i = 0$. Число $k = r(U)$ называют *рангом* конъюнкции U.

Переменные функции f, входящие в конъюнкцию U, назовем *связанными* в ней, а остальные переменные — *свободными* для конъюнкции U. Множество связанных переменных конъюнкции U обозначим $S(U)$.

Говорят, что конъюнкция U *покрывает* множество $V_1 \subset V$ единиц функции f, если на всяком наборе из V_1 значение элементарной конъюнкции U равно единице, а на всех остальных наборах переменных функции f — равно нулю. Такую конъюнкцию называют также *импликантой* функции f.

Импликанту U называют *простой*, если никакая ее собственная часть импликантой функции f не является. Другими словами, импликанта U является простой, если функция f не имеет другой импликанты U^* такой, что $S(U^*) \subset S(U)$, или $V \setminus S(U) \subset V \setminus S(U^*)$.

Легко заметить, что множество V_1 наборов, покрываемое импликантой U, имеет следующие свойства:

- $Card(V_1) = 2^t$, где $t = Card(V \setminus S(U))$;
- t свободных переменных функции f принимают всевозможные значения на множестве наборов V_1;
- значения $(m - t) = k$ связанных переменных функции f принимают фиксированные значения (на которых конъюнкция U равна единице).

Лемма 7.12.1. *Если U_1, U_2 есть простые импликанты булевой функции f, покрывающие соответственно подмножества $V_1 \subset V$ и $V_2 \subset V$ единиц этой функции, то найдутся хотя бы два единичных набора v_1, $v_2 \in V$ такие, что $v_1 \in V_1$, $v_1 \in V_2$ и $v_2 \in V_1$, $v_2 \in V_2$ и которые отличаются друг от друга значениями по крайней мере двух переменных функции f.*

Очевидно. Q.E.D.

Лемма 7.12.2. *Если v_1 и v_2 — наборы, покрываемые раздельно простыми импликантами U_1 и U_2 соответственно, то не существует простой импликанты U такой, которая одновременно покрывала бы эти наборы.*

Пусть выполняются условия леммы 7.12.2.

Предположим, что существует простая импликанта U такая, которая одновременно покрывает наборы v_1 и v_2, раздельно покрываемые простыми импликантами U_1 и U_2.

Согласно лемме 7.12.1, наборы v_1 и v_2 отличаются друг от друга значениями по крайней мере двух переменных функции f. Отсюда следует, что упомянутые переменные являются свободными для простой импликанты U.

Значения связанных переменных импликанты U_3, очевидно, совпадают со значениями таких же переменных как импликанты U_1, так и импликанты U_2. Иначе U_1 и U_3, а также U_2 и U_3 не покрывали бы (каждая пара) общего набора. Но тогда существует такая импликанта U, для которой свободными будут переменные из множеств $S(U_1)$ и $S(U_2)$, то есть $V \setminus S(U_1) \subset V \setminus S(U)$ и $V \setminus S(U_2) \subset V \setminus S(U)$, что противоречит условию леммы 7.12.2, согласно которой импликанты U_1 и U_2 — простые. Q.E.D.

Ясно, что всякую булеву функцию f можно представить в виде дизъюнкции каких-то элементарных конъюнкций:

$$D(f) = U_1 \vee U_2 \vee \ldots \vee U_p.$$

Такую форму представления функции f называют *дизъюнктивной нормальной формой (ДНФ)*. Число p называют *длиной* ДНФ $D(f)$. Сумму рангов всех составляющих $D(f)$ конъюнкций U_i ($i = \overline{1,p}$), то есть число

$$\sum_{i=1}^{p} r(U_i),$$

называют *сложностью* этой ДНФ.

ДНФ $D(f)$ булевой функции f называют *кратчайшей*, если она имеет наименьшую длину по сравнению с другими ДНФ этой же функции.

ДНФ $D(f)$ называют *минимальной*, если она имеет наименьшую сложность по сравнению с другими ДНФ этой же функции.

Итак, пусть требуется найти кратчайшую ДНФ функции $f(x_1, \ldots, x_m)$.

34 Трудноразрешимые проблемы теории графов {???}

- **Гипотеза Каццетты — Хаггвиста** — ориентированный граф, имеющий n вершин, из каждой вершины которого выходит не менее m рёбер, имеет замкнутый контур длиной не более $\left\lceil \frac{n}{m} \right\rceil$.[58]
- **Гипотеза Хадвигера** — каждый n-хроматический граф стягиваем к полному графу K_n[59].
- **Гипотеза Улама**:[60]
  - а) всякий граф с более чем двумя вершинами однозначно определяется *набором* графов, где каждый граф из набора получен удалением одной из вершин исходного графа;
  - б) всякий граф с более чем *тремя* вершинами однозначно определяется *множеством* графов, где каждый граф из множества получен удалением одной из вершин исходного графа.
- **Гипотеза Харари** (слабая форма гипотезы Улама) — если граф имеет более трёх рёбер, то его можно однозначно восстановить по подграфам, полученным удалением единственного ребра[60].
- В любом графе, не содержащем мостов (ребер, удаление которых увеличивает число компонент связности графа), можно выбрать множество простых циклов, такое, что каждое ребро принадлежит ровно двум из них.

- В любом кубическом графе можно выбрать 6 1-факторов так, чтобы каждое ребро принадлежало ровно двум из них.
- **Гипотеза Рамачандрана** Любой орграф N-реконструируем.[61]
- **Гипотеза Бержа** Граф G является совершенным тогда и только тогда, когда ни он, ни его дополнение $\overline{G}$ не содержат порождённых подграфов вида $C_{2k+1,k} \geqslant 2$.[62]
- **Гипотеза о восстановлении** Если заданы классы изоморфизма всех k примарных подграфов некоторого графа, то при $k \geqslant 3$ класс изоморфизма этого графа определяется однозначно.[63]

- раскраска графов
- задача независимого множества
- задача вершинного покрытия
- задача коммивояджера