Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет геосистем и технологий»
(СГУГиТ)

В. Л. Неклюдова, В. П. Вербная

МАТЕМАТИЧЕСКАЯ ЛОГИКА И ТЕОРИЯ АЛГОРИТМОВ

Утверждено редакционно-издательским советом университета
в качестве учебного пособия для обучающихся
по направлению подготовки 10.03.01 Информационная безопасность
(уровень бакалавриата)

Новосибирск
СГУГиТ
2022

УДК 51
Н476

Рецензенты: кандидат физико-математических наук, доцент, СКУ имени М. Козыбаева *А. А. Таджигитов*

кандидат физико-математических наук, доцент, СГУГиТ *О. В. Григоренко*

Неклюдова, В. Л.

Н476 Математическая логика и теория алгоритмов : учебное пособие / В. Л. Неклюдова, В. П. Вербная. – Новосибирск : СГУГиТ, 2022. – 70 с. – Текст : непосредственный.

ISBN 978-5-907513-37-2

Учебное пособие подготовлено сотрудниками кафедры высшей математики СГУГиТ: кандидатом физико-математических наук, доцентом В. Л. Неклюдовой и старшим преподавателем В. П. Вербной.

Настоящее пособие состоит из разделов базового курса дисциплины: логика высказываний, логика предикатов, конечные автоматы, теория алгоритмов. Изложение теоретического материала сопровождается примерами, раскрывающими смысл основных понятий.

Учебное пособие по дисциплине «Математическая логика и теория алгоритмов» предназначено для обучающихся по направлению подготовки 10.03.01 Информационная безопасность (уровень бакалавриата).

Рекомендовано к изданию кафедрой высшей математики, Ученым советом Института геодезии и менеджмента СГУГиТ.

Печатается по решению редакционно-издательского совета СГУГиТ

УДК 51

ISBN 978-5-907513-37-2

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В соответствии с требованиями Федерального государственного образовательного стандарта бакалавр по направлению подготовки «Информационная безопасность» готовится к профессиональной деятельности, связанной с решениями задач обеспечения защищенности объектов информатизации в условиях существования угроз в информационной сфере, проектированием и оценкой качества систем защиты информации, что предполагает знание основных понятий математической логики и теории алгоритмов и умение конструировать и применять информационно-логические алгоритмы и методы. Математическая логика и теория алгоритмов имеет широкий спектр приложений в областях, связанных с организацией и технологиями защиты информации.

Структура изложения и содержание материала обеспечивают формирование у обучающихся общепрофессиональной компетенции, выражающейся способностью применять соответствующий математический аппарат для решения профессиональных задач.

1. ЛОГИКА ВЫСКАЗЫВАНИЙ

1.1. Формулы логики высказываний

Высказыванием называется повествовательное предложение, о котором можно сказать, что оно истинно или ложно. Например, высказывание «Солнце заходит на западе» истинно, а «Три больше пяти» ложно. Высказывание «Сегодня пятница» может быть как истинным, так и ложным.

Каждому высказыванию соответствует *логическая переменная*, или *булева переменная* x, которая принимает значение 1, если высказывание истинно, и 0, если оно ложно.

В дальнейшем будут рассматриваться только логические переменные, вне зависимости от смысла высказываний.

Если имеется несколько высказываний, то из них можно образовать новые высказывания с помощью следующих *логических операций*, или *логических связок*:

– отрицание $\neg x$ или $\bar{x}$ – «не x»;

– конъюнкция $x \wedge y$ – «x и y»;

– дизъюнкция $x \vee y$ – «x или y»;

– импликация $x \to y$ – «если x, то y»;

– эквивалентность $x \leftrightarrow y$ – «x тогда и только тогда, когда y».

– штрих Шефера, или антиконъюнкция $x \mid y = \overline{x \wedge y}$;

– стрелка Пирса, или антидизъюнкция $x \downarrow y = \overline{x \vee y}$;

– сумма Жегалкина, или антиэквивалентность (сложение по модулю 2, исключающее «или») $x \oplus y = \overline{x \leftrightarrow y}$.

Запись высказывания с помощью логических переменных и операций представляет собой *формулу логики высказываний*.

Логические операции могут быть заданы *таблицами истинности*, которые каждому набору истинных или ложных значений логических переменных сопоставляют истинное или ложное значение формулы.

Таблицы истинности для приведенных выше операций выглядят следующим образом:

x	$\bar{x}$
0	1
1	0

x	y	$x \wedge y$	$x \vee y$	$x \to y$	$x \leftrightarrow y$	$x \mid y$	$x \downarrow y$	$x \oplus y$
0	0	0	0	1	1	1	1	0
0	1	0	1	1	0	1	0	1
1	0	0	1	0	0	1	0	1
1	1	1	1	1	1	0	0	0

Используя таблицы истинности для логических связок, можно строить таблицы истинности для произвольных формул.

Пример 1.1. Построить таблицу истинности для формулы

$$\phi = \left(\bar{x} \wedge y\right) \leftrightarrow \bar{z}\,.$$

Строим таблицу сначала для промежуточных формул, а затем для основной:

x	y	z	$\bar{x}$	$\bar{x} \wedge y$	$\bar{z}$	$\phi = \left(\bar{x} \wedge y\right) \leftrightarrow \bar{z}$
0	0	0	1	0	1	0
0	0	1	1	0	0	1
0	1	0	1	1	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	1	0	0	0	1
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Поскольку значения переменных в таблице истинности всегда записываются в фиксированном порядке, то можно считать, что формула задана *вектором значений*, $\phi = (0,1,1,0,0,1,0,1)$.

При составлении формул возникает большое количество скобок. Чтобы избежать этого, некоторые скобки можно опускать в соответствии с принятым порядком действий [1, 4, 6].

В формуле без скобок в первую очередь выполняется отрицание (в вышеприведенных формулах отрицание скобками не выделялось), затем конъюнкция, штрих Шефера и стрелка Пирса (эти действия являются равноправ-

ными в отношении порядка действий), далее – дизъюнкция, потом – импликация, и, наконец, действия эквивалентности и суммы Жегалкина. Схематически порядок действий можно представить следующим образом:

$$(1)\ \bar{}\ ;\ \ (2)\ \wedge, |, \downarrow ;\ \ (3)\ \vee ;\ \ (4)\ \to ;\ \ (5)\ \leftrightarrow, \oplus .$$

Пример 1.2. В формуле $(x \vee y) \leftrightarrow (z \to y)$ скобки можно опустить и записать ее как $x \vee y \leftrightarrow z \to y$. В формуле $x \vee (y \leftrightarrow z) \to y$ скобки опускать нельзя.

Если в формуле без скобок содержится несколько равноправных действий подряд, то они выполняются в том порядке, в котором записаны. Например, формулу $a | b \downarrow c \wedge d$ следует читать как $((a | b) \downarrow c) \wedge d$.

Знак конъюнкции при записи формул часто опускается (так же, как знак умножения в алгебре). Например, формула $x \wedge y \wedge \bar{z}$ может быть записана как $xy\bar{z}$.

Формулы называются *эквивалентными*, или *равносильными*, если они имеют одинаковые таблицы истинности. Если формулы ϕ и ψ эквивалентны, пишут $\phi \sim \psi$. Допускается также запись $\phi = \psi$.

Пример 1.3. Эквивалентность формул $\bar{x} \to \bar{y}$ и $x \vee \bar{y}$ следует из идентичности их таблиц истинности:

x	y	$\bar{x}$	$\bar{y}$	$\bar{x} \to \bar{y}$	$x \vee \bar{y}$
0	0	1	1	1	1
0	1	1	0	0	0
1	0	0	1	1	1
1	1	0	0	1	1

Отношение $\sim$ является отношением эквивалентности [2, 6], поскольку оно рефлексивно ($\phi \sim \phi$), симметрично (если $\phi \sim \psi$, то $\psi \sim \phi$) и транзитивно (если $\phi \sim \psi$ и $\psi \sim \chi$, то $\phi \sim \chi$).

Формула называется *тождественно истинной* или *тавтологией*, если при любых наборах значений переменных она принимает значение 1,

и *тождественно ложной*, если для любого набора значений переменных ее значение равно 0.

Формула называется *выполнимой*, если хотя бы на одном наборе значений переменных она принимает значение 1, и *опровержимой*, если хотя бы на одном наборе значений переменных ее значение равно 0.

Пример 1.4. Формула $a \to (a \vee b)$ является тождественно истинной, а формула $a \downarrow (\bar{a} \vee b)$ – тождественно ложной:

a	b	$a \vee b$	$a \to (a \vee b)$	$\bar{a}$	$\bar{a} \vee b$	$a \downarrow (\bar{a} \vee b)$
0	0	0	1	1	1	0
0	1	1	1	1	1	0
1	0	1	1	0	0	0
1	1	1	1	0	1	0

Тождественно истинные и тождественно ложные формулы обозначают символами 1 и 0 и называют *логическими константами*. Логические константы могут встречаться в записи формул наряду с логическими переменными. Очевидно, что если $\phi \sim \psi$, то формула $\phi \leftrightarrow \psi$ является тавтологией.

Свойства логических операций (законы логики, основные равносильности)

Для любых высказываний x, y и z:

1) $x \wedge x = x$; $x \vee x = x$ – *идемпотентность* конъюнкции и дизъюнкции;

2) $\overline{(\bar{x})} = x$ – *закон двойного отрицания*;

3) $x \vee \bar{x} = 1$ – *закон исключенного третьего*;

4) $x \wedge \bar{x} = 0$ – *закон противоречия*;

5) $x \wedge y = y \wedge x$; $x \vee y = y \vee x$ – *коммутативность* конъюнкции и дизъюнкции;

6) $x \wedge (y \wedge z) = (x \wedge y) \wedge z = x \wedge y \wedge z$; $x \vee (y \vee z) = (x \vee y) \vee z = x \vee y \vee z$ – *ассоциативность* конъюнкции и дизъюнкции;

7) $x \wedge (y \vee z) = xy \vee xz$; $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$ – *дистрибутивность*;

8) $\overline{x \vee y} = \bar{x} \wedge \bar{y}$; $\overline{x \wedge y} = \bar{x} \vee \bar{y}$ – *законы де Моргана*;

9) $x \to y = \bar{x} \vee y$ – раскрытие импликации;

10) $x \sim y = (x \to y) \wedge (y \to x)$ – раскрытие эквивалентности;

11) $x \wedge 1 = x$; $x \wedge 0 = 0$; $x \vee 1 = 1$; $x \vee 0 = x$ – действия с константами;

12) $xy \vee x\bar{y} = x$; $(x \vee y) \wedge (x \vee \bar{y}) = x$ – *склеивание*.

Для доказательства необходимо применить сначала свойство 7, затем закон 3, и, наконец, 11: $xy \vee x\bar{y} = x(y \vee \bar{y}) = x \wedge 1 = x$. Второе равенство доказывается аналогично;

13) $x \wedge (x \vee y) = x$, $x \vee (x \wedge y) = x$ – *поглощение*.

Для доказательства применяется сначала свойство 11, затем 7, и снова 11: $x \wedge (x \vee y) = (x \vee 0) \wedge (x \vee y) = x \vee (0 \wedge y) = x \vee 0 = x$. Второе равенство доказывается аналогично.

1.2. Конъюнктивные и дизъюнктивные нормальные формы

Пусть x – логическая переменная. Далее используется обозначение

$$x^{\delta} = \begin{cases} x, & \text{если } \delta = 1, \\ \bar{x}, & \text{если } \delta = 0. \end{cases}$$

Выражение x^{δ} называется *литерой*. Формула вида $x_1^{\delta_1} \vee x_2^{\delta_2} \vee \ldots \vee x_n^{\delta_n}$ называется *элементарной дизъюнкцией* (*дизъюнктом*), а $x_1^{\delta_1} \wedge x_2^{\delta_2} \wedge \ldots \wedge x_n^{\delta_n}$ – *элементарной конъюнкцией* (*конъюнктом*).

Конъюнкция элементарных дизъюнкций называется *конъюнктивной нормальной формой* (КНФ). Дизъюнкция элементарных конъюнкций называется *дизъюнктивной нормальной формой* (ДНФ).

Пример 1.5. Формулы $x \vee \bar{y} \vee \bar{z}$ и $x \vee y \vee \bar{y} \vee x$ являются элементарными дизъюнкциями, формулы $\bar{x} \wedge y \wedge \bar{z}$ и $x \wedge y \wedge \bar{x}$ – элементарными

конъюнкциями, а $\bar{x}$ – и элементарной дизъюнкцией, и элементарной конъюнкцией одновременно.

Формула $\left(x \wedge \bar{y}\right) \vee \left(y \wedge z\right)$ является ДНФ, $\left(x \vee z \vee \bar{y}\right) \wedge \left(\bar{x} \vee z\right) \wedge y$ – КНФ, а $x \wedge \bar{y}$ является как ДНФ, так и КНФ.

Для любой формулы можно подобрать эквивалентные ей КНФ или ДНФ – привести ее к КНФ / ДНФ. Это позволяет свести все логические операции к конъюнкции, дизъюнкции и отрицанию. Одна и та же формула может иметь несколько представлений в виде КНФ и ДНФ. Особое место среди этих представлений занимают совершенные конъюнктивная нормальная форма и дизъюнктивная нормальная форма (СКНФ и СДНФ).

Пусть $x_1, x_2, \ldots x_k$ – набор всех логических переменных, входящих в формулу. *Совершенной КНФ* называется КНФ, в которой все элементарные дизъюнкции различны и имеют вид $x_1^{\delta_1} \vee x_2^{\delta_2} \vee \ldots \vee x_k^{\delta_k}$ (для каждого значения $i = 1, \ldots, k$ содержат либо переменную x_i, либо ее отрицание $\overline{x_i}$, причем ровно один раз). Аналогично *СДНФ* – это ДНФ, в которой все элементарные конъюнкции различны и имеют вид $x_1^{\delta_1} \wedge x_2^{\delta_2} \wedge \ldots \wedge x_k^{\delta_k}$.

Построение СКНФ и СКНФ по таблице истинности формулы

СДНФ и СКНФ можно построить, если известна таблица истинности формулы, руководствуясь следующими правилами:

1) СДНФ формулы содержит столько элементарных конъюнкций, сколько единиц в ее таблице истинности. Каждому набору значений $x_1 = \delta_1$, $x_2 = \delta_2$, …, $x_k = \delta_k$, для которого формула истинна, соответствует элементарная конъюнкция $x_1^{\delta_1} \wedge x_2^{\delta_2} \wedge \ldots \wedge x_k^{\delta_k}$.

2) СКНФ формулы содержит столько элементарных дизъюнкций, сколько нулей в ее таблице истинности. Каждому набору значений $x_1 = \delta_1$, $x_2 = \delta_2$, …, $x_k = \delta_k$, для которого формула ложна, соответствует элементарная конъюнкция $x_1^{1-\delta_1} \vee x_2^{1-\delta_2} \vee \ldots \vee x_k^{1-\delta_k}$.

Пример 1.6. Построить СДНФ и СКНФ формулы ϕ, заданной таблицей:

x	0	0	0	0	1	1	1	1
y	0	0	1	1	0	0	1	1
z	0	1	0	1	0	1	0	1
ϕ	0	0	1	1	0	1	1	1

Так как $\phi(0,1,0)=\phi(0,1,1)=\phi(1,0,1)=\phi(1,1,0)=\phi(1,1,1)=1$, то в СДНФ набору переменных $(0,1,0)$ соответствует конъюнкт $\bar{x}\wedge y\wedge\bar{z}$ (фактически необходимо расставить отрицания так, чтобы каждая литера и конъюнкт в целом стали истинными); набору $(0,1,1)$ – конъюнкт $\bar{x}\wedge y\wedge z$ и т. д. В итоге СДНФ формулы ϕ имеет вид $\phi_1=\bar{x}y\bar{z}\vee\bar{x}yz\vee x\bar{y}z\vee xy\bar{z}\vee xyz$.

Соответственно $\phi(0,0,0)=\phi(0,0,1)=\phi(1,0,0)=0$ и СКНФ формулы имеет вид $\phi_2=(x\vee y\vee z)\wedge(x\vee y\vee\bar{z})\wedge(\bar{x}\vee y\vee z)$. Здесь отрицания расставляются так, чтобы каждая литера в дизъюнкте и, соответственно, сам дизъюнкт стали ложными.

Очевидно, СДНФ можно построить для любой формулы, кроме тождественно ложной, СКНФ – для любой, кроме тождественно истинной.

Как видно из примера 1.6, даже для трех переменных СКНФ и СДНФ выглядят громоздкими и неудобными для дальнейшего применения, что является существенным недостатком представления формулы в виде СКНФ и СДНФ.

Далее рассматриваются более компактные способы представления формул в виде КНФ и ДНФ.

1.3. Минимизация дизъюнктивной и конъюнктивной нормальных форм

С помощью тождественных преобразований СДНФ и СКНФ могут быть упрощены и приведены к сокращенной форме. Для этого применяются операции склеивания и поглощения (свойства 12 и 13, подраздел 1.1).

