Федеральное агентство по образованию и науке
Сибирский государственный аэрокосмический университет
имени академика М. Ф. Решетнева

А.В. Саяпин, Т. А. Сливина

ДИСКРЕТНАЯ МАТЕМАТИКА

Допущено учебно-методическим объединением вузов по университетскому политехническому образованию в качестве учебного пособия для студентов высших технических заведений, обучающихся по специальности 230100 - «Информатика и вычислительная техника»

Красноярск 2010

УДК 519
ББК 22.174
С 47

Рецензенты:

Зав. кафедрой
Информационных систем
Института космических и информационных технологий
Сибирского федерального Университета
д.ф.-м.н., проф. Б. С. Добронец
Ведущий научный сотрудник
Института вычислительного моделирования СО РАН
д.ф.-м.н., проф. В. И. Сенашов

Саяпин А.В., Сливина, Т. А.
С 47 Дискретная математика: Учебное пособие / А.В. Саяпин, Т. А. Сливина; СибГАУ.-Красноярск, 2010. – 151 с.

В учебном пособии рассматриваются вопросы, связанные с теорией множеств, математической логикой, теорией графов и комбинаторикой, а так же теорией формальных языков и конечных автоматов. Приведено большое количество примеров, а также вопросов и упражнений для закрепления материала. Предлагаемый материал полностью соответствует учебной программе. Учебное пособие предназначено для студентов специальности 230100 - «Информатика и вычислительная техника» всех форм обучения.

УДК 519
ББК 22.174

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Настоящая дисциплина преследует следующие цели:

— познакомить студентов с максимально широким кругом понятий дискретной математики, сформировать у студентов терминологический запас, необходимый для самостоятельного изучения специальной математической и теоретико-программистской литературы;

— сообщить студентам необходимые сведения из дискретной математики, предусматриваемые стандартной программой. Разбор доказательств приведенных утверждений и выполнение упражнений позволят студенту овладеть методами дискретной математики, используемыми чаще других при решении практических задач;

— сформировать навыки мышления, позволяющие решать профессиональные задачи в том числе и в области защиты информации.

Дискретная математика – бурно развивающаяся ветвь математики. Ее роль и место определяются в основном тремя факторами:

— дискретную математику можно рассматривать как теоретическую основу компьютерной математики и защиты информации;

— модели и методы дискретной математики являются хорошим средством и языком для построения и анализа моделей в различных науках.

Математика от рождения делится на дискретную и континуальную математику. Что мы относим к континуальной математике? Все, что явно или неявно содержит идеи теории пределов и непрерывности. Все остальное – дискретная математика. Дискретная математика включает в себя такие разделы как арифметика, алгебра, теория множеств, математическая логика, комбинаторный анализ, теория графов, теория алгоритмов, теория автоматов. Некоторые из перечисленных разделов, такие как теория алгоритмов и теория автоматов, студенты изучают более глубоко в специальных курсах.

Глава 1

ЭЛЕМЕНТЫ ТЕОРИИ МНОЖЕСТВ

В этой главе рассматриваются начальные понятия: множества, отношения и функции. С помощью этих понятий строится, по существу, любая математическая дисциплина.

Любое понятие дискретной математики можно определить с помощью понятия множества.

Множество – это совокупность определенных различаемых объектов таких, что для любого объекта можно установить, принадлежит этот объект данному множеству или нет.

Обозначать будем множества заглавными латинскими буквами A, B, C, ..., а их элементы, соответственно, маленькими латинскими буквами a, b, c, a_1, a_2, ... Множество A считается заданным, если указано характеристическое свойство элементов этого множества, т. е. такое свойство, которым обладают все элементы этого множества и только они. Эта формулировка позволяет рассматривать любые множества с элементами различной природы, например:

— множество студентов на лекции (конечное),

— множество натуральных чисел N (бесконечное),

— множество программ, находящихся в памяти ЭВМ в данный момент времени (конечное),

— множество операций в какой-то программе (конечное),

— множество точек на плоскости (бесконечное).

Однако все эти множества являются достаточно сложными для исследования и изучения, поэтому для большинства примеров мы будем рассматривать некоторые абстрактные множества – числовые.

Множества можно задавать двумя способами – перечислением всех его элементов и с помощью описания свойств, которыми обладает каждый элемент этого множества.

Примеры записи множества

1) $A = \{7, 8, 9\}$ – множество A, содержащее числа 7, 8, и 9, которые являются элементами данного множества;

2. $A = \{x: x$ – целое число и $6 < x < 10\}$ – элементами множества являются числа 7, 8, 9, т. е. целые числа, удовлетворяющие заданному неравенству.

Множества рассматривают как неупорядоченную совокупность элементов, т. е. в нашем примере $A = \{7, 8, 9\} = \{9, 8, 7\} = \{8, 7, 9\}$. Для любого заданного объекта можно определить принадлежит он данному множеству или нет, например: $x = 7$, $7 \in A$ (7 принадлежит A) или $x = 6$, $6 \notin A$ (6 не принадлежит A).

Элементами множества могут быть другие множества, например: $A = \{\{1\}\}$ или $B = \{\{1\}, \{2\}\}$.

В следующих параграфах рассмотрим основные понятия, действия над множествами, отношения на множествах, функции и отображения, алгебраические операции на множествах и их свойства, задачи и упражнения.

§ 1. Основные понятия

Множества называются *равными*, если они состоят из одних и тех же элементов.

Являются ли равными множества $A = \{1, 2\}$ и $B = \{\{1\}, \{2\}\}$? Нет, так как элементами множества A являются числа 1 и 2, а элементами множества B являются множества $\{1\}$, $\{2\}$.

Множество A называется *подмножеством* множества B, если A и B таковы, что из того, что $x \in A$ следует, что $x \in B$. Обозначают это так $A \subseteq B$ (A включается в B).

Если существует элемент B, который не принадлежит A, то A называют *собственным подмножеством* B и записывают в виде $A \subset B$, что означает в некотором смысле, что B больше, чем A.

Свойства включения:

1. $A \subseteq A$.
2. Если $A \subseteq B$ и $B \subseteq C$, то $A \subseteq C$.
3. Если $A \subseteq B$ и $B \subseteq A$, то $A = B$. В этом случае множества A и B называются *эквивалентными*.

Множество, не содержащее элементов, называется *пустым* и обозначается пустое множество символом $\emptyset$.

Пустое множество есть подмножество любого множества. Каждое непустое множество $A \neq \emptyset$, имеет по крайней мере два различных подмножества – само A и $\emptyset$.

Множество всех подмножеств множества A называется *степенью* множества A и обозначается $P(A)$:

$$P(A) = 2^n,$$

где n – количество элементов множества A.

Пример.

Задано множество $A = \{1, 2, 3\}$. Определить степень множества A.

$$P(A) = 2^3 = 8, \quad P(A) = \{0, A, \{1\}, \{2\}, \{3\}, \{1, 2\}\ \{1, 3\}\ \{2, 3\}\}.$$

Мощностью множества A называется число $|A| = \text{card}(A) = n$, где n – количество элементов множества.

Множество A называется *конечным*, если между элементами множества A и элементами множества $\{1, 2, 3, \ldots, n\}$ можно установить взаимно однозначное соответствие.

Если множество $A \neq \varnothing$ и никакого n не может быть найдено, то множество A называется *бесконечным*.

Множество всех рассматриваемых в данной задаче элементов называется *универсальным* и обозначается U.

§ 2. Диаграммы Эйлера–Венна. Операции над множествами. Прямое произведение множеств

Рассмотрим простейшие операции над множествами: объединение, пересечение, разность (дополнение) и симметрическая разность. Множества и операции над множествами можно изображать при помощи диаграмм Эйлера–Венна. Диаграммы Эйлера–Венна дают некоторые геометрические представления о множествах и представляют собой схематическое изображение множеств в виде замкнутых геометрических фигур.

Пересечением множеств A и B называется множество всех элементов, принадлежащих A и B одновременно. Обозначается $A \cap B$, а описание имеет вид $A \cap B = \{x: x \in A \text{ и } x \in B\}$.

Геометрическое изображение пересечения множеств A и B представлено на рис. 1.

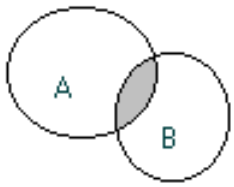


Рис. 1

Объединением множеств A и B называется множество всех элементов, принадлежащих A или B. Обозначается $A \cup B$, а описание имеет вид

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

Геометрическое изображение объединения множеств A и B представлено на рис. 2.

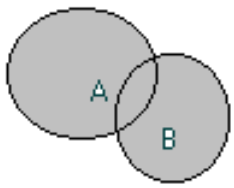


Рис. 2

Пример.

1. $\{1, 2\} \cup \{2, 3\} = \{1, 2, 3\}$.
2. $\{1, 2\} \cap \{2, 3\} = \{2\}$.

Если $A \cap B = \varnothing$, то множества A и B не пересекаются, т. е. не имеют общих элементов.

Разностью A и B (*дополнением* B до A) называется множество, которое содержит элементы множества A и не содержит элементов множества B. Обозначается $A \backslash B$, а описание имеет вид $A \backslash B = \{x: x \in A \text{ и } x \notin B\}$.

Геометрическое изображение разности множеств A и B представлено на рис. 3.

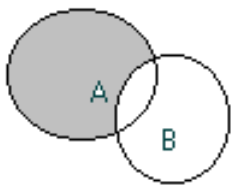


Рис. 3

Симметрической разностью множеств A и B называется множество, которое содержит элементы множества $A \cup B$ и не содержит элементы множества $A \cap B$. Обозначается $A \Delta B$, а описание имеет вид

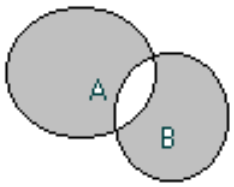


Рис. 4

$$A \Delta B = \{x: x \in A \cup B \text{ и } x \notin A \cap B\}$$

или

$$A \Delta B = A \cup B \backslash A \cap B.$$

Геометрическое изображение симметрической разности множеств A и B представлено на рис. 4.

Если задано универсальное множество U, то *дополнением* множества A до универсального называется множество $\overline{A} = U \backslash A = \{x: x \notin A\}$.

Рис. 5

Геометрическое изображение дополнения множества A до универсального множества U представлено на рис. 5.

В этом параграфе рассмотрели основные действия над множествами.

§ 3. Основные тождества алгебры множеств

Коммутативные законы:

$$A \cup B = B \cup A, \quad A \cap B = B \cap A.$$

Ассоциативные законы:

$$A \cup (B \cup C) = (A \cup B) \cup C, \quad A \cap (B \cap C) = (A \cap B) \cap C.$$

Дистрибутивные законы:

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C), \quad A \cap (B \cup C) = (A \cap B) \cup (A \cap C).$$

Законы де Моргана:

$$\overline{A \cup B} = \overline{A} \cap \overline{B}, \quad \overline{A \cap B} = \overline{A} \cup \overline{B}.$$

Законы идемпотентности:

$$A \cup A = A, \quad A \cap A = A.$$

Законы Ø и U:

$$A \cup U = U, \quad A \cap \varnothing = \varnothing;$$

$$A \cup \varnothing = A, \quad A \cap U = A;$$

$$A \cup \overline{A} = U, \quad A \cap \overline{A} = \varnothing.$$

Законы поглощения:

$$A \cup (A \cap B) = A, \quad A \cap (A \cup B) = A.$$

Справедливость этих законов следует из определений операций над множествами. Для примера приведем доказательство следующих свойств.

Пример.

Доказать, что $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$.

Чтобы доказать справедливость этого равенства, необходимо доказать справедливость следующих двух соотношений:

а) $A \cap (B \cup C) \subseteq (A \cap B) \cup (A \cap C)$;

б) $(A \cap B) \cup (A \cap C) \subseteq A \cap (B \cup C)$.

Доказательство (а):

$x \in A \cap (B \cup C) \Rightarrow x \in A$ и $x \in (B \cup C) \Rightarrow$

$x \in A$ и $(x \in B$ или $x \in C) \Rightarrow$

$(x \in A$ и $x \in B)$ или $(x \in A$ и $x \in C) \Rightarrow$

$(x \in A \cap B)$ или $(x \in A \cap C) \Rightarrow$

$x \in (A \cap B) \cup (A \cap C)$, таким образом, $A \cap (B \cup C) \subseteq (A \cap B) \cup (A \cap C)$.

Сейчас необходимо доказать включение $\subseteq$ в обратную сторону.

Доказательство (б):

$x \in (A \cap B) \cup (A \cap C) \Rightarrow (x \in A \cap B)$ или $(x \in A \cap C) \Rightarrow$

$(x \in A$ и $x \in B)$ или $(x \in A$ и $x \in C) \Rightarrow$

$x \in A$ и $(x \in B$ или $x \in C) \Rightarrow$

$x \in A$ и $x \in (B \cup C) \Rightarrow$

$x \in A \cap (B \cup C)$, таким образом $(A \cap B) \cup (A \cap C) \subseteq A \cap (B \cup C)$.

Из доказательств (а) и (б) следует, что $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$, что и требовалось доказать.

Пример.

Доказать, что $\overline{(X \cap Y)} = \overline{X} \cup \overline{Y}$.

Доказательство: $x \in \overline{(X \cap Y)} \Leftrightarrow x \in (X \cap \overline{Y}) \cup (\overline{X} \cap Y) \cup (\overline{X} \cap \overline{Y}) \Leftrightarrow$

$x \in (X \cap \overline{Y}) \cup (\overline{X} \cap \overline{Y}) \cup (\overline{X} \cap Y) \cup (\overline{X} \cap \overline{Y}) \Leftrightarrow$

(один член здесь продублирован, в силу того, что $A = A \cup A$, имеем

$x \in ((X \cap \overline{Y}) \cup (\overline{X} \cap \overline{Y})) \cup ((\overline{X} \cap Y) \cup (\overline{X} \cap \overline{Y})) \Leftrightarrow$

$x \in ((X \cup \overline{X}) \cap \overline{Y}) \cup (\overline{X} \cap (Y \cup \overline{Y})) \Leftrightarrow$

$x \in (U \cap \overline{Y}) \cup (\overline{X} \cap U) \Leftrightarrow x \in \overline{Y} \cup \overline{X} \Leftrightarrow x \in \overline{X} \cup \overline{Y}$,

следовательно $\overline{(X \cap Y)} = \overline{X} \cup \overline{Y}$, что и требовалось доказать.

Другие свойства студентам предлагается доказать самостоятельно.

§ 4. Прямое произведение множеств

Рассмотрим для иллюстрации множество клеток шахматной доски. Множество столбцов обозначим буквами a, b, …, h, и множество строк цифрами 1, 2, …, 8. Следовательно, каждая клетка может быть однозначно задана двумя символами: один – из множества $F = \{a, b, \ldots, h\}$, другой – из множества $R = \{1, 2, \ldots, 8\}$, например $a1$, $f7$, $e3$ и т. д. Таким образом, из множества столб-

цов F и множества строк R мы образовали множество всех клеток шахматной доски. Этот пример иллюстрирует идею образования произведений множеств.

Обозначим последовательность из n элементов $x_1, x_2, \ldots, x_n$ через $(x_1, x_2, \ldots, x_n)$. Пусть даны n множеств $A_1, A_2, \ldots, A_n$; множество всех наборов $(x_1, x_2, \ldots, x_n)$ таких, что $x_1 \in A_1, x_2 \in A_2, \ldots, x_n \in A_n$, называют *прямым произведением* $A_1, A_2, \ldots, A_n$ и обозначают $A_1 \times A_2 \times \ldots \times A_n$.

На практике часто используется прямое произведение для одинаковых множеств. В этом случае прямое произведение записывают так $\underbrace{A \times A \times \ldots \times A}_{n} = A_n$.

Если R – множество всех вещественных чисел, то прямое произведение $R \times R$ или R^2 – множество всех точек на плоскости, R^3 – множество всех точек пространства.

Пример.

Заданы множества $X = \{0, 1\}$ и $Y = \{x, y\}$. Записать множества $X \times Y$ и $Y \times X$.

Решение:

$$X \times Y = \{(0, x), (0, y), (1, x), (1, y)\},$$

$$Y \times X = \{(x, 0), (y, 0), (x, 1), (y, 1)\}.$$

Таким образом, видим, что $X \times Y \neq Y \times X$. Необходимо отметить, что здесь важна упорядоченность элементов в каждой паре.

Свойства прямого произведения:

1. $A \times B = B \times A$, тогда и только тогда, когда $A = B$.
2. $\varnothing \times A = \varnothing$.
3. $U \times A \neq A$.
4. $A \notin A \times A$.

Доказать свойства самостоятельно.

§ 5. Отношения на множествах

Часто в вычислениях необходимо выбирать элементы множеств, которые удовлетворяют некоторому «отношению». Это понятие довольно общее, поэтому оно широко применимо.

Перед тем как подойти к этому вопросу с математической точки зрения, рассмотрим простую ситуацию, которая приводит к возникновению понятия отношения. Предположим, что для некоторой конечной машины мы имеем множество программ P, конечное множество значений данных D и множество результатов R. Если мы выберем конкретное значение из D, то оно может использоваться в некоторых программах из P, и для каждой программы из P существует совокупность значений из D, которые в ней используются. Таким образом мы имеем соответствие между значениями данных и программами, и, следовательно, существуют элементы в $D \times P$. Или можно использовать такие программы из множества P, которые приведут к получению конкретных

результатов из множества R, следовательно существуют элементы множества $P \times R$.

Перейдем к математическим описаниям.

N-местным отношением ρ на множествах $A_1, A_2, \ldots, A_n$ называется подмножество прямого произведения $A_1 \times A_2 \times \ldots \times A_n$.

Другими словами элементы $x_1, x_2, \ldots, x_n$, (где $x_1 \in A_1, x_2 \in A_2, \ldots, x_n \in A_n$) связаны отношением ρ тогда и только тогда, когда $(x_1, x_2, \ldots, x_n) \in \rho$, где $(x_1, x_2, \ldots, x_n)$ – упорядоченный набор элементов.

При $n = 2$ отношение называется *бинарным*. Бинарное отношение A на B является просто подмножеством $A \times B$. Если множества A и B равны $A = B$, то подмножество A^2 определяет отношение на A. По другому бинарное отношение можно определить как множество упорядоченных пар. Если ρ есть некоторое отношение и пара $(x, y) \in \rho$ (принадлежит этому отношению), то будем обозначать $x\rho y$ (или $x\mu y$, и др.).

Областью определения бинарного отношения ρ называется множество $D(\rho) = \{x: x\rho y\}$, т. е. для конкретного x пара $(x, y) \in \rho$.

Областью значений бинарного отношения ρ называется множество $E(\rho) = \{y: x\rho y\}$.

Пример.

Пусть на множестве $A = \{1, 2, 3, \ldots, 10\}$ задано отношение $\rho = \{(x, y): x, y \in A, x$ – делитель y и $x \leq 5\}$. Требуется записать в явном виде множество ρ и найти $D(\rho)$ и $E(\rho)$.

Решение:

Перечислим элементы множества ρ: ρ = {(1,1), (1,2), (1,3), (1,4), (1,5), (1,6), (1,7), (1,8), (1,9), (1,10), (2,2), (2,4), (2,6), (2,8), (2,10), (3,3), (3,6), (3,9), (4,4), (4,8), (5,5), (5,10)}.

$D(\rho) = \{1, 2, 3, 4, 5\}$ и $E(\rho) = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$.

Обратным отношением для отношения ρ называется отношение ρ^{-1}, определенное следующим образом: $\rho^{-1} = \{(x, y): (y, x) \in R\}$.

Таким образом, ρ^{-1} связывает те же пары элементов, что и ρ, но «в другом порядке».

Следовательно, если $\rho \subseteq A \times B$, то $\rho^{-1} \subseteq B \times A$, $D(R^{-1}) = E(\rho)$ и $E(\rho^{-1}) = D(\rho)$.

Очевидно, что общие отношения, будучи только подмножествами произведения множеств, не особенно интересны, поскольку о них можно сказать очень мало. Однако когда отношения удовлетворяют некоторым дополнительным условиям, относительно них можно сделать более содержательные утверждения. Отношения могут обладать следующими свойствами:

— рефлексивность;
— симметричность;
— транзитивность;
— антисимметричность.

Говорят, что свойство имеет место, если только выполнено следующее условие.

Пусть ρ – отношение на множестве A, тогда:

а) ρ *рефлексивно*, если для любого $x \in A$ выполняется $x\rho x$ (т. е. для любого x пара $(x, x) \in \rho$);

б) ρ *симметрично*, если $x\rho y$ влечет $y\rho x$ (т. е. если пара $(x, y) \in \rho$, то и пара $(y, x) \in \rho$);

в) ρ *транзитивно*, если $x\rho y$ и $y\rho z$ влечет $x\rho z$ (т. е. если пары $(x, y) \in \rho$ и $(y, z) \in \rho$, то пара $(x, z) \in \rho$);

г) ρ *антисимметрично*, если $x\rho y$ и $y\rho x$ влекут $x = y$.

Пример.

Пусть задано следующее отношение ρ на множестве N.

$\rho = \{(x, y)$: $x, y \in N$, и x – делитель $y\}$. Проверить свойства.

Решение:

а) ρ рефлексивно, так как любое $x \in N$, является делителем самого себя, т. е. для $\forall x \in N$, пара $(x, x) \in \rho$;

б) ρ несимметрично, так как $x\rho y$ не влечет $y\rho x$, например $(6, 12) \in \rho$, а $(12, 6) \notin \rho$, число 12 не является делителем числа 6 (чтобы показать, что свойство не выполняется для этого достаточно указать противоречие на примере конкретной пары);

в) ρ транзитивно, так как если y делится на x и z делится на y, то z делится на x. Докажем это. Пусть $\frac{y}{x} = n_1$, а $\frac{z}{y} = n_2$, где n_1 и n_2 целые числа ($\neq 0$ и $\in N$). Отсюда $x = \frac{y}{n_1}$, $z = n_2 y$. Нужно показать, что z делится на x: $\frac{z}{x} = \frac{n_2 y}{\frac{y}{n_1}} = n_1 n_2 \in N$. Это означает, что z делится на x. Таким образом выполняется свойство: $x\rho y$ и $y\rho z$ влечет $x\rho z$;

г) ρ антисимметрично, так как из того, что y делится на x и x делится на y, следует, что $x = y$. Докажем это. Пусть $\frac{y}{x} = n_1 \in N$ и $\frac{x}{y} = n_2 \in N$, отсюда $y = xn_1$, тогда $\frac{x}{y} = \frac{x}{xn_1} = \frac{1}{n_1} = n_2$. Поскольку n_1 и n_2 целые числа из N, то $\frac{1}{n_1} = n_2$ только в случае когда $n_1 = n_2 = 1$, следовательно, $x = y$.

§ 6. Разбиения и отношения эквивалентности

Во многих вычислительных задачах берутся большие множества и разбиваются таким образом, чтобы на правильно выбранных примерах можно было исследовать интересующие нас ситуации. Отсюда возникает понятие разбиения.

Разбиением непустого множества A называется совокупность подмножеств $R(A)$ таких, что объединение всех элементов $R(A)$ совпадает с A и все элементы $R(A)$ взаимно не пересекаются, т. е. A разбито таким образом, что каждый элемент A содержится только в одном подмножестве разбиения.

Пример.

1. $\{A, \overline{A}\}$ – разбиение U.

2. $\{A \cap B, A \cap \overline{B}, \overline{A} \cap B, \overline{A} \cap \overline{B}\}$ – разбиение U.

3. $\{A \setminus B, A \cap B, B \setminus A\}$ – разбиение $A \cup B$.

Разбиение определяется однозначно, и части разбиения индуцируют особый ряд отношения, называемого *отношением эквивалентности*. Эти отношения ведут себя подобно отношению «=» между числами или множествами.

Бинарное отношение на множестве называется *отношением эквивалентности*, если оно рефлексивно, симметрично и транзитивно.

Пример.

1. Отношение равенства на множестве действительных чисел R является отношением эквивалентности.

2. Отношение подобия на множестве треугольников – является отношением эквивалентности.

3. Отношение, определенное на множестве всех программ: $\{(a, b)$: a и b вычисляют одну и туже функцию на определенной машине$\}$.

В вычислениях отношения эквивалентности представляют особый интерес, поскольку они ассоциируются с различными алгоритмами, дающими один и тот же результат, или приводящими к одной и той же обработке данных, или же представляющими одну и ту же информацию о различных эквивалентных структурах данных. Примером таких очевидных ситуаций является борьба с трудностями, которые возникают при исследовании разрешимости.

§ 7. Отношения порядка

Поскольку из понятия равенства (например, между числами) возникает математическое понятие эквивалентности, некоторые неравенства могут также использоваться как модели для более широкого класса отношений.

Отношение на множестве A называется *отношением частичного порядка*, если оно рефлексивно, антисимметрично и транзитивно. *Порядок* (называемый также *отношением порядка*) – это обобщение отношения $\leq$ на N.

Требуемые три свойства предлагается проверить студентам самостоятельно.

Определив отношение $\leq$, можно определить отношение $<$ следующим образом: $a < b \Leftrightarrow a < b$ и $a \neq b$.

Аналогично, если задано $<$, то $a \leq b \Leftrightarrow a = b$ или $a < b$.

Если в качестве определения взять отношение $<$, тогда отношение порядка будет только транзитивным. Поэтому свойство транзитивности является наиболее важным для отношения порядка.

§ 8. Функции и отображения

Понятие функции и близкого к ней понятия отображения в дискретной математике рассматривается как множество бинарных отношений. В этом разделе будут изучены различные свойства, которыми они обладают. В дальнейшем функции будут использоваться для того, чтобы формализовать процесс вычислений. Особый интерес представляют функции, которые удобно определять с помощью операторов.

Бинарное отношение ρ между множествами A и B называется *функцией*, если из того, что выполняется $a\rho b$ и $a\rho c$ следует, что $b = c$. Или другими словами: отношение f называется *функцией* на множествах A и B, если из того, что $f(x) = y$ и $f(x) = z$ следует, что $y = z$.

Функции обычно обозначают строчными латинскими буквами $f, g, h, \ldots$.

Если f – функция между множествами A и B, то этот факт может быть записан следующим образом: $f: A \mapsto B$.

Если множества A и B совпадают, то функция запишется так – $f: A \mapsto A$.

В дальнейшем, если $x \in A$ и xfy, мы будем обозначать это соотношение следующим образом: $f: x \mapsto y$. Это обозначение часто используют для того, чтобы описать правило, определяющее функцию.

Пример.

Функция $f: A \mapsto A$, где $A = \{-1, 0, 1\}$, определяется соотношением $f: x \mapsto x^3$.

Область определения функции обозначают $D(f)$, а область значений функции обозначают $E(f)$ и определяются они так же, как и для бинарных отношений.

Часто приходится сталкиваться с трудностями при определении области значений функции. Поэтому, если $D(f) = A$ и $E(f) \subseteq B$, то говорят, что функция f задана на множестве A со значениями во множестве B и осуществляет *отображение* множества A во множество B (или устанавливает *соответствие* между множествами A и B). Следует заметить, что обратная функция не всегда существует.

Пример.

На множестве $\{-1, 0, 1\}$ отношение $f: x \mapsto x^2$ является функцией, но обратной функции не существует, поскольку $f^{-1}(1) = \{-1, 1\}$.

Функция $f: A \mapsto A$ называется *отображением*, если ее область определения $D(f)$ совпадает с A, т. е. $D(f) = A$.

Отображение называют *биективным* (или *биекцией*), если существует такая функция $f: A \mapsto B$, которая устанавливает взаимно однозначное соответствие между множествами A и B, обозначают $A \sim B$ (A биективно B).

Отображение $f: A \mapsto B$ имеет *обратное отображение* $f^{-1}: B \mapsto A$ тогда и только тогда, когда f – биекция.

Пример.

1. Отношение {(1, 2), (2, 3),(□, ○)}являются функцией.

2. Отношение {(1, 2), (1, 3), (2, 4)} – не является функцией.

3. Отношение $\{(x, x^2 + 2x + 1)$: x – действительное число$\}$ – является функцией, которую обычно обозначают $y = x^2 + 2x + 1$.

Пример.

Функция $f : A \mapsto A$, где $A = \{-1, 0, 1\}$, определяется соотношением f: $x \mapsto x^3$.

Пример.

Какие из указанных ниже отношений на множестве $A = \{-10, -9, -8, \ldots, 8, 9, 10\}$ являются функциями?