Метод Квайна получения сокращенной ДНФ

В СДНФ произвести все возможные склеивания ($xy \vee x\bar{y} = x$), а затем поглощения ($x \vee xy = x$). Один и тот же конъюнкт может многократно участвовать в этих операциях в силу идемпотентности дизъюнкции.

Пример 1.6 (продолжение). Получим сокращенную ДНФ для формулы ϕ из примера 1.6.

Ранее была получена СДНФ формулы: $\phi_1 = \underset{(1)}{\bar{x}\bar{y}\bar{z}} \vee \underset{(2)}{\bar{x}yz} \vee \underset{(3)}{x\bar{y}z} \vee \underset{(4)}{xy\bar{z}} \vee \underset{(5)}{xyz}$.

Конъюнкт (1) может быть склеен с конъюнктами (2) и (4)

$$\underset{(1)}{\bar{x}\bar{y}\bar{z}} \vee \underset{(2)}{\bar{x}yz} = \bar{x}\bar{y} \quad \text{и} \quad \underset{(1)}{\bar{x}\bar{y}\bar{z}} \vee \underset{(4)}{xy\bar{z}} = \bar{y}\bar{z}.$$

Конъюнкт (2) может быть склеен с (5), (3) с (5) и (4) с (5)

$$\underset{(2)}{\bar{x}yz} \vee \underset{(5)}{xyz} = yz; \quad \underset{(3)}{x\bar{y}z} \vee \underset{(5)}{xyz} = xz; \quad \underset{(4)}{xy\bar{z}} \vee \underset{(5)}{xyz} = xy.$$

Следовательно, формула может быть преобразована следующим образом:

$$\phi_1 = \underline{\bar{x}\bar{y}\bar{z}} \vee \underline{\bar{x}yz} \vee \underline{\underline{x\bar{y}z}} \vee \overline{\overline{xyz}} \vee \underline{\underline{xyz}} = \bar{x}\bar{y} \vee y\bar{z} \vee yz \vee xz \vee xy.$$

Конъюнкты $\bar{x}\bar{y}\bar{z}$ и $\bar{x}yz$ поглощаются членом $\bar{x}\bar{y}$ и в новую формулу уже не входят. То же можно сказать и о конъюнктах 3–5. В полученной ДНФ продолжаем производить склеивания

$$\phi_1 = \underline{\bar{x}\bar{y}} \vee \underline{\underline{y\bar{z}}} \vee \underline{\underline{yz}} \vee xz \vee \underline{xy} = y \vee y \vee xz = y \vee xz.$$

Конъюнкт xz не участвует в склеиваниях и сохраняется в формуле до конца; из двух одинаковых конъюнктов y остается один в силу идемпотентности дизъюнкции (см. первое свойство логических операций в подразделе 1.1). В итоге формула ϕ эквивалентна формуле $y \vee xz$.

Аналогично выглядит метод Квайна для получения сокращенной КНФ. В СКНФ производятся все возможные склеивания $(x \vee y) \wedge (x \vee \bar{y}) = x$, а затем поглощения $x \wedge (x \vee y) = x$. Один и тот же дизъюнкт может мно-

гократно участвовать в этих операциях в силу идемпотентности конъюнкции.

Пример 1.6 (продолжение). Получим сокращенную КНФ для формулы ϕ. В полученной ранее СКНФ произведем склеивания

$$\phi_2 = \underline{\overbrace{(x \vee y \vee z)}} \wedge \underline{\left(x \vee y \vee \bar{z}\right)} \wedge \overbrace{\left(\bar{x} \vee y \vee z\right)} = (x \vee y) \wedge (y \vee z).$$

Больше в формуле нельзя произвести ни склеиваний, ни поглощений. Таким образом, получена сокращенная КНФ.

ДНФ и КНФ с минимальным числом вхождений переменных называются *минимальными* ДНФ и КНФ (МДНФ и МКНФ). Иногда сокращенная форма, полученная с помощью склеиваний и поглощений, не является минимальной, поскольку некоторые конъюнкты (дизъюнкты) в ней являются лишними, т. е. их удаление не меняет таблицы истинности формулы. Для исключения лишних членов из сокращенной ДНФ строится *матрица Квайна*, строки которой соответствуют конъюнктам сокращенной ДНФ, а столбцы – конъюнктам СДНФ.

В таблице крестиком отмечаются пересечения строк и столбцов, в которых конъюнкт, соответствующий строке, поглощает конъюнкт, стоящий в заглавии столбца. В матрице необходимо выбрать минимальное число строк таким образом, чтобы каждый столбец содержал хотя бы один крестик на пересечении с выбранными строками. Конъюнкты, соответствующие выбранным строкам, входят в минимальную ДНФ.

Минимальная КНФ строится аналогичным образом.

Пример 1.7. Построить МДНФ для формулы, заданной таблицей:

x	0	0	0	0	1	1	1	1
y	0	0	1	1	0	0	1	1
z	0	1	0	1	0	1	0	1
ψ	0	1	0	1	1	1	0	0

Построим СДНФ формулы: $\psi = \bar{x}\bar{y}z \vee \bar{x}yz \vee x\bar{y}z \vee xy\bar{z}$.

Найдем сокращенную ДНФ: $\psi = \overline{\overline{x}yz} \vee \underline{\overline{x}yz} \vee \underline{\underline{x\overline{y}z}} \vee \underline{\underline{x\overline{y}z}} = \overline{x}z \vee \overline{y}z \vee x\overline{y}$.

Построим матрицу Квайна. В строках матрицы – члены сокращенной ДНФ, в столбцах – члены СДНФ.

	$\overline{x}\overline{y}z$	$\overline{x}yz$	$x\overline{y}z$	$xy\overline{z}$
$\overline{x}z$	+	+		
$\overline{y}z$	+			+
$x\overline{y}$			+	+

Рассмотрим первую строку таблицы. Конъюнкт $\overline{x}z$ поглощает $\overline{x}\overline{y}z$ и $\overline{x}yz$ (так как оба они содержат $\overline{x}$ и z), следовательно, плюсы ставятся в первом и втором столбцах. Вторая и третья строки заполняются аналогично.

Если выбрать в таблице первую и третью строки, то все столбцы таблицы будут содержать плюсы. Теперь можно записать МДНФ: $\psi = \overline{x}z \vee x\overline{y}$.

Существует другой способ получения МДНФ и МКНФ из таблицы истинности формулы.

Построение МДНФ и МКНФ с помощью карт Карно

Карты Карно являются способом представления таблицы истинности формулы в виде, позволяющем легко находить члены, подлежащие склеиванию и поглощению. Для формулы, содержащей n переменных, таблица истинности изображается в виде n-мерного булева куба. В табл. 1.1 и 1.2 продемонстрирован порядок заполнения карты Карно для случая двух переменных и приведен пример заполнения карты для операции дизъюнкции.

Таблица 1.1

Карта Карно для случая двух переменных x и y

x \ y	0	1
0	$\phi(0,0)$	$\phi(0,1)$
1	$\phi(1,0)$	$\phi(1,1)$

Таблица 1.2

Карта Карно для формулы $x \vee y$

x \ y	0	1
0	0	1
1	1	1

Для формулы, содержащей три переменные, изображается развертка трехмерного куба. Если формула логики высказываний задана вектором значений, то эти значения с 1-го по 8-е заносятся в карту Карно в порядке, представленном в табл. 1.3. При таком расположении наборы значений, находящиеся в соседних ячейках таблицы, отличаются значением только одной переменной, следовательно, к ним может быть применена операция склеивания. При этом ячейки 1 и 3, а также 5 и 7, расположенные в одной строке на противоположных краях таблицы, считаются соседними (можно считать, что правый и левый края таблицы соединяются, замыкая ее в кольцо). В табл. 1.4 показан порядок заполнения карты Карно для четырех переменных. В этом случае смыкаются не только правый и левый края таблицы, но и верхний и нижний.

Таблица 1.3

Порядок заполнения карты Карно для трех переменных *x*, *y*, *z*

x \ *yz*	00	01	11	10
0	1	2	4	3
1	5	6	8	7

Таблица 1.4

Порядок заполнения карты Карно для четырех переменных *x*, *y*, *z*, *t*

xy \ *zt*	00	01	11	10
00	1	2	4	3
01	5	6	8	7
11	13	14	16	15
10	9	10	12	11

Для построения МДНФ в заполненной таблице необходимо выделить *k*-мерные кубы (прямоугольные блоки из 1, 2, 4, … 2^k ячеек), заполненные единицами. Каждому блоку соответствует конъюнкт в МДНФ. Блок должен быть как можно больше, а количество блоков как можно меньше. При этом любая из ячеек, содержащая единицу, может быть использована многократно (блоки могут пересекаться) [7].

Пример 1.8. Построить МДНФ для формулы, заданной таблицей:

x	0	0	0	0	1	1	1	1
y	0	0	1	1	0	0	1	1
z	0	1	0	1	0	1	0	1
ψ	1	1	0	0	1	1	1	0

Заполним карту Карно в соответствии с табл. 1.3:

x \ yz	00	01	11	10
0	1	1	0	0
1	1	1	0	1

В таблице можно выделить два блока, состоящих из единиц. Первый из них, отмеченный серым цветом, содержит 2 единицы (напомним, что левый и правый края таблицы смыкаются). В этом блоке происходит склеивание по переменной y, которая принимает значения 0 и 1, в то время как остальные переменные сохраняют значения $x=1,\ z=0$. Конъюнкт, который соответствует этому блоку, равен $x\bar{z}$ (как и при построении СДНФ, отрицания нужно расставить так, чтобы каждая литера была истинной).

Второй блок содержит 4 единицы. Склеивание здесь происходит по переменным x и z. Неизменное значение сохраняет переменная $y=0$. Ей соответствует конъюнкт, равный $\bar{y}$.

В итоге МДНФ имеет вид $x\bar{z}\vee\bar{y}$.

Аналогично для построения МКНФ в карте Карно выделяются прямоугольные блоки из 2^k ячеек, заполненных нулями. Каждому блоку соответствует дизъюнкт в МКНФ. Количество блоков должно быть минимальным, при этом каждый блок содержит наибольшее возможное количество ячеек. Любая из ячеек, содержащая ноль, может быть использована многократно.

Пример 1.8 (продолжение). Построить МКНФ для формулы из примера 1.8.

В полученной ранее карте Карно выделим блоки, состоящие из нулей:

x \ yz	00	01	11	10
0	1	1	0	0
1	1	1	0	1

В горизонтальном блоке происходит склеивание по переменной z, при этом $x=0,\ y=1$, в итоге имеем дизъюнкт $x\vee\bar{y}$ (каждая литера в нем должна

иметь значение 0). В вертикальном блоке происходит склеивание по x, соответствующий дизъюнкт равен $\overline{y} \vee \overline{z}$. МКНФ имеет вид $\left(x \vee \overline{y}\right) \wedge \left(\overline{y} \vee \overline{z}\right)$.

Если операция склеивания по карте Карно представляет сложность, можно выписать значения переменных на всех наборах, объединенных в блок. В итоговый конъюнкт или дизъюнкт войдут только те переменные, значение которых не меняется.

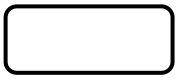
Пример 1.9. У мальчика Коли есть мама, папа, дедушка и бабушка. Коля пойдет гулять на улицу, если ему разрешит папа и еще хотя бы один из родственников. Составить формулу, показывающую, состоится прогулка или нет.

Поставим в соответствие каждому родственнику Коли переменные: x – мама, y – папа, z – дедушка, t – бабушка. Условимся обозначать согласие родственников единицей, несогласие – нулем. Пусть формула f описывает возможность пойти на прогулку. Составим таблицу истинности формулы и карту Карно:

x	y	z	t	f
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

xy \ zt	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	1	1	1	1
10	0	0	0	0

На карте Карно можно отметить три блока, соответствующие трем конъюнктам.

Рассмотрим блок 1. 

x	y	z	t
0	1	0	1
0	1	1	1
1	1	0	1
1	1	1	1

Не меняются переменные $y = 1$ и $t = 1$. В итоге получим yt.

Рассмотрим блок 2.

x	y	z	t
0	1	1	1
0	1	1	0
1	1	1	1
1	1	1	0

Ему соответствует yz.

Рассмотрим блок 3.

x	y	z	t
1	1	0	0
1	1	0	1
1	1	1	1
1	1	1	0

Ему соответствует xy.

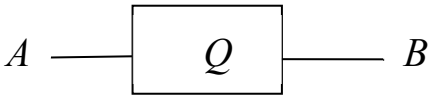
В результате получим формулу $f = xy \vee yz \vee yt$.

1.4. Некоторые приложения алгебры логики в технике. Релейно-контактные схемы

Релейно-контактные схемы (РКС, переключательные схемы) используются при проектировании автоматической техники и находят широкое применение в телефонии, телемеханике, на железнодорожном транспорте, а также в вычислительной технике.

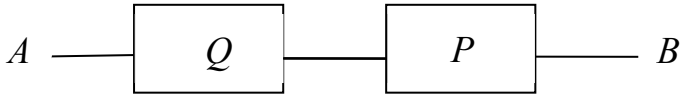
Переключательная схема представляет собой устройство, состоящее из переключателей (контактов), соединяющих их проводников, а также входов в схему и выходов из нее. На вход и выход РКС подается электрическое напряжение. Контакт может находиться либо в замкнутом, либо в разомкнутом положении.

Существует тесная связь между переключательными схемами и формулами алгебры логики. Пусть переключательная схема содержит один переключатель Q, имеет один вход A и один выход B. Переключателю Q ставится в соответствие высказывание q: «Переключатель Q замкнут». Если q истинно, то схема проводит электрический ток. Если q ложно, то переключатель разомкнут, и схема ток не проводит. Высказыванию q соответствует релейно-контактная схема:

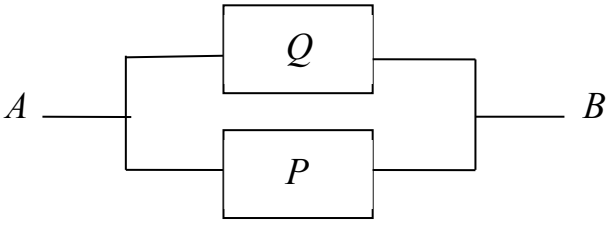


Основным логическим операциям также могут быть поставлены в соответствие переключательные схемы.

Конъюнкция двух высказываний q и p представляется схемой с последовательным соединением двух переключателей.



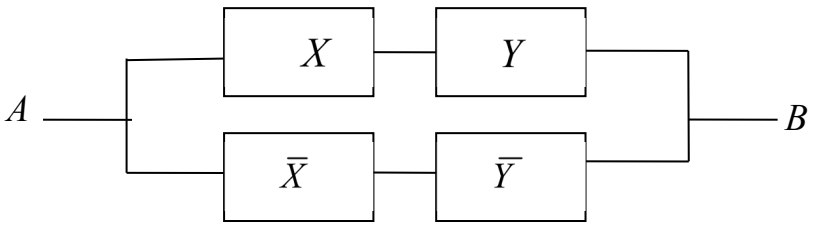
Дизъюнкция двух высказываний q и p будет представлена схемой с параллельным соединением двух переключателей.



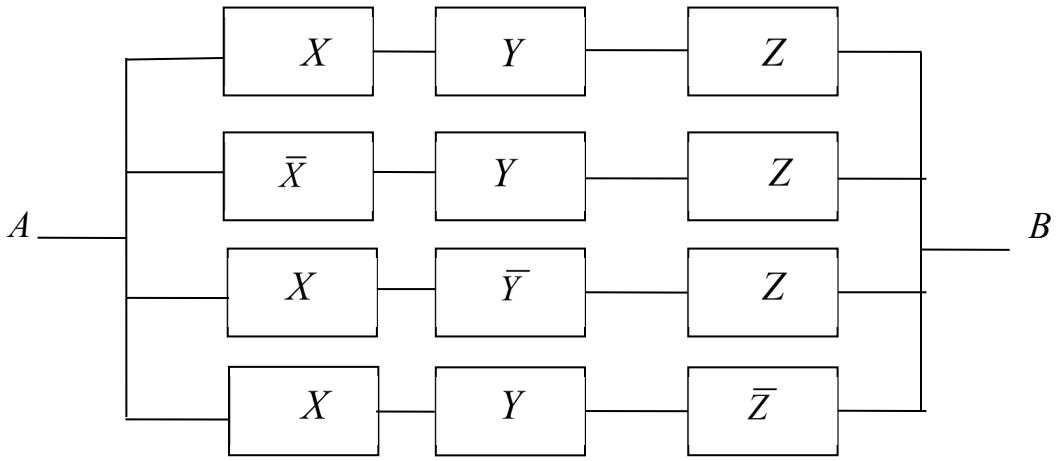
Так как любая формула может быть представлена операциями конъюнкции, дизъюнкции и отрицания (например, в виде КНФ или ДНФ), то всякая формула алгебры логики может быть изображена релейно-контактной схемой, и напротив, любой РКС можно поставить в соответствие некоторую формулу алгебры логики, которая называется *функцией проводимости РКС*.

Пример 1.10. Изобразить РКС, заданную функцией проводимости $L = xy \vee \overline{x}\,\overline{y}$.

Данная формула изображается на схеме параллельным соединением двух конъюнкций, каждая из которых есть последовательное соединение соответствующих высказываний. А именно, на одной из параллелей изображаются последовательно истинные высказывания, на другой – отрицания высказываний.



Пример 1.11. Упростить переключательную схему:



Эта схема представляется функцией проводимости

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

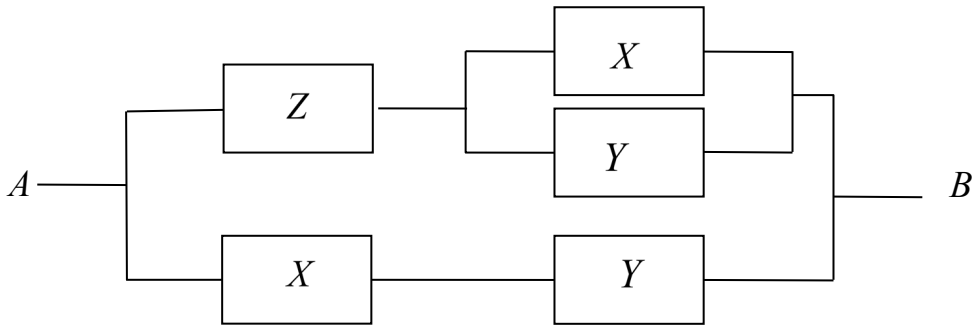
Формулу L можно сократить методом Квайна, склеивая первый конъюнкт последовательно со вторым, третьим и четвертым

$$L = yz \vee xz \vee xy\,.$$

Полученную формулу также можно упростить, вынося за скобку z в первых двух конъюнктах

$$L = \left(z \wedge \left(y \vee x\right)\right) \vee xy\,.$$

Последней формуле соответствует схема:



1.5. Булевы функции. Принцип двойственности

Функции, заданные на множестве булевых переменных $\{0,1\}$, называются *булевыми функциями*. Булева функция *n* переменных $f:\{0,1\}^n \to \{0,1\}$ каждому набору $(x_1, x_2, \ldots, x_n)$ нулей и единиц ставит в соответствие значение $f(x_1, x_2, \ldots, x_n)$, равное нулю или единице. Булевы функции также называются *логическими функциями*, *переключательными функциями*.

Булева функция может быть задана следующими способами:

1) формулой логики высказываний;

2) таблицей истинности;

3) вектором значений;

4) перечислением всех наборов, на которых значение функции равно нулю (или всех наборов, на которых она принимает значение 1);

5) описанием.

Пример 1.12. Рассмотрим булеву функцию трех переменных, значение которой равно 1, если значения всех ее аргументов равны между собой, и только в этом случае. Используя данное описание (способ 5), мы можем задать функцию набором равенств: $f(0,0,0) = f(1,1,1) = 1$ (способ 4), а также составить ее таблицу истинности (способ 2):

x	0	0	0	0	1	1	1	1
y	0	0	1	1	0	0	1	1
z	0	1	0	1	0	1	0	1
$f(x,y,z)$	1	0	0	0	0	0	0	1

Вектор значений функции имеет вид $f(x,y,z) = (1,0,0,0,0,0,0,1)$ (способ 3). Формульным представлением данной функции может быть, например, ее СДНФ: $f(x,y,z) = \overline{x}\,\overline{y}\,\overline{z} \vee xyz$ (способ 1).

Функция $f^+(x_1, x_2, \ldots x_n)$ называется *двойственной* по отношению к функции $f(x_1, x_2, \ldots x_n)$, если $f^+(x_1, x_2, \ldots x_n) = \overline{f\ (\overline{x_1}, \overline{x_2}, \ldots \overline{x_n})}$.

Двойственную функцию можно получить, заменив значения всех переменных и самой функции в таблице истинности на противоположные. Очевидно, что функцией, двойственной к f^+, будет функция f, т. е. $\left(f^+\right)^+ = f$.

Функция $f\left(x_1, x_2, \ldots x_n\right)$ называется *самодвойственной*, если она двойственна сама себе, $f^+\left(x_1, x_2, \ldots x_n\right) = f\left(x_1, x_2, \ldots x_n\right)$.

Пример 1.13. Найдем функции, двойственные к конъюнкции, дизъюнкции и отрицанию.

Пусть $f_1(x,y) = x \wedge y$, $f_2(x,y) = x \vee y$, $f_3(x) = \overline{x}$.

Так как $f_1(0,0) = 0$, $f_1(0,1) = 0$, $f_1(1,0) = 0$, $f_1(1,1) = 1$, то

$$f_1^+(1,1) = 1,\ f_1^+(1,0) = 1,\ f_1^+(0,1) = 1,\ f_1^+(0,0) = 0.$$

Легко заметить, что $f_1^+(x,y) = f_2(x,y)$, т. е. функцией, двойственной к конъюнкции, является дизъюнкция. Как было сказано ранее, это означает, что конъюнкция является функцией, двойственной к дизъюнкции, $f_2^+(x,y) = f_1(x,y)$.

И, наконец, рассмотрим отрицание. Поскольку $f_3(0) = 1$ и $f_3(1) = 0$, то $f_3^+(1) = 0$ и $f_3^+(0) = 1$. Таким образом, $f_3^+(x) = f_3(x)$, т. е. отрицание является самодвойственной функцией.

Полученные в примере результаты можно использовать для нахождения двойственной функции без построения таблицы истинности.

Теорема 1 (Принцип двойственности). Если функция представлена формулой, содержащей только конъюнкции, дизъюнкции, отрицания и логические константы, то для получения двойственной функции в исходной формуле следует заменить конъюнкции на дизъюнкции, дизъюнкции на конъюнкции, логическую константу 0 на 1, и 1 на 0.

Пример 1.14. Найти функцию, являющуюся двойственной к $g = \overline{a} \wedge (a \vee b) \vee 1$.

В соответствии с принципом двойственности $g^+ = \left(\overline{a} \vee (a \wedge b)\right) \wedge 0$.

1.6. Полные системы булевых функций

Система булевых функций называется *полной*, если любая булева функция может быть представлена как суперпозиция функций из этой системы.

Как было показано в подразделах 1.2 и 1.3, любая логическая функция может быть представлена конъюнктивной и / или дизъюнктивной нормальной формой, т. е. выражена через конъюнкцию, дизъюнкцию и отрицание. Таким образом, система $\left\{\wedge, \vee, \bar{}\right\}$ является полной. Важен вопрос о том, является ли некоторый набор функций полной системой, а также о том, является ли эта система избыточной, т. е. может ли количество функций в ней быть уменьшено без потери полноты.

Во множестве булевых функций можно выделить классы, замкнутые относительно суперпозиции, т. е. такие, что если функция представляется посредством нескольких функций, принадлежащих к определенном классу, то и она сама будет принадлежать к этому же классу.

Существует пять замкнутых классов логических функций. Они называются *классами Поста*:

1) P_0 – класс функций, сохраняющих ноль, т. е. функций, принимающих значение ноль, если значения всех переменных равны нулю,

$$P_0 = \left\{ f\left(x_1, x_2, \ \ldots, \ x_n\right) \middle| f\left(0, 0, \ \ldots, \ 0\right) = 0 \right\};$$

2) P_1 – класс функций, сохраняющих единицу, т. е.

$$P_1 = \left\{ f\left(x_1, x_2, \ \ldots, \ x_n\right) \middle| f\left(1, 1, \ \ldots, \ 1\right) = 1 \right\};$$

3) S – класс самодвойственных функций,

$$S = \left\{ f\left(x_1, x_2, \ \ldots, \ x_n\right) \middle| f^{+} = f \right\};$$

4) M – класс монотонных функций.

Функция $f\left(x_1, x_2, \ \ldots, x_n\right)$ называется *монотонной*, если для любых кортежей значений логических переменных $\left(t_1, t_2, \ \ldots, t_n\right)$ и $\left(s_1, s_2, \ \ldots, s_n\right)$ таких, что $t_i \le s_i$ для всех i, верно $f\left(t_1, t_2, \ \ldots, t_n\right) \le f\left(s_1, s_2, \ \ldots, s_n\right)$. Если значение одной из переменных увеличивается, то значение функции либо

увеличивается, либо остается неизменным (функция является неубывающей по всем переменным). Таким образом,

$$M = \left\{ f\left(x_1, x_2, \ \ldots, \ x_n\right) \middle| \ f \text{ монотонна} \right\};$$

5) L – класс линейных функций.

Булева функция называется *линейной*, если она представима в виде $f = c_0 \oplus c_1 x_1 \oplus c_2 x_2 \oplus \ldots \oplus c_n x_n$, где c_i – логические константы при $i = 0,1,\ldots,n$,

$$L = \left\{ f\left(x_1, x_2, \ \ldots, \ x_n\right) \middle| \ f = c_0 \oplus c_1 x_1 \oplus c_2 x_2 \oplus \ldots \oplus c_n x_n \right\}.$$

Пример 1.15. Выяснить, является ли дизъюнкция линейной функцией.

Пусть $f = x \vee y$. Предположим, что функция представима в виде $f = c_0 \oplus c_1 x \oplus c_2 y$. Попробуем найти c_0, c_1, c_2 методом неопределенных коэффициентов. Из таблицы истинности для дизъюнкции:

$0 \vee 0 = 0$, тогда $f\left(0,0\right) = c_0 \oplus c_1 0 \oplus c_2 0 = c_0 = 0$, откуда $c_0 = 0$;

$0 \vee 1 = 1$, тогда $f\left(0,1\right) = c_0 \oplus c_1 0 \oplus c_2 1 = 0 \oplus 0 \oplus c_2 = c_2 = 1$, откуда $c_2 = 1$;

$1 \vee 0 = 1$, тогда $f\left(1,0\right) = c_0 \oplus c_1 1 \oplus c_2 0 = 0 \oplus c_1 \oplus 0 = c_1 = 1$, откуда $c_1 = 1$.

Теперь все коэффициенты найдены и $f = 0 \oplus 1x \oplus 1y = x \oplus y$. Из таблицы истинности для дизъюнкции: $1 \vee 1 = 1$, однако $f\left(1,1\right) = 1 \oplus 1 = 0$. Следовательно, функция $f = x \vee y$ не представима в требуемом виде и не является линейной.

Теорема 2 (Критерий Поста или теорема Поста о функциональной полноте). Система булевых функций является полной тогда и только тогда, когда для каждого класса P_0, P_1, S, M, L в этой системе найдется хотя бы одна функция, не принадлежащая этому классу.

Пример 1.16. Доказать полноту системы $\left\{ \wedge, \vee, \bar{} \right\}$.

Проверим принадлежность конъюнкции, дизъюнкции и отрицания каждому из классов Поста.

P_0: $0 \wedge 0 = 0$, $0 \vee 0 = 0$, $\bar{0} = 1$, следовательно, $\wedge \in P_0$, $\vee \in P_0$, $\bar{} \notin P_0$.

P_1: $1 \wedge 1 = 1$, $1 \vee 1 = 1$, $\bar{1} = 0$, следовательно, $\wedge \in P_1$, $\vee \in P_1$, $\bar{} \notin P_1$.

S: В примере 1.13 показано, что $\wedge \notin S$, $\vee \notin S$, $\bar{} \in S$.

M: Из таблицы истинности конъюнкции и дизъюнкции (см. подраздел 1.1) видно, что если значение одной из переменных увеличивается с 0 до 1, то значение каждой из этих функций либо не меняется, либо также увеличивается. Для отрицания это не так. Если значение переменной x меняется с 0 на 1, то значение $\bar{x}$ изменится с 1 на 0, т. е. уменьшится. Таким образом, конъюнкция и дизъюнкция монотонны, а отрицание не является монотонной функцией. Таким образом, $\wedge \in M$, $\vee \in M$, $\bar{} \notin M$.

L: В примере 1.15 показано, что дизъюнкция не является линейной функцией. Аналогично доказывается, что конъюнкция не линейна. Отрицание является линейной функцией и представимо в виде $\bar{x} = 1 \oplus x$ (проверяется сравнением таблиц истинности). Следовательно, $\wedge \notin L$, $\vee \notin L$, $\bar{} \in L$.

Занесем полученные результаты в таблицу

	P_0	P_1	S	M	L
$\wedge$	+	+	–	+	–
$\vee$	+	+	–	+	–
$\bar{}$	–	–	+	–	+

В каждом из столбцов таблицы есть хотя бы один минус, следовательно, условие критерия Поста выполнено, и рассматриваемая система булевых функций является полной. Кроме того, видно, что полными также являются системы $\{\wedge, \bar{}\}$ и $\{\vee, \bar{}\}$.

Полная система булевых функций, исключение любой из которых делает ее неполной, называется *базисом*.

Пример 1.17. Система $\{\wedge, \oplus, 1\}$ называется *базисом Жегалкина*. Ниже приведена таблица, позволяющая установить полноту системы и показывающая, что ни одна из функций не может быть исключена из нее без потери полноты.

В качестве упражнения предлагается исследовать функции $\oplus$ и 1 на принадлежность к классам Поста.

	P_0	P_1	S	M	L
$\wedge$	+	+	–	+	–
$\oplus$	+	–	–	–	+
1	–	+	–	+	+

Другими примерами базисов являются системы: $\{\to, \bar{}\}$, $\{\leftrightarrow, \vee, 0\}$, $\{\leftrightarrow, \wedge, \oplus\}$, $\{\downarrow\}$, $\{|\}$.

1.7. Полином Жегалкина

Представление булевой функции в базисе Жегалкина $\{\wedge, \oplus, 1\}$ (пример 1.17) называется *полиномом (многочленом) Жегалкина*. Действительно, операция конъюнкции имеет сходство с умножением, а сумма Жегалкина (сложение по модулю 2) – со сложением; в результате булева функция, полученная с помощью этих операций, напоминает многочлен [4].

В общем случае полином Жегалкина имеет вид

$$P = \bigoplus_{\substack{1 \le i_1 < \ldots < i_k \le n \\ k = 0, 1, \ldots, n}} a_{i_1, \ldots, i_k} \wedge x_{i_1} \wedge \ldots \wedge x_{i_k},$$

где $a_{i_1, \ldots, i_k}$ – коэффициенты полинома, причем $a_{i_1, \ldots, i_k} \in \{0, 1\}$.

Число слагаемых в формуле полинома Жегалкина равно 2^n, где n – число переменных. В частности, полиномы Жегалкина нулевой, первой, второй и третьей степеней имеют вид

$$f_0 = a_0;$$

$$f_1(x_1) = a_0 \oplus a_1 x_1;$$

$$f_2(x_1, x_2) = a_0 \oplus a_1 x_1 \oplus a_2 x_2 \oplus a_{12} x_1 x_2;$$

$$f_3(x_1, x_2, x_3) = a_0 \oplus a_1 x_1 \oplus a_2 x_2 \oplus a_3 x_3 \oplus a_{12} x_1 x_2 \oplus a_{13} x_1 x_3 \oplus$$
$$\oplus a_{23} x_2 x_3 \oplus a_{123} x_1 x_2 x_3.$$

Построение полинома Жегалкина сводится к нахождению коэффициентов $a_{i_1,\ldots,i_k}$.

Далее рассматриваются основные методы построения полинома Жегалкина.

Алгоритм построения полинома Жегалкина по СДНФ

1. В СДНФ заменить знак дизъюнкции на знак суммы Жегалкина.

2. Упростить полученное выражение, используя формулы $\overline{x_i} \oplus x_i = 1$, $x_i \wedge 1 = x_i$ (т. е. вынести за скобки выражение вида $x_i \wedge x_j$).

3. Избавиться от отрицаний, используя формулу $\overline{x_i} = x_i \oplus 1$.

4. Раскрыть скобки в полученной формуле.

5. Привести подобные, используя формулы: $x_i \oplus x_i = 0$, $x_i \wedge 0 = 0$, $x_i \oplus 0 = x_i$ (вычеркнуть из формулы пару одинаковых слагаемых).

Полученное выражение является полиномом Жегалкина.

Пример 1.18. Построить полином Жегалкина для функции, заданной вектором значений $f(x_1, x_2, x_3) = (0,1,1,1,0,0,1,1)$.