1) $f = \{(x, y)$: $x^2 = y\}$;

2) $f = \{(x, y)$: $x = y^2\}$.

Решение:

1) $f = \{(x, y)$: $x^2 = y\}$ – это функция, так как при $x = 3$, $y = 9$ определено однозначно, других значений нет;

2) $f = \{(x, y)$: $x = y^2\}$ – это отношение не является функцией, так как при $x = 4$, $y = 2$ и $y = -2$, т. е. $y_1 \neq y_2$.

Пример.

Какие из следующих функций являются отображениями?

1) $f = \{(x, \sin x), x \in R\}$;

2) $f = \{(x, \frac{1}{x}), x \in R\}$.

Решение:

1) $f = \{(x, \sin x), x \in R\}$ – это отображение, так как $D(f) = R$.

2) $f = \{(x, \frac{1}{x}), x \in R\}$ – это не отображение, так как $D(f) \neq R$.

$D(f) = (-\infty, 0) \cup (0, \infty) \neq R$.

§ 9. Операции

Систематизируем понятие алгебраической операции, с которым мы уже встречались в различных разделах курса математики.

Пусть дано множество A.

На множестве A определена *алгебраическая операция*, если всякому упорядоченному набору элементов множества A по некоторому закону ставится в соответствие вполне определенный элемент этого же множества A, т. е. упорядоченный набор из n элементов функция f (соответствующее правило) переводит только в один элемент множества A.

Это можно записать с помощью функции следующим образом: f: $A^n \mapsto A$, $n \in N$. Говорят, что операция имеет порядок n.

Наиболее распространены операции порядка 1 и 2 (унарные и бинарные).

Примеры бинарных операций:

1) операция сложения на множестве целых чисел;

2) операция умножения на множестве целых чисел;

3) векторное умножение векторов пространства;

4) умножение квадратных матриц порядка n;

5) объединение и пересечение множеств;

Для обозначения операций используют знаки: $\otimes, \oplus$ и т. д. (любой значок в кружочке). Например, $x \otimes y = x + y$ на N или $x \otimes y = xy - 1$ на Z.

Кроме такого задания алгебраических операций можно так же задавать операцию непосредственным перечислением всех результатов операции для конечных множеств, с помощью так называемых таблиц Кэли. Например:

$\otimes$	a	b	c
a	a	a	b
b	b	a	c
c	a	b	b

В этой таблице определена операция $\otimes$ и означает

$$a \otimes a = a,$$

$$a \otimes b = a,$$

$$c \otimes b = b, \text{ и т. д.}$$

Использование таблиц имеет важное значение, так как некоторые операции, с которыми приходиться иметь дело в компьютерной математике, непригодны для словесного задания.

§ 10. Свойства операций

Бинарная операция на множестве A *коммутативна*, если $a \otimes b = b \otimes a$, для любых $a, b \in A$.

Пример.

Коммутативные операции:

1) сложение и умножение чисел;

2) сложение матриц одного порядка;

Некоммутативные операции:

1) векторное произведение векторов;

2) произведение матриц порядка n ($n \geq 2$) и др.

Бинарная операция на множестве A *ассоциативна*, если $(a \otimes b) \otimes c = a \otimes (b \otimes c)$, для всех $a, b, c \in A$. (Если операция ассоциативна, то порядок выполнения действий не важен и, следовательно, скобки не требуются).

Пример.

1. На множестве Z операция сложения ассоциативна, так как

$$(1 + 2) + 3 = 1 + (2 + 3) = 1 + 2 + 3 = 6.$$

2. На множестве Z операция вычитания не ассоциативна, так как

$$(1 – 2) – 3 = –4, \quad 1 – (2 – 3) = 2.$$

Коммутативность и ассоциативность являются двумя важными свойствами, которые могут быть определены для простых операций.

Пример.

Пусть на некотором множестве A задана операция между a и b следующим образом: $a \otimes b = \dfrac{ab+1}{a+b}$. Проверить, выполняются ли для этой операции свойства коммутативность и ассоциативность.

Решение:

1. Коммутативность: $a \otimes b = b \otimes a,$ для любых $a, b \in A.$

Определим $b \otimes a = \dfrac{ba+1}{b+a} = a \otimes b,$ следовательно, операция коммутативна.

2. Ассоциативность: $(a \otimes b) \otimes c = a \otimes (b \otimes c),$ для любых $a, b \in A.$

Определим левую часть равенства

$$(a \otimes b) \otimes c = \frac{ab+1}{a+b} \otimes c = \frac{\dfrac{ab+1}{a+b} \cdot c + 1}{\dfrac{ab+1}{a+b} + c} = \frac{((ab+1)c + a + b)(a+b)}{(a+b)(ab+1+c(a+b))} = \frac{abc + c + a + b}{ab + ac + bc + 1}.$$

Определим правую часть равенства

$$a \otimes (b \otimes c) = a \otimes \frac{bc+1}{b+c} = \frac{a \cdot \dfrac{bc+1}{b+c} + 1}{a + \dfrac{bc+1}{b+c}} =$$

$$= \frac{a(bc+1) + b + c}{(b+c)} \cdot \frac{(b+c)}{a(b+c) + bc + 1} = \frac{abc + a + b + c}{ab + ac + bc + 1}.$$

Сравнивая результат имеем $(a \otimes b) \otimes c = a \otimes (b \otimes c),$ следовательно, операция ассоциативна.

Кроме того, операции могут обладать *единичным элементом*, *правой и левой единицей*, и *обратным элементом*.

Пусть $\otimes$ – бинарная операция на множестве A и $l \in A$ такой, что $l \otimes a = a,$ для всех $a \in A$. Тогда l называется *левой единицей* по отношению к $\otimes$ на A.

Пусть $\otimes$ – бинарная операция на множестве A и $r \in A$ такой, что $a \otimes r = a,$ для всех $a \in A$. Тогда r называется *правой единицей* по отношению к $\otimes$ на A.

Если существует элемент e, который является и левой, и правой единицей, т. е. $e \otimes a = a \otimes e = a$ для всех $a \in A$, то e называется (*двусторонней*) *единицей* по отношению к $\otimes$.

Пример.

На множестве R число 0 является правой единицей по отношению к вычитанию, так как $a - 0 = a$, но $0 - a \neq a$, (если $a \neq 0$).

На множестве R число 0 является единицей по отношению к сложению, так как $a + 0 = a$ и $0 + a = a$ для всех $a \in R$.

Пусть $\otimes$ – операция на A с единицей e и $x \otimes y = e$. Тогда говорят, что x – *левый обратный* элемент к y, а y *правый обратный* элемент к x.

Далее, если x и y такие, что $x \otimes y = e = y \otimes x$, то y – называется *обратным элементом* к x по отношению к $\otimes$, и наоборот.

Докажем единственность единичного элемента.

Пусть $\otimes$ – операция на множестве A и существует единица по отношению к $\otimes$. Тогда единичный элемент единственный.

Доказательство: Предположим, что x и y – единицы по отношению к $\otimes$, т. е.

$x \otimes a = a \otimes x = a$,

$y \otimes a = a \otimes y = a$, для всех $a \in A$.

Тогда $x = x \otimes y$, так как y – единица и $x \otimes y = y$, так как x – единица, следовательно, $x = y$. Доказали единственность.

Докажем единственность обратного элемента.

Пусть $\otimes$ – ассоциативная операция на множестве A и существует e – единица по отношению к $\otimes$. Тогда, если $x \in A$ и x имеет обратный, то обратный элемент единственный, по отношению к $\otimes$.

Доказательство: Допустим, что x_1 и x_2 – обратные элементы к x, так что

$$x \otimes x_1 = x_1 \otimes x = e \quad \text{и} \quad x \otimes x_2 = x_2 \otimes x = e,$$

тогда

$$x_1 = x_1 \otimes e = x_1 \otimes (x \otimes x_2) = (x_1 \otimes x) \otimes x_2 = e \otimes x_2 = x_2, \text{ т. е. } x_1 = x_2.$$

Доказали единственность.

Задачи и упражнения

1. Равны ли множества A и B, если

а) $A = \{1, 2, 3\}$ и $B = \{\{1, 2\}, \{2, 3\}\}$

б) $A = \{2, 3\}$ и $B = \{\{2\}, \{3\}\}$?

2. Заданы множества: $U = \{1, 2, 4, 5\}$, $X = \{1, 5\}$, $Y = \{1, 2, 4\}$, $Z = \{2, 5\}$. Найти

1) $X \cap \overline{Y}$.

2) $(X \cap Z) \cup \overline{Y}$.

3) $X \cup (Y \cap Z)$.

4) $(X \cup Y) \cap (X \cup Z)$.

5) $\overline{(X \cup Y)}$.

6) $\overline{X} \cap \overline{Y}$.

7) $\overline{(X \cap Y)}$.

8) $\overline{X} \cup \overline{Y}$.

9) $(X \cup Y) \cup Z$.

10) $X \setminus Z$.

11) $(X \setminus Z) \setminus (Y \setminus Z)$.

3. Существуют ли такие множества A, B и C, что $A \cap B \neq \varnothing$, $A \cap C = \varnothing$, $(A \cap B) \setminus C = \varnothing$?

4. Доказать следующие тождества:

а) $(A \cap B) \cup (A \cap \overline{B}) = (A \cup B) \cap (A \cup \overline{B}) = A$;

б) $(\overline{A} \cup B) \cap A = A \cap B$;

в) $(A \setminus B) \setminus C = (A \setminus C) \setminus (B \setminus C)$.

5. Проверить, что $X \times Y \neq Y \times X$, , если $X = \{3, 4, 5\}$ и $Y = \{0, 1\}$.

6. Записать в явном виде отношение $R = \{(x, y)$: $x, y \in A$, x – делитель y, $x \leq 7\}$, где $A = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$.

7. Записать область определения $D(R)$ и область значений $E(R)$, если отношение $R = \{(x, y)$: $x, y \in A$, x – делитель y, $x \leq 7\}$, где $A = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$.

8. Проверить, является ли отношение $\rho = \{(a,b) : a,b \in A u\ a+b$ четное$\}$, где $A = \{1, 2, 3, 4, 5, 6\}$, рефлексивным, симметричным, транзитивным и антисимметричным.

9. На множестве = {–4, –3, –2, –1, 0, 1, 2, 3, 4} определены следующие отношения:

1) $\rho = \{(a, b)$: $a < b\}$;

2) $\sigma = \{(a, b)$: $b - 1 < a < b + 2\}$;

3) $\tau = \{(a, b)$: $a^2 \leq b\}$.

Записать в явном виде множества $\rho(0)$, $\sigma(0)$, $\tau(0)$, $\rho(1)$, $\sigma(-1)$, $\tau(-1)$.

10. Проверить, правильно ли заданы следующие операции:

а) $x \otimes y = x - y$ на N;

б) $x \otimes y = x \cdot y - 1$ на Z;

в) $x \otimes y = \max(x, y)$ на N.

Если правильно, то проверить свойства (коммутативность и ассоциативность) и найти, если это возможно, единичный и обратный элементы.

Глава 2

ЭЛЕМЕНТЫ МАТЕМАТИЧЕСКОЙ ЛОГИКИ

Математика является наукой, в которой все утверждения доказываются с помощью умозаключений, т. е. путем использования законов человеческого мышления.

Предметом математической логики является изучение законов человеческого мышления.

Основным понятием матлогики является понятие высказывания.

Высказыванием называется всякое повествовательное предложение, о котором мы можем сказать – истинно оно или ложно, т. е. логическими значениями высказывания являются истина или ложь.

Примеры высказываний:

1) г. Красноярск расположен на р. Енисей.

2) Число 10 делится на 2 и на 5.

3) Париж – столица Англии.

4) Карась – это не рыба.

Высказывания 1) и 2) принимают значения истина, высказывания 3) и 4) – ложь.

Высказывания, которые представляют собой одно утверждение – называются *простыми* или *элементарными*. Например 1) и 2).

Высказывания, которые получают из простых с помощью связок «и», «или», «не», «если …, то», «тогда и только тогда» называются *составными* или *сложными*. Например 3) и 4).

Очевидно, что не каждое предложение может быть высказыванием.

Пример: «Да здравствуют спортсмены нашей страны!» – это не высказывание, оно не может принять значение истина или ложь.

В алгебре логики все высказывания рассматриваются с точки зрения их логического значения, т. е. высказывание может принимать одно из двух значений – истина или ложь, одновременно высказывание не может принять то и другое значение.

В дальнейшем высказывание условимся обозначать малыми буквами латинского алфавита *a, b, c, d, …, x, y, z*. А их значения «и» и «л» или «1» и «0». Например, если высказывание а – истинно, то будем писать $a = 1$, а если ложно, то $a = 0$.

§ 1. Логические операции над высказываниями

Над высказываниями можно выполнять логические операции, которые играют важную роль в математических доказательствах.

Отрицанием высказывания x называется новое высказывание, которое является истинным, если x – ложно и ложным, если x – истинно.

Обозначается $\bar{x}$ и читается «не x».

Логические значения высказывания $\bar{x}$ можно представить следующей таблицей, которую в дальнейшем будем называть *таблицей истинности*:

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

Если x – высказывание, то $\bar{x}$ – тоже высказывание. Можно образовать отрицание высказывания $\bar{x}$, т. е. $\bar{\bar{x}}$, которое называется *двойным отрицанием*. Значения $\bar{x}$ и $\bar{\bar{x}}$ совпадают.

Коньюнкцией двух высказываний x и y (логическим умножением) называется новое высказывание, которое считается истинным, если оба высказывания x и y истинны, и ложным, если хотя бы одно из них ложно.

Обозначается символом $x \wedge y$ или $(x \,\&\, y)$. Читается «x и y».

Логические значения коньюнкции описываются следующей таблицей истинности:

x	y	$x \wedge y$
1	1	1
1	0	0
0	1	0
0	0	0

Из определения операции коньюнкции и отрицания следует, что высказывание $x \wedge \bar{x}$ всегда ложно.

Дизьюнкцией двух высказываний x и y (логическим сложением) называется новое высказывание, которое считается истинным, если хотя бы одно из высказываний x или y – истинно, и ложным, если они оба – ложны.

Обозначается $x \vee y$. Читается «x или y».

Логические значения описываются следующей таблицей истинности:

x	y	$x \vee y$
1	1	1
1	0	1
0	1	1
0	0	0

Из определения операции дизьюнкции и отрицания следует, что высказывание $x \vee \bar{x}$ всегда истинно.

Импликацией двух высказываний x и y называется новое высказывание, которое считается ложным, если x – истинно, а y – ложно, и истинным, во всех остальных случаях.

Обозначается $x \to y$. Читается «если x, то y» или «из x следует y».

Высказывание x называют *условием*, а y – *следствием* или заключением.

Логические значения этой операции описываются следующей таблицей истинности:

x	y	$x \to y$
1	1	1
1	0	0
0	1	1
0	0	1

Импликация играет важную роль в математических доказательствах, так как многие теоремы в условной форме формулируются «если *x*, то *y*».

Эквиваленцией (или *эквивалентностью*) двух высказываний *x* и *y* называется новое высказывание, которое считается истинным, когда оба высказывания *x* и *y* либо одновременно истинны, либо одновременно ложны, и ложным, во всех остальных случаях.

Обозначается $x \leftrightarrow y$. Читается «для того, чтобы *x*, необходимо и достаточно, чтобы *y*» или «*x* тогда и только тогда, когда *y*».

Логические значения этой операции описываются следующей таблицей истинности:

x	y	$x \leftrightarrow y$
1	1	1
0	0	1
1	0	0
0	1	0

Логические операции играют важную роль в математических доказательствах, так как значительное число теорем формулируется в условной форме «если *x*, то *y*» или в форме необходимых и достаточных условий «для того чтобы *x*, необходимо и достаточно чтобы *y*».

§ 2. Формулы алгебры логики. Основные равносильности и преобразования

С помощью логических операций над простыми высказываниями можно строить различные сложные высказывания. При этом порядок выполнения операций указывается скобками.

С помощью логических операций над простыми высказываниями можно строить различные сложные высказывания.

Например:

$(x \wedge y) \vee \bar{z}$ или $x \to \overline{(y \vee (x \wedge z))}$.

Скобки указывают порядок выполнения действий.

Формулой алгебры логики называется всякое сложное высказывание, которое может быть получено из элементарных высказываний посредством применения логических операций отрицания, коньюнкции, дизьюнкции, импликации и эквиваленции.

Формулы алгебры логики (ФАЛ) будем обозначать большими буквами латинского алфавита *A, B, C, …*

Скобки в ФАЛ можно опускать, придерживаясь следующего порядка выполнения действий: коньюнкция, дизьюнкция, импликация и эквиваленция.

Пример.

1) $(x \wedge y) \vee \bar{z}$ равносильно $x \wedge y \vee \bar{z}$.

2) $x \to \overline{(y \vee (x \wedge z))}$ равносильно $x \to \overline{y \vee x \wedge z}$.

Логическое значение ФАЛ полностью определяется логическими значениями входящих в нее элементарных высказываний.

Пример.

При $x = 1, y = 1, z = 0$ формула $\overline{x \wedge y} \vee \bar{z} = 1$.

Логическое значение формулы изменяется в зависимости от изменений значений элементарных высказываний, входящих в формулу. Все возможные логические значения формулы могут быть описаны полностью с помощью таблицы истинности.

Пример.

Таблица истинности логических значений формулы $\bar{x} \vee y \to x \wedge \bar{y}$ будет следующая:

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

Если формула содержит n элементарных высказываний, то она принимает 2^n значений. Таблица истинности будет содержать 2^n строк.

Две формулы алгебры логики *A* и *B* называются *равносильными*, если они принимают одинаковые логические значения на любом наборе значений, входящих в формулы элементарных высказываний.

Обозначается равносильность ≡, т. е. $A \equiv B$.

Пример.

Следующие формулы являются равносильными:

$$\bar{\bar{x}} \equiv x,$$

$$x \vee x \equiv x,$$

$$(x \wedge \bar{x}) \vee y \equiv y.$$

Формула *A* называется *тождественно истинной* (или *тавтологией*), если она принимает значение 1 при всех значениях входящих в нее переменных.

Пример.

Следующие формулы являются тавтологиями:

$$x \vee \bar{x}, \quad x \to (y \to x).$$

Формула A называется *тождественно ложной,* если она принимает значение 0 при всех значениях входящих в нее переменных.

Пример.

Формула $x \wedge \bar{x}$ является тождественно ложной.

Отношение равносильности обладает следующими свойствами: оно рефлексивно, симметрично и транзитивно.

Между понятиями равносильности и эквивалентности существует следующая связь: если формулы A и B равносильны, то формула $A \leftrightarrow B$ – тавтология, и обратно, если формула $A \leftrightarrow B$ – тавтология, то формулы A и B равносильны.

Равносильности алгебры логики используются для того, чтобы любую формулу алгебры логики можно заменить равносильной ей формулой.

Важнейшие равносильности алгебры логики можно разбить на три группы.

1. Основные равносильности

$$\left.\begin{array}{l} 1.\ x \wedge x \equiv x \\ 2.\ x \vee x \equiv x \end{array}\right\} \text{– законы идемпотентности.}$$

3. $\bar{\bar{x}} \equiv x$ – закон снятия двойного отрицания.

$$\left.\begin{array}{l} 4.\ x \wedge (y \vee x) \equiv x \\ 5.\ x \vee (y \wedge x) \equiv x \end{array}\right\} \text{– законы поглощения.}$$

Докажем формулу 4.

Пусть $A \equiv x \wedge (y \vee x)$, при $x = 1$, значение $A = 1$, при $x = 0$, значение $A = 0$. Итак во всех случаях значения формулы A совпадают со значениями x, следовательно, $A \equiv x$.

2. Равносильности, выражающие одни логические операции через другие

1. $x \leftrightarrow y \equiv (x \to y) \wedge (y \to x)$.
2. $x \to y \equiv \bar{x} \vee y$.
3. $\overline{x \wedge y} \equiv \bar{x} \vee \bar{y}$.
4. $\overline{x \vee y} \equiv \bar{x} \wedge \bar{y}$.
5. $x \wedge y \equiv \overline{\bar{x} \vee \bar{y}}$.
6. $x \vee y \equiv \overline{\bar{x} \wedge \bar{y}}$.

Замечание. Формулы 5 и 6 получаются из 3 и 4, если от обеих частей последних взять отрицания и воспользоваться законом снятия двойного отрицания.

Докажем формулы 1–4.

Докажем формулу 1.

1) *при одинаковых логических значениях* x *и* y формулы $x \leftrightarrow y$, $x \to y$ и $y \to x$ – *истинны*, следовательно, истинной будет и коньюнкция $(x \to y) \wedge (y \to x)$, т. е. обе части равносильности имеют одинаковые истинные значения.

2) пусть теперь *x и y имеют разные логические значения*, тогда будут ложными $x \leftrightarrow y$ и одна из $x \to y$ или $y \to x$. При этом будет ложной и коньюнкция $(x \to y) \wedge (y \to x)$, т. е. обе части равносильности имеют одинаковые ложные значения. Что и требовалось доказать.

Докажем формулу 3.

1) *пусть x и y одновременно принимают истинные значения*, тогда будет истинной и коньюнкция $x \wedge y$ и ложным ее отрицание $\overline{x \wedge y}$. В то же время будут ложными $\bar{x}$ и $\bar{y}$, следовательно, будет ложной и дизьюнкция $\bar{x} \vee \bar{y}$;

2) *пусть хотя бы одна из переменных x или y принимает значение ложь*, тогда $x \wedge y$ тоже ложь, а $\overline{x \wedge y}$ – истина. В то же время отрицание хотя бы одной из переменных будет истинным, следовательно, будет истиной и дизьюнкция $\bar{x} \vee \bar{y}$.

Следовательно, во всех случаях обе части равносильности 3 принимают одинаковые логические значения.

Аналогично доказываются равносильности 2 и 4.

Из равносильностей группы 2 следует, что всякую формулу алгебры логики можно заменить равносильной ей формулой, содержащей только две логические операции: коньюнкцию и отрицание или дизьюнкцию и отрицание.

3. Равносильности, выражающие основные законы алгебры логики

$$\left.\begin{array}{l} 1.\ x \wedge y \equiv y \wedge x \\ 2.\ x \vee y \equiv y \vee x \end{array}\right\}$$ – коммутативность коньюнкции и дизьюнкции.

$$\left.\begin{array}{l} 3.\ x \wedge (y \wedge z) \equiv (x \wedge y) \wedge z \\ 4.\ x \vee (y \vee z) \equiv (x \vee y) \vee z \end{array}\right\}$$ – ассоциативность коньюнкции и дизьюнкции.

$$\left.\begin{array}{l} 5.\ x \wedge (y \vee z) \equiv (x \wedge y) \vee (x \wedge z) \\ 6.\ x \vee (y \wedge z) \equiv (x \vee y) \wedge (x \vee z) \end{array}\right\}$$ – дистрибутивность коньюнкции относительно дизьюнкции и наоборот.

Докажем формулу 6.

При $x = 1$, формулы $x \vee (y \wedge z)$, $x \vee y$ и $x \vee z$ будут истинны, тогда и $(x \vee y) \wedge (x \vee z)$ – тоже истинна.

При $x = 0$, $x \vee (y \wedge z) \equiv y \wedge z$, $x \vee y \equiv y$, $x \vee z \equiv z$, следовательно, $(x \vee y) \wedge (x \vee z) \equiv y \wedge z$.

Таким образом, обе части формулы 6 равносильны одной и той же формуле $y \wedge z$, и поэтому принимают одинаковые логические значения. Что и требовалось доказать.

Равносильности 3-ей группы выражают основные законы алгебры логики: коммутативность, ассоциативность и дистрибутивность (относительно логических операций – коньюнкции и дизьюнкции). Эти же законы имеют место в алгебре чисел. Поэтому над формулами алгебры логики можно производить те же преобразования, которые проводятся в алгебре чисел, т. е.

1) раскрытие скобок;

2) заключение в скобках;

3) вынесения за скобки общего множителя.

Кроме этих преобразований над формулами алгебры логики можно производить и преобразования, основанные на использовании равносильностей.

Равносильные преобразования формул используют

1) для доказательства равносильностей,

2) для приведения формул к заданному виду,

3) для упрощения формул.

Формула A считается проще формулы B, если A содержит меньше букв и меньше логических операций. При этом логические операции эквивалентность ($\leftrightarrow$) и импликация ($\rightarrow$), заменяются операциями дизьюнкция ($\vee$) и коньюнкция ($\wedge$), а отрицание относят к элементарным высказываниям.

Пример.

1. Доказать равносильность $x \leftrightarrow y \equiv \bar{x} \wedge \bar{y} \vee x \wedge y$.

Доказательство:

$$x \leftrightarrow y \equiv (x \rightarrow y) \wedge (y \rightarrow x) \equiv (\bar{x} \vee y) \wedge (\bar{y} \vee x) \equiv \bar{x} \wedge \bar{y} \vee \bar{x} \wedge x \vee y \wedge \bar{y} \vee y \wedge x \equiv$$
$$\equiv \bar{x} \wedge \bar{y} \vee 0 \vee 0 \vee y \wedge x \equiv \bar{x} \wedge \bar{y} \vee y \wedge x \equiv \bar{x} \wedge \bar{y} \vee x \wedge y.$$

2. Упростить формулу $(\overline{x \vee y} \rightarrow x \vee y) \wedge y$.

$$(\overline{x \vee y} \rightarrow x \vee y) \wedge y \equiv (\overline{\overline{x \vee y}} \vee x \vee y) \wedge y \equiv (x \vee y \vee x \vee y) \wedge y \equiv (x \vee y) \wedge y \equiv y.$$

3. Доказать тождественную истинность формулы

$$(x \rightarrow y) \rightarrow ((y \rightarrow z) \rightarrow (x \vee y \rightarrow z)).$$

Запишем цепочку равносильных формул:

$$(x \rightarrow y) \rightarrow ((y \rightarrow z) \rightarrow (x \vee y \rightarrow z)) \equiv \overline{\overline{(\bar{x} \vee y)} \vee (\overline{\bar{y} \vee z} \vee \overline{x \vee y} \vee z)} \equiv$$
$$\equiv \bar{\bar{x}} \wedge \bar{y} \vee \bar{\bar{y}} \wedge \bar{z} \vee \bar{x} \wedge \bar{y} \vee z \equiv x \wedge \bar{y} \vee y \wedge \bar{z} \vee \bar{x} \wedge \bar{y} \vee z \equiv$$
$$\equiv (x \wedge \bar{y} \vee \bar{x} \wedge \bar{y}) \vee (y \wedge \bar{z} \vee z) \equiv \bar{y} \wedge (x \vee \bar{x}) \vee (y \vee z) \wedge (\bar{z} \vee z) \equiv$$
$$\equiv \bar{y} \wedge 1 \vee (y \vee z) \wedge 1 \equiv \bar{y} \vee y \vee z \equiv (\bar{y} \vee y) \vee z \equiv 1 \vee z \equiv 1.$$

Что и требовалось доказать.

§ 3. Алгебра Буля. Функции Буля. Представление произвольной функции алгебры логики в виде формулы алгебры логики

Равносильности 3 – ей группы говорят о том, что алгебра логики обладает коммутативными и ассоциативными законами относительно операций коньюнкции и дизьюнкции и дистрибутивным законом коньюнкции относительно дизьюнкции, эти же законы имеют место и в алгебре чисел. Поэтому над формулами алгебры логики можно производить те же преобразовантя, которые проводятся в алгебре чисел (раскрытие скобок, заключение в скобки, вынесение за скобки общего множителя).

Но в алгебре логики возможны и другие преобразования, основанные на использовании равносильностей:

$$(x \wedge y) \vee z \equiv (x \vee z) \wedge (y \vee z),$$
$$x \wedge (y \vee x) \equiv x,$$
$$x \vee (y \wedge x) \equiv x,$$
$$\overline{x \wedge y} \equiv \bar{x} \vee \bar{y},$$
$$\overline{x \vee y} \equiv \bar{x} \wedge \bar{y}.$$

Рассмотрим непустое множество M элементов любой природы $\{x, y, z, \ldots\}$, в котором определены отношение «=» и три операции «+», «·» и «–» (отрицание), подчиняющиеся следующим аксиомам:

Коммутативные законы

$x + y = y + x$,
$x \cdot y = y \cdot x$.