Составим таблицу истинности для данной функции:

x_1	0	0	0	0	1	1	1	1
x_2	0	0	1	1	0	0	1	1
x_3	0	1	0	1	0	1	0	1
$f(x_1, x_2, x_3)$	0	1	1	1	0	0	1	1

Составим СДНФ функции, используя те наборы переменных, при которых функция принимает значение 1

$$f(x_1, x_2, x_3) = \overline{x_1}\,\overline{x_2}\,x_3 \vee \overline{x_1}\,x_2\,\overline{x_3} \vee \overline{x_1}\,x_2\,x_3 \vee x_1\,x_2\,\overline{x_3} \vee x_1\,x_2\,x_3.$$

Далее применяем последовательно этапы алгоритма для преобразования СДНФ

$$f(x_1,x_2,x_3)=\overline{x_1}\,\overline{x_2}x_3\vee\overline{x_1}x_2\overline{x_3}\vee\overline{x_1}x_2x_3\vee x_1x_2\overline{x_3}\vee x_1x_2x_3=\left|\text{заменим}\vee\text{на}\oplus\right|=$$

$$=\underline{\overline{x_1}\ \overline{x_2}\ x_3}\oplus\underline{\overline{x_1}\ x_2\ \overline{x_3}}\oplus\underline{\overline{x_1}\ x_2\ x_3}\oplus\underline{\underline{x_1\ x_2\ \overline{x_3}}}\oplus\underline{\underline{x_1\ x_2\ x_3}}=$$

$$=\left|\begin{array}{l}\text{вынесем общие множители:}\\ \text{для слагаемых (1) и (3)}\ \ \overline{x_1}\ x_3,\\ \text{для слагаемых (4) и (5)}\ \ x_1\ x_2\end{array}\right|=\overline{x_1}x_3\left(\overline{x_2}\oplus x_2\right)\oplus\overline{x_1}x_2\overline{x_3}\oplus x_1x_2\left(\overline{x_3}\oplus x_3\right)=$$

$$=\left|\begin{array}{l}\text{так как}\\ \overline{x_i}\oplus x_i=1\\ x_i\wedge 1=x_i\end{array}\right|=\overline{x_1}\ x_3\oplus\overline{x_1}\ x_2\ \overline{x_3}\ \oplus x_1\ x_2=\left|\begin{array}{l}\text{так как}\\ \overline{x_i}=x_i\oplus 1\end{array}\right|=$$

$$=\left(x_1\oplus 1\right)x_3\oplus\left(x_1\oplus 1\right)x_2\left(x_3\oplus 1\right)\oplus x_1x_2=\left|\text{раскрываем скобки}\right|=$$

$$=x_1x_3\oplus x_3\oplus\left(x_1\oplus 1\right)\left(x_2x_3\oplus x_2\right)\oplus x_1x_2=$$

$$=x_1x_3\oplus x_3\oplus x_1x_2x_3\oplus\underline{x_1x_2}\oplus x_2x_3\oplus x_2\oplus\underline{x_1x_2}=$$

$$=\left|\text{приводим подобные: удаляем пары одинаковых слагаемых}\right|=$$

$$=x_1x_3\oplus x_3\oplus x_1x_2x_3\oplus x_2x_3\oplus x_2=\underline{x_2\oplus x_3\oplus x_1x_3\oplus x_2x_3\oplus x_1x_2x_3}\ .$$

Алгоритм построения полинома Жегалкина методом неопределенных коэффициентов

1. Для функции $f(x_1,x_2,\ldots x_n)$ составляется таблица истинности.

2. Записывается полином Жегалкина в общем виде с неопределенными коэффициентами.

3. Последовательно в выражение полинома Жегалкина подставляются наборы значений переменных; в результате получаются линейные уравнения, решениями которых являются значения коэффициентов полинома Жегалкина. Для решения полученных уравнений используются свойства сложения по модулю 2: $a\oplus\overline{a}=1,\ a\oplus 0=a,\ a\oplus a=0$. Каждая подстановка позволяет найти один коэффициент.

Пример 1.19. Для функции $f(x_1, x_2, x_3, x_4) = (0000\,0010\,1001\,1010)$ составить полином Жегалкина.

Составим таблицу истинности данной функции:

x_1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
x_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_3	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
f	0	0	0	0	0	0	1	0	1	0	0	1	1	0	1	0

Запишем полином Жегалкина с неопределенными коэффициентами для функции четырех переменных

$$f(x_1, x_2, x_3, x_4) = a_0 \oplus a_1 x_1 \oplus a_2 x_2 \oplus a_3 x_3 \oplus a_4 x_4 \oplus a_{12} x_1 x_2 \oplus a_{13} x_1 x_3 \oplus$$
$$\oplus a_{14} x_1 x_4 \oplus a_{23} x_2 x_3 \oplus a_{24} x_2 x_4 \oplus a_{34} x_3 x_4 \oplus a_{123} x_1 x_2 x_3 \oplus a_{124} x_1 x_2 x_4 \oplus$$
$$\oplus a_{134} x_1 x_3 x_4 \oplus a_{234} x_2 x_3 x_4 \oplus a_{1234} x_1 x_2 x_3 x_4 .$$

Подставим набор переменных $(0;0;0;0)$ в полином. В соответствии с таблицей истинности на этом наборе функция равна нулю, $f(0;0;0;0) = 0$. В результате получается тождество $a_0 \oplus 0 = 0$, откуда $a_0 = 0$.

Теперь найдем коэффициент a_1. Для этого в полином подставим значения переменных $(1;0;0;0)$. Значение функции на этом наборе равно 1. Уравнение примет вид $f(1;0;0;0) = a_0 \oplus a_1 = 0 \oplus a_1 = 1$. Тогда $a_1 = 1$.

Далее аналогично подставляем все остальные наборы, причем сначала берутся наборы с одной единицей, затем с двумя и т. д. Места единиц в наборе указывают на коэффициент при соответствующих логических переменных. Так, например, если подставить в формулу набор переменных $(1;0;1;0)$, то будет найден коэффициент при конъюнкции $x_1 x_3$.

Вычислим последовательно все коэффициенты

$f(1;0;0;0) = a_0 \oplus a_1 = a_1 \oplus 0 = 1 \Rightarrow a_1 = 1$;

$$f(0;1;0;0) = a_0 \oplus a_2 = a_2 \oplus 0 = 0 \Rightarrow a_2 = 0;$$

$$f(0;0;1;0) = a_0 \oplus a_3 = a_3 \oplus 0 = 0 \Rightarrow a_3 = 0;$$

$$f(0;0;0;1) = a_0 \oplus a_4 = a_4 \oplus 0 = 0 \Rightarrow a_4 = 0;$$

$$f(1;1;0;0) = a_0 \oplus a_1 \oplus a_2 \oplus a_{12} = a_1 \oplus a_{12} = 1 \oplus a_{12} = 1 \Rightarrow a_{12} = 0;$$

$$f(1;0;1;0) = a_0 \oplus a_1 \oplus a_3 \oplus a_{13} = a_1 \oplus a_{13} = 1 \oplus a_{13} = 0 \Rightarrow a_{13} = 1;$$

$$f(1;0;0;1) = a_0 \oplus a_1 \oplus a_4 \oplus a_{14} = a_1 \oplus a_{14} = 1 \oplus a_{14} = 0 \Rightarrow a_{14} = 1;$$

$$f(0;1;1;0) = a_0 \oplus a_2 \oplus a_3 \oplus a_{23} = 0 \oplus a_{23} = 1 \Rightarrow a_{23} = 1;$$

$$f(0;1;0;1) = a_0 \oplus a_2 \oplus a_4 \oplus a_{24} = 0 \oplus a_{24} = 0 \Rightarrow a_{24} = 0;$$

$$f(0;0;1;1) = a_0 \oplus a_3 \oplus a_4 \oplus a_{34} = 0 \oplus a_{34} = 0 \oplus a_{34} = 0 \Rightarrow a_{34} = 0;$$

$$f(1;1;1;0) = a_0 \oplus a_1 \oplus a_2 \oplus a_3 \oplus a_{12} \oplus a_{13} \oplus a_{23} \oplus a_{123} = a_{123} \oplus 1 = 1 \Rightarrow a_{123} = 0;$$

$$f(1;1;0;1) = a_0 \oplus a_1 \oplus a_2 \oplus a_4 \oplus a_{12} \oplus a_{14} \oplus a_{24} \oplus a_{124} = a_{124} \oplus 0 = 0 \Rightarrow a_{124} = 0;$$

$$f(1;0;1;1) = a_0 \oplus a_1 \oplus a_3 \oplus a_4 \oplus a_{13} \oplus a_{14} \oplus a_{34} \oplus a_{134} = a_{134} \oplus 1 = 1 \Rightarrow a_{134} = 0;$$

$$f(0;1;1;1) = a_0 \oplus a_2 \oplus a_3 \oplus a_4 \oplus a_{23} \oplus a_{24} \oplus a_{34} \oplus a_{234} = a_{234} \oplus 1 = 0 \Rightarrow a_{234} = 1;$$

$$f(1;1;1;1) = a_0 \oplus a_1 \oplus a_2 \oplus a_3 \oplus a_4 \oplus a_{12} \oplus a_{13} \oplus a_{14} \oplus a_{23} \oplus$$

$$\oplus a_{24} \oplus a_{34} \oplus a_{123} \oplus a_{124} \oplus a_{134} \oplus a_{234} \oplus a_{1234} = a_{1234} \oplus 1 = 0 \Rightarrow a_{1234} = 1.$$

В результате полином Жегалкина выглядит так

$$f(x_1, x_2, x_3, x_4) = x_1 \oplus x_1x_3 \oplus x_1x_4 \oplus x_2x_3 \oplus x_2x_3x_4 \oplus x_1x_2x_3x_4.$$

Полином Жегалкина является канонической формой представления булевой функции, т. е. любая булева функция может быть представлена в виде полинома Жегалкина единственным образом с точностью до порядка слагаемых. Такими же каноническими формами являются СКНФ и СДНФ. Минимальные КНФ и ДНФ не обладают этим свойством, так как их вид зависит от выбора, производимого в ходе получения этих форм, и некоторые булевы функции могут иметь несколько различных представлений в виде МКНФ / МДНФ.

2. ЛОГИКА ПРЕДИКАТОВ

2.1. Понятие предиката

Предикатом называется функция, принимающая логические значения. Внутреннюю структуру предиката можно разделить на субъект и высказывание относительно субъекта – предикат. Например, в высказывании «Андрей сдал экзамен по математике», «Андрей» – субъект, а «сдал экзамен по математике» – предикат. Если в данном высказывании заменить «Андрей» на переменную x, тогда получится высказывательная форма «x – сдал экзамен по математике», которая определяет функцию одной переменной x, определенную на множестве «студенты», принимающая значения из множества $\{0;1\}$.

Таким образом, предикат выражает свойство субъекта и является функцией субъекта.

В логике высказываний булева функция $f(x_1, x_2, ..., x_n)$ – функция логических переменных, которые могут принимать только значения 0 или 1.

В логике предикатов предикат $P(x_1, x_2, ..., x_n)$ – функция, в которой переменные, определенные на множествах M_1, M_2, …, M_n, могут иметь разную природу.

Одноместным предикатом $P(x)$ называется $P(x): M \to \{0,1\}$, функция переменной x, определенная на множестве M, принимающая значения из множества $\{0,1\}$. Множество M называется *областью определения предиката.*

Областью истинности I_P предиката $P(x)$ называется множество значений x, при которых он принимает значение 1, т. е. $I_P = \{x \in M \,|\, P(x) = 1\}$.

Пример 2.1. Предикат $P(x): x^2 + 2x + 1 = 0;\ x \in \mathbb{R}$ – одноместный предикат, который является истинным только при $x = -1$, т. е. $I_P = \{1\}$.

Тождественно истинным называется предикат, определенный на множестве M и принимающий значение 1 для всех $x \in M$. Для такого предиката $I_P = M$.

Пример 2.2. Предикат $P(x): x^2 \geq 0;\ x \in \mathbb{R}$ является тождественно истинным.

Если требуется описать отношение между двумя субъектами, вводят двуместный предикат.

Двуместным предикатом $P(x, y)$ называется функция двух переменных x и y, определенная на множестве $M = M_1 \times M_2$ и принимающая значения из множества $\{0;1\}$.

Пример 2.3. Предикат $P(x, y): x > y$, где $x, y \in \mathbb{R}$ задает отношение «больше» на множестве действительных чисел. Если вместо переменных подставить числа, получится высказывание. Если вместо y подставить 0, тогда получится одноместный предикат $P(x): x > 0$, который описывает положительные действительные числа.

Пример 2.4. В штабе армии во время войны использовали пароли и отзывы, где в качестве паролей использовались названия городов СССР, а в качестве отзывов – реки СССР. Каждые сутки в штабе задавалось соответствие (код) между ними. Этот код является предикатом $P(x, y): (x \to y)$, где $x \in M_1,\ y \in M_2,\ M_1$ – множество городов, M_2 – множество рек.

В общем случае вводится *n*-местный предикат. *n-местным предикатом* $P(x_1, x_2, ..., x_n)$, определенным на множестве $M = M_1 \times M_2 \times ... \times M_n$, называется функция *n*-переменных $P: M_1 \times M_2 \times ... \times M_n \to \{0,1\}$.

Однородным предикатом называется такой предикат, у которого все предметные переменные определены на одном и том же множестве, $P: M^n \to \{0,1\}$.

Пример 2.5. Однородным трехместным предикатом будет $P(x, y, z)$: $x + y < z$, где $x, y, z \in \mathbb{R}$.

Пример 2.6. Однородным четырехместным предикатом будет $P(x,a,b,c)$: $ax^2+bx+c\geq 0$, где $a,b,c,x\in\mathbb{R}$.

Пример 2.7. Неоднородным трехместным предикатом будет $P(x;y;z)$: «человек x родился в году y в городе z».

Пусть два одноместных предиката $P(x)$ и $Q(x)$ определены на множестве *M*, тогда к ним применимы все операции логики высказываний (см. подраздел 1.1).

Конъюнкция $P(x)\wedge Q(x)$ – предикат, который принимает значение 1 только при тех значениях $x\in M$, при которых каждый из предикатов $P(x)$ и $Q(x)$ принимает значение 1. Областью истинности предиката $P(x)\wedge Q(x)$ является пересечение областей истинности предикатов $P(x)$ и $Q(x)$, т. е. $I_{P\wedge Q}=I_P\cap I_Q$.

Пример 2.8. Для предикатов $P(x)$: «x – четное число» и $Q(x)$: «x кратно 5» конъюнкцией является предикат $P(x)\wedge Q(x)$: «x – четное число и x кратно 5», т. е. «x делится на 10».

Дизъюнкция $P(x)\vee Q(x)$ – предикат, который принимает значение 0 только при тех значениях $x\in M$, при которых каждый из предикатов принимает значение 0. Очевидно, что $I_{P\vee Q}=I_P\cup I_Q$.

Пример 2.9. Для предикатов $P(A)$: «определитель матрицы A положителен» и $Q(A)$: «определитель матрицы A отрицателен» дизъюнкцией является предикат $P(A)\vee Q(A)$: «определитель матрицы A не равен нулю» или, в иной формулировке, $P(A)\vee Q(A)$: «матрица A – невырожденная».

Отрицание $\overline{P(x)}$ – предикат, который принимает значение 1 при тех $x\in M$, при которых предикат $P(x)$ принимает значение 0, и наоборот. Областью истинности предиката $\overline{P(x)}$ будет $I_{\overline{P}}=M\setminus I_P$.

Пример 2.10. Если предикат $P(x)$: «$x^2 - 4 = 0$», $x \in \mathbb{R}$, тогда предикат $\overline{P(x)}$: «$x^2 - 4 \neq 0$», $x \in \mathbb{R}$.

Импликация $P(x) \to Q(x)$ – предикат, который принимает значение 0 только для тех значений $x \in M$, при которых $P(x)$ принимает значение 1, а $Q(x)$ – 0. Так как для любого значения $x \in M$ верно $P(x) \to Q(x) = \overline{P(x)} \vee Q(x)$, то $I_{P \to Q} = I_{\overline{P}} \cup I_Q$.

2.2. Кванторы

В логике предикатов рассматриваются еще две операции, которые обращают одноместный предикат в высказывание – *кванторы* (от латинского quantum – сколько). Существуют два квантора: *квантор всеобщности* $\forall$ и *квантор существования* $\exists$.

Рассмотрим одноместный предикат $P(x)$, определенный на множестве *M.* Высказывание $\forall x P(x)$, которое читается «для всех x верно $P(x)$», истинно в том и только в том случае, когда предикат $P(x)$ тождественно истинен на *M*.

Пример 2.11. Если x – студент, а $P(x)$: «x имеет зачетную книжку», то высказывание $\forall x P(x)$ означает «любой студент имеет зачетную книжку».

Рассмотрим одноместный предикат $P(x)$, определенный на множестве *M.* Высказывание $\exists x P(x)$, которое читается «существует x, при котором верно $P(x)$», истинно тогда и только тогда, когда существует хотя бы один элемент $x \in M$, для которого $P(x)$ принимает значение 1.

Пример 2.12. Если x – студент, а $P(x)$: «x живет в общежитии», то высказывание $\exists x P(x)$ означает «некоторые студенты живут в общежитии».

Формула, на которую распространяется действие квантора, называется *областью действия* квантора. Переменная, стоящая под знаком

квантора и попадающая в его область действия, называется *связанной*. Переменная, не попадающая в область действия квантора, называется *свободной*.

Пример 2.13. В предикате $\forall x\, P(x, y)$ переменная x является связанной, а переменная y – свободной. В предикате $P(x, y) \wedge \forall x\, Q(x, y)$ переменная y является свободной, x – и связанной, и свободной.

Пусть область определения предиката $P(x)$ – конечное множество $M = \{a_1, a_2, ..., a_n\}$. Тогда кванторы могут быть выражены через конъюнкцию и дизъюнкцию

$$\forall x\, P(x) = P(a_1) \wedge P(a_2) \wedge ... \wedge P(a_n);$$

$$\exists x\, P(x) = P(a_1) \vee P(a_2) \vee ... \vee P(a_n).$$

Кванторы $\forall$ и $\exists$ связаны друг с другом по принципу двойственности (по законам де Моргана)

$$\overline{\forall x P(x)} = \exists x \overline{P(x)};$$

$$\overline{\exists x P(x)} = \forall x \overline{P(x)}.$$

Пример 2.14. Если M – множество птиц, $P(x)$: «x летает», $x \in M$, то высказывание $\overline{\forall x P(x)}$ означает «не все птицы летают» или «существуют нелетающие птицы», т. е. $\exists x \overline{P(x)}$. Высказывание $\overline{\exists x P(x)}$ – «не существует летающих птиц» – эквивалентно высказыванию «все птицы не летают», т. е. $\forall x \overline{P(x)}$.

Кванторы широко применяются в математическом анализе и других разделах математики.

Пример 2.15. Рассмотрим определение предела функции в точке x_0: «Число A называется пределом функции $y = f(x)$ в точке x_0, если для любого $\varepsilon > 0$ существует $\delta(\varepsilon) > 0$, такое, что для любого x, удовлетворяющего условию $|x - x_0| < \delta$, верно $|f(x) - A| < \varepsilon$».

На языке кванторов и предикатов определение предела запишется так:
$P(A, y, x_0)$: « $\lim\limits_{x \to x_0} y = A$», $\forall \varepsilon > 0$, $\exists \delta(\varepsilon) > 0$: $\forall x \ |x - x_0| < \delta \ \rightarrow \ |y - A| < \varepsilon$.

2.3. Тавтологии логики предикатов

Из подраздела 1.1 известно, что тавтологией логики высказываний называется тождественно истинное высказывание. Примерами тавтологий являются высказывания:

$\overline{(x \wedge \overline{x})}$ – закон отрицания противоречия;

$(x \leftrightarrow y) \leftrightarrow (\overline{x} \leftrightarrow \overline{y})$ – закон противоположности;

$(x \rightarrow y) \wedge (y \rightarrow z) \rightarrow (x \rightarrow z)$ – правило цепного заключения;

$(x \wedge (x \rightarrow y)) \rightarrow y$ – правило *modus ponens*.

$(x \rightarrow y) \leftrightarrow (\overline{y} \rightarrow \overline{x})$ – закон контрапозиции;

Наиболее важные тавтологии логики высказываний получаются из равносильностей, приведенных в подразделе 1.1, заменой знака = на $\leftrightarrow$ ($\phi = \psi$ тогда и только когда, когда формула $\phi \leftrightarrow \psi$ является тавтологией).

Если $F(x_1, ..., x_n)$ является тавтологией и $\phi_1, ..., \phi_n$ – формулы, то $F(\phi_1, ..., \phi_n)$ также является тавтологией.

Формула исчисления предикатов называется *тавтологией* (*тождественно истинной* или *общезначимой* формулой), если она принимает значение «истина» на любых наборах переменных.

Если $F(x_1, ..., x_n)$ является тавтологией в исчислении высказываний и $P_1, ..., P_n$ – предикаты, то $F(P_1, ..., P_n)$ – тавтология в исчислении предикатов.

Основные тавтологии логики предикатов

1) $\overline{\forall x P(x)} \leftrightarrow \exists x \overline{P(x)}$, $\overline{\exists x P(x)} \leftrightarrow \forall x \overline{P(x)}$ – закон де Моргана;

2) $(\forall x)(P(x) \wedge Q(x)) \leftrightarrow (\forall x) P(x) \wedge (\forall x) Q(x)$,

$(\exists x)(P(x) \vee Q(x)) \leftrightarrow (\exists x) P(x) \vee (\exists x) Q(x)$ – перенесение квантора через конъюнкцию и дизъюнкцию;

3) $(\forall x)\big(P(x)\vee Q\big)\leftrightarrow(\forall x)P(x)\vee Q$,

$(\exists x)\big(P(x)\wedge Q\big)\leftrightarrow(\exists x)P(x)\wedge Q$, где Q не зависит от x;

$Q\leftrightarrow(\exists x)Q$, $Q\leftrightarrow(\forall x)Q$, где Q не зависит от x;

4) $(\forall x)P(x)\to P(x)$ – закон удаления квантора всеобщности;

5) $P(x)\to(\exists x)P(x)$ – закон введения квантора существования;

6) $(\forall x)(\forall y)P(x,y)\leftrightarrow(\forall y)(\forall x)P(x,y)$,

$(\exists x)(\exists y)P(x,y)\leftrightarrow(\exists y)(\exists x)P(x,y)$ – коммутативность кванторов;

7) $(\exists x)(\forall y)P(x,y)\to(\forall y)(\exists x)P(x,y)$.

2.4. Нормальная и предваренная нормальная формы логики предикатов

Нормальной формой формулы логики предикатов называется форма, содержащая только конъюнкции, дизъюнкции и кванторы, а также операцию отрицания, примененную только к элементарным формулам.

Всякую формулу логики предикатов можно привести к нормальной форме, используя равносильности логики высказываний и предикатов.

Пример 2.16. Привести формулу $\big(\exists xP(x)\to\forall yQ(y)\big)\to R(z)$ к нормальной форме.

После использования равносильности логики высказываний получим

$$\big(\exists xP(x)\to\forall yQ(y)\big)\to R(z)=\big|10)\text{ подраздел }1.1\big|=\overline{\overline{\exists xP(x)}\vee\forall yQ(y)}\vee R(z)=$$

$$=\big|8)\text{ подраздел }1.1\big|=\overline{\overline{\exists xP(x)}}\wedge\overline{\forall yQ(y)}\vee R(z)=\big|2)\text{ подраздел }1.1\big|=$$

$$=\exists xP(x)\wedge\exists y\overline{Q(y)}\vee R(z).$$

Предваренной нормальной формой формулы логики предикатов называется нормальная форма, в которой либо кванторы полностью отсутствуют, либо находятся в начале формулы, и область действия каждого квантора распространяется на всю формулу.

Предваренная нормальная форма формулы логики предикатов имеет вид

$$(\sigma x_1)(\sigma x_2)...(\sigma x_n)\,A(x_1, x_2, ..., x_m),\ n \le m,$$

где под символом σx_i понимается один из кванторов $\forall x_i$ или $\exists x_i$; при этом формула A – нормальная форма, не содержащая кванторов.

Теорема 3. Всякая формула логики предикатов может быть приведена к предваренной нормальной форме.

Пример 2.17. Привести к предваренной нормальной форме формулу $A = \forall x \exists y P(x, y) \wedge \overline{\exists x \forall y Q(x, y)}$.

После использования тавтологий логики предикатов получим

$$A = \forall x \exists y P(x, y) \wedge \overline{\exists x \forall y Q(x, y)} = |1) \text{ подраздел } 2.3| =$$

$$= \forall x \exists y P(x, y) \wedge \forall x \exists y \overline{Q(x, y)} = |2) \text{ подраздел } 2.3| = \forall x \left(\exists y P(x, y) \wedge \exists z \overline{Q(x, z)} \right) =$$

$$= |3) \text{ подраздел } 2.3| = \forall x \exists y \left(P(x, y) \wedge \exists z \overline{Q(x, z)} \right) = |3) \text{ подраздел } 2.3| =$$

$$= \forall x \exists y \exists z \left(P(x, y) \wedge \overline{Q(x, z)} \right).$$

3. КОНЕЧНЫЕ АВТОМАТЫ

3.1. Конечные автоматы. Понятие языка. Способы задания конечных автоматов

Конечный автомат представляет собой абстрактную математическую модель дискретного устройства, которое имеет конечное множество возможных состояний $Q = \{q_0, q_1, ..., q_m\}$, среди которых выделяют одно входное состояние q_0 (*вход*) и одно или несколько заключительных состояний (*выходов*), образующих множество заключительных состояний $F \subset Q$. В каждый момент времени устройство может находиться только в одном из состояний $q_i \in Q$. Входные данные в виде цепочки символов поступает на вход устройства и последовательно обрабатываются, в результате чего происходит переход устройства из одного состояния в другое.

Непустое конечное множество символов $V = \{a_1, ..., a_n\}$, которые могут обрабатываться конечным автоматом, называют *входным алфавитом* конечного автомата. Элементы алфавита V называются *буквами* или *символами*.

Входными данными могут являться не только буквы какого-либо алфавита, но и сигналы от датчиков, кнопок, клавиатуры или других устройств. В этом случае задается взаимно однозначное соответствие между поступающими дискретными сигналами и символами некоторого алфавита.

Из букв или символов формируются входные цепочки или слова. *Слово* или *цепочка* в алфавите V – это произвольный кортеж из символов алфавита, то есть элемент множества V^k для различных $k = 0, 1, 2, ...$ [2, 3].

Устройство, моделируемое конечным автоматом, состоит из блока управления, считывающей головки автомата и входной ленты. Оно может считывать и обрабатывать цепочку символов, записанных на входной ленте

(входную цепочку); результат обработки определяет состояние, в котором находится устройство в тот или иной момент времени.

Входная лента представляет собой полубесконечную неограниченную справа ленту, разделенную на отдельные ячейки. В ячейках записываются символы $a_1,...,a_k,...$ алфавита V, составляющие входную цепочку. Все символы цепочки записываются в ячейки ленты последовательно, начиная с первой, без пропусков.

На рис. 3.1 изображена схема конечного автомата и расположение на входной ленте цепочки символов *aacbab.*

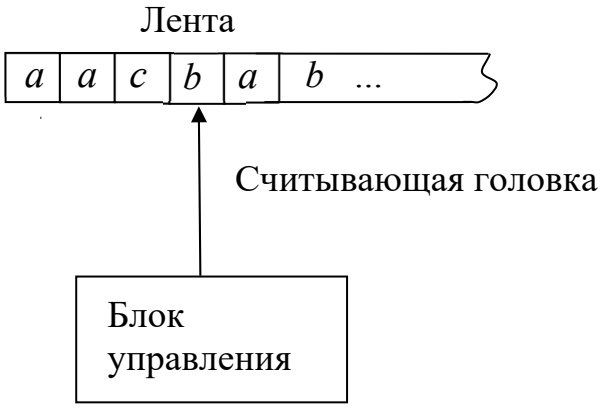


Рис. 3.1. Схематическое представление конечного автомата

Считывающая головка автомата всегда указывает ровно на одну ячейку входной ленты и после прочтения очередного символа сдвигается на одну ячейку вправо или остается на прежнем месте. Устройство в каждый момент времени может находиться только в одном состоянии $q_i \in Q$.

На вход конечного автомата в последовательные дискретные моменты времени (называемые *тактами*) $t_1,t_2,...,t_k,...$ поступают символы входной цепочки (входные данные) $a_1,a_2,...,a_k,...$ алфавита V.

Пусть в момент времени t_k считывающая головка автомата установлена напротив k-й ячейки ленты, автомат находится в некотором состоянии $q_i \in Q$. Один такт работы автомата состоит в следующем: автомат считывает содержимое ячейки и после обработки входных данных переходит

в некоторое новое состояние q_j, которое зависит от символа, содержащегося в ячейке, и текущего состояния. Новое состояние может совпадать с предшествующим состоянием q_i.

В блоке управления могут быть предусмотрены переходы из одного состояния в другое по пустому слову; при этом автомат не читает ленту и не продвигает головку. Такой такт называют λ-тактом и рассматривают как переход из одного состояния в другое по пустой цепочке. Пустая цепочка не является аналогом пробела в последовательности символов или пустого множества.

Если после полного прочтения входной цепочки конечный автомат переходит в одно из заключительных состояний, то говорят, что автомат распознал цепочку.

Возможна ситуация, когда после прочтения очередного входного символа считывающая головка не движется, а блок управления остается в текущем состоянии, то есть автомат «зависает». В этом случае говорят, что автомат не может обработать входную цепочку.

Множество всех слов в заданном алфавите V, которые может распознать (обработать) конечный автомат M, называется *языком конечного автомата* L или $L(M)$.

Работа блока управления конечного автомата определяется системой команд, каждая из которых задается записью $q_i a \to q_j$ или $q_i \lambda \to q_j$. Первая команда означает, что из состояния q_i по символу $a \in V$ автомат может перейти в состояние q_j, вторая команда означает, что из состояния q_i по пустой цепочке можно перейти в состояние q_j. Возможна ситуация, когда $q_i = q_j$; в этом случае состояние автомата не изменяется. Команда $q_i a \to q_j$ и команда $q_i \lambda \to q_j$ являются взаимоисключающими. Если при прочтении символа a в состоянии q_i автомат зависает, это означает, что в блоке управления отсутствует соответствующая команда. Множество всех команд задает *функцию перехода состояний* $g : Q \times V \cup \{\lambda\} \to Q$.

Конечный автомат может быть представлен в виде размеченного ориентированного графа $W = (G; \varphi)$, где $G = (Q, E)$ – ориентированный граф

[2, 4], заданный множеством вершин Q, совпадающим со множеством состояний, и множеством дуг E, а φ – весовая функция (или функция разметки), ставящая в соответствие каждой дуге $e_i \in E$ ориентированного графа G некоторую метку – символ входного алфавита или пустое слово λ. Множество дуг и функция разметки определяются системой команд конечного автомата. Если в системе команд автомата есть команда $q_i a \to q_j$, это означает, что в ориентированном графе из вершины q_i в вершину q_j ведет дуга с меткой a, где $a \in V$. Если в системе команд есть команда $q_i \lambda \to q_j$, это означает, что существует дуга из вершины q_i в вершину q_j с меткой λ. Тогда цепочка символов $a_1,...,a_k$ будет распознаваться автоматом, если в ориентированном графе существует путь $q_0, q_1,..., q_k$ такой, что для любого m существует команда $q_{m-1} a_m \to q_m$ и соответствующая ей дуга $(q_{m-1}, q_m) \in E$ с меткой a_m.

Вершина графа, соответствующая начальному состоянию автомата, далее будет обозначаться короткой заходящей стрелкой, а вершины, соответствующие заключительным состояниям, – короткой стрелкой (рис. 3.2). Возможны и другие способы отметить начальное и заключительные состояния конечного автомата.

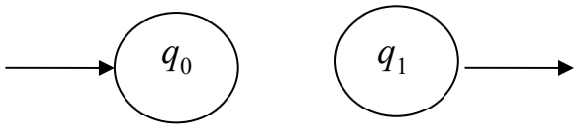
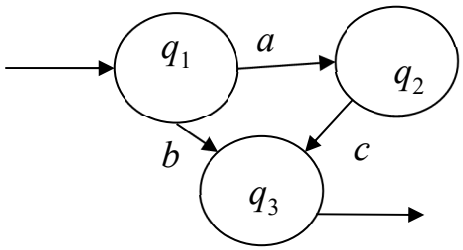


Рис. 3.2. Начальное (q_0) и заключительное (q_1) состояния конечного автомата

Последовательность вершин $q_0, q_1,..., q_k,...$ в ориентированном графе, представляющем конечный автомат, может быть конечной или бесконечной. Наличие бесконечной последовательности вершин (бесконечного пути) означает существование контура в графе.

Пример 3.1. Пусть конечный автомат M задан множеством состояний $Q=\{q_1,q_2,q_3\}$; входным алфавитом $V=\{a,b,c\}$ и системой команд $q_1a\to q_2$, $q_1b\to q_2$, $q_2c\to q_3$.

Входным состоянием является q_1, заключительным – q_3. Граф конечного автомата M изображен на рисунке.



Цепочки ac и b распознаются автоматом M. Примерами цепочек, которые не распознаются автоматом, могут быть слова ab и ca.

Таким образом, $L(M)=\{ac,b\}$.

Еще одним способом задания конечного автомата является система канонических уравнений, которые описывают состояние автомата в любой момент времени.

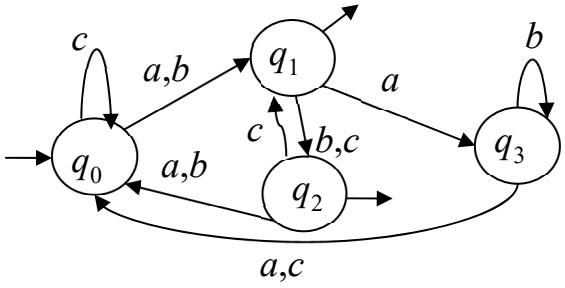
Через $S_k(\alpha,t)$ обозначается состояние, в котором окажется конечный автомат через t тактов времени, получив на вход слово $\alpha=a_1a_2...a_m$. Система канонических уравнений имеет вид

$$\begin{cases} S_k(\alpha,0)=q_0 \\ S_k(\alpha,t)=g\left(S_k(\alpha,t-1),a_t\right), t=1,...m. \end{cases}$$

Первое уравнение системы является начальным условием, оно показывает, что в начальный момент времени, то есть на нулевом такте, автомат находится в состоянии q_0.