Ассоциативные законы

$x + (y + z) = (x + y) + z$,
$x \cdot (y \cdot z) = (x \cdot y) \cdot z$.

Дистрибутивные законы

$(x + y) \cdot z = (x \cdot z) + (y \cdot z)$,
$(x \cdot y) + z = (x + z) \cdot (y + z)$.

Законы идемпотентности

$x + x = x$,
$x \cdot x = x$.

Закон двойного отрицания

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

Законы де-Моргана

$\overline{x + y} = \bar{x} \cdot \bar{y}$,
$\overline{x \cdot y} = \bar{x} + \bar{y}$.

Законы поглощения

$x + (y \cdot x) = x$,
$x \cdot (y + x) = x$.

Такое множество М называется *булевой алгеброй*.

Если можно подобрать конкретные объекты $x, y, z, \ldots$ и конкретные соотношения между ними, так что все приведенные аксиомы выполняются, то говорят, найдена *интерпретация* (или *модель*) данной системы аксиом.

Пример.

1) алгебра логики является интерпретацией булевой алгебры ($x, y, z, \ldots$ – высказывания и операции «=», «+», «·», «–», соответствующие – равносильности, дизъюнкции, коньюнкции, отрицанию);

2) алгебра множеств – также является интепретацией булевой алгебры (x, y, z, … – множества и операции «=», «+», «·», «–», соответствующие равенству множеств, объединению, пересечению и дополнению множеств).

Среди различных интерпретаций булевой алгебры имеются интерпретации технического характера в области современной автоматики (релейно-контактные схемы).

Как уже отмечалось, значение формулы алгебры логики полностью зависит от значений входящих в эту формулу высказываний. Поэтому формула алгебры логики является функцией входящих в нее элементарных высказываний.

Пример.

Формула $(x \wedge y) \to \bar{z}$ является функцией трех переменных f(x, y, z). Особенность этой функции состоит в том, что ее переменные принимают – лишь два значения 0 и 1, при этом значение функции тоже может быть только 0 и 1.

Функцией Буля n переменных называется функция, где каждая переменная принимает два значения 0 и 1, при этом функция может принимать только одно из двух значений 0 или 1.

Замечание. Тождественно ложные и тождественно истинные формулы представляют собой постоянные функции, а две равносильные формулы выражают одну и ту же функцию.

Каждая функция алгебры логики также как и формула задается таблицей истинности, которая будет содержать 2^n строк. А число различных функций n переменных будет равно 2^{2^n}.

Выпишем все функции одной и двух переменных.

Таблица истинности для различных функций одной переменной имеет следующий вид:

x	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$
1	1	1	0	0
0	1	0	1	0

Из этой таблицы следует, что $f_1(x) \equiv 1$, $f_4(x) \equiv 0$ – постоянные, а $f_2(x) \equiv x$, $f_3(x) \equiv \bar{x}$.

Таблица истинности для всевозможных функций двух переменных $f_i = f_i(x, y)$ имеет следующий вид:

x	y	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}
1	1	1	1	1	1	0	1	1	0	0	0	1	0	0	0	1	0
1	0	1	1	1	0	1	1	0	0	1	1	0	0	0	1	0	0
0	1	1	1	0	1	1	0	0	1	1	0	1	0	1	0	0	0
0	0	1	0	1	1	1	0	1	1	0	1	0	1	0	0	0	0

Строк – 2^2, функций – 2^4.

Аналитические выражения этих функций могут быть записаны следующим образом:

$$f_1 \equiv 1,\ f_2 \equiv x \vee y,\ f_3 \equiv y \to x,\ f_4 \equiv x \to y,\ f_5 \equiv \overline{x \wedge y},\ f_6 \equiv x,$$

$$f_7 \equiv x \leftrightarrow y,\ f_8 \equiv x,\ f_9 \equiv \overline{x \leftrightarrow y},\ f_{10} \equiv \bar{y},\ f_{11} \equiv y,\ f_{12} \equiv \overline{x \vee y},$$

$$f_{13} \equiv \overline{y \to x},\ f_{14} \equiv \overline{x \to y},\ f_{15} \equiv x \wedge y,\ f_{16} \equiv 0.$$

Пусть $F(x_1, x_2, \ldots, x_n)$ – произвольная функция алгебры логики n переменных. Рассмотрим формулу

$$F(1,1,...,1) \wedge x_1 \wedge x_2 \wedge ... \wedge x_n \vee F(1,1,...,1,0) \wedge x_1 \wedge x_2 \wedge ... \wedge \overline{x_n} \vee$$
$$F(1,1,...,0,1) \wedge x_1 \wedge x_2 \wedge ... \wedge \overline{x_{n-1}} \wedge x_n \vee ... \vee F(0,0,...,0) \wedge \overline{x_1} \wedge \overline{x_2} \wedge ... \wedge \overline{x_n}, \quad (1)$$

которая составлена следующим образом: каждое слагаемое этой логической суммы есть коньюнкция, в которой первый член является значением функции $F(x_1, x_2, \ldots, x_n)$ при определенных значениях переменных $x_1, x_2, \ldots, x_n$, остальные члены есть сами эти переменные или их отрицания. При этом под знаком отрицания находятся те переменные, которые в первом члене коньюнкции имеют значение 0.

Значения функции F и формулы (1) совпадают на всех наборах значений переменных $x_1, x_2, \ldots, x_n$.

Например, пусть $x_1 = 0$, а все остальные 1. Тогда F принимает значение $F(0, 1, \ldots, 1)$, при этом логическое слагаемое $F(0,1,...,1) \wedge \overline{x_1} \wedge x_2 \wedge ... \wedge x_n$ принимает значение $F(0, 1, \ldots, 1)$. А все остальные логические слагаемые формулы (1) равны 0. Так как в них отрицания над переменными располагаются иначе, чем в рассмотренном слагаемом. При подстановке значений переменных в коньюнкцию войдет значение 0 без знака отрицания и значение 1 под знаком отрицания. В таком случае один из членов коньюнкции будет равен 0, следовательно и вся коньюнкция будет равна 0. На основании равносильности $x \vee 0 \equiv x$ значение формулы (1) равно $F(0, 1, \ldots, 1)$.

Вид формулы (1) может быть значительно упрощен, если в ней отбросить те логические слагаемые, в которых первый член коньюнкции равен 0 (следовательно и вся коньюнкция равна 0).

Если же первый член коньюнкции равен 1, то в силу равносильности $1 \wedge x \equiv x$, этот член коньюнкции можно не выписывать. В результате получается формула (1), которая содержит только элементарные переменные высказывания и обладает следующими свойствами, которые называют свойствами совершенства:

1) каждое логическое слагаемое формулы содержит все переменные, входящие в функцию $F(x_1, x_2, \ldots, x_n)$;

2) все логические слагаемые формулы различны;

3) ни одно логическое слагаемое формулы не содержит одновременно переменную и ее отрицание;

4) ни одно логическое слагаемое формулы не содержит одну и ту же переменную дважды.

Пример.

Составить формулу для функции $F(x_1, x_2, x_3)$, заданную следующей таблицей истинности:

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

Формула может быть получена следующим образом. Для каждого набора значений переменных, на котором $F(x_1, x_2, x_3)$ равна 1, запишем коньюнкцию элементарных переменных высказываний

$$(1, 1, 0) - x_1 \wedge x_2 \wedge \overline{x_3}.$$

$$(1, 0, 1) - x_1 \wedge \overline{x_2} \wedge x_3.$$

$$(0, 1, 0) - \overline{x_1} \wedge x_2 \wedge \overline{x_3}.$$

$$(0, 0, 0) - \overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3}.$$

Искомая формула будет иметь вид

$$x_1 \wedge x_2 \wedge \overline{x_3} \vee x_1 \wedge \overline{x_2} \wedge x_3 \vee \overline{x_1} \wedge x_2 \wedge \overline{x_3} \vee \overline{x_1} \wedge \overline{x_2} \wedge \overline{x_3}.$$

Пусть формула A содержит операции коньюнкции, дизьюнкции и отрицания. Будем называть операцию коньюнкции *двойственной* операции дизьюнкции, а операция дизьюнкции *двойственной* операции коньюнкции.

Формулы A и A^* называются *двойственными*, если формула A^* получается из A путем замены в ней каждой операции на двойственную.

Пример.

Формулы $A \equiv (x \vee y) \wedge z$ и $A^* \equiv (x \wedge y) \vee z$ являются двойственными.

Теорема.

Если формулы A и B равносильны, то равносильны и двойственные им формулы, т. е. $A^* \equiv B^*$.

Формулировку теоремы примем без доказательства.

§ 4. Дизьюнктивная нормальная форма и совершенная дизьюнктивная нормальная форма. Коньюнктивная нормальная форма и совершенная коньюнктивная нормальная форма

Для любой формулы алгебры логики путем равносильных преобразований можно получить *дизьюнктивную нормальную форму* (ДНФ).

Дизьюнктивной нормальной формой формулы A называется равносильная ей формула, представляющая собой дизьюнкцию элементарных коньюнкций.

ДНФ для любой формулы алгебры логики не единственная, но среди этих всех ДНФ будет такая, для которой выполняются свойства совершенства. Такая ДНФ называется *совершенной дизьюнктивной нормальной формой* формулы A (СДНФ).

Пример.

1. Преобразовать формулу $A \equiv x \wedge (x \to y)$ к СДНФ.

$$СДНФ\ A \equiv x \wedge (\overline{x} \vee y) \equiv (x \wedge \overline{x}) \vee (x \wedge y) \equiv x \wedge y.$$

2. Преобразовать формулу $A \equiv x \vee y \wedge (x \vee \overline{y})$ к СДНФ.

$$СДНФ\ A \equiv x \vee y \wedge x \vee y \wedge \overline{y} \equiv x \wedge (y \vee \overline{y}) \vee (y \wedge x) \vee (y \wedge \overline{y}) \equiv$$
$$\equiv (x \wedge y) \vee (x \wedge \overline{y}) \vee (x \wedge y) \vee (y \wedge \overline{y}) \equiv (x \wedge y) \vee (x \wedge \overline{y}).$$

Коньюнктивной нормальной формой (КНФ) формулы A называется равносильная ей формула, представляющая собой коньюнкцию элементарных дизьюнкций.

Для любой формулы алгебры логики с помощью равносильных преобразований можно получить ее КНФ (причем не единственную), но среди этих всех КНФ будет такая, для которой выполняются свойства совершенства. Такая КНФ называется *совершенной коньюнктивной нормальной формой* формулы A (СКНФ).

Все формулы алгебры логики делятся на три класса:

1) тождественно истинные;

2) тождественно ложные;

3) выполнимые.

Формула A называется *выполнимой*, если она принимает значение «и», хотя бы на одном наборе значений, входящих в нее переменных и не является тождественно истинной.

Задача, состоящая в определении к какому классу относится формула, носит название *проблемы разрешимости*.

Один из способов решения этой проблемы – это составление таблиц истинности. По таблице можно определить к какому классу принадлежит формула.

Другой способ основан на приведении формулы к КНФ или ДНФ и использовании алгоритма, который позволяет определить, является ли данная формула тождественно истинной или не является.

Одновременно с этим решается вопрос: будет ли данная формула выполнимой.

Сформулируем *критерии тождественной истинности и тождественной ложности формул алгебры логики*.

Теорема 1. *Для того чтобы элементарная дизьюнкция была тождественно истинной, необходимо и достаточно чтобы в ней содержалась переменная и ее отрицание*.

Теорема 2. *Для того чтобы формула алгебры логики А была тождественно истинна, необходимо и достаточно чтобы любая элементарная дизьюнкция, входящая в КНФ А, содержала переменную и ее отрицание*.

Теорема 3. *Для того чтобы элементарная коньюнкция была тождественно истинной, необходимо и достаточно чтобы в ней содержалась переменная и ее отрицание.*

Теорема 4. *Для того чтобы формула алгебры логики А была тождественно ложной, необходимо и достаточно чтобы любая элементарная коньюнкция, входящая в ДНФ А, содержала переменную и ее отрицание.*

§ 5. Логика предикатов. Логические операции над предикатами. Кванторные операции. Формулы

Логика предикатов – это такая логика, которая рассматривает элементарное высказывание, не только с точки зрения его истинности или ложности, но и с точки зрения его структуры и содержания.

Логика предикатов делит элементарное высказывание на субъект (подлежащее) и предикат (сказуемое).

Субъект – это то, о чем что-то утверждается в высказывании.

Предикат – это то, что утверждается о субъекте.

Пример.

Высказывание – «7 простое число».

«7» – субъект, «простое число» – предикат.

Это высказывание утверждает, что «7» обладает свойством «быть простым числом».

Если в этом примере заменить число 7 переменной x из множества натуральных чисел, то получим высказывание «x – простой число». При разных значениях x это высказывание будет принимать различные значения – {0, 1}.

Можно сказать, что это высказывание определяет функцию одной переменной $x \in N$ и принимающую значения из множества {0, 1}. Здесь предикат становится функцией субъекта и выражает свойство субъекта.

Одноместным предикатом $P(x)$ называется произвольная функция переменного x, определенного на множестве М и принимающая значения из множества {0, 1}.

Множество M на котором определен предикат $P(x)$, называется *областью определения предиката*.

Множество всех элементов $x \in M$, при которых предикат принимает значение «и», называется *множеством истинности предиката P(x)*:

$$I_p = \{x : x \in M,\ P(x) = 1\}.$$

Пример.

1. $P(x)$ – «x – простое число», определен на множестве N, а множество истинности I_P для него есть множество всех простых чисел.

2. $Q(x)$ – «sin x = 0» определен на множестве R, а множество истинности $I_Q = \{k\pi, k \in Z\}$.

3. $F(x)$ – «Диагонали параллелограмма x перпендикулярны» определен на множестве всех параллелограммов, а его множество истинности I_F – множество всех ромбов.

Приведенные примеры одноместных предикатов выражают свойства предметов.

Предикат $P(x)$, определенный на множестве M, называется *тождественно истинным*, если $I_P = M$, и *тождественно ложным*, если $I_P = \varnothing$.

Двухместным 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 $\{1, 0\}$.

Пример.

1. Бинарное отношение «меньше» на множестве целых чисел Z.

Это отношение между двумя предметами, оно может быть записано в виде высказывания «$x < y$», где $x, y \in Z$, т. е. является функцией двух переменных $P(x, y)$, определенный на множестве $\{1, 0\}$.

2. $Q(x, y)$ – «$x = y$» предикат равенства, определенный на множестве

$$R^2 = R \times R.$$

Аналогично определяется n – местный предикат.

Далее рассмотрим логические и кванторные операции над предикатами. Формулы логики предикатов, основные равносильности.

Предикаты, так же, как высказывания, принимают значения «и» и «л» (или 1 и 0), поэтому к ним применимы все операции логики высказываний.

Коньюнкцией двух предикатов $P(x)$ и $Q(x)$ называется новый предикат $P(x) \wedge Q(x)$, который принимает значение «и» при тех и только тех значениях $x \in M$, при которых каждый из предикатов принимает значение «и», и принимает значение «л» во всех остальных случаях.

Областью истинности предиката $P(x) \wedge Q(x)$ является общая часть областей истинности предикатов $P(x)$ и $Q(x)$, т. е. пересечение $I_P \cap I_Q$.

Пример.

$P(x)$ – «x – четное число», $Q(x)$ – «x кратно 3», тогда $P(x) \wedge Q(x)$ – «x – четное число и кратное 3», или «x – делится на 6».

Дизьюнкцией двух предикатов $P(x)$ и $Q(x)$ называется новый предикат $P(x) \vee Q(x)$, который принимает значение «л» при тех и только тех значениях $x \in M$, при которых каждый из предикатов принимает значение «л» и принимает значение «и» во всех остальных случаях.

Областью истинности предиката $P(x) \vee Q(x)$ является объединение областей истинности предикатов $P(x)$ и $Q(x)$, т. е. $I_P \cup I_Q$.

Отрицанием предиката $P(x)$ называется новый предикат $\overline{P(x)}$, который принимает значение «и» при всех значениях $x \in M$, при которых предикат $P(x)$ принимает значение «л» и наоборот.

Областью истинности предиката $\overline{P(x)}$ является $I_{\overline{P}} = M \setminus I_P$.

Импликацией предикатов $P(x)$ и $Q(x)$ называется новый предикат $P(x) \to Q(x)$, который является «л» при тех и только тех значениях $x \in M$, при которых одновременно $P(x)$ принимает значение «и», а $Q(x)$ – значение «л» и принимает значение «и» во всех остальных случаях.

Областью истинности будет $I_{P \to Q} = I_{\bar{P}} \cup I_Q$.

Квантор всеобщности.

Пусть $P(x)$ – предикат, определенный на множестве M. Выражение $\forall x P(x)$ – есть высказывание, которое является «и» для каждого $x \in M$ и «л» в противном случае. Это высказывание уже не зависит от x. Оно читается следующим образом: «Для всякого x, $P(x)$ истинно». Символ $\forall$ называется *квантором всеобщности*. Переменную x в предикате $P(x)$ называют *свободной*, а в высказывании $\forall x P(x)$ называют *связанной квантором всеобщности* $\forall$.

Квантор существования.

Пусть $P(x)$ – предикат, определенный на множестве M. Выражение $\exists x P(x)$ – есть высказывание, которое является истинным, если существует элемент $x \in M$, для которого $P(x)$ истинно, и ложно, в противном случае.

Читается: «Существует x для которого $P(x)$ истинно». Символ $\exists$ называется *квантором существования*. В высказывании $\exists x P(x)$ переменная x считается *связанной квантором существования* $\exists$.

Примеры употребления кванторов.

Пусть на множестве N задан предикат $P(x)$ – «число x кратно 5».

Используя кванторы $\forall$ и $\exists$ можно получить следующие высказывания:

1) $\forall x P(x)$ – «Все натуральные числа кратны 5». Это высказывание всегда ложно;

2) $\exists x P(x)$ – «Существует натуральное число, кратное 5». Это высказывание всегда истинно.

Кванторные операции применяются и к многоместным предикатам.

Для записи формул логики предикатов пользуются следующими символами:

1) $p, q, r, \ldots$ – переменные высказывания, которые принимают значения 0 или 1;

2) $x, y, z, \ldots$ – предметные переменные, которые принимают значения из некоторого множества M. $x^0, y^0, z^0, \ldots$ – предметные константы, т. е. значения предметных переменных;

3) $P(\cdot), F(\cdot), \ldots$ – одноместные предикаты,
$P(\cdot,\cdot, \ldots,), F(\cdot,\cdot, \ldots,)$ – n-местные предикаты,
$P^0(\cdot), P^0(\cdot,\cdot,\ldots,)$ – постоянные предикаты;

4) Символы логических операций: $\wedge, \vee, \to, -$;

5) Символы кванторных операций: $\forall x, \exists x$;

6) Вспомогательные символы: скобки ( ), запятые.

Определение формулы логики предикатов.

1. Каждое высказывание как переменное, так и постоянное, является формулой.

2. $F(x_1, x_2, \ldots, x_n)$ является формулой, если $F(\cdot,\cdot, \ldots,)$ – есть n-местный предикат (переменный или постоянный), а $x_1, x_2, \ldots, x_n$ – предметные переменные или постоянные.

3. Если A и B формулы, то $A \wedge B$, $A \vee B$, $A \to B$ – тоже формулы. Причем, переменные, которые в исходных формулах были свободными, остаются свободными, а связанные – остаются связанными.

4. Если A – формула, то $\overline{A}$ – тоже формула.

5. Если $A(x)$ – формула, в которую x входит свободно, то высказывания $\forall x A(x)$ и $\exists x A(x)$ являются формулами, причем переменная x входит в них связанно.

Все остальные высказывания, не удовлетворяющие 1–5, не являются формулами.

Примеры формул.

q, $P(x)$, $P(x) \wedge Q(x^0, y)$, $\forall x P(x) \to \exists x Q(x, y)$, $(\overline{Q(x, y)} \vee q) \to r$.

Не является формулой следующее выражение: $\forall x Q(x, y) \to P(x)$. Здесь нарушено условие п. 3, так как в это выражение переменная x входит связанно $(\forall x Q(x, y))$ и свободно $(P(x))$.

Замечание. Всякая формула алгебры логики высказываний является формулой логики предикатов.

Формула логики предикатов может принимать логическое значение, если задано множество M, на котором определены, входящие в эту формулу предикаты.

Логическое значение формулы зависит от:

1) значений, входящих в формулу переменных высказываний;

2) значений свободных переменных из множества M;

3) значений предикатных переменных.

При конкретных значениях, каждого из трех видов переменных, формула логики предикатов примет значение «и» или «л», (0, 1).

Две формулы логики предикатов A и B называются *равносильными* на области M, если они принимают одинаковые логические значения при всех значениях, входящих в них переменных, отнесенных к области М.

Две формулы логики предикатов A и B называются *равносильными*, если они равносильны на всей области M.

Основные равносильности логики предикатов:

Пусть $A(x)$, $B(x)$ – предикаты, C – высказывание.

1. $\overline{\forall x A(x)} \equiv \exists x \overline{A(x)}$, если не для всех x истинно $A(x)$, то $\exists x$ при котором истинно $\overline{A(x)}$.

2. $\overline{\exists x A(x)} \equiv \forall x \overline{A(x)}$, если не $\exists x$ при котором истинно $A(x)$, то для всех x будет истинно $\overline{A(x)}$.

3. $\forall x A(x) \equiv \overline{\exists x \overline{A(x)}}$ (следствие 1).

4. $\exists x A(x) \equiv \overline{\forall x \overline{A(x)}}$ (следствие 2).

5. $\forall xA(x) \wedge \forall xB(x) \equiv \forall x\left[A(x) \wedge B(x)\right]$.

6. $C \wedge \forall xB(x) \equiv \forall x\left[C \wedge B(x)\right]$.

7. $C \vee \forall xB(x) \equiv \forall x\left[C \vee B(x)\right]$.

8. $C \to \forall xB(x) \equiv \forall x\left[C \to B(x)\right]$.

9. $\forall x\left[B(x) \to C\right] \equiv \exists xB(x) \to C$.

10. $\exists x\left[A(x) \vee B(x)\right] \equiv \exists xA(x) \vee \exists xB(x)$.

11. $\exists x\left[C \vee B(x)\right] \equiv C \vee \exists xB(x)$.

12. $\exists x\left[C \wedge B(x)\right] \equiv C \wedge \exists xB(x)$.

13. $\exists xA(x) \wedge \exists yB(y) \equiv \exists x\exists y\left[A(x) \wedge B(y)\right]$.

14. $\exists x\left[C \to B(x)\right] \equiv C \to \exists xB(x)$.

15. $\exists x\left[B(x) \to C\right] \equiv \forall xB(x) \to C$.

Формулы логики предикатов также можно приводить с помощью равносильностей к нормальной форме.

Формулы логики предикатов, также как и формулы алгебры логики высказываний делятся на тождественно истинные, ложные, выполнимые и не выполнимые.

Формула A логики предикатов называется *выполнимой* в области M, если существуют значения переменных, входящих в эту формулу и отнесенных к области M, при которых формула A принимает истинные значения.

Формула A называется *выполнимой*, если существует область, на которой эта формула выполнима.

Из этого определения следует, что, если формула выполнима, то это еще не означает, что она выполнима в любой области.

Формула A называется *тождественно истинной* в области M, если она принимает тождественно истинные значения для всех значений переменных, входящих в эту формулу и отнесенных к этой области.

Формула A называется *общезначимой*, если она тождественно истинная на всякой области.

Формула A называется *тождественно ложной* в области M, если она принимает ложные значения для всех значений переменных, входящих в эту формулу и отнесенных к этой области.

Из приведенных определений следует:

1. Если формула A общезначима, то она и выполнима на всякой области.

2. Если формула A тождественно истинная в области M, то она и выполнима в этой области.

3. Если формула A тождественно ложная в области M, то она не выполнима в этой области.

4. Если формула A не выполнима, то она тождественно ложна на всякой области.

Отметим, что общезначимую формулу называют *логическим законом*.

Проблема разрешимости в логике предикатов также как и в алгебре логики состоит в том, чтобы определить к какому классу относится формула – т. е. является ли она общезначимой, выполнимой или тождественно ложной.

Замечание. В отличие от алгебры логики в логике предикатов не применим метод перебора всех вариантов значений переменных, входящих в формулу, так как таких вариантов может быть бесконечно много.

Проблема разрешимости в логике предикатов достаточно сложная задача и решается лишь в отдельных частных случаях.

Далее приведем примеры применения языка логики предикатов для записи математических предложений и определений.

1. Определение предела числовой последовательности:

$$a = \lim_{n\to\infty} a_n \Leftrightarrow \forall\varepsilon > 0\ \exists n_0\ \ \forall n \in N\ \ (n \ge n_0 \to |a_n - a| < \varepsilon).$$

Здесь использован трехместный предикат

$$Q(\varepsilon, n, n_0) : (n \ge n_0 \to |a_n - a| < \varepsilon).$$

2. Определение предела функции в точке:

$$b = \lim_{x\to x_0} f(x) \Leftrightarrow \forall\varepsilon > 0\ \exists\delta > 0\ \forall x \in E\ \ (0 < |x - x_0| < \delta \Rightarrow |f(x) - b| < \varepsilon).$$

Здесь использован трехместный предикат

$$P(\varepsilon, \delta, x) : (0 < |x - x_0| < \delta \Rightarrow |f(x) - b| < \varepsilon).$$

3. Определение непрерывности функции в точке: Функция $f(x)$, определенная на множестве E, непрерывная в точке $x_0 \in E$, если

$$\forall\varepsilon > 0\ \exists\delta > 0\ \forall x \in E\ \ (|x - x_0| < \delta \Rightarrow |f(x) - f(x_0)| < \varepsilon).$$

Здесь использован трехместный предикат

$$\mathrm{P}(\varepsilon, \delta, x) : (|x - x_0| < \delta \Rightarrow |f(x) - f(x_0)| < \varepsilon).$$

4. Определение возрастающей функции: Функция $f(x)$, определенная на множестве E, возрастает на этом множестве, если

$$\forall x_1 \in E\ \ \forall x_2 \in E\ \ (x_1 < x_2 \Rightarrow f(x_1) < f(x_2)).$$

Здесь использован двухместный предикат $Q(x_1, x_2)$: $(x_1 < x_2 \Rightarrow f(x_1) < f(x_2))$.

Задачи и упражнения

1. Какие из следующих предложений являются высказываниями:

а) Москва – столица России;

б) студент ФИСУ;

в) $\sqrt{3} + 2\sqrt{7} - 28$;

г) $a > 0$;

д) Сегодня – понедельник.

2. Приведите примеры предложений, а) являющихся высказываниями; б) не являющихся высказываниями.

3. Установите, истинно или ложно следующие высказывания:

а) $5 \in \left\{ x : 2x^3 - 3x - 235 = 0,\ x \in R \right\}$;

б) $-3 \in \left\{ x : \dfrac{x^3 - 1}{x^2 + 2} \le -2,\ x \in R \right\}$;

в) $\{1\} \in N$;

г) $\{1\} \in P(N)$, где $P(N)$ – множество всех подмножеств множества N;

д) $\varnothing \in \varnothing$;

е) $\varnothing \in \{\varnothing\}$.

4. Среди следующих высказываний указать элементарные и составные. В составных высказываниях выделить грамматические связки:

а) число 32 делится на 4;

б) число 28 делится на 4 и на 7;

в) если число 125 делится на 25, то оно делится на 5;

г) число 7 является делителем числа 42;

д) число 1 269 делится на 9 тогда и только тогда, когда 18 делится на 9.

5. Обозначьте элементарные высказывания буквами и запишите следующие высказывания с помощью символов алгебры логики:

а) 27 кратно 3 и 15 кратно 3;

б) 27 кратно 3 и 9 не кратно 3;

в) $\sqrt{25} = 5$ или $\sqrt{25} = -5$;

г) если число 120 делится на 3 и на 5, то оно делится на 15.

6. Пусть p и q обозначают высказывания:

p – «Я учусь в школе»,

q – «Я люблю математику».

Прочтите следующие сложные высказывания:

а) $\overline{p}$; б) $\overline{\overline{p}}$; в) $p \wedge q$; г) $p \wedge \overline{q}$; д) $\overline{p} \wedge q$; е) $\overline{p} \wedge \overline{q}$; ж) $\overline{p \wedge q}$.

7. Проверить, не составляя таблиц истинности, являются ли следующие формулы тождественно истинными:

а) $p \to p$;

б) $\overline{p \wedge \overline{p}}$;

в) $\overline{p} \to p$;

г) $p \vee \overline{p}$;

д) $p \leftrightarrow \overline{p}$;

е) $p \leftrightarrow p$;

ж) $(p \vee p) \to p$;

з) $\overline{p \wedge (p \leftrightarrow \overline{p})}$;

и) $(p \to p) \vee \overline{p}$;

к) $p \leftrightarrow p \wedge (\overline{p} \to p \wedge p)$;

л) $p \vee (p \leftrightarrow \bar{p})$;

м) $\overline{p \to \bar{p}}$;

н) $\overline{p \leftrightarrow \bar{p}}$;

о) $(p \vee p) \to (p \wedge p)$.

8. Найдите логические значения x и y, при которых выполняются равенства:

1) $(1 \to x) \to y = 0$; 2) $x \vee y = \bar{x}$.

9. Пусть $x = 0$, $y = 1$, $z = 1$. Определить логические значения нижеследующих сложных высказываний:

а) $x \wedge (y \wedge z)$;

б) $(x \wedge y) \wedge y)$;

в) $x \to (y \to z)$;

г) $x \wedge y \to z$;

д) $(x \wedge y) \leftrightarrow (z \vee \bar{y})$;

е) $((x \vee y) \wedge z) \leftrightarrow ((x \wedge z) \vee (y \wedge z))$.

10. а) Постройте с помощью отрицания и дизъюнкции формулу, таблица истинности для которой совпала бы с таблицей для импликации.

б) Аналогично этому постройте с помощью отрицания и импликации формулу, таблица истинности для которой совпадает с таблицей для дизъюнкции, и вторую формулу с таблицей, совпадающей с таблицей для коньюнкции.

11. Составить таблицы истинности для формул:

а) $\bar{x} \vee \bar{y}$;

б) $(x \vee y) \to (x \wedge \bar{y} \vee \bar{x} \to \bar{y})$

в) $(x \wedge y) \vee z$;

г) $x \wedge \bar{y} \to (y \vee \bar{x} \to \bar{z})$;

д) $(x \to \bar{y}) \to (\overline{x \vee y} \wedge \bar{z})$;

е) $(\bar{x} \vee z) \wedge (y \to (u \to x))$;

ж) $(x_1 \to (x_2 \to (... \to x_n)...)$.

12. Установить, какие из следующих формул являются тождественно истинными, тождественно ложными:

а) $\overline{\overline{x \vee y}} \to \overline{\overline{x \wedge y}}$;

б) $(x \to y) \to (\bar{y} \to \bar{x})$;

в) $\overline{x \to (y \to x)}$;

г) $\bar{x} \to (x \to y)$;

д) $\big((x \wedge y) \leftrightarrow y\big) \leftrightarrow \big(y \to x\big)$.

13. Доказать равносильность:

а) $\overline{x \to y} \equiv x \wedge \bar{y}$;

б) $\big(x \vee y\big) \wedge \big(x \vee \bar{y}\big) \equiv x$;

в) $x \vee \big(\bar{x} \wedge y\big) \equiv x \vee y$;

г) $x \leftrightarrow y \equiv \bar{x} \leftrightarrow \bar{y}$;

д) $x \to \bar{y} \equiv y \to \bar{x}$;

е) $x \to (y \to z) \equiv x \wedge y \to x$.

14. Упростить формулу:

а) $(x \to x) \to x$;

б) $x \to (x \to y)$;

в) $(x \leftrightarrow y) \wedge (x \vee y)$;

г) $(x \to y) \wedge (y \to z) \to (z \to x)$;

д) $(x \vee \bar{y} \to (z \to y \vee \bar{y} \vee x)) \wedge \left(x \vee \overline{x \to (x \to x)}\right) \to y$.

15. Доказать тождественную истинность или тождественную ложность формул:

а) $x \wedge y \to x$;

б) $x \to (x \vee y)$;

в) $(x \to y) \to (\bar{y} \to \bar{x})$;

г) $(\bar{y} \to \bar{x}) \to (x \to y)$;

д) $(x \to y) \wedge (x \to \bar{y}) \to \bar{x}$.

16. Для следующих формул найти СДНФ и СКНФ, каждую двумя способами (путем равносильных преобразований и таблиц истинности):

а) $x \wedge (x \to y)$;

б) $(x \to y) \to (y \to x)$;

в) $\left(x \vee \bar{z}\right) \to y \wedge z$;

г) $(x \vee \bar{y} \to x \wedge z) \to \left(\overline{x \to \bar{x}}\right) \vee y \wedge \bar{z}$;

д) $\left(\bar{a} \to c\right) \to \left(\overline{\bar{b} \to \bar{a}}\right)$.

17. Докажите равносильность формул $\overline{x\bar{y}} \to (\bar{y} \to x)$ и $\overline{x \to y} \vee x \vee y$ сравнением их совершенных нормальных форм (коньюнктивных и дизьюнктивных).

Глава 3

ТЕОРИЯ ГРАФОВ

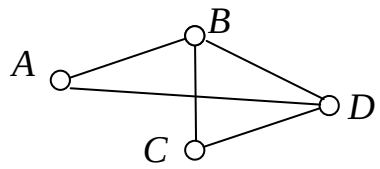
В этой главе излагаются основы теории графов (ориентированных и неориентированных). Приводятся задачи теории графов, являющиеся математическими моделями ряда прикладных задач. Методы их решения доведены до уровня простых алгоритмов, реализуемых на ЭВМ.

Введем основные понятия и рассмотрим способы задания графов.

Многие отношения на конечных множествах можно изобразить в виде рисунков. Например, отношение $x\,ry = \{(x, y) \in A: x < y\}$, где $A = \{1, 2, 3, 4\}$, или в явном виде $r = \{(1, 2), (1, 3), (1, 4), (2, 3). (2, 4), (3, 4)\}$, можно изобразить на плоскости в виде точек с координатами (x, y).

Представим на плоскости некоторое конечное множество V точек и конечный набор X линий, соединяющих некоторые пары точек из V. Простым примером подобной геометрической конфигурации может служить любая схема автомобильных дорог, связывающих города некоторой области.

Пример схемы автомобильных дорог между пунктами A, B, C, D, изображенной в виде графа:



Для многих задач оказывается несущественным, как соединяются точки – отрезками или дугами. Важно лишь то, что каждая линия соединяет какие-либо две точки из заданного набора точек.

Введенная пара множеств (V, X) допускает также и другие интерпретации и является предметом детального изучения в математике.

При рассмотрении подобных задач достаточно ограничиться исследованием совокупности двух конечных множеств V и X, где V – непустое множество, X – некоторый набор элементов из V, вида $\{v_i, v_j\}$.

Элементы множества V называются *вершинами*, а элементы множества X – *ребрами*.

В множестве X могут встречаться пары с одинаковыми элементами вида $\{v, v\}$, а также одинаковые пары.

Ребра вида $\{v, v\}$ называются *петлями*.

Одинаковые пары в X называются *кратными* (или *параллельными*) *ребрами*.

Количество одинаковых пар $\{v_i, v_j\}$ в X называются *кратностью* ребра $\{v_i, v_j\}$.

Про множество V и множество X будем говорить, что они определяют *граф с кратными ребрами и петлями* (или *псевдограф*) $G = (V, X)$.

Псевдограф без петель называется *графом с кратными ребрами* (или *мультиграфом*).

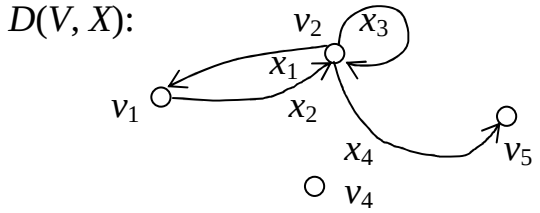
Если в X ни одна пара не встречается более одного раза, то мультиграф $G = (V, X)$ называется *графом*.

Если пары в наборе X являются упорядоченными, то граф $D = (V, X)$ называется *ориентированным* (или *орграфом*). Ребра орграфа называются *дугами*.

Если пары в наборе X неупорядочены, то граф называется *неориентированным графом* (или просто *графом*). Ребра в неориентированном графе будем обозначать $\{v_i, v_j\}$, дуги в ориентированном графе будем обозначать (v_i, v_j).

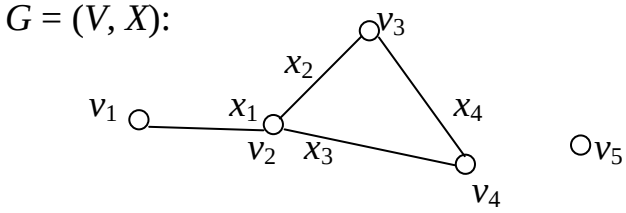
Примеры орграфа или графа.

Пусть $V = \{v_1, v_2, v_3, v_4\}$, $X = \{x_1 = (v_1, v_2), x_2 = (v_1, v_2), x_3 = (v_2, v_2), x_4 = (v_2, v_3)\}$, тогда $D = (V, X)$ – есть ориентированный псевдограф (есть одинаковые дуги, петля) и имеет следующий вид:



Пример.

Пусть $V = \{v_1, v_2, v_3, v_4, v_5\}$, $X = \{x_1 = \{v_1, v_2\}, x_2 = \{v_2, v_3\}, x_3 = \{v_2, v_4\}, x_4 = \{v_3, v_4\}\}$. Тогда $G = (V, X)$ – неориентированный граф (или просто граф) и имеет следующий вид:



Приведем ряд определений для ориентированных и неориентированных графов. Те же определения справедливы для ориентированных и неориентированных псевдографов.

Если $x = \{v_i, v_j\}$ – ребро графа, то вершины v_i и v_j называются *концами* ребра x.

Если $x = (v_i, v_j)$ – дуга орграфа, то вершина v_i называется *началом*, а вершина v_j – называется *концом* дуги x.

Если вершина v является концом ребра x, то говорят, что v и x *инцидентны*. *В орграфе* v и x *инцидентны*, если v является началом или концом дуги x.

Вершины v_i, v_j графа $G = (V, X)$ называются *смежными*, если $\{v_i, v_j\} \in X$.

Два ребра называются *смежными*, если они имеют общую вершину.

Степенью вершины v графа G называется число ребер $\delta(v)$ графа G, инцидентных вершине v.

Вершина графа, имеющая степень $\delta(v) = 0$, называется *изолированной*, а степень $\delta(v) = 1$ – называется *висячей*.

В рассмотренном *примере 2* вершина v_5 – изолированная; v_1 – висячая; концами ребра x_1 являются вершины v_1, v_2; вершина v_2 инцидентна ребрам x_1, x_2, x_3; степень вершины v_2 равна 3, $\delta(v_2) = 3$; вершины v_1 и v_2 – смежные; ребра x_1 и x_2 – смежные.

В ориентированном псевдографе из *примера 1* дуга x_1 исходит из вершины v_1 и заходит в вершину v_2; вершина v_2 инцидентна дугам x_1, x_2, x_3, x_4; вершина v_2 имеет 2 – исхода и 3 – захода.

Число исходов будем обозначать $\delta^+(v)$ – количество дуг, исходящих из вершины v.

Число заходов будем обозначать $\delta^-(v)$ – количество дуг, заходящих в вершину v.

Свойства графов:

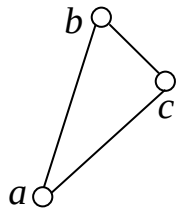
1. $\sum\limits_{v \in V} \delta(v) = 2|X|$, где $|X|$ – число ребер G.
2. В любом графе число вершин нечетной степени четно.
3. $\sum\limits_{v \in V} \delta^+(v) = \sum\limits_{v \in V} \delta^-(v) = |X|$.

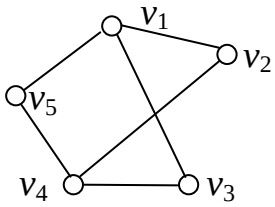
Графы можно задавать различными способами.

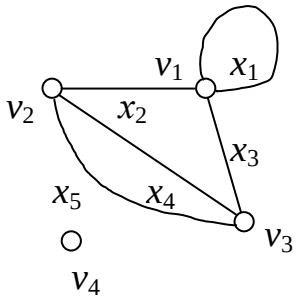
1. *Аналитический способ*: в этом случае перечисляются множества V и X.

Пример. $G = (V, X)$, $V = \{a, b, c\}$, $X = \{\{b, a\}, \{b, c\}, \{c, a\}\}$.

2. *Графический способ*: в этом случае все элементы множества V обозначаются точками (или кружками) на плоскости и если $\{v_i, v_j\} \in X$, то проводится линия, соединяющая вершины v_i и v_j. Например:







3. *Матричный способ*: матрицей *смежности* орграфа D называется квадратная матрица $A(G) = [a_{i,j}]$ порядка n, элементы которой определяются по следующей формуле

$$a_{i,j} = \begin{cases} 1, \text{ если } (v_i, v_j) \in X, \\ 0, \text{ если } (v_i, v_j) \notin X. \end{cases}$$

Замечание 1. Понятие матрицы смежности для неориентированного графа такое же. Отличие их состоит лишь в том, что матрица смежности для графа симметричная, а для орграфа нет.

Замечание 2. Матрицу смежности можно определить и для псевдографа. Тогда $a_{i,j} = k$, где k – кратность дуги (v_i, v_j) (или ребра).

Матрицей *инцидентности* графа G называется матрица $B(G) = [b_i, _j]$ размерности $(n \times m)$, элементы которой определяются по формуле

$$b_{i,j} = \begin{cases} 1, \text{ если вершина } v_i \text{ инцидентна ребру } x_j, \\ 0, \text{ если вершина } v_i \text{ не инцидентна ребру } x_j. \end{cases}$$

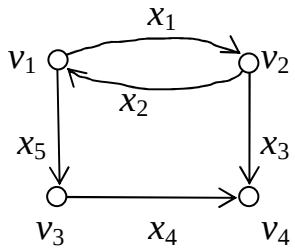
Определение матрицы инцидентности без изменения переносится на неориентированные мультиграфы и псевдографы.

Матрицей *инцидентности* орграфа D называется матрица $B(D) = [b_i, _j]$ размерности $(n \times m)$, элементы которой определяются по формуле

$$b_{i,j} = \begin{cases} 1, \text{ если вершина } v_i \text{ является концом дуги } x_j, \\ -1, \text{ если вершина } v_i \text{ является началом дуги } x_j, \\ 0, \text{ если вершина } v_i \text{ не инцидентна дуге } x_j. \end{cases}$$

Пример.

Для приведенного ниже графа записать матрицу смежности и инцидентности.



Решение:

Матрица смежности A и матрица инцидентности B будут иметь следующий вид:

матрица смежности:

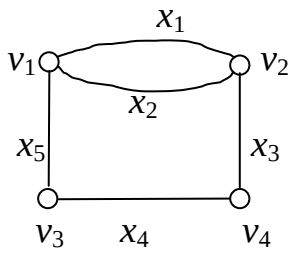
	v_1	v_2	v_3	v_4
v_1	0	1	0	1
v_2	1	0	1	0
v_3	0	0	0	0
v_4	0	0	1	0

матрица инцидентности:

	x_1	x_2	x_3	x_4	x_5
v_1	–1	1	0	0	–1
v_2	1	–1	–1	0	0
v_3	0	0	1	1	0
v_4	0	0	0	–1	1

Пример.

Для мультиграфа, построить матрицу смежности и инцидентности.



Матрица смежности A будет иметь вид

	v_1	v_2	v_3	v_4
v_1	0	2	0	1
v_2	2	0	1	0
v_3	0	1	0	1
v_4	1	0	1	0

Матрица инцидентности B будет иметь вид

	x_1	x_2	x_3	x_4	x_5
v_1	1	1	0	0	1
v_2	1	1	1	0	0
v_3	0	0	1	1	0
v_4	0	0	0	1	1

По матрице смежности графа (орграфа) всегда можно определить ребра графа (дуги орграфа) как пары инцидентных вершин, а для псевдографов, кроме того, и кратности ребер (дуг). Однако если ребра (дуги) были пронумерованы, то восстановить их номера по матрице смежности невозможно. В этом смысле матрица инцидентности является более информативной, чем матрица смежности, так как позволяет получить полную информацию о ребрах (дугах), включая их нумерацию.

С помощью введенных матриц удобно задавать графы (орграфы) для обработки на ЭВМ. Однако следует отметить, что при большом количестве вершин матрица смежности оказывается громоздкой и число элементов в ней может превысить допустимый объем оперативной памяти ЭВМ. То же можно сказать и о матрице инцидентности, причем ее размеры зависят, кроме того, и от количества ребер (дуг).

Свойства матриц смежности и инцидентности.

1. Сумма элементов матрицы смежности $A(G)$, где $G(V, X)$ – мультиграф, $V = \{v_1, v_2, \ldots, v_n\}$, по i-й строке (или по j-му столбцу) равна $\delta(v_i)$.

2. Сумма элементов матрицы смежности $A(G)$, где $G(V, X)$ – ориентированный псевдограф, $V = \{v_1, v_2, \ldots, v_n\}$, по i-й строке и по j-му столбцу равны $\delta^+(v_i)$, $\delta^-(v_i)$.

3. Пусть G – ориентированный мультиграф с непустым множеством дуг. Тогда:

а) сумма строк матрицы инцидентности $B(G)$ является нулевой строкой;

б) любая строка матрицы инцидентности $B(G)$ является линейной комбинацией остальных строк;

в) ранг матрицы инцидентности $B(G)$ не превосходит $(n-1)$;

г) для любого контура в орграфе G сумма столбцов матрицы $B(G)$ соответствующих дугам, входящим в этот контур, равна нулевому столбцу.

§ 1. Маршруты, пути. Поиск маршрутов в графе

При решении некоторых прикладных задач возникает необходимость найти маршрут, соединяющий заданные вершины в графе G. В этом параграфе введем основные понятия и приведем различные алгоритмы решения этой задачи.

Понятие маршрута определяется для графа $G(V, X)$; понятие пути – для орграфа $D(V, X)$.

Последовательность $v_1x_1v_2x_2v_3\ldots x_kv_{k+1}$, (где $k \geq 1$, $v_i \in V$, $i = 1, \ldots k+1$, $x_j \in X$, $j = 1, \ldots, k$) в которой чередуются вершины и ребра (дуги) и для каждого $j = 1, \ldots, k$ ребро (дуга) x_j имеет вид $\{v_j, v_{j+1}\}$ $((v_j, v_{j+1}))$, называется маршрутом, соединяющим вершины v_1, v_{k+1} (путем из v_1 в v_{k+1}).

При этом v_1 называется *начальной*, а v_{k+1} – *конечной* вершинами маршрута (пути), а остальные вершины – *внутренними*.

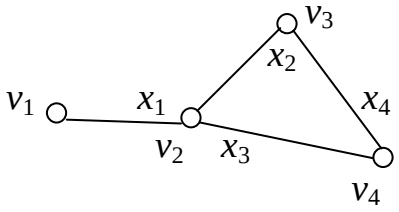
Одна и та же вершина может одновременно оказаться начальной, конечной и внутренней.

Последовательность вершин в маршруте определяет ориентацию.

Например, запись $\{v_2, v_3\}$ указывает на то, что ребро x_2 ориентировано от вершины v_2 к вершине v_3.

Пример.

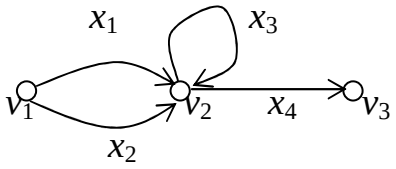
Рассмотрим граф



Последовательность $v_1x_1v_2x_3v_4x_4v_3$ – маршрут, соединяющий вершины v_1 и v_3.

Пример.

Рассмотрим ориентированный граф



Последовательность $v_1x_2v_2x_3v_2x_4v_3$ – путь из v_1 в v_3.

Замечание. Можно использовать также более краткие записи, например в графе указать тот же маршрут можно следующей последовательностью ребер $x_1x_3x_4$ или последовательностью вершин $v_1v_2v_4v_3$. Для ориентированного псевдографа можно использовать сокращенные записи: $x_2x_3x_4$ или $v_1x_2v_2v_2v_3$.

Число ребер (дуг) в маршруте (пути) называется *длиной* маршрута (пути).

Маршрут (путь) называется *замкнутым*, если его начальная вершина совпадает с конечной, т. е. $v_1 = v_{k+1}$.

Замечание. Далее всюду при подсчете числа вхождений вершин в замкнутый маршрут (путь) начальную и конечную вершины будем считать за одно вхождение этой вершины в маршрут (путь).

Замечание. При замкнутом маршруте (пути) $x_1x_2\ldots x_k$ последовательности $x_2\ldots x_kx_1$ или $x_3\ldots x_kx_1x_2$ – это просто различные записи одного и того же маршрута (пути).

Незамкнутый маршрут (путь), в котором все ребра (дуги) попарно различны, называется *цепью*.

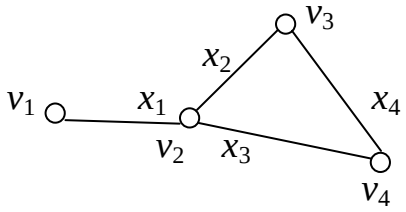
Цепь, в которой все вершины попарно различны, называется *простой цепью*.

Замкнутый маршрут (путь), в котором все ребра (дуги) попарно различны, называется *циклом* (*контуром*).

Цикл (контур), в котором все вершины попарно различны называется *простым циклом*.

Пример.

Рассмотрим граф



1. $v_1x_1v_2x_3v_4x_4v_3$ – маршрут длины 3, соединяющий вершины v_1 и v_3; это простая цепь, так как все ребра и вершины попарно различны.

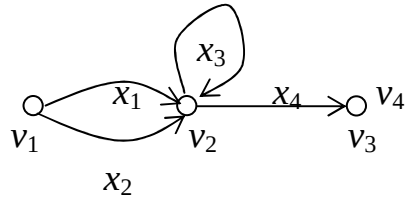
2. $v_2x_2v_3x_4v_4x_3v_2$ – замкнутый маршрут длины 3; это простой цикл, так как все ребра и вершины попарно различны.

3. $v_1x_1v_2x_2v_3x_4v_4x_3v_2$ – маршрут длины 4, соединяющий вершины v_1 и v_2; это цепь не является простой, так как вершина v_2 встречается дважды.

4. $v_1x_1v_2x_2v_3x_2v_2$ – маршрут длины 3, соединяющий вершины v_1 и v_2 и не являющийся цепью, так как ребро $x_2 = \{v_2, v_3\}$ встречается дважды.

Пример.

Рассмотрим ориентированный псевдограф



1. $v_1x_1v_2x_4v_3$ – путь длины 2 из вершины v_1 в v_3; это простая цепь, так как все дуги и вершины попарно различны.

2. $v_2x_3v_2$ – простой контур длины 1.

3. $v_1x_2v_2x_3v_2x_4v_3$ – цепь из v_1 в v_3, длины 3, которая не является простой.

Утверждение 1. В псевдографе (в ориентированном псевдографе) из всякого цикла (контура) можно выделить простой цикл (простой контур).

Утверждение 2. Из всякого незамкнутого маршрута (пути) можно выделить простую цепь с теми же начальной и конечной вершинами.

Подграфом графа G называется граф, все вершины и ребра которого содержатся среди вершин и ребер графа G. Определение аналогично и для орграфа.

Подграф называется *собственным*, если он отличен от самого графа.

Связность. Матрица связности

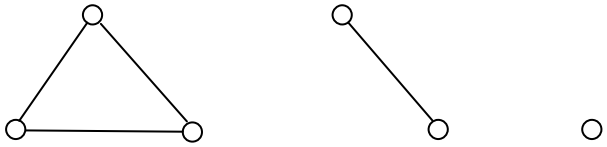
Вершина w графа (орграфа) *достижима* из вершины v, если $v = w$ или существует маршрут из v в w (путь, соединяющий v и w).

Граф (орграф) называется *связным* (*сильно связным*), если для любых двух его вершин v, w существует маршрут (путь), соединяющий v, w (из v в w).

Компонентой связности (*сильной связности*) графа (орграфа) называется его связный (сильно связный) подграф, не являющийся собственным подграфом никакого другого связного (сильно связного) подграфа графа G (орграфа).

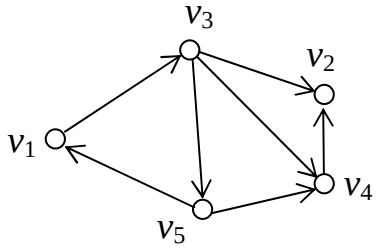
Пример.

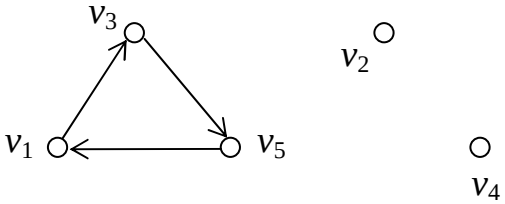
Граф, изображенный на рисунке, имеет три компоненты связности.



Пример.

Орграф, изображенный на рисунке, имеет три компоненты сильной связности:





Из определения компоненты связности (сильной связности) заключаем, что справедливы следующие утверждения.

1. Пусть $G_1 = (V_1, X_1)$ – компонента связности графа G. Тогда G_1 – подграф графа G, порожденный множеством V_1.

2. Пусть $G_1 = (V_1, X_1)$ – компонента сильной связности графа G. Тогда G_1 – подграф графа G, порожденный множеством V_1.

Матрицей достижимости орграфа D называется квадратная матрица $T(D) = [t_{ij}]$ порядка n, элементы которой определяются по следующей формуле:

$$t_{i,j} = \begin{cases} 1, \text{ если } v_j \text{ достижима из } v_i, \\ 0, \text{ в противном случае.} \end{cases}$$

Матрицей сильной связности орграфа D называется квадратная матрица $S(D) = [s_{i,j}]$ порядка n, элементы которой определяются по следующей формуле

$$s_{i,j} = \begin{cases} 1, \text{если вершина } v_j \text{ достижима из } v_i \, u, \text{ одновременно, } v_i \text{ достижима из } v_j, \\ 0, \text{ в противном случае.} \end{cases}$$

Замечание. $s_{ij} = 1$, тогда и только тогда, когда вершины v_i и v_j принадлежат одной компоненте сильной связности орграфа G.

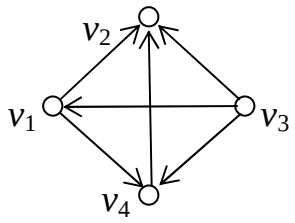
Матрицей связности графа G называется квадратная матрица $S(G) = [s_{ij}]$ порядка n, элементы которой определяются по следующей формуле

$$s_{i,j} = \begin{cases} 1, \text{ если } i = j \text{ или существует маршрут из вершины } v_i \text{ в вершину } v_j, \\ 0, \text{ в противном случае.} \end{cases}$$

Замечание. $s_{i,j} = 1$, тогда и только тогда, когда вершины v_i и v_j принадлежат одной компоненте связности графа G.

Пример.

Определить матрицы достижимости и сильной связности для орграфа, изображенного на рисунке



Решение:

$$T = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \qquad S = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

При решении некоторых прикладных задач нередко возникает необходимость найти маршрут, соединяющий заданные вершины в графе G. Эту задачу можно решить с помощью алгоритма Тери.

Пусть задан связный неориентированный граф $G = (V, X)$.

Требуется найти маршрут, соединяющий вершины v и w, принадлежащие X, причем $v \neq w$.

Алгоритм Тери. Необходимо, исходя из вершины v и осуществляя последовательный переход от каждой достигнутой вершины к смежной ей вершине, выполнять следующие правила:

1. Проходя по произвольному ребру, отмечать направление, в котором оно было пройдено.

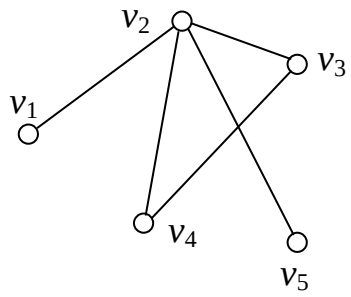
2. Исходя из некоторой вершины v', всегда следовать только по тому ребру, которое не было пройдено или было пройдено в противоположном направлении.

3. Для всякой вершины v', отличной от v, отмечать первое заходящее в v' ребро, если вершина v' встречается в первый раз.

4. Исходя из некоторой вершины v', отличной от v, по первому заходящему в v' ребру идти лишь тогда, когда нет других возможностей.