Второе уравнение системы показывает состояние, в котором будет находиться автомат на такте времени t при чтении слова α. Функция $g:Q\times V\cup\{\lambda\}\to Q$ – описанная выше функция переходов состояний.

Пример 3.2. Задан конечный автомат K с входным алфавитом $V=\{a,b,c\}$, q_1, q_2 – конечные состояния, то есть $F=\{q_1,q_2\}$.



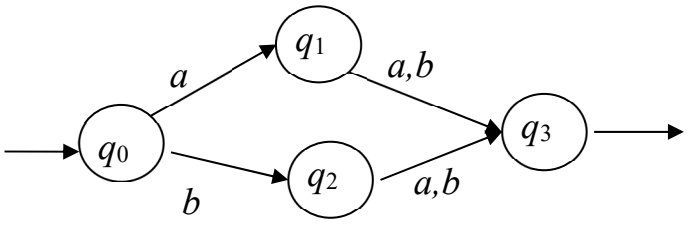
Вычислить $S_k(\alpha,t)$, где $\alpha=abcc$, $t=3$.

Решение. $S_k(abcc,0)=q_0$, $S_k(abcc,1)=g(q_0,a)=q_1$,

$S_k(abcc,2)=g(q_1,b)=q_2$, $S_k(abcc,3)=g(q_2,c)=q_1$.

Ответ: q_1.

Пример 3.3. Задан конечный автомат K с входным алфавитом $V=\{a,b\}$, q_3 – конечное состояние, то есть $q_3\in F$.



Какие из следующих слов принадлежат языку $L(K)$: aab, aa, ba, bab, $aaab$?

Решение. $S_k(aab,3)=\varnothing$ так как нет дуги с меткой b из вершины q_3, отсюда $aab\notin L$;

$S_k(aa,2)=q_3\in F$, по этому слову мы пришли в конечное состояние, отсюда $aa\in L$;

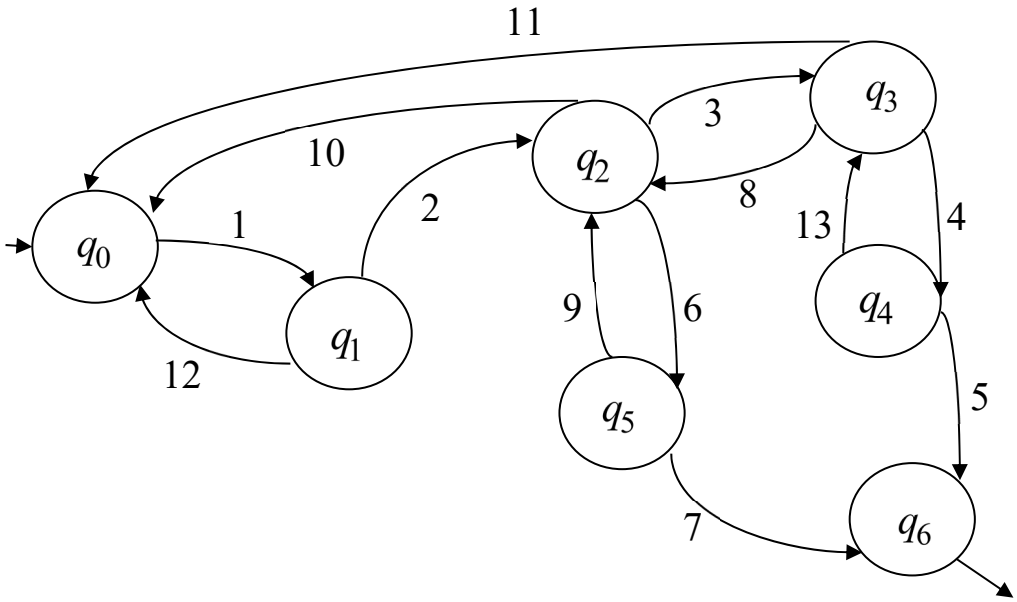
$S_k(ba,2)=q_3\in F$, отсюда $ba\in L$;

$S_k(aaab,3)=\varnothing$, отсюда $aaab\notin L$.

Ответ: $aa\in L$, $ba\in L$.

Конечные автоматы используются для моделирования поведения различных приборов и систем, а также для описания работы алгоритмов, например, при создании пользовательских интерфейсов, решении задач управления памятью, для задания поведения персонажей в компьютерных играх.

Пример 3.4. Рассмотрим конечный автомат, который описывает работу кофемашины



Перечислим состояния автомата и команды перехода:

q_0 – начальное состояние (выключено), команда 1 – включение:

$$q_0\ 1 \to q_1;$$

q_1 – включено (режим ожидания), команда 2 – нагрев, 12 – выключение:

$$q_1\ 2 \to q_2,\ q_1 12 \to q_0;$$

q_2 – готов к работе, команда 3 – включение генератора пара, 6 – приготовление эспрессо, 10 – отключение:

$$q_2\ 3 \to q_3,\ q_2\ 6 \to q_5,\ q_2\ 10 \to q_0;$$

q_3 – работа генератора пара, 4 – ручка пара вперед, 8 – отключение пара, 11 – отключение:

$$q_3\ 4 \to q_4,\ q_3\ 8 \to q_2,\ q_3\ 11 \to q_0;$$

q_4 – пар готов, 5 – подача пара в чашку, 13 – ручка пара назад:

$$q_4\ 5 \to q_6,\ q_4\ 13 \to q_3;$$

q_5 – эспрессо готов, 7 – подача эспрессо в чашку, 9 – остановка приготовления эспрессо:

$$q_5\ 7 \to q_6,\ q_5\ 9 \to q_3;$$

q_6 – работа завершена.

Цепочка 1, 2, 6, 7 задает процесс приготовления эспрессо, а цепочка 1, 2, 3, 4, 5 – подачу пара для взбивания молока.

3.2. Детерминированный конечный автомат

В описании конечного автомата возможна ситуация, когда блок управления, находящийся в состоянии q_i, при прочтении символа a может перейти в одно из нескольких состояний q_{j1}, q_{j2}, …, q_{jk}. Поведение такого конечного автомата носит случайный характер и не может быть предсказано однозначно.

Отношение g, описывающее множество всех команд, в этом случае уже не является функцией, хотя по-прежнему называется функцией переходов состояний.

Конечный автомат M называется *детерминированным*, если в нем нет команд, содержащих пустую цепочку (нет дуг с меткой λ в ориентированном графе, задающем этот конечный автомат), и из любого состояния по любому входному символу возможен переход не более чем в одно состояние, т. е. $\forall q_i \in Q$ и $\forall a \in V$ существует не более одного $q_j \in Q$, такое, что $q_i a \to q_j$ в соответствии с функцией переходов g. В данном случае термин «функция» применим к $g : Q \times V \cup \{\lambda\} \to Q$ в полной мере.

Два конечных автомата называются *эквивалентными*, если их языки совпадают. Для любого конечного автомата может быть построен эквивалентный ему детерминированный конечный автомат.

Пусть конечный автомат M задан как упорядоченная пятерка

$$M = (V, Q, q_0, F, g),$$

где V – входной алфавит; Q – множество состояний; q_0 – начальное состояние; F – множество заключительных состояний; $g : Q \times V \cup \{\lambda\} \to Q$ – функция переходов.

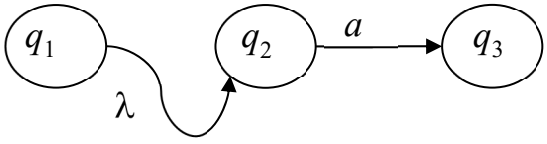
Тогда эквивалентный M детерминированный конечный автомат строится в два этапа.

1. Находится конечный автомат $M_1 = (V, Q_1, q_0, F_1, g_1)$, эквивалентный конечному автомату M, не содержащий команд вида $q_i\lambda \to q_j$. Иными словами, по графу исходного автомата строится граф нового конечного автомата, не содержащего дуг с меткой λ.

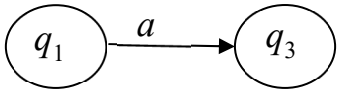
2. Находится конечный автомат $M_2 = (V, Q_2, \{q_0\}, F_2, g_2)$, эквивалентный M_1, в котором из любого состояния автомата по любому входному символу возможен переход не более чем в одно состояние.

Задача удаления λ-переходов является довольно простой, и способы ее решения далее демонстрируются на примерах.

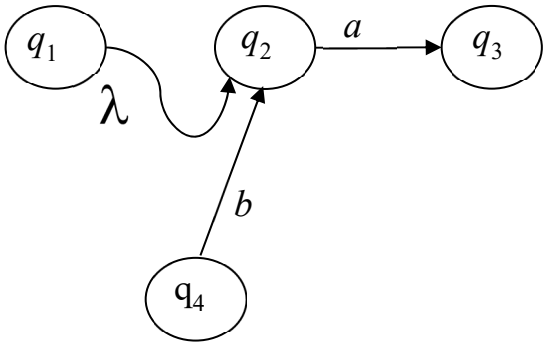
Пример 3.5. Пусть из состояния q_1 в состояние q_2 ведет дуга с меткой λ, и из состояния q_2 в состояние q_3 ведет дуга с меткой a, при этом в состояние q_2 заходят только дуги с меткой λ



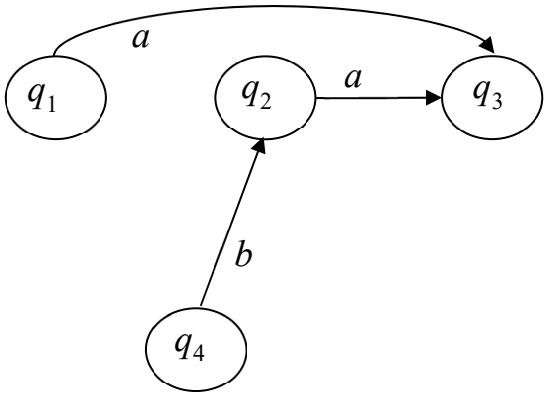
Результат удаления λ-дуг представлен ниже.



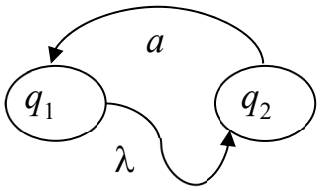
Пример 3.6. Дан конечный автомат



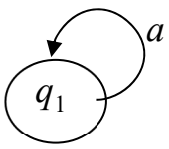
Конечный автомат без λ-переходов выглядит следующим образом



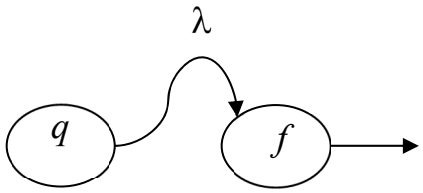
Пример 3.7. Дан конечный автомат



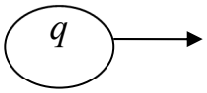
В результате удаления λ-дуги получим дугу, ведущую из состояния q_1 в то же состояние (петлю).



Пример 3.8. Пусть из состояния q в состояние f ведет путь ненулевой длины, состоящий из дуг с меткой λ, и состояние f является заключительным.



Тогда после удаления λ-дуг состояние q станет заключительным.



Пусть $M_1 = (V, Q_1, q_0, F_1, g_1)$ – конечный автомат без λ-переходов. Необходимо построить эквивалентный ему детерминированный конечный автомат M_2, то есть такой, в котором из любого состояния конечного автомата по любому входному символу возможен переход не более чем в одно состояние.

Состояния нового конечного автомата Q_2 являются подмножествами множества состояний конечного автомата M_1, то есть $Q_2 \subseteq P(Q_1)$, где $P(Q_1)$ – булеан множества Q_1. Таким образом, каждое отдельное состояние S конечного автомата M_2 является множеством, $S \subseteq Q_1$, $S = \{q_{s1}, ..., q_{sk}\}$. Состояния конечного автомата M_2 будем называть *состояниями-множествами*.

Начальным состоянием конечного автомата M_2 является множество $\{q_0\}$, содержащее только начальное состояние конечного автомата M_1.

Заключительными состояниями конечного автомата M_2 являются подмножества Q_1, которые содержат хотя бы одну заключительную вершину конечного автомата M_1.

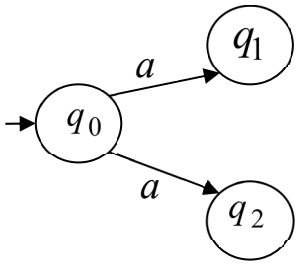
Функция переходов нового конечного автомата M_2 определяется следующим образом: из состояния-множества S по входному символу a ко-

нечный автомат M_2 переходит в состояние-множество, представляющее собой объединение всех множеств состояний конечного автомата M_1, в которые этот конечный автомат переходит по символу a из каждого состояния множества S, т. е.

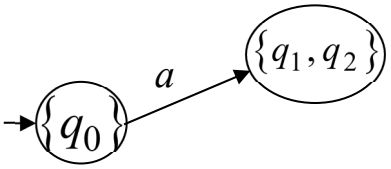
$$g_2(S,a) = \bigcup_{q \in S} g_1(q,a).$$

В итоге получается конечный автомат $M_2 = (V, Q_2, \{q_0\}, F_2, g_2)$, который является детерминированным по построению. Здесь $Q_2 \subseteq P(Q_1)$ – множество состояний автомата M_2; F_2 – множество заключительных состояний конечного автомата M_2, состоящее из таких состояний-множеств S, что в каждом из них имеется хотя бы одно заключительное состояние автомата M_1, то есть $F_2 = \{S : S \cap F_1 \neq \varnothing, S \in Q_2\}$; g_2 – функция переходов автомата M_2.

Пример 3.9. В результате детерминизации конечного автомата, заданного графом,



получим конечный автомат с меньшим количеством состояний.



Среди состояний нового конечного автомата может быть состояние, соответствующее $\varnothing$, причем $g_2(\varnothing, a) = \varnothing$ для любого символа a. Попав

в такое состояние, конечный автомат M_2 уже его не покинет. Состояние $\varnothing$ детерминированного конечного автомата M_2 является поглощающим.

Поглощающим состоянием конечного автомата называют такое его состояние q, что для всех входных символов a верно $g(q,a)=q$.

$g_2(S,a)=\varnothing$ тогда и только тогда, когда в старом конечном автомате M_1 для каждого состояния $q \in S$ верно $g_1(q,a)=\varnothing$, то есть в графе M_1 из каждого такого состояния q не выходит ни одна дуга, помеченная символом a.

Далее рассматривается метод построения детерминированного конечного автомата, называемый методом вытягивания.

Алгоритм метода вытягивания

В конечном автомате M_1 без λ-дуг необходимо найти множество всех состояний, достижимых из начального.

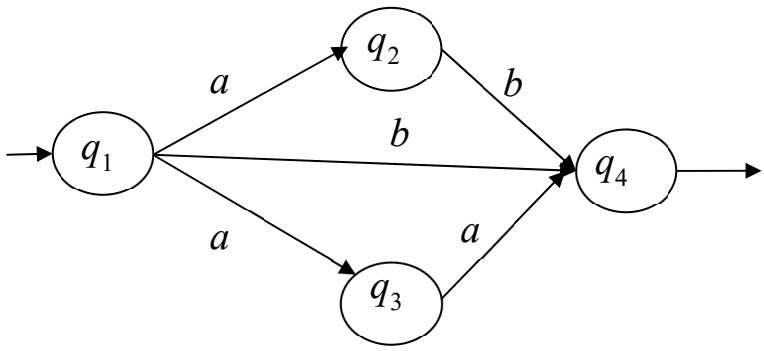
1. Для каждого символа входного алфавита a находим множество состояний, в которые автомат переходит из начального по дугам с метками a. Каждое такое множество $g_1(q_0,a)$ в новом автомате является состоянием-множеством, достижимым из начального за один такт.

$2-n$). Для каждого из определенных на предыдущем шаге состояний-множеств S и каждого символа входного алфавита a находим множество $\bigcup_{q\in S} g_1(q,a)$, $q \in S$. Все полученные на этом шаге состояния будут состояниями нового детерминированного автомата M_2, достижимыми из начальной вершины $\{q_0\}$ по пути длины 2 (за два такта).

Повторяем этот шаг алгоритма до тех пор, пока не перестанут появляться новые состояния-множества (включая пустое). В результате определятся все состояния нового детерминированного автомата M_2, достижимые из начальной вершины $\{q_0\}$ по пути длины n, $n>2$.

$n+1$). Формируем множество заключительных состояний, включая в него все найденные состояния-множества, содержащие в качестве элемента хотя бы одно из заключительных состояний конечного автомата M_1.