Пример.

Используя алгоритм Тери, найти маршрут, соединяющий вершины v_1 и v_5, в следующем связном неориентированном графе:



Решение:

Эти пути будут следующие:

1) $v_1v_2v_3v_4v_2v_5$ – длина пути 5;

2) $v_1v_2v_4v_3v_2v_5$ – длина пути 5.

Как видим из примера, пути могут определяться неоднозначно.

При решении прикладных задач возникает необходимость найти путь из v в w ($v \neq w$), который содержит наименьшее число дуг, т. е. является минимальным путем из всех возможных. Эту задачу можно решить с помощью алгоритма «Фронт волны».

Путь в орграфе из вершины v в w ($v \neq w$), называется *минимальным*, если он имеет минимальную длину среди всех путей орграфа из v в w. Аналогично определяется и *минимальный маршрут*.

Пусть задан орграф $D(V, X)$, имеющий n вершин ($n \geq 2$).

Требуется найти путь из v в w ($v \neq w$), который содержит наименьшее число дуг, т. е. является минимальным путем из всех возможных.

Алгоритм «Фронт волны». 1. Пометить вершину v индексом 0, (v – начальная или заданная вершина).

2. Пометить вершины, смежные с v, индексом 1 (такие вершины называют – образы вершины v). Вершины с индексом 1 обозначим через множество $W_1(v) = \{v_i : (v, v_i) \in X\}$. Полагаем $k = 1$.

3. Пусть построено множество $W_k(v)$. Если $W_k(v) = \varnothing$, или $w \notin W_k(v)$ для $k = n - 1$, то путь из v в w не существует, т. е. вершина w не достижима из v. Работа алгоритма на этом заканчивается.

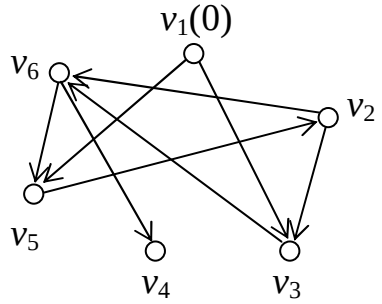
В противном случае (т. е., если $w \in W_k(v)$) путь существует, длины k, и этот путь является минимальным. Чтобы записать его, переходим к следующему шагу.

4. Путь записывается в виде последовательности вершин $vw_1w_2\ldots w_{k-1}w$, где

$$w_{k-1} \in W_{k-1}(v) \cap W^{-1}(w);$$
$$w_{k-2} \in W_{k-2}(v) \cap W^{-1}(w_{k-1});$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$w_1 \in W_1(v) \cap W^{-1}(w_2).$$

Это и есть искомый путь и на этом работа алгоритма заканчивается.
Множество $W_k(v)$ – называют **фронтом волны** k-го уровня.

Пример.
Найти минимальный путь в орграфе G из v_1 в v_4, изображенном на рисунке.



Помечаем $v_1 - 0$.

Матрица смежности заданного графа имеет вид

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

Строим множества $W_k(v_1)$:
$W_1(v_1) = \{v_3, v_5\}$, $W_1 \neq \varnothing$, $v_4 \notin W_1$,
$W_2(v_1) = \{v_2, v_6\}$, $W_1 \neq \varnothing$, $v_4 \notin W_2$,
$W_3(v_1) = \{v_4\}$, $W_1 \neq \varnothing$, $v_4 \in W_3$.
Минимальный путь существует и длина его равна 3 ($k = 3$).

Запишем этот путь: $v_1w_1w_2v_4$. Определим неизвестные вершины w_1, w_2.

$$w_2 = W_2(v_1) \cap W^{-1}(v_4) = \{v_2, v_6\} \cap \{v_6\} = \{v_6\},$$

$$w_1 = W_1(v_1) \cap W^{-1}(v_6) = \{v_3, v_5\} \cap \{v_2, v_3\} = \{v_3\}.$$

Итак, имеем путь $v_1v_3v_6v_4$. Этот путь минимальный, длина его равна 3.

Замечание. В общем случае, вершины могут быть выделены неоднозначно.

§ 2. Поиск минимального пути в нагруженном орграфе (графе). Алгоритм Форда–Беллмана

Орграф $D(V, X)$ называется *нагруженным*, если каждой дуге (v_i, v_j) поставлено в соответствие неотрицательное число $l_{i,j}$, называемое *весом дуги* (или *длиной дуги*).

Длиной пути в нагруженном орграфе называется сумма длин дуг, входящих в путь.

Путь с наименьшей суммой длин дуг называется *путем минимального веса*.

Замечание. Аналогично определяется минимальный маршрут в нагруженном графе.

Свойства минимальных путей (маршрутов) в нагруженном графе.

1. Если для $\forall\, x \in X$, $l(x) > 0$, то любой минимальный путь (маршрут) является простой цепью.

2. Если $v_1v_2v_3 \ldots v_k$ – минимальный путь (маршрут), то для любых номеров i, j, таких что $1 \le i < j \le k$, путь (маршрут) $v_iv_{i+1} \ldots v_j$, также является минимальным.

Рассмотрим задачу поиска минимальных путей (маршрутов) в нагруженном орграфе (графе). Все рассуждения будем проводить для орграфа (для графа они аналогичны).

Пусть задан нагруженный орграф $D(V, X)$. Требуется найти минимальный путь из v в w.

Введем матрицу длин дуг нагруженного графа $C = [c_{ij}]$ порядка n, элементы которой определяются по формуле

$$c_{i,j} = \begin{cases} l_{i,j}, \text{ если } (v_i, v_j) \in X, \\ \infty, \text{ если } (v_i, v_j) \notin X. \end{cases}$$

Введем величины λ_i^k, где $i = 1, \ldots, n$, $k = 1, 2, \ldots$. Для каждых фиксированных i и k величина λ_i^k равна длине минимального пути среди путей из v_1 в v_i, содержащих не более k дуг.

Если таких путей нет, то $\lambda_i^k = \infty$.

Если произвольную вершину $v \in V$ считать путем из v в v нулевой длины, то величины λ_i^k при $k = 0$ будут равны: $\lambda_1^0 = 0$, $\lambda_i^0 = \infty$, $i = 2, 3, \ldots, n$.

Величины λ_i^k вычисляются по формулам:

а) для $i = 2, 3, \ldots, n$, $k \geq 0$

$$\lambda_i^{k+1} = \min_{1 \leq j \leq n} \{\lambda_j^k + c_{ij}\};$$

б) для $i = 1$, $k \geq 0$

$$\lambda_1^{k+1} = \min\left\{0, \min_{1 \leq j \leq n} \{\lambda_j^k + c_{j1}\}\right\}.$$

Величины λ_i^k лучше всего записывать в виде матрицы (i – номер строки, ($k + 1$) – номер столбца), начиная с $k = 0$ и т. д.

Если при некотором k_0, $0 \leq k_0 \leq n - 2$ выполняются равенства $\lambda_i^{k_0} = \lambda_i^{k_0+1}$, $i = 1, 2, \ldots, n$, то и для $\forall k > k_0$, $\lambda_i^k = \lambda_i^k$, $i = 1, 2, \ldots, n$, тогда следует оборвать процесс определения наборов величин $\lambda_1^k, \ldots, \lambda_n^k$, на значении $k = k_0$.

Алгоритм Форда–Беллмана. Этот алгоритм позволяет по таблице λ_i^k ($i = 1, \ldots, n$, $k = 0, \ldots, n - 1$) определить минимальный путь из v_1 в любую достижимую вершину v_{i_1}, где i_1 – номер конечной вершины.

1 шаг. Строится матрица длин дуг C и таблица λ_i^k – длина минимального пути из вершины v_1 в v_{i_1}, с числом дуг не более k. Если $\lambda_{i_1}^{n-1} = \infty$, то вершина v_{i_1} не достижима из v_1 и на этом алгоритм заканчивается. Если $\lambda_{i_1}^{n-1} < \infty$, то переходим к следующему шагу.

2 шаг. $\lambda_{i_1}^{n-1} < \infty$, следовательно маршрут существует и $\lambda_{i_1}^{n-1}$ – длина этого маршрута. Минимальное число дуг k_1 этого маршрута определяется из условия $\lambda_{i_1}^{n-1} = \lambda_{i_1}^{k_1}$.

3 шаг. На этом шаге последовательно определяются номера вершин i_2, i_3, $\ldots$, i_{k_1}, из следующих условий:

$\lambda_{i_2}^{k_1-1} + c_{i_2 i_1} = \lambda_{i_1}^{k_1}$, отсюда определяем i_2;

$\lambda_{i_3}^{k_1-2} + c_{i_3 i_2} = \lambda_{i_2}^{k_1-1}$, отсюда определяем i_3;

$\lambda_{i_4}^{k_1-3} + c_{i_4 i_3} = \lambda_{i_3}^{k_1-2}$, отсюда определяем i_4;

$\lambda_{i_{k_1}}^{1} + c_{i_{k_1} i_{k_1-1}} = \lambda_{i_{k_1-1}}^{2}$, отсюда определяем i_{k_1},

и маршрут записывается следующим образом:

$$v_1 v_{i_{k_1}} v_{i_{k_1-1}} \ldots v_{i_3} v_{i_2} v_{i_1}.$$

Это и есть минимальный путь из v_1 в v_{i_1}, имеющий k_1 дуг, длиной $\lambda_{i_1}^{k_1}$.

Алгоритм закончен.

Пример.

Найти минимальный путь из v_1 в v_6, если орграф задан матрицей длин дуг, которая представлена следующей таблицей.

	V_1	V_2	V_3	V_4	V_5	V_6
V_1	∞	4	1	∞	5	10
V_2	∞	∞	∞	3	∞	∞
V_3	∞	2	∞	∞	2	∞
V_4	∞	∞	∞	∞	∞	2
V_5	∞	∞	∞	5	∞	∞
V_6	∞	∞	∞	∞	∞	∞

Решение:

Справа от матрицы припишем 6 столбцов значений λ_i^k (i = 1, 2, …, 6, k = 0, 1, …, 5), которые будем определять по формулам, приведенными в *шаге 1* алгоритма Форда–Беллмана.

	V_1	V_2	V_3	V_4	V_5	V_6
V_1	∞	4	1	∞	5	10
V_2	∞	∞	∞	3	∞	∞
V_3	∞	2	∞	∞	2	∞
V_4	∞	∞	∞	∞	∞	2
V_5	∞	∞	∞	5	∞	∞
V_6	∞	∞	∞	∞	∞	∞

λ_i^0	λ_i^1	λ_i^2	λ_i^3	λ_i^4	λ_i^5
0	0	0	0	0	0
∞	4	3	3	3	3
∞	1	1	1	1	1
∞	∞	7	6	6	6
∞	5	3	3	3	3
∞	10	10	9	8	8

$k = 0$, $i = 1$; $\quad \lambda_1^1 = \min\left\{0, \min\limits_{1\le j\le 6}\left\{\lambda_j^0 + c_{j1}\right\}\right\} = 0.$

$i = 2$; $\quad \lambda_2^1 = \min\limits_{1\le j\le 6}\left\{\lambda_j^0 + c_{j2}\right\} = \min\left\{\begin{matrix}4\\ \infty\\ \infty\\ \infty\\ \infty\\ \infty\end{matrix}\right\} = 4.$

$i = 3$; $\quad \lambda_3^1 = \min\limits_{1\le j\le 6}\left\{\lambda_j^0 + c_{j3}\right\} = \min\left\{\begin{matrix}1\\ \infty\\ \infty\\ \infty\\ \infty\\ \infty\end{matrix}\right\} = 1$ и т. д.

$k = 1, i = 1;$ $\lambda_1^2 = \min\left\{0, \min\limits_{1\le j\le 6}\left\{\lambda_j^1 + c_{j1}\right\}\right\} = 0.$

$i = 2;$ $\lambda_2^2 = \min\limits_{1\le j\le 6}\left\{\lambda_j^1 + c_{j2}\right\} = \min\left\{\begin{matrix}4\\ \infty\\ 3\\ \infty\\ \infty\\ \infty\end{matrix}\right\} = 3.$

$i = 3;$ $\lambda_3^2 = \min\limits_{1\le j\le 6}\left\{\lambda_j^1 + c_{j3}\right\} = \min\left\{\begin{matrix}1\\ \infty\\ \infty\\ \infty\\ \infty\\ \infty\end{matrix}\right\} = 1$ и т. д.

Шаг 2.

Находим минимальное k_1 для которого $\lambda_6^{k_1} = \lambda_6^5$: $k_1 = 4$, следовательно, минимальное количество дуг в пути равно 4.

Шаг 3.

Так как $i_1 = 6$ (номер конечной вершины), то далее найдем номера вершин i_2, i_3 и i_4 из соотношений:

$\lambda_{i_2}^3 + c_{i_2 6} = \lambda_6^4 = 8,$ следовательно $i_2 = 4$;

$\lambda_{i_3}^2 + c_{i_3 4} = \lambda_4^3 = 6,$ следовательно $i_3 = 2$;

$\lambda_{i_4}^1 + c_{i_4 2} = \lambda_2^2 = 3,$ следовательно $i_4 = 3$;

$\lambda_{i_5}^0 + c_{i_5 3} = \lambda_3^1 = 1,$ следовательно $i_5 = 1$.

Итак, получили минимальный путь v1v3v2v4v6, длина которого равна 8, причем он содержит минимальное количество дуг, равное 4, среди всех возможных минимальных путей из v1 в v6.

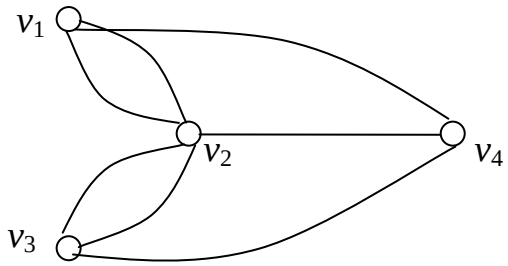
§ 3. Деревья и циклы

Классической задачей, которая привела к понятию эйлеровой цепи (цикла), является задача о Кенигсбергских мостах (XVIII век).

Задача состоит в следующем: г. Кенигсберг расположен на двух берегах реки и двух островах, соединенными между собой семью мостами.

Требуется осуществить прогулку по городу таким образом, чтобы пойти по одному разу по каждому мосту и вернуться в исходное место.

Для этой задачи можно построить неориентированный граф, где v_1, v_2, v_3, v_4 – это суша (берега и острова), ребра – мосты.



Пусть задан псевдограф $G = (V, X)$.

Цепь (цикл) называется *эйлеровой* (*эйлеровым*), если она (он) проходит по одному разу через каждое ребро псевдографа G.

На языке теории графов, задача будет звучать следующим образом: найти эйлеров цикл в мультиграфе G.

Решение этой задачи было приведено Эйлером. Он доказал следующие теоремы о существовании эйлерова цикла и цепи.

Теорема 1.

Связный неориентированный псевдограф G имеет эйлеров цикл тогда и только тогда, когда степени всех его вершин четные.

Теорема 2.

Связный неориентированный псевдограф G имеет эйлерову цепь тогда и только тогда, когда ровно две вершины имеют нечетную степень.

Перешейком называется ребро, при удалении которого, связный граф становится двусвязным, с компонентами связности, имеющими хотя бы по одному ребру.

Алгоритм построения эйлерова цикла. Пусть задан граф G, степени всех вершин которого четные.

Шаг 1. Выбрать произвольную вершину v и ребро, инцидентное этой вершине. Присвоить ему номер 1.

Шаг 2. Каждое пройденное ребро вычеркивать и присваивать ему номер на единицу больше номера предыдущего вычеркнутого ребра.

Шаг 3. Находясь в вершине v_i нельзя идти по ребру, которое является перешейком.

Шаг 4. Находясь в вершине v_i нельзя идти по ребру, соединяющему v_i и v, пока есть другие возможности.

Шаг 5. Если в графе занумерованы все ребра, то последовательность n ребер образует эйлеров цикл.

Пусть G – псевдограф.

Цепь (цикл) в G называется *гамильтоновой* (*гамильтоновым*), если она (он) проходит через каждую вершину псевдографа G ровно один раз.

С понятием гамильтоновых циклов тесно связана так называемая задача коммивояжера.

Задача коммивояжера.

Коммерсант должен совершить поездку по городам и вернуться обратно, побывав в каждом городе ровно один раз, и при этом его затраты на поездку должны быть минимальными.

Граф называется *полным*, если каждая его вершина смежна со всеми остальными вершинами.

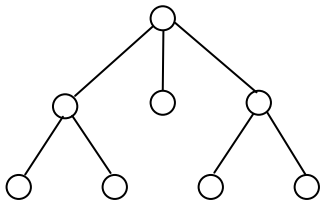
Утверждение. В полном графе всегда существует гамильтонов цикл.

Более ста лет ученые решают вопрос, каковы должны быть необходимые и достаточные условия, чтобы граф был гамильтоновым. Но до сих пор вопрос этот остается открытым. Вопрос о существовании гамильтонова цикла достаточно сложный и в курсе дискретной математике не рассматривается.

Деревья и их свойства

Граф G называется *деревом*, если он является связным и не имеет циклов.

Например:



Дерево не имеет петель, кратных дуг.

Свойства деревьев:

1. Следующие утверждения являются эквивалентными:

а) граф $G = (V, X)$ – есть дерево, $|V| = n$;

б) граф G является связным и не содержит циклов;

в) граф G является связным и число его ребер равно $(n – 1)$;

г) любые две вершины графа G соединены между собой цепью и только одной;

д) граф G – связный граф, но он утрачивает это свойство, если удалить любое ребро;

е) граф G не содержит циклов, но добавляя к нему любое новое ребро, получим ровно один и притом простой цикл;

2. Всякое дерево имеет по крайней мере две висячие вершины.

3. Граф G – является деревом тогда и только тогда, когда определитель квадратной матрицы порядка $(n – 1)$, полученный из матрицы инцидентности вычеркиванием i-й строки ($i = 1, \ldots, n$), равен ±1. В остальных случаях этот определитель равен 0.

Остовным деревом связного графа G называется любой его подграф, содержащий все вершины графа G и являющийся деревом.

Алгоритм выделения остовного дерева связного графа *G*.

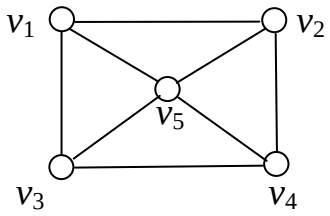
Шаг 1. Выбираем в G произвольную вершину v_1, которая образует подграф G_1 псевдографа G. Полагаем $i = 1$.

Шаг 2. Если $i = n$, то G_n – остовное дерево псевдографа G. В противном случае перейти к шагу 3.

Шаг 3. Строим граф G_{i+1}, добавляя к графу G_i новую вершину $v_{i+1} \in V$, смежную с некоторой вершиной v_j графа G_i, и новое ребро (v_j, v_{i+1}). Присваиваем $i = i + 1$ и переходим к шагу 2.

Пример.

Выделить остовное дерево связного графа, используя алгоритм.

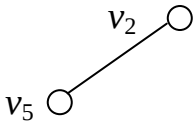


1. Выбираем произвольную вершину $v_5 = 1$. Получили подграф G_1, псевдографа G. Полагаем $i = 1$:

$i = 1$.

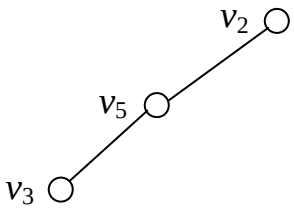
2. $1 \neq 5$.

3. Выбираем новую вершину, смежную с v_5 графа G_1, например v_2, и строим новое ребро (v_5, v_2). Полагаем $i = i + 1 = 2$. Получили граф G_2:



4. Переходим к шагу 2. $2 \neq 5$.

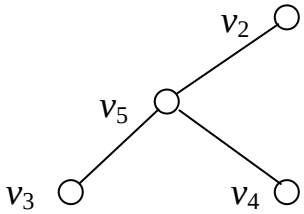
5. Выполняем шаг 3. Добавляем к графу G_2 смежную вершину v_3 и новое ребро (v_5, v_3). Полагаем $i = i + 1 = 3$. Получили граф G_3:



6. $3 \neq 5$.

7. Добавляем к графу G_3 смежную вершину v_4 и новое ребро (v_5, v_4).

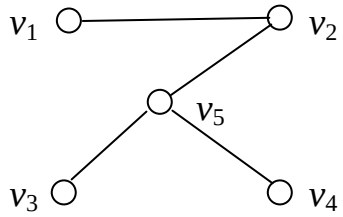
Полагаем $i = i + 1 = 4$. Получили граф G_4:



8. $4 \neq 5$.

9. Добавляем к графу G_4 смежную вершину v_1 и новое ребро (v_2, v_1).

Полагаем $i = i + 1 = 5$. Получили граф G_5:



10. 5 = 5. G_5 – остовное дерево.

Замечание. Остовное дерево выделяется неоднозначно.

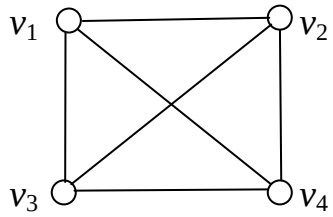
Теорема.

Число различных остовных деревьев связного графа равно минору любого из элементов главной диагонали квадратной матрицы $D–A$, где A – матрица смежности; D – диагональная матрица, у которой элемент d_{ii} равен степени i-й вершины.

Замечание. Для полного графа, имеющего n вершин, число остовных деревьев равно n^{n-2}.

Пример.

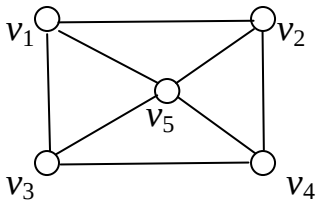
Пусть имеется полный граф G с $n = 4$ вершинами:



Число остовных деревьев равно $4^{4-2} = 4^2 = 16$.

Вектор – циклы. Цикловой базис. Цикломатическая матрица. *Цикломатическим числом* $\nu(G)$ графа $G = (V, X)$ с n вершинами и m ребрами называется число $\nu(G) = m – n + p$, где p – число компонент связности.

Пример.



$\nu(G) = m – n + p = 8 – 5 + 1 = 4.$

Смысл цикломатического числа – сколько нужно удалить ребер из связного графа, чтобы граф превратился в остовное дерево.

Свойства цикломатического числа.

1. Для любого графа справедливо неравенство $\nu(G) \geq 0$.
2. Равенство $\nu(G) = 0$ эквивалентно отсутствию циклов в графе G.

В дальнейшем под «циклом» будем понимать произвольный замкнутый маршрут.

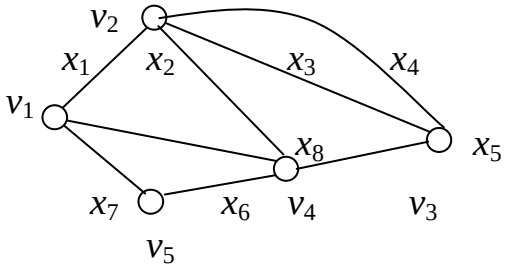
Пусть $G = (V, X)$ – неориентированный граф с количеством ребер равным m. Введем в графе G произвольную ориентацию ребер, т. е. придадим каждому ребру произвольное направление.

В графе G выберем произвольный цикл μ. Если цикл μ проходит i-е ребро в выбранном направлении $C_i^+(\mu)$ – раз, а в противоположном $C_i^-(\mu)$ – раз, то каждый такой цикл μ можно представить в виде некоторого вектора $C(\mu)$.

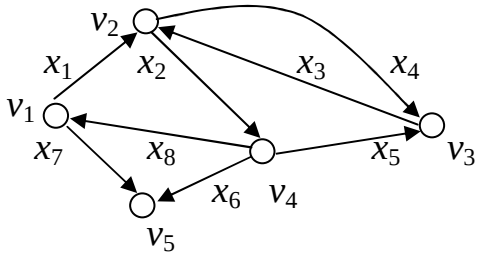
Вектор $C(\mu)$ – называется *вектор-циклом* соответствующим циклу μ, если i-я координата $C_i(\mu)$ вектора $C(\mu)$ равна разности $C_i^+(\mu) - C_i^-(\mu)$, где $C_i^+(\mu)$ – число проходов в цикле μ через ребро x_i в направлении выбранной ориентации; $C_i^-(\mu)$ – число проходов в цикле μ через ребро x_i в направлении, противоположном выбранной ориентации.

Пример.

Пусть задан граф $G(V, X)$:



Введем в G ориентацию на ребрах, получим орграф



Запишем несколько циклов в неориентированном графе G:

$\mu_1 = v_1x_1v_2x_2v_4x_8v_1$;

$\mu_2 = v_1x_8v_4x_2v_2x_3v_3x_5v_4x_2v_2x_1v_1$;

$\mu_3 = v_1x_7v_5x_6v_4x_5v_3x_4v_2x_2v_4x_8v_1$.

Найдем координаты вектор-циклов $C(\mu_i)$:

$C(\mu_1) = (1, 1, 0, 0, 0, 0, 0, 1)$;

$C(\mu_2) = (–1, –2, –1, –1, 0, 0, 0, –1)$;

$C(\mu_3) = (0, 1, 0, –1, 1, –1, 1, 1)$.

Пусть $\mu_1, \mu_2, \ldots, \mu_k$ – циклы в графе G.

Цикл μ – называется *линейной комбинацией* циклов $\mu_1, \mu_2, \ldots, \mu_k$, если выполняется равенство:

$$C(\mu) = \alpha_1 C(\mu_1) + \alpha_2 C(\mu_2) + \ldots + \alpha_k C(\mu_k), \qquad (*)$$

где α_k – рациональные числа, одновременно не равные 0.

Или сокращенно это равенство будем записывать так:

$$\mu = \alpha_1 \mu_1 + \alpha_2 \mu_2 + \ldots + \alpha_k \mu_k.$$

Циклы $\mu_1, \mu_2, \ldots, \mu_k$ называются *независимыми,* если соответствующие им вектор-циклы линейно-независимы.

Теорема.

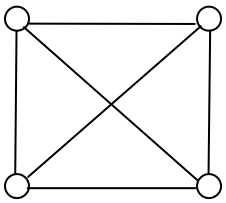
Пусть для некоторой ориентации ребер выполняется соотношение (*). Тогда оно справедливо и для любой ориентации ребер, т. е. выбор ориентации ребер не влияет на независимость циклов.

Теорема.

Цикломатическое число $\nu(G)$ графа $G = (V, X)$ равно наибольшему числу независимых циклов.

Пример.

Для графа максимальное число независимых циклов равно 3.



$\nu(G) = 6 - 4 + 1 = 3,$

Независимые циклы $\{\mu_1, \mu_2, \ldots, \mu_k\}$ называются *цикловым базисом* графа G, если любой цикл из G является линейной комбинацией независимых циклов.

Теорема.

Пусть G – граф, для которого $\nu(G) > 0$. Тогда в G существует цикловой базис, элементами которого являются простые циклы.

Алгоритм нахождения циклового базиса связного графа

Шаг 1. Если $\nu(G) = 0$, то stop. Циклового базиса не существует. Если $\nu(G) > 0$, то выполнить шаг 2.

Шаг 2. Выделить в G произвольное остовное дерево T. T содержит $(n - 1)$ ребро. Остальные $\nu(G)$ ребер удалить.

Шаг 3. Добавить к T любое из удаленных ребер и выделить простой цикл, содержащий это ребро.

Шаг 4. Процесс повторять до тех пор, пока не используются все удаленные ребра.

Шаг 5. Простые циклы $\mu_1, \mu_2, \ldots, \mu_\nu$, в каждом из которых имеется ребро, не содержащееся в других циклах, образуют цикловой базис (являются независимыми).

Пусть $G = (V, X)$ – граф, $|V| = n$, $|E| = m$.. Матрица $C(G)$ размерности $\nu \times m$, строками которой являются вектор-циклы циклового базиса называется *цикломатической матрицей графа G.*

Цикломатическая матрица широко применяется в теории электрических цепей, теории релейных схем, теории конечных автоматов.

Уравнения Кирхгофа для напряжений и токов электрической цепи. Пусть имеется некоторая электрическая цепь S, представляющая собой набор двухполюсных элементов $a_1, a_2, \ldots, a_n$ (например – конденсаторов, напряжений, источников ЭДС и т. д.), соединенных проводниками.

Представим электрическую цепь S в виде мультиграфа $G = G(S)$.

Для этого каждому j-му узлу соединений элементов цепи поставим в соответствие вершину v_j, а каждому элементу цепи a_i поставим в соответствие ребро x_i, соединяющее соответствующие вершины. Введем произвольную ориентацию на ребрах.

Согласно закону Кирхгофа для напряжений имеем

$$C(G) \cdot U = 0, \tag{1}$$

где $C(G)$ – цикломатическая матрица графа G, а $U = (u_1, u_2, \ldots, u_n)$ – вектор напряжений в электрической цепи S. Каждое u_i – это напряжение между полюсами элемента a_i. Знак величины u_i определяется в соответствии с выбранной ориентацией на ребре x_i, а именно u_i – есть величина равная разности потенциалов на концах элемента a_i (из потенциала полюса элемента a_i, соответствующего началу дуги x_i, вычитается потенциал, соответствующий концу дуги x_i).

Для расчета электрической цепи кроме уравнений Кирхгофа для напряжений (1) используют еще уравнения Кирхгофа для токов.

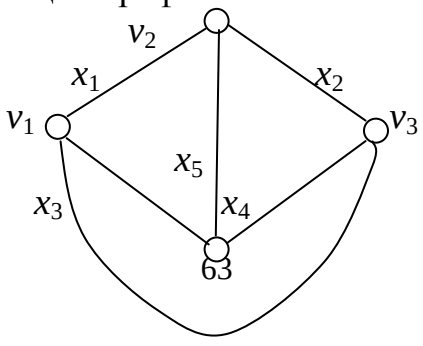
Уравнения Кирхгофа для токов составляют относительно каждой вершины ориентированного мультиграфа G. Эти уравнения математически выражают тот факт, что при установившемся процессе в электрической цепи S сумма входящих в некоторый узел этой цепи токов равна сумме токов, выходящих из него. С учетом условных направлений токов, определяемых ориентированным мультиграфом G, система уравнений Кирхгофа для токов в цепи S имеет вид

$$B(G)\, I = 0, \tag{2}$$

где $B(G)$ – матрица инцидентности ориентированного мультиграфа G, $I = (I_1, I_2, \ldots, I_n)$ – вектор токов в электрической цепи S, каждое значение I_i – величина тока, проходящего через элемент a_i.

Пример.

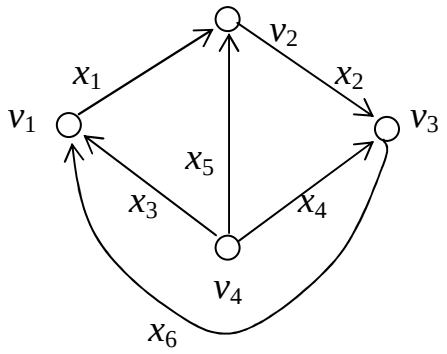
Составить уравнения для напряжений и токов электрической цепи S, которой соответствует следующий граф G:



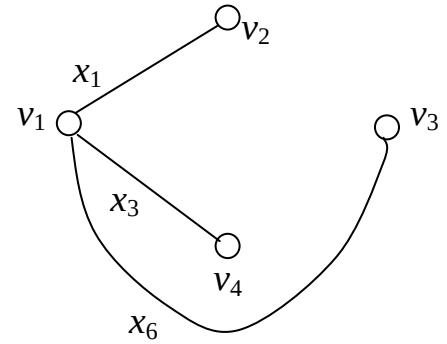
v_4

x_6

Ребра графа x_1, x_2, x_3, x_4, x_5 соответствуют сопротивлениям электрической цепи, ребро x_6 соответствует источнику ЭДС. Введем произвольную ориентацию на ребрах графа G, например:



Выделим остовное дерево графа G, удалив ребра x_2, x_4, x_5.



Определим цикловой базис (используя алгоритм нахождения циклового базиса):

1. $\nu(G) = 6 - 4 + 1 = 3$ – цикломатическое число.

2. Выделенное остовное дерево содержит 3 ребра ($n - 1 = 3$).

3. Добавляя к остовному дереву поочередно по одному из удаленных ребер, записываем простые циклы:

$\mu_1 = v_1x_1v_2x_2v_3x_6v_1$;

$\mu_2 = v_1x_3v_4x_4v_3x_6v_1$;

$\mu_3 = v_1x_1v_2x_5v_4x_3v_1$;

μ_1, μ_2, μ_3 – есть цикловой базис.

Выпишем вектор-циклы:

$C(\mu_1) = (1, 1, 0, 0, 0, 1)$

$C(\mu_2) = (0, 0, -1, 1, 0, 1)$;

$C(\mu_3) = (1, 0, 1, 0, -1, 0)$.

Тогда, цикломатическая матрица будет равна:

$$C(G) = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & -1 & 0 \end{bmatrix}.$$

Заметим, что столбцы цикломатической матрицы $C(G)$, соответствующие ребрам x_2, x_4, x_5, не вошедшим в остовное, образуют диагональную матрицу порядка $\nu(G) = 3$, с элементами, равными +1 или –1.

Зная матрицу $C(G)$, получим уравнения Кирхгофа для напряжений, подставив ее в уравнение $C(G)\cdot U = 0$:

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix} = 0,$$

$$u_1 + u_2 + u_6 = 0,$$

$$-u_3 + u_4 + u_6 = 0,$$

$$u_1 + u_3 - u_5 = 0.$$

В полученной системе переменные u_2, u_4, u_5, соответствующие ребрам x_2, x_4, x_5, являются *базисными переменными* и легко выражаются через остальные переменные u_1, u_3, u_6, которые называются *свободными*.

В ряде случаев при расчете электрических цепей, используя выражение базисных переменных через свободные, мы можем существенно сократить общее число неизвестных. Такая возможность имеется, например, в случае, когда элементами электрической цепи являются лишь источники ЭДС и сопротивления.

Определим уравнения Кирхгофа для токов в электрической цепи

$$B(G)\cdot I = 0.$$

Составим матрицу инцидентности $B(G)$ в соответствии с выбранной ориентацией графа G:

$$B(G) = \begin{bmatrix} -1 & 0 & 1 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & -1 \\ 0 & 0 & -1 & -1 & -1 & 0 \end{bmatrix}.$$

Тогда система уравнений для токов будет иметь вид

$$-I_1 + I_3 + I_6 = 0,$$

$$I_1 - I_2 + I_5 = 0,$$

$$I_2 + I_4 - I_6 = 0,$$

$$-I_3 - I_4 - I_5 = 0.$$

Из определения матрицы $B(G)$ следует, что для произвольного ориентированного мультиграфа G сумма всех строк матрицы $B(G)$ является линейной комбинацией остальных строк. Таким образом, из системы уравнений Кирхгофа $B(G)\cdot I = 0$ можно исключить любое уравнение и получить при этом систему, равносильную исходной, поскольку исключенное уравнение является линейной комбинацией оставшихся

§ 4. Транспортные сети

К возникновению теории транспортных сетей привели задачи, связанные с перевозкой грузов. В последствии оказалось, что основные понятия и алгоритмы применимы к различным прикладным задачам. Например задачи об «источниках» и «потребителях». «Источниками» могут быть – заводы, электростанции, контрольно-измерительные приборы и т. д., которые поставляют электроэнергию, изделия, информацию и т. д. для «потребителей». «Источники» и «потребители» связаны сетью линий передач с заданными пропускными способностями. Возникает задача о нахождении такого распределения потока энергии или грузов, или информации по линиям связи, чтобы в максимальной степени удовлетворить потребителей.

Транспортной сетью называется связный орграф $G(V, X)$, для которого выполняются следующие условия:

1) существует одна и только одна вершина v_1, называемая *источником*, т. е. такая, что ни одна дуга не заходит в нее, $W^{-1}(v_1) = \varnothing$;

2) существует одна и только одна вершина v_n, называемая *стоком*, т. е. такая, что ни одной дуги не исходит из нее, $W(v_n) = \varnothing$;

3) каждой дуге $x \in X$ поставлено в соответствие целое число $C(x) \geq 0$, называемое *пропускной способностью дуги*.

Вершины в транспортной сети, отличные от источника и стока, называются *промежуточными вершинами*.

Пусть v – произвольная вершина. Обозначим через X_v^- – множество дуг, заходящих в v, через X_v^+ – множество дуг, выходящих из v.

Функция $\varphi(x)$, определенная на множестве дуг X транспортной сети T и принимающая целочисленные значения, называется *допустимым потоком* (или просто *потоком*) транспортной сети T, если:

1) для любой дуги $x \in X$ величина $\varphi(x)$, называемая потоком по дуге x, удовлетворяет условию $0 \leq \varphi(x) \leq C(x)$;

2) для любой промежуточной вершины v выполняется равенство $\sum_{x \in X_v^-} \varphi(x) - \sum_{x \in X_v^+} \varphi(x) = 0$, т. е. сумма потоков по дугам, заходящим в v, равна сумме потоков по дугам, исходящим из v;

3) $\sum_{x\in X^+_{v_1}} \varphi(x) = \sum_{x\in X^-_{v_n}} \varphi(x) = \Phi$, т. е. поток, выходящий из v_1 равен потоку, входящему в v_n.

Ф – называется *величиной потока* транспортной сети. Дуга x называется *насыщенной*, если $\varphi(x) = C(x)$.

Поток Ф называется *полным*, если каждый путь из v_1 в v_n содержит хотя бы одну насыщенную дугу.

Алгоритм построения полного потока транспортной сети.

Шаг 1. Построить нулевой поток, т. е. для $\forall x \in X, \varphi(x) = 0$.

Шаг 2. Удалить из орграфа все насыщенные ребра.

Шаг 3. Найти простую цепь из v_1 в v_n. Если такой не существует, то поток полный. В противном случае переходим к следующему шагу.

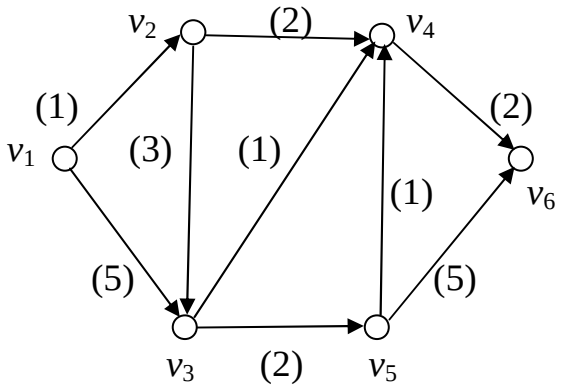
Шаг 4. Увеличить поток $\varphi(x)$ по каждой дуге x из μ на одинаковую величину $k > 0$, такую, что по крайней мере одна дуга из μ станет насыщенной, т. е. $k = \min_{x\in\mu}(C(x) - \varphi(x))$.

Далее повторяем этот процесс с шага 2.

Каждый раз, после выполнения шага 4, величина потока увеличивается на число k и в конечном итоге имеем $\Phi = \varphi(\mu_1) + \varphi(\mu_2) + \varphi(\mu_3) + ...$

Пример.

Построить полный поток транспортной сети T



Выполняем алгоритм.

1. Строим нулевой поток.

2. –

3. Рассмотрим путь $\mu_1 = v_1v_2v_4v_6$.

4. Можем увеличить величину потока на 1, так как $k_1 = \min\begin{Bmatrix} 1-0 \\ 2-0 \\ 2-0 \end{Bmatrix} = 1$.

$\Phi = \varphi(\mu_1) = 1$.

5. Удаляем насыщенную дугу v_1v_2.

6. Рассмотрим путь $\mu_2 = v_1v_3v_4v_6$.

7. Можем увеличить величину потока на 1, так как $k_2 = \min\begin{Bmatrix}5-0\\1-0\\2-0\end{Bmatrix} = 1.$

$\Phi = \varphi(\mu_1) + \varphi(\mu_2) = 2.$

8. Удаляем насыщенные дуги v_3v_4 и v_4v_6.

9. Рассмотрим путь $\mu_3 = v_1v_3v_5v_6$.

10. Можем увеличить величину потока на 2, так как $k_3 = \min\begin{Bmatrix}5-1\\2-0\\5-0\end{Bmatrix} = 2.$

$\Phi = \varphi(\mu_1) + \varphi(\mu_2) + \varphi(\mu_3) = 4.$

11. Удаляем насыщенную дугу v_3v_5.

12. Больше путей нет, следовательно полный поток $\Phi = 4$.

Замечание. Полный поток для транспортной сети не является строго определенной величиной и зависит от направления потоков в отдельных дугах.

Введем орграф приращений:

Пусть имеется транспортная сеть $T = (V, X)$, $\varphi(x)$ – заданный поток этой сети.

Орграф приращений $I(T, \varphi)$ содержит все вершины транспортной сети T и множество дуг P, которое определяется по следующим правилам:

1) $x \in P$, если $x \in X$ и $\varphi(x) < C(x)$;

2) $x' \in P$, если $x \in X$ и $\varphi(x) > 0$, где $x = (v_i, v_j)$, а $x' = (v_j, v_i)$.

Припишем дугам x и x' орграфа приращений $I(T, \varphi)$ длину l следующим образом:

$$l(x) = \begin{cases} 0, \text{если } \varphi(x) \prec C(x), \\ \infty, \text{если } \varphi(x) = C(x), \end{cases}$$

$$l(x') = \begin{cases} 0, \text{если } \varphi(x) \succ 0, \\ \infty, \text{если } \varphi(x) = 0. \end{cases}$$

Таким образом, длина любого пути из v_1 в v_n в орграфе приращений $I(T, \varphi)$ равна либо 0, либо ∞.

Орграф приращений для транспортной сети T из предыдущего примера с потоком $\Phi = 4$ можно представить следующими матрицами:

$$C(I(T, \varphi)) = \begin{bmatrix} \infty & \infty & 0 & \infty & \infty & \infty \\ 0 & \infty & 0 & 0 & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 0 & \infty & \infty & \infty \\ \infty & \infty & 0 & 0 & \infty & 0 \\ \infty & \infty & \infty & 0 & 0 & \infty \end{bmatrix}, \quad A(I(T, \varphi)) = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

§ 5. Максимальный поток транспортной сети. Алгоритм построения максимального потока

Поток называется *максимальным*, если его величина $\varphi(x)$ принимает максимальное значение по сравнению с другими допустимыми потоками в транспортной сети T.

Теорема Форда–Фалкерсона. Для того чтобы допустимый поток φ транспортной сети T был максимальным, необходимо и достаточно, чтобы сток не был достижим из источника в орграфе приращений (без доказательства), т. е. минимальный путь из v_1 в v_n в орграфе приращений должен быть равен ∞, чтобы поток φ был максимальным.

Следствием этой теоремы является **алгоритм построения максимального потока в транспортной сети.**

Шаг 1. Полагаем $i = 0$. Пусть φ_0 – любой допустимый поток в транспортной сети T (рекомендуется брать полный поток).

Шаг 2. По сети T и потоку φ_i строим орграф приращений $I(T, \varphi_i)$.

Шаг 3. Находим минимальный путь μ_i из v_1 в v_n в орграфе приращений $I(T, \varphi_i)$ (можно использовать алгоритм Форда–Беллмана).

Если длина этого пути равна ∞, то поток φ_i максимален.

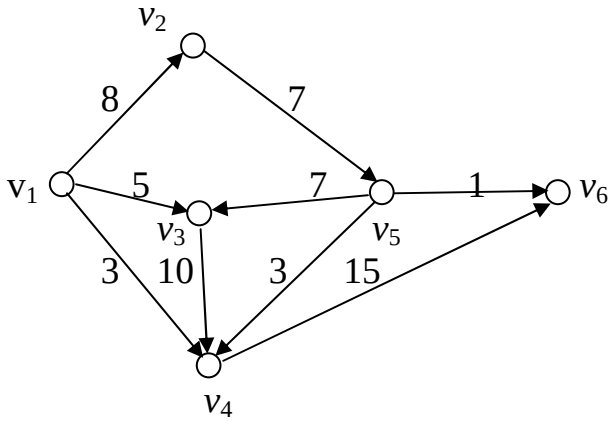
В противном случае увеличиваем поток вдоль пути μ_i на максимально допустимую величину $A_i > 0$, прибавляя A_i к величине потока, походящего через каждую дугу пути μ_i, если направления дуги и пути μ_i совпадают, и, вычитая, если направления дуги и пути μ_i противоположны.

Поток в транспортной сети T изменится на φ_{i+1}, при этом $\Phi_{i+1} = \Phi_i + A_i$.

Присваиваем $i = i + 1$ и переходим к шагу 2.

Пример.

Найти максимальный поток в транспортной сети T



Строим полный поток:

$\mu_1 = v_1v_2v_5v_6$, $k = 1$, удаляем ребро v_5v_6;

$\mu_2 = v_1v_2v_5v_3v_4v_6$, $k = 6$, удаляем ребро v_2v_5;

$\mu_3 = v_1v_3v_4v_6$, $k = 4$, удаляем ребро v_3v_4;

$\mu_4 = v_1v_4v_6$, $k = 3$, удаляем ребро v_1v_4.

Больше путей нет. Полный поток $\Phi_0 = 1 + 6 + 4 + 3 = 14$.

Проверяем, является ли полученный поток максимальным.

Строим орграф приращений, ему соответствуют следующие матрицы:

$$A(I(T, \Phi_0)) = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}, \quad C(I(T, \Phi_0)) = \begin{bmatrix} \infty & 0 & 0 & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & 0 & \infty \\ 0 & \infty & 0 & \infty & \infty & 0 \\ \infty & 0 & 0 & 0 & \infty & \infty \\ \infty & \infty & \infty & 0 & 0 & \infty \end{bmatrix}.$$

Используя алгоритм Форда–Беллмана, получим следующую таблицу:

	v_1	v_2	v_3	v_4	v_5	v_6
v_1	∞	0	0	∞	∞	∞
v_2	0	∞	∞	∞	∞	∞
v_3	0	∞	∞	∞	0	∞
v_4	0	∞	0	∞	∞	0
v_5	∞	0	0	0	∞	∞
v_6	∞	∞	∞	0	0	∞

λ_i^0	λ_i^1	λ_i^2	λ_i^3	λ_i^4	λ_i^5
0	0	0	0	0	0
∞	0	0	0	0	0
∞	0	0	0	0	0
∞	∞	∞	0	0	0
∞	∞	0	0	0	0
∞	∞	∞	0	0	0

$\lambda_6^5 \neq \infty$, следовательно, поток $\Phi = 14$ не является максимальным.
Находим минимальный путь из v_1 и v_6:

$$\lambda_{i_1}^3 + l_{i_1 6} = 0 \Rightarrow i_1 = 4,$$

$$\lambda_{i_2}^2 + l_{i_2 4} = 0 \Rightarrow i_2 = 5,$$

$$\lambda_{i_3}^1 + l_{i_3 5} = 0 \Rightarrow i_3 = 3.$$

Минимальный путь $\mu = v_1 v_3 v_5 v_4 v_6$.
Добавляем вдоль этого пути величину $A_0 = 1$.

$$\Phi_1 = \Phi_0 + A_0 = 14 + 1 = 15.$$

Проверяем, является ли полученный поток максимальным.
Строим орграф приращений, ему соответствуют следующие матрицы:

$$A(I(T, \Phi_0)) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}, \quad C(I(T, \Phi_0)) = \begin{bmatrix} \infty & 0 & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & 0 & \infty \\ 0 & \infty & 0 & \infty & 0 & 0 \\ \infty & 0 & 0 & 0 & \infty & \infty \\ \infty & \infty & \infty & 0 & 0 & \infty \end{bmatrix}.$$

Используя алгоритм Форда–Беллмана, получим:

	v_1	v_2	v_3	v_4	v_5	v_6
v_1	∞	0	∞	∞	∞	∞
v_2	0	∞	∞	∞	∞	∞
v_3	0	∞	∞	∞	0	∞
v_4	0	∞	0	∞	0	0
v_5	∞	0	0	0	∞	∞
v_6	∞	∞	∞	0	0	∞

λ_i^0	λ_i^1	λ_i^2	λ_i^3	λ_i^4	λ_i^5
0	0	0	0	0	0
∞	0	0	0	0	0
∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞

$\lambda_6^5 = \infty$, следовательно, поток Ф = 15 является максимальным.

Задачи и упражнения

1. Показать, что в любом графе количество вершин нечетной степени четно.

2. Показать, что из всякого замкнутого маршрута нечетной длины можно выделить простую цепь.

3. Показать, что ребро, входящее в цикл графа, входит в некоторый его простой цикл.

4. Показать, что любая вершина, входящая в цикл графа, не является висячей.

5. Определить, имеют ли контуры орграфы с матрицами смежности:

а) $\begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$; б) $\begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}$; в) $\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$; г) $\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}$.

6. Определить матрицы достижимости и сильной связности для орграфов с матрицами смежности из задачи 5.

7. Для заданных матриц нарисовать соответствующие им графы.

а)

	x_1	x_2
v_1	1	1
v_2	1	0
v_3	0	1

б)

	v_1	v_2	v_3
v_1	1	2	0
v_2	0	0	0
v_3	2	3	2

8. Для заданных матриц A и B нарисовать соответствующие им графы.

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

9. Определить, какие заданы матрицы и изобразить графы, представленные этими матрицами:

а) $\begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$;

б) $\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$;

в) $\begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$;

г) $\begin{pmatrix} -1 & 0 & 0 & -1 & 0 & -1 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 \end{pmatrix}$.

10. Доказать, что в сильно связном орграфе с симметричной матрицей смежности существует контур, проходящий по одному разу через каждую дугу орграфа.

11. Найти минимальный путь из v_1 в v_7 в орграфах, заданных матрицами смежности:

а) $\begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$;

б) $\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$.

12. Найти минимальный путь из v_1 в v_7 в нагруженных орграфах, заданных матрицами длин дуг:

а) $\begin{pmatrix} \infty & 4 & \infty & \infty & 5 & 1 & \infty \\ 3 & \infty & 2 & 1 & \infty & \infty & \infty \\ 1 & 1 & \infty & \infty & \infty & \infty & 3 \\ \infty & 3 & 1 & \infty & 1 & \infty & \infty \\ \infty & \infty & 2 & \infty & \infty & 1 & 5 \\ \infty & 3 & \infty & 2 & 2 & \infty & \infty \\ \infty & \infty & 2 & \infty & \infty & 2 & \infty \end{pmatrix}$; б) $\begin{pmatrix} \infty & \infty & 9 & \infty & \infty & 2 & 12 \\ 1 & \infty & \infty & \infty & 1 & 2 & 4 \\ 2 & 1 & \infty & \infty & 1 & \infty & 2 \\ \infty & 1 & 1 & \infty & \infty & 1 & \infty \\ 1 & 2 & \infty & 2 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 1 & \infty & 8 \\ \infty & 2 & 1 & \infty & 1 & 2 & \infty \end{pmatrix}$.

13. Проверить, существуют ли в мультиграфах, заданных матрицами смежности, эйлеровы цепи и циклы? Если существуют, то найти их. Рассмотреть следующие случаи:

а) $\begin{pmatrix} 0 & 1 & 0 & 0 & 2 & 1 \\ 1 & 0 & 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 2 & 0 & 1 \\ 0 & 0 & 2 & 0 & 1 & 1 \\ 2 & 0 & 0 & 1 & 0 & 1 \\ 1 & 2 & 1 & 1 & 1 & 0 \end{pmatrix}$; б) $\begin{pmatrix} 0 & 1 & 0 & 0 & 2 & 1 \\ 1 & 0 & 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 2 & 0 & 1 \\ 0 & 0 & 2 & 0 & 1 & 1 \\ 2 & 0 & 0 & 1 & 0 & 0 \\ 1 & 2 & 1 & 1 & 0 & 0 \end{pmatrix}$.

Глава 4

ЭЛЕМЕНТЫ КОМБИНАТОРИКИ

Комбинаторика – это раздел математики, изучающий расположение объектов в соответствии со специальными правилами и методами, число всех возможных способов, которыми эти расположения могут быть сделаны.

Основная задача комбинаторики – пересчет и перечисление элементов в конечных множествах.

Задача пересчета – сколько элементов, принадлежащих заданному конечному множеству, обладает некоторым свойством.

Задача перечисления – необходимо выделить все элементы множества, удовлетворяющие заданным свойствам.

§ 1. Правила суммы и произведения

При решении задач комбинаторики используются два правила: правило суммы и правило произведения.

Правило суммы. Если элемент A_1 может быть выбран n_1 способами, A_2 – n_2 способами, и т. д., A_k – n_k способами, отличными от первых (k–1) способов, то выбор одного из элементов: или A_1, или A_2, …, или A_k может быть осуществлен $n_1 + n_2 + \ldots + n_k$ способами.

Правило умножения. Пусть требуется выполнить одно за другим k действий. Если действие A_1 можно выполнить n_1 способами, A_2 – n_2 способами, …, A_k – n_k способами, то все k действий вместе, могут быть выполнены $n_1 \cdot n_2 \cdot \ldots \cdot n_k$ способами.

§ 2. Выборки без повторений и с повторениями. Перестановки, сочетания, размещения

В этом параграфе введем понятия и формулы, с помощью которых решается задача пересчета.

Пусть $X = \{x_1, x_2, \ldots, x_n\}$, все x_i различны.

Набор элементов $x_{i_1}, x_{i_2}, ..., x_{i_r}$ из множества X называется *выборкой* объема r из n элементов или (n, r) – выборкой.

Одна выборка объема r может отличаться от другой выборки такого же объема, если:

а) различен по крайней мере один элемент;

б) различен порядок расположения элементов.

Выборка называется *упорядоченной*, если порядок следования элементов в ней задан.

Перестановкой из элементов называется всякая конечная последовательность, которая получается в результате упорядоченности некоторого конечного множества, состоящего из n элементов.

$$P_n = n! \quad (n! = 1 \cdot 2 \cdot \ldots \cdot n).$$

Примеры.

1. Каким числом способов можно рассадить 8 зрителей в ряду из 8 мест.

Решение: $P_8 = 8! = 40\,320$ (способов).

2. Сколько различных 4-значных чисел можно образовать из цифр 2, 5, 0, 4, 7?

Решение: $N = P_5 - P_4 = 5! - 4! = 120 - 24 = 96$ (различных чисел).

Пусть A – множество, состоящее из n элементов. Всякое подмножество множества A, содержащее k элементов из данных n элементов, называется *сочетанием* из n элементов по k элементов.

$$C_n^k = \frac{n!}{k!(n-k)!}.$$

Пример.

1. Найти число диагоналей выпуклого 10-угольника.

Решение: $C_{10}^2 = \dfrac{10!}{2!(10-2)!} = 45,$ так как стороны не являются диагоналями, то получим 45 – 10 = 35 (диагоналей).

Всякое упорядоченное множество, содержащее k элементов, из данных n элементов, называется *размещением* из n элементов по k элементов.

$$A_n^k = \frac{n!}{(n-k)!}.$$

Пример.

1.Сколько различных 6-значных телефонных номеров можно составить из цифр 1, 2, 3, 4, 5, 6, 7, 8, 9 (все цифры в номере различны)?

Решение: $A_9^6 = \dfrac{9!}{(9-6)!} = 84.$

Выборки с повторениями. Сочетания и размещения с повторениями

Пусть X – множество элементов различных классов, т. е. $X = \{x, x, \ldots, x, y, y, \ldots, y, z, z, \ldots, z\}$ и число элементов в каждом из этих классов неограниченно.

Набор r элементов из множества X называется *выборкой* с повторениями объема r.

Одна выборка объема r может отличаться от другой выборки такого же объема, если:

а) различен по крайней мере один элемент;

б) различен порядок расположения элементов.

Упорядоченная выборка, в которой элементы могут повторяться, называется *размещением с повторениями*.

$$\overline{A}_k^r = k^r.$$

Неупорядоченная выборка, в которой элементы могут повторяться, называется (k, r) – *сочетанием с повторениями*.

$$\overline{C}_k^r = C_{k+r-1}^r.$$

Задачи и упражнения

1. В ящике 300 деталей. Из них 150 – 1-го сорта, 120 – 2-го сорта, остальные – 3-го сорта. Сколько существует способов извлечения из ящика одной детали 1-го или 2-го сорта?

2. В группе 30 чел. Необходимо выбрать старосту, его заместителя и профорга. Сколько существует способов это сделать?

3. В розыгрыше по футболу принимает участие 16 команд. Сколькими способами могут быть распределены золотая и серебряная медали?

4. Порядок выступления 7 участников конкурса определяется жеребьевкой. Сколько различных вариантов жеребьевки при этом возможно?