Пример 3.10. Построить детерминированный конечный автомат, эквивалентный заданному графом



Из условия задачи $M_1 = (V, Q_1, q_0, F_1, g_1)$ – конечный автомат с множеством состояний $Q_1 = \{q_1, q_2, q_3, q_4\}$, входным алфавитом $V = \{a, b\}$, входным состоянием q_1, множеством заключительных состояний $F_1 = \{q_4\}$, функцией переходов g_1, для которой $g_1(q_1, a) = \{q_2, q_3\}$, $g_1(q_1, b) = \{q_4\}$, $g_1(q_2, b) = \{q_4\}$, $g_1(q_3, a) = \{q_4\}$. Конечный автомат M_1 не содержит λ-переходов, поэтому сразу переходим к этапу 2, собственно, детерминизации.

Применим метод вытягивания. Процедуру вытягивания начнем с начального состояния q_1, из которого возможен переход по дуге с меткой a в состояния q_2 и q_3. В результате получим первое состояние-множество $\{q_2, q_3\}$. Кроме того, из состояния q_1 по дуге с меткой b можно перейти в состояние q_4, откуда получим второе состояние-множество $\{q_4\}$.

То есть

1. $g_2(q_1, a) = \{q_2, q_3\}, g_2(q_1, b) = \{q_4\}$.

На втором шаге рассмотрим состояние-множество $\{q_2, q_3\}$ и найдем функцию переходов из этого состояния.

Из состояния q_2 дуги с меткой a нет, поэтому $g_1(q_2, a) = \varnothing$, из состояния q_3 по дуге с меткой a возможен переход в состояние q_4. В результате $g_2(\{q_2, q_3\}, a) = \{q_4\} \cup \varnothing = \{q_4\}$. Аналогично по метке b из того

же состояния-множества $\{q_2, q_3\}$: $g_1(q_3, b) = \varnothing$, $g_1(q_2, b) = q_4$, в результате $g_2(\{q_2, q_3\}, b) = \{q_4\} \cup \varnothing = \{q_4\}$.

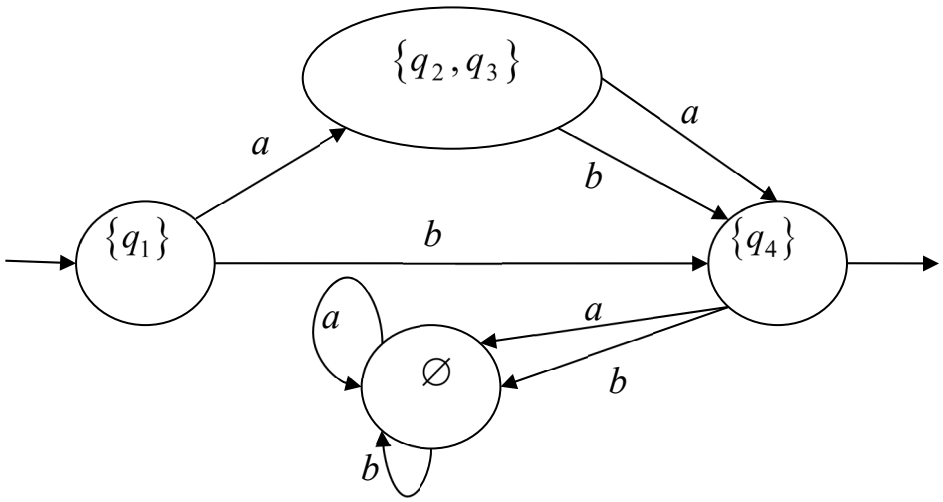
2. $g_2(\{q_2, q_3\}, a) = \{q_4\} \cup \varnothing = \{q_4\}$, $g_2(\{q_2, q_3\}, b) = \{q_4\} \cup \varnothing = \{q_4\}$.

В итоге множество заключительных состояний состоит из единственного состояния $\{q_4\}$. Продолжая процедуру вытягивания, убедимся, что по меткам a и b из состояния $\{q_4\}$ можно попасть только в поглощающее состояние $\varnothing$.

3. $g_2(\{q_4\}, a) = \varnothing$, $g_2(\{q_4\}, b) = \varnothing$.

4. $g_2(\varnothing, a) = \varnothing$, $g_2(\varnothing, b) = \varnothing$.

Все состояния-множества, достижимые из начального, выявлены; работа алгоритма на этом завершается. Результатом является детерминированный конечный автомат M_2, граф которого представлен на рисунке.



4. ТЕОРИЯ АЛГОРИТМОВ

4.1. Понятие алгоритма

Термин «алгоритм» происходит от имени среднеазиатского математика Аль-Хорезми (VIII–IX вв.), однако алгоритмы – инструкции для пошагового решения класса однотипных задач – были известны и раньше (например, алгоритм Евклида нахождения наибольшего общего делителя двух чисел).

Интуитивное определение алгоритма были сформулированы неоднократно, примерами чего могут служить определения А. Н. Колмогорова и А. А. Маркова [8].

Алгоритм – это всякая система вычислений, выполняемых по строго определенным правилам, которая после какого-либо числа шагов заведомо приводит к решению поставленной задачи (Колмогоров).

Алгоритм – это точное предписание, определяющее вычислительный процесс, идущий от варьируемых исходных данных к искомому результату (Марков).

Различные определения алгоритма предполагают выполнения нескольких требований (*свойств алгоритмов*):

1) дискретность – процесс решения разбит на простые шаги, выполнение каждого из которых требует конечного отрезка времени;

2) определенность (детерминированность) – результаты каждого шага определяются только исходными данными, полученными на предыдущем шаге, и не зависят от исполнителя;

3) результативность – процесс вычислений должен быть конечным и давать результат – решение задачи;

4) массовость – исходные данные могут выбираться из некоторого множества, называемого областью применимости алгоритма. Таким образом, посредством данного алгоритма можно решить большое число однотипных задач [7, 8].

Приведенные выше определения и свойства хорошо описывают уже известные алгоритмы решения конкретных задач, однако их недостаточно, когда речь идет о задаче, решение которой не найдено. В этом случае важным является вопрос о том, существует ли такое решение (вопрос алгоритмической разрешимости). Для того, чтобы доказать существование алгоритма, достаточно дать его описание, но для того, чтобы доказать его отсутствие, необходимо строгое математическое определение алгоритма.

Современная теория алгоритмов опирается на фундаментальные труды А. Тьюринга, Э. Поста и А. Чёрча, опубликованные в 1936 г., где предложены формальные модели представления алгоритма. Важным следствием этих работ стала формулировка алгоритмически неразрешимых проблем и доказательство их неразрешимости.

Таким образом, задачами теории алгоритмов являются:

1) формализация понятия алгоритма и исследование различных алгоритмических систем;
2) доказательство алгоритмической неразрешимости ряда задач;
3) классификация задач, определение и исследование классов сложности;
4) асимптотический анализ сложности алгоритмов;
5) получение явных функций трудоемкости алгоритмов и разработка критериев сравнительной оценки их качества.

Существует несколько подходов к решению поставленных задач. Э. Постом и А. Тьюрингом независимо был предложен подход, согласно которому алгоритм определяется как набор инструкций, осуществляемый машиной (машина Поста и машина Тьюринга). Второй подход, предложенный Чёрчем, связан с понятием вычислимой функции. Впоследствии была доказана эквивалентность этих подходов.

4.2. Машина Поста

Машина Поста – это абстрактная вычислительная машина, созданная для формализации понятия алгоритма. Она представляет собой универсальный исполнитель, позволяющий вводить исходные данные и читать результат выполнения программы, а также обладает способностью определять, является ли та или иная задача алгоритмически разрешимой.

Машина Поста состоит из бесконечной ленты, поделенной на одинаковые ячейки, которые могут быть пустыми (0 или пустота) или содержать метку (1 или любой другой знак), и считывающей головки (каретки), способной перемещаться по ленте на одну ячейку в ту или иную сторону, а также проверять наличие метки, стирать существующую метку, записывать метку в пустую ячейку и завершить работу (рис. 4.3).

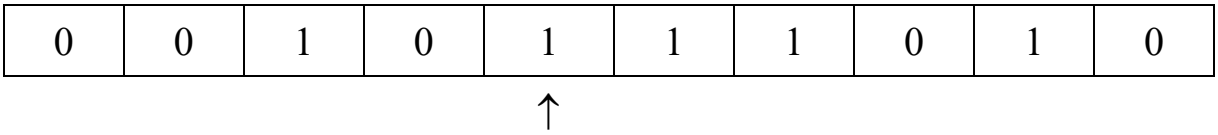


Рис. 4.3. Схематическое представление машины Поста

Исходные данные определяются состоянием ленты и положением каретки на момент начала работы. Кареткой управляет программа, состоящая из пронумерованных строк команд, каждая из которых имеет вид iKj, где i – номер команды, K – действие каретки, j – номер следующей команды (отсылка).

Существует шесть типов команд:

1) $i\,\mathbf{V}j$ – поставить метку в пустую ячейку, перейти к j-й строке;

2) $i\,\mathbf{X}j$ – стереть метку в помеченной ячейке, перейти к j-й строке;

3) $i \leftarrow j$ – сдвинуться влево, перейти к j-й строке программы;

4) $i \rightarrow j$ – сдвинуться вправо, перейти к j-й строке программы;

5) $i\,?\,j_1; j_2$ – если в ячейке нет метки, то перейти к строке j_1 программы, иначе перейти к строке j_2;

6) $i\,!$ – конец программы (отсылки нет).

При работе с целыми неотрицательными числами принято считать, что машина Поста работает в единичной системе счисления, то есть ноль представляется одной помеченной ячейкой, а целое положительное число – помеченными ячейками в количестве на единицу больше его значения ($0:010$; $1:0110$; $2:01110$; $3:011110$ и т. д.).

Пример 4.1. Составить программу, вычисляющую число $n+1$ на машине Поста.

Пусть на момент начала работы число n представлено $n+1$ единицами, записанными на ленте, а каретка находится напротив крайней левой из единиц так, как изображено на рисунке.

По окончании работы программы лента должна содержать $n+2$ единицы, а каретка может располагаться произвольно. Тогда программа может иметь вид:

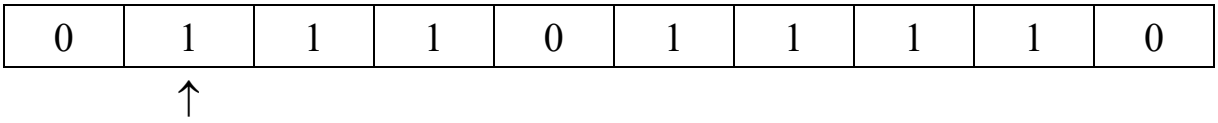
$1 \leftarrow 2$

$2\,\mathbf{V}3$

$3\,!$

Пример 4.2. Составить программу, вычисляющую сумму чисел.

Пусть числа n и m представлены, соответственно, $n+1$ и $m+1$ единицами, записанными на ленте, а каретка находится напротив крайней левой из единиц первого из чисел так, как изображено на рисунке.



Общее число единиц на ленте равно $n+m+2$. Число $n+m$ должно содержать $n+m+1$ единиц. Таким образом, сумма может быть вычислена с помощью следующей программы:

$1\,\mathbf{X}2$	$4\,\mathbf{X}5$	$7\,\mathbf{V}8$
$2 \rightarrow 3$	$5 \rightarrow 6$	$8\,!$
$3\ ?\ 8;4$	$6\ ?\ 7;5$	

Заметим, что команда 3 реализуется только если первое из чисел равно 0 (то есть задается одной единицей).

При запуске программы на машине Поста реализуется один из трех вариантов:

1) машина дойдет до команды «конец программы», в этом случае программа считается выполненной – результативная остановка;

2) в ходе работы машина дойдет до невыполнимой команды (стирание несуществующей метки или запись в помеченную ячейку) – безрезультатная остановка;

3) работа машины никогда не закончится.

Пусть задача представляет собой класс однотипных проблем с различными исходными данными из некоторого множества. Если при заданных исходных данных работа машины Поста заканчивается результативной остановкой, то говорят, что набор инструкций применим к этим данным; в противном случае набор инструкций называется неприменимым к исходным данным.

Предположение (тезис) Поста состоит в том, что алгоритм решения задачи существует тогда и только тогда, когда можно составить набор инструкций (программу) для машины Поста, применимый и дающий правильное решение задачи для тех исходных данных, для которых оно существует (результативная остановка) и неприменимый при тех исходных данных, когда решения нет. Фактически такая формулировка предполагает эквивалентность интуитивного понятия алгоритма и определения, данного Постом [9].

Очевидно, что из разрешимости задачи на машине Поста следует алгоритмическая разрешимость, так как программа для машины Поста является алгоритмом ее решения. Обратное утверждение о том, что из алгоритмической разрешимости задачи следует разрешимость на машине Поста, является гипотезой.

На современном этапе развития математики не представляется возможным доказать гипотезу Поста, поскольку интуитивное понятие алгоритма не является строго математическим, однако эквивалентность понятий алгоритмической разрешимости и разрешимости на машине Поста подтверждается практикой. При этом доказано, что определение разрешимости задачи в терминах Поста эквивалентно определениям, данным Тьюрингом и Чёрчем.

Рассматриваемая в следующем пункте машина Тьюринга имеет более сложный набор команд, отчего количество команд, необходимых для решения различных задач, как правило, оказывается меньше, чем для машины Поста.

4.3. Машина Тьюринга

Машина Тьюринга – абстрактное устройство, состоящее из бесконечной в обе стороны ленты, разбитой на ячейки, считывающей и печатающей головки (каретки), способной перемещаться вправо и влево, и блока управления. Каретка перемещается вдоль ленты так, что в каждый момент времени она указывает ровно на одну ячейку.

В ячейках могут быть записаны символы некоторого конечного алфавита

$$A = \{a_0, a_1, ..., a_k\},$$

в числе которых имеется особый пустой символ, заполняющий все клетки ленты, кроме тех из них, на которых записаны данные.

Устройство может находиться в одном из состояний из множества

$$Q = \{q_0, q_1, ..., q_n\}.$$

Программа для машины Тьюринга представляет собой конечный список команд вида

$$q_i a_i \to q_j a_j D,$$

где $a_i, a_j \in A$, $q_i, q_j \in Q$, $D \in \{R, L, S\}$ – движение каретки; R, L, S – «вправо», «влево» и «остаться» соответственно.

Команда читается следующим образом: машина, находящаяся в состоянии q_i, считывает с ленты символ a_i, после чего переходит в состояние q_j, печатает в текущей ячейке символ a_j, затем выполняет одно из движений D, а именно смещается на одну ячейку вправо, если $D = R$, на одну ячейку влево, если $D = L$, остается на той же ячейке, если $D = S$.

В программе не должно быть команд с одинаковыми входами вида

$$q_i a_i \to q_{j1} a_{j1} D \quad \text{и} \quad q_i a_i \to q_{j2} a_{j2} D$$

– это противоречит однозначности алгоритма.

Начальным состоянием машины Тьюринга считается q_1, заключительным – q_0. Перейдя в состояние q_0, машина останавливается.

Машина называется применимой к некоторым входным данным, записанным на ленте, если при работе над этими данными после конечного числа шагов она приходит в заключительное состояние q_0.

В противном случае машина называется неприменимой к этим входным данным.

Пример 4.3. Найти результат применения машины Тьюринга, заданной программой:

$$q_1 0 \to q_1 0R, \qquad q_1 1 \to q_2 0R, \qquad q_2 0 \to q_0 1S, \qquad q_2 1 \to q_1 0R.$$

к входным данным $P_1 = 0\,1\,1\,1\,0\,1\,0$ и $P_2 = 0\,1\,1\,1\,1\,0$.

Решение. Применяя команды к данным P_1 и P_2, имеем:

$$P_1 : 0\,\underset{q_1}{\underset{\sqcup}{1}}\,1\,1\,0\,1\,0 \to 0\,0\,\underset{q_2}{\underset{\sqcup}{1}}\,1\,0\,1\,0 \to 0\,0\,0\,\underset{q_1}{\underset{\sqcup}{1}}\,0\,1\,0 \to$$

$$\to 0\,0\,0\,0\,\underset{q_2}{\underset{\sqcup}{0}}\,1\,0 \to 0\,0\,0\,0\,\underset{q_0}{\underset{\sqcup}{1}}\,1\,0;$$

$$P_2 : 0\,\underset{q_1}{\underset{\sqcup}{1}}\,1\,1\,1\,0\,0 \to 0\,0\,\underset{q_2}{\underset{\sqcup}{1}}\,1\,1\,0\,0 \to 0\,0\,0\,\underset{q_1}{\underset{\sqcup}{1}}\,1\,0\,0 \to 0\,0\,0\,0\,\underset{q_2}{\underset{\sqcup}{1}}\,0\,0 \to$$

$$\to 0\,0\,0\,0\,0\,\underset{q_1}{\underset{\sqcup}{0}}\,0 \to 0\,0\,0\,0\,0\,0\,\underset{q_1}{\underset{\sqcup}{0}} \to 0\,0\,0\,0\,0\,0\,0\,\underset{q_1}{\underset{\sqcup}{0}} \to \ldots$$

Во втором случае машина никогда не остановится и будет работать бесконечно. Таким образом, заданная машина Тьюринга применима к набору данных P_1 и неприменима к P_2.