5. Собрание из 40 чел выбирает председателя, секретаря и 3-х членов редакционной комиссии. Сколькими способами это можно сделать?

6. В вещевой лотерее разыгрывается 8 предметов. Первый подошедший к урне вынимает из нее 5 билетов. Каким числом способов он может их вынуть, чтобы:

а) ровно 2 из них были выигрышными;

б) по крайней мере 2 из них были выигрышными.

Всего 50 билетов.

7. Расписание одного дня занятий состоит из 5 уроков. Сколько возможных вариантов расписания при выборе из 10 дисциплин (все уроки в один день различные)?

8. В конкурсе по 5 номинациям участвуют 10 кинофильмов. Сколько существует вариантов распределения призов, если по каждой номинации установлены: а) различные призы; б) одинаковые призы?

9. Сколько существует семизначных чисел, состоящих из цифр 4, 5 и 6, в которых цифра 4 повторяется 3 раза, а цифры 5 и 6 – по 2 раза?

Глава 5

ФОРМАЛЬНЫЕ ГРАММАТИКИ

В этой главе излагаются основы формальных грамматик и теории автоматов. Приводятся примеры реализации практических задач.

В дискретной математике, теория автоматов изучает абстрактные объекты, называемые автоматами, и проблемы, которые они могут решать.

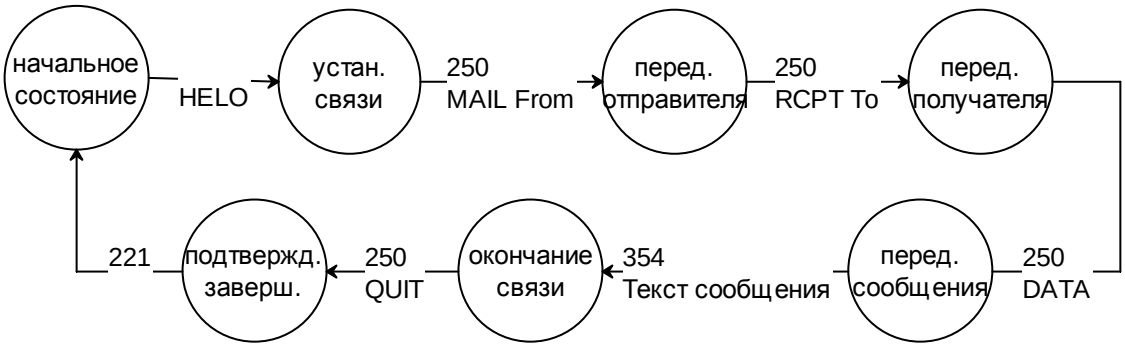
Теория автоматов наиболее тесно связана с теорией алгоритмов и теорией языков. Фактически, в некоторых случаях теория автоматов предоставляет инструмент для теории алгоритмов (средство выполнения, в частности, машина Тьюринга-Поста) и для теории языков (средство описания/интерпретации/визуализации). Помимо того, есть прикладные области, например, в теориях автоматического управления и автоматного программирования.

Рассмотрим пример задачи, которая может быть решена в терминах теории автоматов.

Пример.

Рассмотрим процедуру передачи сообщения электронной почты по протоколу SMTP.

Порядок работы можно описать в виде диаграммы:



Над линиями указаны сообщения, получаемые абонентом, под линиями - отправляемые абонентом серверу.

Обратите внимание, насколько наглядна такая диаграмма, и насколько легко искать ошибки в такой системе. Данная диаграмма называется диаграммой состояний, фактически описывает конечный автомат, и может быть однозначно преобразована в программу (причем допустимо обратное преобразование), причем в автоматическом режиме.

Такой подход называется автоматным программированием (мы о нем немного поговорим далее), и он очень удобен именно в системах обеспечения безопасности, где одним из главных требований как раз и является однозначность описания функционирования системы.

§ 1. Алфавит, строка, язык

Конечное множество символов, неделимых в данном рассмотрении, называется *словарем* или *алфавитом*, а символы, входящие в множество, - *буквами алфавита*.

Например, алфавит *A = {a, b, c, +, !}* содержит 5 букв, а алфавит *B = {00, 01, 10, 11}* содержит 4 буквы, каждая из которых состоит из двух символов.

C={a,b,c,A,B,C} – это еще один пример алфавита.

Из алфавитных символов составляются, путем последовательного соединения, *строки* или *слова*.

Если строка ω составлена из букв алфавита *A*, то говорят, что ω - строка над алфавитом *A*.

Например, строка *11001101* - это слово над алфавитом *{0, 1}*, *group* – слово над алфавитом латинских букв.

Следует учитывать, что не всегда по виду слова можно определить, над каким алфавитом это слово.

Например, слово *11001101* - может быть словом и над алфавитом *{0, 1}*, и над афлавитом *{00, 01,10, 11}*.

Слово над алфавитом характеризуется *длиной* - то есть количеством составляющих слово букв. Это записывают следующим образом $r = |\omega|$.

Введем еще несколько определений:

Пустой строкой (пустым словом) будем называть слово, не содержащее букв. Обозначают пустую строку обычно символом ε. Длина пустого слова - 0.

Множество всех слов над алфавитом *A*, имеющих длину *n*, обычно обозначают A^n.

Иными словами, пусть $\omega \in A^r$, тогда длина слова ω равна *r*.

Множество всех слов над алфавитом А назовем *замыканием A* (обозначается *A**).

Замыкание алфавита А можно записать следующим образом:

$A^* = A^0 \cup A^1 \cup A^2 \cup \ldots = \bigcup_{n=0}^{\infty} A^n$, где $A^0 = \varepsilon$.

Кроме множества всех слов над алфавитом А вводят так же множество непустых слов над алфавитом А:

$A^+ = A^* \setminus \{\varepsilon\}$.

Множество всех слов над алфавитом *A* - бесконечно, если не налагать никаких дополнительных условий (например, на количество букв в слове).

Однако, если по каким-либо критериям выделить некоторое подмножество слов из множества всех слов над заданным алфавитом, мы получим то, что называется языком над заданным алфавитом.

Фактически, *языком L* над алфавитом *A* называется некоторое подмножество *L* множества всех слов над алфавитом *A*:

$L \subset A^*$.

Кроме того, на множестве слов над некоторым языком *A*, как и над любым

другим носителем, можно ввести операции. Одной из важнейших таких операций является *конкатенация*.

Конкатенация – это бинарная операция, определенная следующим образом:

$\oplus : A^{*^2} \to A^*$

$\oplus : (\alpha, \beta) \to \alpha\beta$

Конкатенация выполняется следующим образом: к строке α дописывается срока β.

Очевидно, операция конкатенации ассоциативна.

Кроме того, выполняется такое свойство:

$|\omega \oplus \upsilon| = |\omega| + |\upsilon|$.

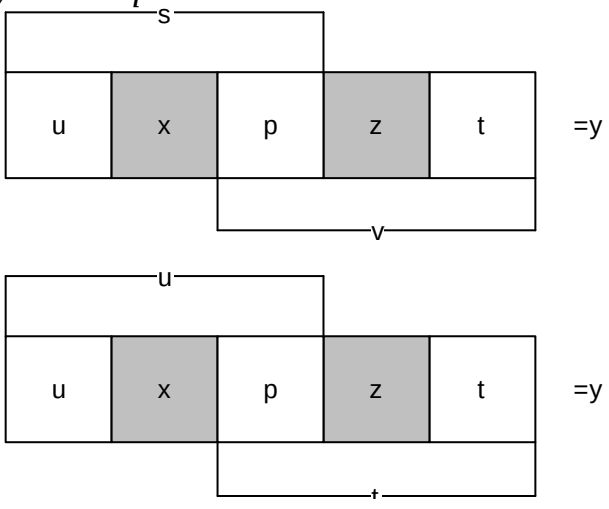
Следующая операция, которую мы рассмотрим - вхождение.

Вхождением слова $x \in A^*$ в слово $y \in A^*$ называется упорядоченная тройка слов *(u, x, v)*, такая, что $y=u \oplus x \oplus v$. При этом слово *u* называют *левым*, а слово *v* - *правым крылом* данного вхождения. Слово *x* называют *основой* вхождения.

Говорят, что слово *x* входит в слово *y*, если существует вхождение *x* в *y*. При этом слово *x* называют *подсловом* слова *y*. Подслово *x* слова *y* называют *началом* (*префиксом*) слова *y*, если $y=x \oplus z$ для некоторого непустого слова *z*. Аналогично определяют *конец* или *постфикс* слова.

Слово *x* может иметь несколько вхождений в слово *y*. При этом в одном из вхождений длина левого клыла будет наименьшей. Такое вхождение называется *первым* или *главным* вхождением.

Говорят, что вхождения *(u, x, v)* и *(s, z, t)* слов *x* и *z* в одно и то же слово *y* не пересекаются, если существуют такие слова *p* и *q* (возможно, пустые), что $y=u \oplus x \oplus p \oplus z \oplus t$ или $y=s \oplus z \oplus q \oplus w$.



Пример.

Например, пусть имеется алфавит *A={-,.,0,1,2,3,4,5,6,7,8,9}*. К словам над данным алфавитом относятся *0.1.2, 11.222, -4.5-*. А в качестве языка *L* над алфавитом *A* можно выбрать такое множество строк, которые являются корректной записью целых чисел.

Рассмотрим алфавит *A* и множество слов над ним *A**. Определив на данном носителе операцию конкатенации $\oplus$, как описано выше, получаем алгебраиче-

скую систему.
Классифицируем данную систему.
На носителе определена единственная операция, следовательно, данная система является группоидом. Операция конкатенации ассоциативна, следовательно данная система является полугруппой.
Пустая строка по отношению к операции конкатенации является единицей, следовательно, данная система является моноидом.
Следует отметить, что любое слово входит само в себя, и пустое слово входит в пустое слово.
Существует несколько способов описания языка.
Но прежде чем рассматривать эти способы, посмотрим, что представляет из себя сам язык.

Особенности естественных языков

Рассмотрим фразу "Собака кусает почтальона".
Эту фразу можно рассматривать как упорядоченную совокупность слов, а каждое слово - как упорядоченную совокупность букв. Смысл фразы в таком контексте значения не имеет, однако порядок слов в предложении важен. Такое рассмотрение – это рассмотрение с точки зрения синтаксиса.
Если рассматривать смысл, вкладываемый в каждое слово и в предложение в целом, мы будем рассматривать данное предложение семантически.
Воздействие данного предложения на какой либо объект, например, на собеседника (который бросится отгонять собаку от почтальона) – это рассмотрение фразы с точки зрения прагматики.
Ну, и собственно построение слов, входящих в данную фразу, есть грамматика.
В формальных грамматиках под собственно грамматикикой подразумевается более широкое понятие, чем в лингвистике. В лингвистике под грамматикой подразумевают получение слов из букв (обратите внимание на совпадение терминологии), а в формальных грамматиках термины "буква" и "слово" понимаются в более широком смысле – как последовательности символов и слов соответственно.
Таким образом, с точки зрения человека, формальная грамматика позволяет получать как слова, так и из слов – предложения. В данном случае речь идет о двух разных грамматиках.
Исследования в области лингвистики показывают, что большинство языков народов планеты Земля имеют схожие особенности. Во многом это определяется, естественно, тем, что языки принадлежат к одинаковым группам, то есть фактически восходят к общим предкам, праязыкам. Но возможна и другая трактовка этого явления: язык определяется мозгом человека, а строение мозга у всех народов, населяющих землю, одинаково. Таким образом, можно говорить о так называемой универсальной грамматике, частным случаем которой являются все языки на Земле.
Исторически, идея универсальной грамматики восходит к идеям таких философов, как Роджер Бэкон и Рене Декарт, но в настоящее время она практически всегда ассоциируется с именем американского лингвиста Ноама Хомского. Хом-

ский выдвинул гипотезу о том, что дети обладают врождённым механизмом усвоения языка (англ. Language Acquisition Device). Главным аргументом Хомского было практически полное отсутствие обратной связи при усвоении детьми языка (родители практически никогда не поправляют своих детей), что делает процесс усвоения языка невозможным без наличия какой-то заранее заданной информации.

Универсальная грамматика резко ограничивает количество вариантов построения языка. Главную задачу лингвистики Хомский видел в формальном описании универсальной грамматики, то есть грамматики, в рамках которой были бы описаны все языки мира, грамматики, частным случаем которой был бы любой земной язык.

Основой теории Хомского стало описание языка как внешнего проявления некоторых внутренних состояний мозга человека, а не просто следствием, или еще одним видом деятельности человека. Хомский сосредоточил внимание науки на изучении способности человека к языковой деятельности (linguistic competence), а не на самой этой деятельности (linguistic performance).

Уже Чарльз Дарвин выделял язык в качестве одного из отличительных признаков человека как вида. Однако Дарвин, как и большинство его современников, не рассматривал язык как отдельную биологическую систему, а считал его производным общих способностей человеческого разума.

Однако позднейшие исследования показали непосредственную связь между отдельными структурами мозга и речью. А именно, учеными Пьером Полем Броком и Карлом Вернике были детально описаны случаи афазии, нарушения способности человека к производству (афазия Брока) или восприятию (афазия Вернике) речи (при сохранении большинства прочих когнитивных функций), вызванные поражением определённых областей головного мозга (названных, соответственно, область Брока и область Вернике).

Наличие специфических речевых центров в мозге подтверждает предположения Хомского о врождённом характере языка и наличии универсальной грамматики. Дальнейшее изучение области Брока выявило, что она активируется только при конструировании предложений языка на основе иерархической структуры непосредственно составляющих, и не активируется при конструировании предложений языка, основанного на простом линейном порядке слов, что является сильным аргументом в пользу существования универсальной грамматики.

Счетность слов языка

Лемма о счетности множества слов языка. Замыкание алфавита A^* - счетно.

Докажем, что множество A^* счетно. Для этого достаточно построить любую нумерацию этого множества. Одним из примеров такой нумерации является так называемая лексикографическая.

Пусть алфавит A состоит из n букв $a_1, a_2, \ldots, a_n$.

В выбранной нами нумерации пустому слову сопоставляется 0, буквам алфавита $a_1, a_2, \ldots, a_n$ - соответсвенно $1, 2, ..., n$.

Если слово w имеет лексикографический номер l_w, то слову wa_i будет присвоен номер $n \cdot l_w + i$. Следовательно, номер слова $a_{i_1} a_{i_2} \ldots a_{i_k}$, будет определяться вы-

ражением $n^{k-1} \cdot i_1 + n^{k-2} \cdot i_2 + \ldots + i_k$.

Таким образом, любому слову однозначно сопоставляется число. Однако в определении счетного множества указано взаимно-однозначное соответствие, то есть произвольному числу должно соответствовать определенное слово. Фактически, ставится задача разложения числа по степеням *n*. Известно, что такая задача решается однозначно. Таким образом, построено взаимно-однозначное соответствие между любым словом над заданным алфавитом и натуральным числом. Таким образом, множество произвольных слов над заданным алфавитом – счетно.

Определение счетности множества всех слов заданного языка имеет не только теоретическое, но и практическое значение, даже несколько.

Во-первых, сопоставление каждому слову языка определенного числа позволяет сортировать слова. Спектр применений – очень широк.

Во-вторых, сопоставление каждому слову определенного числа позволяет реализовать в информационных системах быстрый поиск определенного слова по его коду.

Во многих системах программирования имеются структуры, называемые словарями. Они обеспечивают поиск каких-либо объектов не по их индексу, а, например, по названию. Компьютер обрабатывает информацию в числовом виде, так что представление слова в виде числа позволяет ускорить обработку текстовых данных. В частности, можно организовать быстрый поиск в базе данных, сохраняя не сами слова, а их числовые эквиваленты.

§ 2. Алгебры языков

Пусть имеется заданный алфавит *A*. Над данным алфавитом можно построить некоторое множество языков.

Вводя операции над множеством языков на алфавитом *A*, мы получим алгебру языков, возможно, обладающую некоторыми важными свойствами.

Определим операции на множестве языков.

Поскольку каждый язык является множеством слов, над языками вводятся все операции, определенные на множествах - *объединение*, *пересечение*, *дополнение*, *разность*, и так далее. В этом случае множество A^* называется *универсальным языком*, по аналогии с универсальным множеством.

Помимо перечисленных, вводятся несколько специальных операций.

Соединением (конкатенацией) языков L_1 и L_2 называют язык L_1L_2, состоящий из всех возможных конкатенаций слов $x \oplus y$, в которых слово x принадлежит языку L_1, а слово y - языку L_2:

$$L_1L_2=\{x \oplus y: x \in L_1, y \in L_2\}$$

Нетрудно заметить, что в качестве аналога конкатенации языков можно рассматривать прямое произведение множеств.

Можно так же рассматривать операцию конкатенации языков как аналог опера-

ции умножения. В этом случае можно провести следующую аналогию: операция конкатенации языков можно рассматривать как операцию умножения, а перечисление слов конечного языка – как операцию сложения. В этом случае операция конкатенации языков дистрибутивна относительно операции перечисления слов языка.

Операция соединения языков не обладает свойством коммутативности

По аналогии со степенью множества можно ввести операцию *возведения языка в натуральную степень*.

Действительно,

$L^0=\{\varepsilon\}$ для любого $L \subseteq V^*$,

$L^n=L^{n-1}L$ при $n>0$.

Итерацией языка L называют объединение всех его степеней:

$$L^* = \bigcup_{n=0}^{\infty} L^n$$

Отдельно выделяют *позитивную итерацию языка*:

$$L^+ = \bigcup_{n=1}^{\infty} L^n$$

Пример.

Пусть дан алфавит $A=\{0, 1, a, b\}$, и над данным алфавитом определены языки $L_1=\{01, 10\}$ и $L_2=\{ab, ba\}$, тогда

$L_1L_2=\{01ab, 01ba, 10ab, 10ba\}$

и

$L_2L_1=\{ab01, ab10, ba01, ba10\}$

Рассматривая операцию конкатенации как аналог умножения, а перечисление - как сложение, можно записать:

$(01+10)(ab+ba)= 01ab+01ba+10ab+10ba$

§ 3. Формальная грамматика

Формальной порождающей грамматикой G называется следующая совокупность четырех объектов: $G= \{ V_m, V_A, <I> \in V_A, R \}$,

где V_m - *терминальный алфавит* (словарь); буквы этого алфавита называются *терминальными символами*; из них строятся *строки порождаемые грамматикой*;

V_A - *нетерминальный, вспомогательный алфавит* (словарь); буквы этого алфавита используются при построении строк; они могут входить в промежуточные строки, но не должны входить в результат порождения;

<I> - начальный символ грамматики $<I> \in V_A$**.**

R - множество правил вывода или *порождающих правил* вида $\alpha \to \beta$, где α и β - строки , построенные из букв алфавита $V_m \cup V_A$**,** который называют *полным алфавитом* (словарем) грамматики *G*.

Пример.

Определим грамматику $G= \{ V_m, V_A, <I> \in V_A, R \}$, где

Vm={*собака_, почтальона, кусает_*},
VA={<*предложение*>, <*подлежащее*>, <*сказуемое*>, <*дополнение*>},
<*I*>=<*предложение*>,
R={
<*предложение*>→<*подлежащее*> <*сказуемое*> <*дополнение*>,
<*подлежащее*> →*собака_*,
<*сказуемое*> →*кусает_*,
<*дополнение*>→*почтальона*
}
Данная грамматика порождает единственное слово «*собака_кусает_почтальона*».

Обратите внимание, что грамматика может использоваться и для описания отдельных слов языка, и для описания предложений.
Если мы будем рассматривать слова естественного языка как *символы*, а *предложения* как строки, то получим описание всех предложений на естественном языке!
Непосредственный вывод Пусть $r = \tau \to \gamma$ - правило грамматики G и $\alpha = \chi'\tau\chi''$ - строка символов, причем $\chi', \chi'' \in (Vm \cup VA)^*$. Тогда строка $\beta = \chi'\gamma\chi''$ может быть получена из строки путем применения правила r (т.е. заменой в m строки τ на γ). В этом случае говорят, что строка β *непосредственно выведена* из строки α и обозначают $\alpha \Rightarrow \beta$.
Бинарное отношение на множестве слов над алфавитом A, образованное парами *(v, w)*, такими, что *w* может быть непосредственно выведено из *v* $v \Rightarrow w$, называется *отношением непосредственной выводимости*.
Если задана совокупность строк $\Omega = (\varpi_0, \varpi_1, ..., \varpi_n)$, таких что существует последовательность непосредственных выводов:
$\varpi_0 \Rightarrow \varpi_1, \varpi_1 \Rightarrow \varpi_2, ..., \varpi_{n-1} \Rightarrow \varpi_n$,
то такую последовательность называют *выводом* ϖ_n из ϖ_0 в грамматике G и обозначают
$\varpi_0 \Rightarrow^* \varpi_n$.
Множество конечных строк терминального алфавита V_m грамматики G, выводимых из начального символа <*I*>, называется *языком, порождаемым грамматикой G* и обозначается $L(G)$.
$L(G) = \{\varpi \in V_m^* : <I> \Rightarrow^* \varpi\}$.

Пример.
Рассмотрим несколько примеров, иллюстрирующих введенные понятия:
а) Задана грамматика G и требуется определить язык, порождаемый этой грамматикой:
G: $V_m = \{a, b, c\}$, $V_a = \{<I>\}$, $R = \{<I> \to abc\}$.
Схема грамматики содержит одно правило, поэтому G порождает язык из одно-

го слова
$L(G) = \{abc\}$.
b) Задана грамматика G и требуется определить язык, порождаемый этой грамматикой.
$G: \quad V_m = \{a, b, c\}, V_a = \{<I>, <B>, <C>\} = \{ <I> \to a<B>,$
$<B> \to <C>d,$
$<B> \to dc,$
$<C> \to \varepsilon\}.$
Построим все выводы в этой грамматике:
$<I> \Rightarrow a<B> \Rightarrow a<C>d \Rightarrow ad,$
$<I> \Rightarrow a<B> \Rightarrow adc.$
Следовательно язык $L(G) = \{adc, ad\}$.
в) Задана грамматика G и требуется определить язык, порождаемый этой грамматикой.
$G: \quad V_a = \{<I>, <A>\}, V_m = \{0, 1\},$
$R = \{<I> \to 0<A>1,$
$0<A>1 \to 00<A>11,$
$<A> \to \varepsilon\}.$
Рассмотрим несколько выводов с помощью правил грамматики G. Применяя первое и третье правила, получаем:
$<I> \Rightarrow 0<A>1 \Rightarrow 01.$
Применяя первое правило, два раза второе правило и третье, имеем
$<I> \Rightarrow 0<A>1 \Rightarrow 00<A>11 \Rightarrow 0011.$
В общем случае, применяя K раз второе правило, получим в результате строку, содержащую K нулей и K единиц.
Следовательно, язык, порождаемый грамматикой G, содержит всевозможные строки, в которых число нулей равно числу единиц.
г) Задана грамматика G и требуется построить язык, порождаемый этой грамматикой.
$G: \quad V_m = \{0, 1\}, V_a = \{<I>, <A>\}, = \{ <I> \to 0<A>,$
$<A> \to 1<A>\}.$
Попытка построения вывода в этой грамматике приводит нас к строке:
$<I> \Rightarrow 0<A> \Rightarrow 01<A> \Rightarrow 011<A> \Rightarrow \ldots,$
которая оказывается бесконечной. Другими словами, данная грамматика порождает пустой язык.

Если язык, порождаемый грамматикой G, не содержит ни одной конечной строки (конечного слова), то он называется *пустым*.
Для того, чтобы язык $L(G)$ не был пустым, в множестве R должно быть хотя бы одно правило вида $r = \chi \to \psi$, где $\psi \in V_m^*$ и должен существовать вывод $<I> \Rightarrow^* \chi$.

§ 4. Классификация грамматик

Грамматики типа 0, которые называют *грамматиками общего вида*, не имеют никаких ограничений на правила порождения. Любое правило
$r = \eta \to \psi$
может быть построено с использованием произвольных строк
$\eta, \psi \in (V_m \cup V_a)^*$. Например,
$<I>a<A> \to <A><I>$ или
$0<A>1<B><D> \to x<H><D>$.

Грамматики типа 1, которые называют также *контекстно-зависимыми грамматиками*, не допускают использования любых правил. Правила вывода в таких грамматиках должны иметь вид:

$\chi_1<A>\chi_2 \to \chi_1 \omega \chi_2$,

где χ_1, χ_2 – строки, возможно пустые, из множества $(V_m \cup V_a)^*$**,** символ $<A> \in V_a$ и строка $w \in (V_m \cup V_a)^*$. Строки χ_1 *и* χ_2 остаются неизменными при применении правила, поэтому их называют контекстом (соответственно левым и правым), а грамматику - контекстно-зависимой.
Грамматики типа 1 значительно удобнее на практике, чем грамматики типа 0, поскольку в левой части правила заменяется всегда один нетерминальный символ, который можно связать с некоторым синтаксическим понятием, в то время как в грамматике типа 0 можно заменять сразу несколько символов, в том числе и терминальных.
Если в КЗ-правиле снять требование непустоты строки *w*, то мы получим грамматику, называемую *обобщенной контекстно-зависимой грамматикой*.

Пример.
Грамматика
$G: V_m = \{a, b, c, d\}, V_a = \{<I>, <A>, <B>\},$
$R = \{ <I> \to a<I>,$
$<I> \to \alpha<A>,$
$a<I><A> \to \alpha<I><B>a,$
$<A> \to b,$
$b<B><A> \to bcd<A>,$
$b<I> \to ba \}$

является контекстно-зависимой, поскольку правила 2, 3 имеют непустой левый контекст, а 5 и 6 правила содержат оба контекста. Правила 1, 4 имеют пустой левый и правый контекст.
Большинство естественных и искусственных языков строятся на основе контекстно-зависимых грамматик. Например, практически в любом разговорном языке существует определенный порядок слов в предложении. Порядок слов как раз и гарантируется контекстностью правил.
Другой пример связан с построением слов.

В русском языке словообразование включает определенное строение слова - приставка, корень, суффикс, окончание, например: строить, строим, построим, построенный.

Грамматики типа 2 называют *контектно-свободными* и *бесконтекстными грамматиками (КС-грамматики или Б-грамматики).*
Правила вывода таких грамматик имеют вид:

$<A> \to \alpha,$

где $<A> \in V_a$ *и* $\alpha \in (V_m \cup V_a)^*$.
Очевидно, что эти правила получаются из правил грамматики типа 1 при условии $\chi_1 = \chi_2 = \varepsilon$**.** Поскольку контекстные условия отсутствуют, то правила КС-грамматик получаются проще, чем правила грамматик типа 1. Именно такие грамматики используются для описания языков программирования.

Пример.
Примером КС-грамматики может служить следующая:

$G: V_m = \{a, b\}, V_a = \{<I>\},$
$R = \{ <I> \to a<I>a,$
$<I> \to b<I>b,$
$<I> \to aa,$
$<I> \to bb\}.$

Эта грамматика порождает язык, который состоит из строк, каждая из которых представляет собой палиндром. С помощью правил этой грамматики может быть построена, например, следующая строка:
$<I> \Rightarrow a<I>a \Rightarrow ab<I>ba \Rightarrow aba<I>aba \Rightarrow ababbaba.$
Палиндро́м (от греч. «назад, снова» и греч. «бег») - число (например, 404), буквосочетание, слово или текст, одинаково (или почти одинаково) читающиеся в обоих направлениях.

Грамматики типа 3 называют *автоматными грамматиками* **(***А - грамматиками***)**. Правила вывода в таких грамматиках имеют вид:

$<A> \to a$ или $<A> \to a<B>$ или $<A> \to <B> a,$

где $a \in V_m$, $<A>$, $<B> \in V_a$**,** причем грамматика может иметь только правила вида $<A> \to a<B>$ - правосторонние правила, либо только вида $<A> \to <B>a$ - левосторонние правила.