Как и машина Поста, машина Тьюринга может работать с целыми неотрицательными числами в единичной системе счисления. В этом случае

алфавит A состоит из символов 0 и 1, где 0 играет роль пустого символа. Число n здесь записывается в виде последовательности из $n+1$ единицы, то есть

$$0:010;\ 1:0110;\ 2:01110 \text{ и т. д.},$$

а кортеж $(n_1, n_2, ..., n_k)$ как

$$...0\underbrace{11\ldots1}_{n_1+1}\ 0\underbrace{11\ldots1}_{n_2+1}\ 0...0\underbrace{11\ldots1}_{n_k+1}\ 0...$$

Пример 4.4. Составить программу для машины Тьюринга, вычисляющую число $n+1$. Пусть каретка изначально указывает на крайнюю правую единицу, а в ходе работы программы дойдет до конца массива из единиц, заменит первый из нулей на единицу и вернется назад, то есть

$$0\underbrace{11\ldots1}_{q_1\ n_1+1}\ 0 \to 0\ \underbrace{11\ldots1}_{q_0\ n_1+2}\ 0.$$

Программа записывается так:

$$q_1 1 \to q_1 1R, \qquad q_1 0 \to q_2 1L, \qquad q_2 1 \to q_2 1L, \qquad q_2 0 \to q_0 0R.$$

В качестве упражнения рекомендуется реализовать на машине Тьюринга алгоритм, использованный для решения этой же задачи в примере 4.1 [5].

Универсальная кодировка машины Тьюринга

Построим универсальную кодировку программы машины Тьюринга M. Занумеруем движения $D \in \{R, L, S\}$, алфавит $\{a_0, a_1, ..., a_k\}$ и состояния машины $\{q_0, q_1, ..., q_n\}$ натуральными числами:

R	L	S	a_0	a_1	…	a_k	q_0	q_1	…	q_n
1	3	5	7	9	…	$2k+7$	0	2	…	$2n$

Теперь любую команду можно закодировать набором из пяти чисел и представить его на ленте с помощью алфавита $\{1,*\}$ стандартным образом, помещая вместо пустого символа символ *. Обозначив код команды через K, можно составить код всей программы

$$K(M) = K_1 * K_2 * ... * K_p,$$

где K_i – коды всех команд программы.

Пример 4.5. Построить машину Тьюринга с программой:

$$q_1 1 \to q_1 1R, \quad q_1 0 \to q_2 1L, \quad q_2 1 \to q_2 1L, \quad q_2 0 \to q_0 0R.$$

Решение. Закодируем каждую команду, используя таблицу кодов:

2*9*2*9*1, 2*7*4*9*3, 4*9*4*9*3, 4*7*0*7*1.

Теперь представим коды команд с помощью алфавита $\{1,*\}$:

$$1^3 * 1^{10} * 1^3 * 1^{10} * 1^2 * 1^3 * 1^8 * 1^5 * 1^{10} * 1^4 * 1^5 * 1^{10} * 1^5 * 1^{10} * 1^4 * 1^5 * 1^8 * 1 * 1^8 * 1^2,$$

где 1^k есть единица, повторенная *k* раз.

4.4. Рекурсивные функции. Тезис Чёрча

Все известные примеры алгоритмов сводятся к вопросу вычисления значений некоторой функции. В свою очередь, функция называется *вычислимой*, если существует алгоритм, вычисляющий ее значения.

Далее рассматриваются функции, заданные на множестве целых неотрицательных чисел $f: Z_+^n \to Z_+$ где $Z_+ = \{0,1,2,...\}$.

Частичными числовыми функциями $f(x_1, x_2, ..., x_n),\ x_i \in Z_+\ (1 \le i \le n)$ называются функции, определенные не на всех кортежах $(x_1, x_2, ..., x_n) \in Z_+^n$.

Простейшими называются следующие всюду определенные функции:

1) $o(x) = 0$ – нулевая функция;

2) $s(x) = x + 1$ – функция следования;

3) $I_n^m(x_1, ..., x_n) = x_m, 1 \le m \le n,$ – функция выбора аргумента.

Каждая из простейших функций может быть вычислена на машине Тьюринга:

1) $o(x)=0$:

$$q_1 1 \to q_1 1R, \quad q_1 0 \to q_2 0L, \quad q_2 1 \to q_2 0L, \quad q_2 0 \to q_0 1S.$$

2) $s(x)=x+1$:

$$q_1 1 \to q_1 1R, \quad q_1 0 \to q_2 1L, \quad q_2 1 \to q_2 1L, \quad q_2 0 \to q_0 0R.$$

3) $f(x,y)=y$:

$$0\underbrace{11...1}_{q_1\ x+1}0\underbrace{11...1}_{y+1}0 \to 0\underbrace{11...1}_{q_0\ y+1}0$$

$$q_1 1 \to q_1 0R, \qquad q_1 0 \to q_0 0R.$$

В качестве упражнения рекомендуется построить машину Тьюринга для функции $f(x,y)=x$ [5].

Из простейших функций можно построить новые функции с помощью перечисленных ниже операций суперпозиции, примитивной рекурсии и минимизации [3].

Пусть даны функция $f(y_1,...,y_m)$ от m аргументов и функции $g_1(x_1,...,x_n)$, …, $g_m(x_1,...,x_n)$ от n аргументов. Говорят, что функция

$$h(x_1,...,x_n)=f\left(g_1(x_1,...,x_n),...,g_m(x_1,...,x_n)\right)$$

получается из f и $g_1,...,g_m$ с помощью операции *суперпозиции*.

Пример 4.6. 1) $s(o(x))=1$; 2) $s(s(x))=(x+1)+1=x+2.$

Функция $h(x_1,...,x_n)$ получается из функций $f(x_1,...,x_{n-1})$ и $g(x_1,...,x_{n+1})$ с помощью *операции примитивной рекурсии*, если

$$h(x_1,...,x_{n-1},0)=f(x_1,...,x_{n-1});$$

$$h(x_1,...,x_{n-1},x_n+1)=g\left(x_1,...,x_{n-1},x_n,h(x_1,...,x_{n-1},x_n)\right).$$

Операция минимизации по *i*-й переменной функции $f(x_1, x_2,..., x_n)$ обозначается $h(x_1,..., x_n) = \mu_y \left(f(x_1, x_2,..., x_{i-1}, y, x_{i+1},..., x_n) = x_i \right)$ и дает минимальное значение x_i, при котором $f(x_1, x_2,..., x_n)$ равна 0.

Значения $\mu_y(f)$ определяются следующим образом.

Пусть $(x_1, x_2,..., x_n)$ – произвольный кортеж чисел из множества $Z_+ = \{0,1,2,...\}$.

Пусть соотношение

$$f(x_1, x_2,..., x_{i-1}, y, x_{i+1},..., x_n) = x_i, \qquad (*)$$

рассматривается как уравнение относительно y, которое решается подбором. Для этого вместо *y* подставляются последовательно числа 0, 1, 2, …

Тогда возможны следующие случаи.

1. При некоторых y левая часть соотношения (*) не определена. В этом случае считается, что для кортежа $(x_1, x_2,..., x_n)$ значение функции не определено.

2. При всех y левая часть соотношения (*) определена, но ни при каких y равенство не выполняется. В этом случае также считается, что для кортежа $(x_1, x_2,..., x_n)$ значение функции не определено.

3. Левая часть соотношения (*) определена при $y = 0$, $y = 1$, …, $y = y_0 - 1$, $y = y_0$, но при $y < y_0$ равенство (*) не выполнялось, а при $y = y_0$ оно выполняется. В этом случае полагаем $g(x_1, x_2,..., x_n) = y_0$, то есть число y_0 считается значением операции минимизации для кортежа $(x_1, x_2,..., x_n)$.

Функции, которые могут быть получены из простейших функций с помощью конечного числа операций суперпозиции, примитивной рекурсии и минимизации называются *частично рекурсивными.*

Общерекурсивные функции – это подмножество частично рекурсивных функций, определенных для всех значений аргументов.

Пример 4.7. Функция $g(x)=\begin{cases} x-1, \text{ при } x>0, \\ \text{не определено при } x=0 \end{cases}$ частично рекурсивна.

Действительно, $g(x)=\mu_y\{s(y)=y+1=x\}$. Эта функция получена из простейшей с помощью операции минимизации.

Для любой частично рекурсивной функции, можно указать алгоритм вычисления ее значений, то есть все частично рекурсивные функции являются вычислимыми. Обратное утверждение является гипотезой, которая выглядит убедительно, но не может быть строго доказана.

Тезис Чёрча. Всякая вычислимая частичная числовая функция является частично рекурсивной функцией.

Тезис Тьюринга. Всякий алгоритм представим в форме машины Тьюринга.

Согласно этому тезису, всякая вычислимая в интуитивном смысле функция вычислима с помощью некоторой машины Тьюринга.

Тезисы Поста, Тьюринга и Чёрча эквивалентны, то есть определяют одно и то же множество задач, являющихся алгоритмически разрешимыми. Существуют и другие способы формализации понятия алгоритма – λ-исчисление Чёрча, нормальные алгоритмы Маркова и т. д. [3]

4.5. Основные меры сложности вычислений

Сложность алгоритма − количественная характеристика потребляемых ресурсов, необходимых программе или алгоритму для работы (успешного решения задачи). Основными ресурсами являются время (временная сложность) и объем памяти (емкостная сложность). Наиболее важной (критической) характеристикой является время.

Временная сложность алгоритма (в худшем случае) – это функция размера входных и выходных данных, равная максимальному количеству элементарных операций, проделываемых алгоритмом для решения экземпляра задачи указанного размера. В задачах, где размер выхода не превосходит или пропорционален размеру входа, можно рассматривать временную сложность как функцию размера только входных данных.

Аналогично понятию временной сложности в худшем случае определяется понятие «временная сложность алгоритма» в наилучшем случае. Также рассматривают понятие «среднее время работы алгоритма», т. е. математическое ожидание времени работы алгоритма. Иногда говорят просто: «временная сложность алгоритма» или «время работы алгоритма», имея в виду временную сложность алгоритма в худшем, наилучшем или среднем случае (в зависимости от контекста) [7, 8].

Емкостная сложность алгоритма определяется числом ячеек памяти, используемых в процессе его исполнения. Эта величина не может превосходить числа действий n, умноженного на определенную константу (число ячеек, используемых в данной модели на одном шаге). В свою очередь, число шагов может сильно превосходить объем памяти (за счет циклов по одним и тем же ячейкам). Следует отметить, что проблемы памяти технически преодолеваются легче, чем проблемы быстродействия, которое имеет физический предел – скорость распространения физических сигналов. Поэтому трудоемкость (временная сложность) считается более существенной характеристикой алгоритма.

Нижняя оценка временной сложности алгоритма получается, если проанализировать лучший случай исходных данных алгоритма – выполнение алгоритма осуществляется при минимальном количестве операций. *Верхняя оценка* сложности алгоритма характеризует ситуацию максимального количества операций обработки (худший случай).

Для многих алгоритмов отмечается ситуация «редкости» крайних значений: только на относительно небольшом количестве сочетаний исходных данных реализуются близкие к верхним или нижним оценкам значения сложности. Поэтому интересно бывает отыскать некоторое «усредненное» по всем данным число операций (получить среднюю оценку временной сложности).

Различные алгоритмы имеют различную временную сложность и выяснение того, какие из них окажутся достаточно эффективными, а какие нет, определяется многими факторами. Речь идет о различии между полиномиальными и экспоненциальными алгоритмами.

Полиномиальным называется алгоритм, временная сложность которого выражается некоторой полиномиальной функцией «размера» задачи n. «Размер» задачи – это объем входных данных, необходимых для ее решения.

Экспоненциальным называется алгоритм, временная сложность которого не может быть выражена некоторой полиномиальной функцией размера задачи n.

Различие между указанными двумя типами алгоритмов становятся особенно заметными при решении задач большого размера. В связи с этим полиномиальные алгоритмы называют *эффективными*, а полиномиально решаемые задачи – *легкорешаемыми*.

Класс сложности – это множество задач распознавания, для решения которых существуют алгоритмы, схожие по вычислительной сложности.

Задача называется *полиномиальной*, т. е. относится к классу P, если существует константа k и алгоритм, решающий эту задачу за время $O\left(n^k\right)$.

Преимущества задач класса P: для большинства реальных задач из класса P константа k меньше 6; класс P обладает свойством естественной замкнутости (можно комбинировать полиномиальные алгоритмы, используя один в качестве «подпрограммы» другого и при этом результирующий алгоритм будет полиномиальным).

Класс EXP – это класс всех экспоненциальных задач.

Задача относится к классу NP, если ее решение, полученное некоторым алгоритмом, может быть полиномиально проверено. Очевидно, что любая задача, принадлежащая классу P, принадлежит и классу NP, так как она может быть полиномиально проверена, при этом задача проверки решения может состоять просто в повторном решении задачи.

Задача называется NP-*полной* (принадлежит классу NP-полных задач, то есть классу NPC), если выполнены два условия:

1) задача принадлежит классу NP;

2) к этой задаче полиномиально сводятся все задачи из класса NP.

Понятие NP-полноты основывается на понятии сводимости одной задачи к другой.

Задача, для которой не удается построить полиномиальный алгоритм, считается *труднорешаемой*. Хотя это утверждение не является категорическим, поскольку известны задачи, в которых достаточно эффективно работают и экспоненциальные алгоритмы.

ЗАКЛЮЧЕНИЕ

Математическая логика и теория алгоритмов применяются при решении задач на компьютере в терминах аппаратных средств и программного обеспечения с привлечением организации символов и манипуляции данными. Без их освоения невозможно понимание сути работы современных цифровых компьютеров. Таким образом, основная цель изучения математической логики и теории алгоритмов – овладение инструментами и технологиями, необходимыми для понимания, проектирования и защиты информационных систем.

Навыки и умения, приобретаемые обучающимися в результате изучения дисциплины «Математическая логика и теория алгоритмов», необходимы для дальнейшего формирования профессиональных компетенций при освоении таких специализированных дисциплин, как «Проектирование и техническое сопровождение компьютерных сетей», «Моделирование процессов и систем защиты информации», «Методы и средства криптографической защиты информации» и др.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Агарева О. Ю., Селиванов Ю. В. Математическая логика и теория алгоритмов : учеб. пособие. – М. : МАТИ, 2011. – 80 с.

2. Дискретная математика : учеб. пособие / В. Л. Неклюдова, О. В. Григоренко, О. Г. Павловская, В. П. Вербная. – Новосибирск : СГУГиТ, 2020. – 109 с.

3. Марченков С. С. Избранные главы дискретной математики : учеб. пособие. – М. : Издательский отдел факультета ВМиК МГУ им. М. В. Ломоносова (лицензия ИД $У^5$ 05899 от 24.09.2001); МАКС Пресс, 2015. – 134 с.

4. Новиков Ф. А. Дискретная математика для программистов : учебник для вузов. – СПб. : Питер, 2009. – 384 с.

5. Игошин В. И. Сборник задач по математической логике и теории алгоритмов: учеб. пособие. – М. : КУРС: ИНФРА-М, 2017. – 392 с.

6. Судоплатов С. В., Овчинникова Е. В. Дискретная математика : учебник. – Новосибирск : НГТУ, 2010. – 256 с.

7. Игошин В. И. Теория алгоритмов : учеб. пособие. – М. : ИНФРА-М, 2012. – 318 с.

8. Теория алгоритмов : учеб. пособие / сост. А. А. Брыкалова. – Ставрополь : Изд-во СКФУ, 2016. – 129 с.

9. Успенский В. А. Машина Поста / Гл. ред. физ.-мат. лит. – 2-е изд., испр. – М. : Наука, 1988. – 96 с.

Учебное издание

Неклюдова Вера Леонидовна

Вербная Валентина Павловна

МАТЕМАТИЧЕСКАЯ ЛОГИКА И ТЕОРИЯ АЛГОРИТМОВ

Редактор *Ю. С. Мерзликина*

Компьютерная верстка *Н. Ю. Леоновой*

Изд. лиц. ЛР № 020461 от 04.03.1997.
Подписано в печать 21.03.2022. Формат 60 × 84 1/16.
Усл. печ. л. 4,07. Тираж 85 экз. Заказ 31.
Гигиеническое заключение
№ 54.НК.05.953.П.000147.12.02. от 10.12.2002.

Редакционно-издательский отдел СГУГиТ
630108, Новосибирск, ул. Плахотного, 10.

Отпечатано в картопечатной лаборатории СГУГиТ
630108, Новосибирск, ул. Плахотного, 8.