Праволинейная грамматика, в которой, соответственно, все правила имеют вид

$<A> \rightarrow a<B>$, или
$<A> \rightarrow c$,

причем с - либо терминальный символ, либо пустая строка, называется *регулярной*.
Это самый простой вид грамматики, который имеет особое значение.
Примерами автоматных грамматик могут служить следующие грамматики:

G1: $V_m = \{a, b\}$, $V_a = \{<I>, <A>\}$, $= \{ <I> \rightarrow a<I>$,
$<I> \rightarrow a<A>$,
$<A> \rightarrow b<A>$,
$<A> \rightarrow b<Z>$,
$<Z> \rightarrow \varepsilon \}$.
G2:
$V_m = \{a, b\}$, $V_a = \{<I>, <A>\}$,
$R = \{ <I> \rightarrow <A>b$,
$<A> \rightarrow <A>b$,
$<A> \rightarrow <Z>a$,
$<Z> \rightarrow <Z>a$,
$<Z> \rightarrow \varepsilon \}$.

Эти грамматики являются эквивалентными и порождают язык, состоящий из последовательностей символов a и b.
Между множествами языков различных типов существует отношение включения:

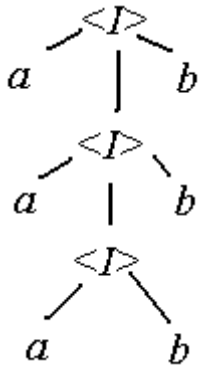
$\{ L_{типа\ 3} \} \subset \{ L_{типа\ 2} \} \subset \{ L_{типа\ 1} \} \subset \{ L_{типа\ 0} \}$.

Доказано, что существуют языки типа 0, не являющиеся языками типа 1, языки типа 2, не являющиеся языками типа 1, и языки типа 3, не являющиеся языками типа 2.
Учитывая, что наибольшее практическое применение находят грамматики типа 2 и типа 3, дальнейшее изложение посвящается рассмотрению именно этих типов грамматик.
Из всех классов ФГ только контекстно-свободные ФГ (КС-грамматики) продуктивно используются в синтаксическом анализе. Они дают дополнительную смысловую нагрузку термину «нетерминальный символ»
Нетерминальный символ - обозначение элемента синтаксиса и места его вхождения (подстановки) в другие элементы синтаксиса.
Формальные грамматики позволяют задавать языки, представляющие множества строк, построенных по определенным правилам. Используемый способ задания позволяет построить любую строку, принадлежащую языку.
Учитывая, что вывод любой строки языка, принадлежащей языку, порождаемому заданной грамматикой, должен начинаться с начального символа, правила

построения дерева можно сформулировать так:
1) В качестве начальной вершины или корня дерева возьмем вершину, которую обозначим начальным символом грамматики <*I*>; эта вершина образует нулевой уровень дерева, *k=0*.
2) Для каждого нетерминального символа <*A*>, расположенного на текущем уровне дерева, необходимо на основе правил грамматики добавить следующий уровень *k+1*, где каждый символ строки правой части использованного правила α представляет собой вершину дерева, соединенную дугой с вершиной <*A*>.
Результатом вывода является множество конечных узлов - листьев, которые выписываются при обходе дерева слева - вниз - направо - вверх. Рассмотрим, например, грамматику *G*
G:
$V_m = \{a, b\}$, $V_a = \{<I>\}$,
$R = \{<I> \rightarrow a<I>b,$
$<I> \rightarrow ab\ \},$

которая порождает язык ***L(G)*** *= {aa...abb...b}*, где *a* и *b* повторяются по *n* раз, *n=1,2,...*

Вывод строки с помощью правил этой грамматики имеет вид:



данному дереву соответствует строка aaabbb.
Дерево можно использовать и для обратной задачи - разбора строки, то есть анализа может или не может заданная строка содержаться в *L(G)*. В этом случае такое дерево называют *деревом грамматического разбора*.
Поскольку заранее неизвестно, как из последовательности правил получить заданную строку, построение дерева разбора - задача нетривиальная. Мы будем ее рассматривать далее.
Вывод строки с помощью правил грамматики может быть задан не только в виде синтаксического дерева. Если пронумеровать правила грамматики, то последовательность номеров используемых правил также задает вывод.
Последовательность номеров правил грамматики *G*, применение которых позволяет построить вывод рассматриваемой строки α из начального символа грамматики, называется *синтаксическим разбором* α.
Например, в следующей грамматике

G: $V_m = \{ i, +, *, (,) \}$, $V_a = \{<E>, <T>, <P>\} = \{$

(1) $<E> \rightarrow <E> + <E>$,
(2) $<E> \rightarrow (<E>)$,
(3) $<E> \rightarrow <E> * <E>$,
(4) $<E> \rightarrow i$ },

правила которой пронумерованы, вывод

$<E> \Rightarrow <E> + <E> \Rightarrow <E> + (<E>) \Rightarrow <E> + (<E> + <E>) \Rightarrow$
$i + (<E> + <E>) \Rightarrow i + (<E> + <E> * <E>) \Rightarrow i + (i + <E> * <E>) \Rightarrow$
$i + (i + i * <E>) \Rightarrow i + (i + i * i)$
имеет синтаксический разбор [1,2,1,4,3,4,4,4].
Если в процессе построения вывода появляются промежуточные строки, содержащие несколько нетерминальных символов, то можно продолжать вывод, заменяя любой из них. Таким образом, одни и те же правила могут быть использованы при выводе строки в разном порядке.
Среди всевозможных выводов наибольший интерес представляют следующие два типа выводов.
Если при построении вывода строки α при каждом применении правила заменяется самый левый нетерминальный символ, то такой вывод называется *левым* или *левосторонним* выводом α. Если при построении вывода α, всегда заменяется самый правый нетерминальный символ промежуточной строки, то вывод называется *правым* или *правосторонним* выводом α. Следует отметить, что различным выводам строки *может соответствовать* одно и то же синтаксическое дерево.
Неоднозначные грамматики Существуют грамматики, в которых одна и та же строка может быть получена с помощью различных выводов. Например:

G:
$V_m = \{a, b, c\}$, $V_a = \{<I>, <A>, <B>\}$,
$R = \{ <I> \rightarrow <A><B>,$
$<A> \rightarrow a,$
$<A> \rightarrow ac,$
$<B> \rightarrow b,$
$<B> \rightarrow cb\}.$

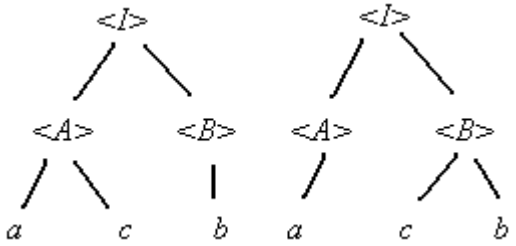
В данной грамматике вывод строки *acb* может быть получен разными способами:
Первый вывод этой строки имеет вид :

1) $<I> \Rightarrow <A><B> \Rightarrow <A>b \Rightarrow acb,$

а второй можно получить так :

2) $<I> \Rightarrow <A><B> \Rightarrow <A>cb \Rightarrow acb.$

Этим выводам соответствуют разные синтаксические деревья и разборы.



Такое свойство грамматик называется неоднозначностью. Оно может быть определено следующим образом.

Строка языка *L(G)* называется *неоднозначной*, если для её вывода существует более чем одно синтаксическое дерево. Если грамматика *G* порождает хотя бы одну неоднозначную строку, то она называется *неоднозначной*

Неоднозначность - одно из почти обязательных свойств естественных языков. Например, в английском языке выражение «Times fly» можно понимать и как «время летит», и как «мухи времени».

В искусственных языках неоднозначности стараются всячески избегать.

В общем случае можно сделать следующий вывод:

1) каждой строке, выводимой в грамматике, может соответствовать одно или несколько синтаксических деревьев,

2) каждому синтаксическому дереву могут соответствовать несколько выводов,

3) каждому синтаксическому дереву соответствует единственный правый и единственный левый выводы.

Кроме того, следует подчеркнуть, что один и тот же язык может быть получен с помощью различных грамматик.

Две грамматики G_1 *и* G_2 называются *эквивалентными*, если они порождают один и тот же язык, т.е. $L(G_1) = L(G_2)$.

§ 5. Рекурсия в языке

Рекурсивное правило - правило, в правой части которого содержится его левая часть, то есть правило вида $A \to xAy$.

Кроме того, говорят, что грамматика *леворекурсивная*, если в ней имеются выводы вида $<X> =>^* <X>\alpha$, или *праворекурсивная*, если в ней имеются выводы вида $<X> =>^* \alpha<X>$, или *самовключающая*, если имеются выводы вида $<X> =>^* \alpha<X>\beta$,

где $<X> \in Va$, $\alpha,\beta \in (Va \cup Vm)^*$.

Примерами рекурсивных грамматик могут служить следующие:

$<S> \to <S>a$

$<S> \to \varepsilon$

или

$<S> \rightarrow ab<S>$

$<S> \rightarrow a$

Большинство естественных языков рекурсивны.

В качестве примера рассмотрим термин "племянник". Племянник – это сын брата/сестры.

Фраза "Племянник Петра" означает "Сын брата Петра". В данном случае эта фраза рекурсивна, что можно показать на иллюстрации:

В разговорной речи допустимы более глубокие уровни рекурсии, а наиболее известным и запоминающимся является известное (английское народное) стихотворение в переводе Самуила Яковлевича Маршака

Вот дом,
Который построил Джек.

А это пшеница,
Которая в темном чулане хранится
В доме,
Который построил Джек.

А это веселая птица-синица,
Которая часто ворует пшеницу,
Которая в темном чулане хранится
В доме,
Который построил Джек.

Вот кот,
Который пугает и ловит синицу,
Которая часто ворует пшеницу,
Которая в темном чулане хранится
В доме,
Который построил Джек.

Вот пес без хвоста,
Который за шиворот треплет кота,
Который пугает и ловит синицу,
Которая часто ворует пшеницу,
Которая в темном чулане хранится

В доме,
Который построил Джек.

А это корова безрогая,
Лягнувшая старого пса без хвоста,
Который за шиворот треплет кота,
Который пугает и ловит синицу,
Которая часто ворует пшеницу,
Которая в темном чулане хранится
В доме,
Который построил Джек.

А это старушка, седая и строгая,
Которая доит корову безрогую,
Лягнувшую старого пса без хвоста,
Который за шиворот треплет кота,
Который пугает и ловит синицу,
Которая часто ворует пшеницу,
Которая в темном чулане хранится
В доме,
Который построил Джек.

А это ленивый и толстый пастух,
Который бранится с коровницей строгою,
Которая доит корову безрогую,
Лягнувшую старого пса без хвоста,
Который за шиворот треплет кота,
Который пугает и ловит синицу,
Которая часто ворует пшеницу,
Которая в темном чулане хранится
В доме,
Который построил Джек.

Вот два петуха,
Которые будят того пастуха,
Который бранится с коровницей строгою,
Которая доит корову безрогую,
Лягнувшую старого пса без хвоста,
Который за шиворот треплет кота,
Который пугает и ловит синицу,
Которая часто ворует пшеницу,
Которая в темном чулане хранится
В доме,
Который построил Джек.

§ 6. Способы задания грамматик

Схема грамматики содержит правила вывода, определяющие синтаксис языка. Для задания правил используются различные формы описания: символическая, форма Наура-Бэкуса, итерационная форма и синтаксические диаграммы.
В работах, связанных с рассмотрением общих свойств грамматик, обычно применяют символическую форму задания правил. Эта форма была рассмотрена нами ранее. Она предусматривает использование в качестве элементов нетерминального словаря отдельных символов и стрелки в качестве разделителя правой и левой частей правила.
Нотация Бэкуса-Наура (больше известная как БНФ или Форма Бэкуса-Наура) – формальный математический прием для описания языка, разработанный Джоном Бэкусом (а также, возможно, Питером Науром) для описания синтаксиса языка программирования Algol 60.
Она используется для формального однозначного определения грамматики какого-либо языка.
Особенностью данной нотации является использование в качестве нетерминальных символов слов и словосочетаний естественного языка, заключенных в угловые скобки, а в качестве разделителя - специального знака, применяется при описании синтаксиса языков программирования.
Например, если правила $<L> \rightarrow <L>$ и $<L> \rightarrow <E>$ записаны в символической форме, и символ $<L>$ соответствует синтаксическому понятию "список", а символ $<E>$ - "элемент списка", то их можно представить в форме Наура-Бэкуса так:

<список>::= <элемент списка><список>,
<список>::= <элемент списка>.

Для сокращения схемы грамматики, в ФБН разрешается объединять правила с одинаковой левой частью в одно правило, правая часть которого должна включать правые части объединяемых правил, разделенные вертикальной чертой. Используя объединение правил, для рассматриваемого примера получаем:

<список>::=<элемент списка><список>|<элемент списка>.

Для получения более компактных описаний синтаксиса применяют итерационную форму описания. Такая форма аналогична операции замыкания. Итерация вида $\{a\}^*$ определяется как множество, включающее строки всевозможной длины, построенные с использованием символа *a*, и пустую строку.

$\{a\}^* = \{\varepsilon, a, aa, aaa, aaaa,...\}.$

Используя итерацию для описания множества строк, задаваемых символическими правилами, для списка получаем:

$<L> \rightarrow <E> \{<E>\}^*$.

Например, описание множества строк, каждая из которых должна начинаться знаком + и может состоять из произвольного числа букв *x* и *y*, может быть представлено в итерационной форме так:

$<I> \rightarrow +\{x \mid y\}^*$.

В итерационных формах описания наряду с итерационными скобками часто применяют квадратные скобки для указания того, что строка, заключенная в них, является необязательной. С помощью таких скобок правила:

$<A> \rightarrow x<A>y<B>z$ и $<A> \rightarrow x<B>z$

могут быть записаны так:

$<A> \rightarrow x[<A>y]<B>z$.

Регулярный язык *Регулярным языком* называют язык над регулярной грамматикой.

Регулярный язык обладает определенными свойствами, которые перечислены ниже, и которые можно использовать в том числе и для определения регулярного языка. Пусть определен некоторый алфавит *A*. *Регулярным языком над алфавитом A* (то есть языком, принадлежащим к множеству регулярных языков *R(A)*) называется язык, обладающий следующими свойствами:

1. Пустое множество является регулярным языком.

$\varnothing \in R(A)$

2. Язык, состоящий из пустого слова, является регулярным языком

$\{\varepsilon\} \in R(A)$

3. Язык, состоящий из одного или нескольких символов алфавита *A*, является регулярным языком

$\forall a \in A^* \{a\} \in R(A)$

4. Если два языка над алфавитом *A* $L_1(A)$ и $L_2(A)$ являются регулярными, их объединение является регулярным языком

$L_1(A) \in R(A)$ *и* $L_2(A) \in R(A) \Rightarrow L_1(A) \cup L_2(A) \in R(A)$

5. Если два языка над алфавитом A $L_1(A)$ и $L_2(A)$ являются регулярными, их конкатенация является регулярным языком

$L_1(A) \in R(A)$ *и* $L_2(A) \in R(A) \Rightarrow L_1(A) \oplus L_2(A) \in R(A)$

6. Если *L(A)* - регулярный язык, то конкатенация всевозможных строк этого языка также является регулярным языком

$L(A) \in R(A), x \in L(A) \Rightarrow x^* \in R(A)$

7. Ничто, кроме вышеперечисленного, не является регулярным языком.

Легко заметить, что определение регулярного языка рекурсивно, то есть опре-

деляется через ранее определенные элементы.
Операция конкатенации всевозможных слов языка, описанная в п. 6, называется *замыканием Клини*.
Иногда помимо термина регулярный язык употребляется термин регулярное множество. Его можно понимать следующим образом: *регулярным множеством* называется множество языков, полученных применением регулярных операций (к которым относятся объединение, итерация и конкатенация) к элементарным языкам. Иногда термин регулярное множество используют в качестве синонима регулярного языка.
На практике часто используются так называемые *регулярные выражения*. На практике регулярные выражения представляют собой специальные шаблоны, при помощи которых можно описать произвольный регулярный язык.
Например, рассмотрим регулярный язык, элементами которого являются все возможные корректные описания адресов электронной почты. Такой язык описывается следующим регулярным выражением:
([A-z]+\S+[A-z]+)@([A-z]+\S+[A-z]+)

Пример.
Рассмотрим практические примеры применения формальных грамматик, и, в частности, регулярных языков.
Первый случай, имеющий косвенное отношение к безопасности информационных систем – ввод информации.
Не секрет, что в общем случае основную угрозу работе системы несет не аппаратная или программная часть, а человеческий фактор. Сюда можно отнести и случайные, ненамеренные ошибки при вводе информации в систему, и сознательное искажение информации, и социальную инженерию, широко применяемую хакерами.
Одним из способов борьбы с перечисленными уязвимостями и способами их использования служит контроль входных данных. Осуществлять его можно поразному, но одним из самых действенных и удобных в использовании способов состоит в проверке введенных данных регулярными выражениями.
Допустим, пользователь вводит имя файла, доступного на локальной или удаленной машине. Как известно, этот адрес может быть оформлен в одном из четырех форматов.

COMPUTER:c:\path\filename.ext - протокол TCP/IP;
COMPUTER@c:\path\filename.ext - протокол IPX/SPX;
\\COMPUTER\c:\path\filename.ext - протокол NetBEUI;
c:\path\filename.ext - локальный файл;

Задача состоит в том, чтобы реализовать в программе следующие функции:

1. Определить, правильно ли пользователь ввел путь к файлу
2. Определить, к какому из четырех вариантов относится введенный пользователем путь
3. Определить имя компьютера и непосредственный путь к файлу из

введенной строчки

4. Удалить лишние пробелы

Для решения этих задач введем следующие регулярные выражения.

```
RE_PROTOCOL_TCPIP   = '^\s*(\w+?):([^\\].*?)\s*$';
RE_PROTOCOL_IPXSPX  = '^\s*(\w+?)@(.*?)\s*$';
RE_PROTOCOL_NETBEUI = '^\s*\\\\(\w+?)\\(.*?)\s*$';
RE_PROTOCOL_LOCAL='^(.*?)\s*$';
```

Будем проверять введенную пользователем строку на соответствие одному из вышеприведенных шаблонов. Если введенная строка соответствует шаблону, то будем считать, что это и есть соответствующий путь. Если введенная строка не соответствует ни одному из вышеприведенных шаблонов, значит, в ней либо допущена ошибка, либо она намеренно искажена.

Лемма о разрастании, называемая так же *леммой о накачке* (в англ. терминологии pumping lemma) позволяет проверить, является ли заданный язык автоматным.

Следует учитывать, что любой конечный язык может быть представлен как автоматный, соответственно, данную лемму имеет смысл использовать для проверки, является ли автоматным какой-либо бесконечный язык.

Лемма о разрастании. Пусть L-автоматный язык над алфавитом A. Тогда справедливо следующее утверждение

$\exists n \in N \ \forall \alpha \in L \ |\alpha| > n \ \exists u,v,w \in A^*$ такие, что

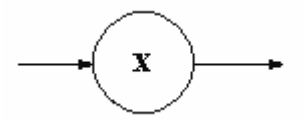
$\alpha = uvw \ u \ |uv| \le n \ u \ |v| \ge 1 \ u \ \forall i \in N \ uv^i w \in L$

Другими словами, найдется такое натуральное число *n*, что для всех строк языка *L* длиной более *n* найдутся подстроки u, v и w, такие, что слово с повтором подстроки *v* произвольное число раз так же принадлежит языку *L*.

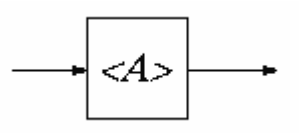
Синтаксическая диаграмма Для того, чтобы улучшить зрительное восприятие и облегчить понимание сложных синтаксических описаний, применяют представление правил грамматики в виде синтаксических диаграмм. Правила построения таких диаграмм можно сформулировать в следующем виде:

1) Каждому правилу вида $<A> \rightarrow a_1 \mid a_2 \mid ... \mid a_k$ ставится в соответствие диаграмма, структура которой определяется правой частью правила.

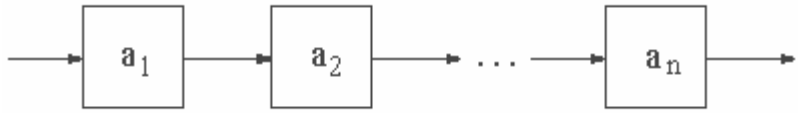
2) Каждое появление терминального символа x в строке ai изображается на диаграмме дугой, помеченной этим символом x, заключенным в кружок.



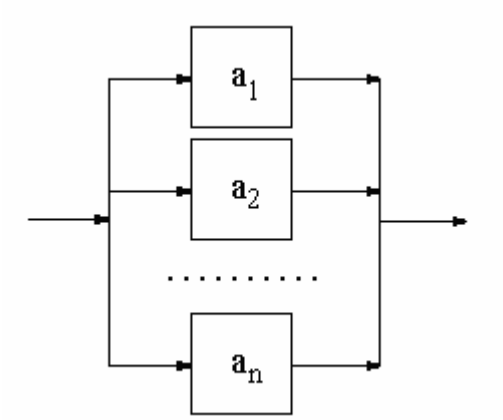
3) Каждому появлению нетерминального символа $<A>$ в строке a_i ставится в соответствие на диаграмме дуга, помеченная символом, заключённым в квадрат.



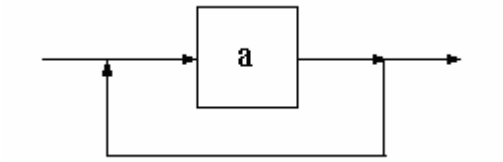
4) Порождающее правило, имеющее вид:
$<A> \rightarrow a_1 a_2 ... a_n$
изображается на диаграмме следующим образом:



5) Порождающее правило, имеющее вид:
$<A> \rightarrow a_1 \mid a_2 \mid ... \mid a_n$
изображается на диаграмме так:



6) Если порождающее правило задано в виде итерации:
$<A> \rightarrow \{a\}^*$,
то ему соответствует диаграмма:



Число синтаксических диаграмм, которые можно построить для заданной схемы грамматики, определяется числом правил. Чтобы сократить число диаграмм, их объединяют, заменяя нетерминальные символы, входящие в диаграмму, построенными для них диаграммами.
Правила 3-6 предусматривают, что в качестве строки a_1 на объединенной диаграмме могут быть использованы диаграммы построенные для этих строк.

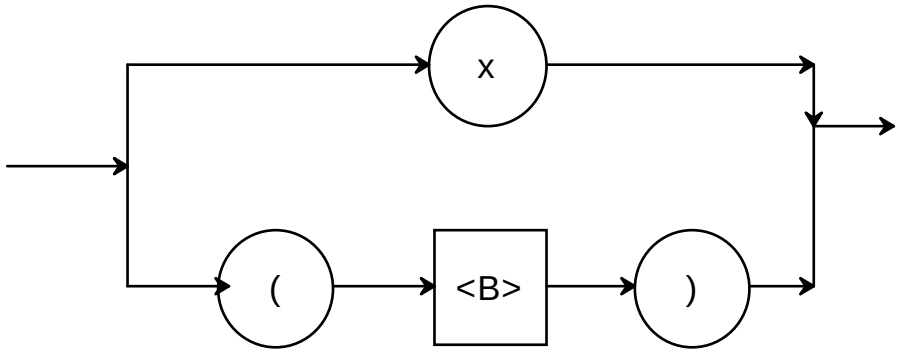
Пример.
В качестве примера рассмотрим следующую грамматику с начальным символом <A> :

G: V_m *= { x, +, (,) },* V_a *= {<A>, <B>, <C>},*
R = {<A> $\rightarrow$ *x | (<B>) (1),*
<B> $\rightarrow$ *<A><C> (2),*

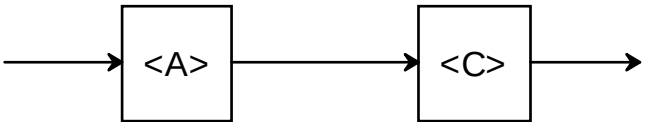
<C> →{+<A>} (3)}.*

Представим правила следующим образом:

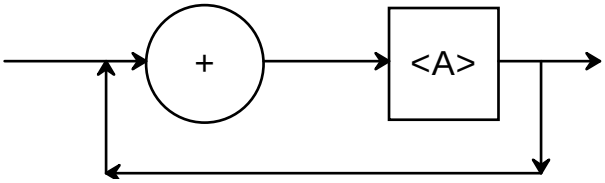
1.



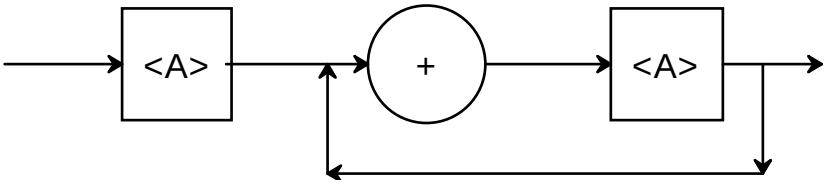
2.



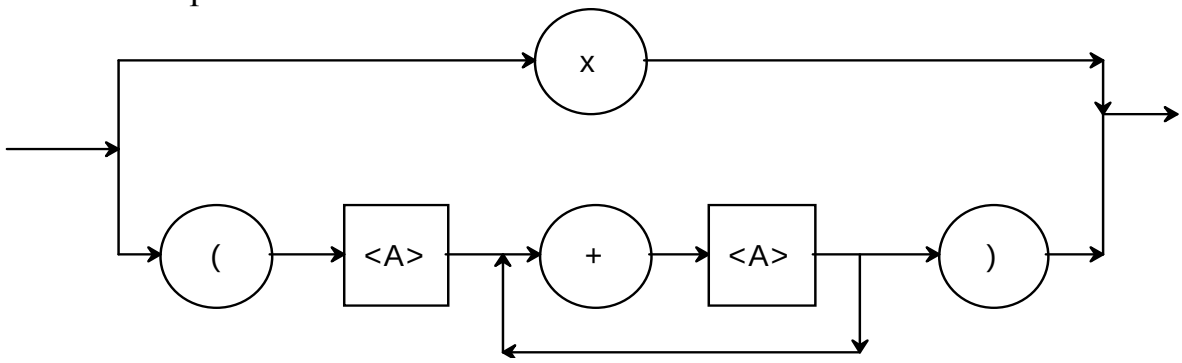
3.



Совмещая диаграммы, получим второе правило:



а затем и первое:



Обратите внимание - на диаграмме присутствует нетерминальный символ <*A*>. Присутствие нетерминального символа на диаграмме означает наличие рекурсии. При прохождении диаграммы, встретив нетерминальный символ, необходимо запомнить текущую позицию и перейти к определению данного нетерминального символа. По окончании обхода этого символа необходимо вернуться в запомненную позицию.

Нормальные виды грамматик Известно, что любая КС-грамматика $G= \{ V_m, V_A, <I> \in V_A, R \}$ может быть приведена к *нормальному виду Хомского*, в котором все правила имеют один из следующих видов:

- $<A> \to <B><C>$, где $<A>, <B>, <C> \in V_A$;
- $<A> \to a$, где $a \in V_m$, $<A> \in V_A$;
- $<A> \to \varepsilon$, при условии, что символ ε принадлежит языку, а нетерминал $<A>$ не встречается в правых частях правил.

Другим широко распространенным стандартным видом КС-грамматик является *нормальная форма Грейбах*, в которой все правые части правил начинаются с терминалов.

Грамматика в нормальной форме Грейбах (grammar in Greibach normal form) - контекстно-свободная грамматика $G= \{ V_m, V_A, <I> \in V_A, R \}$, в которой каждое правило имеет один из следующих четырех видов:

- $<A> \to \varepsilon$;
- $<A> \to a$, где $a \in V_m$, $<A> \in V_A$;
- $<A> \to a <B>$, где $<A>, <B> \in V_A$, $a \in V_m$;
- $<A> \to a <B><C>$, где $<A>, <B>, <C> \in V_A$, $a \in V_m$;

Задачи и упражнения

1. Приведите пример слова над алфавитом $A=\{1, 2, 3, 4\}$.
2. Даны два языка $L_1=\{01, 10\}$ и $L_2=\{1a, 2b\}$. Найдите конкатенацию этих языков.
3. Постройте отношение непосредственной выводимости для грамматики $G: V_m = \{a, b, c\}$, $V_a = \{<I>, <B>, <C>\} = \{ <I> \to a<B>, <B> \to <C>d, <B> \to dc, <C> \to \varepsilon\}$.
4. Определите тип грамматики $G: V_m = \{a, b, c\}$, $V_a = \{<I>, <B>, <C>\} = \{ <I> \to a<B>, <B> \to <C>d, <B> \to dc, <C> \to \varepsilon\}$.
5. Постройте дерево вывода для грамматики $G: V_m = \{ i, +, * , (,) \}$, $V_a = \{<E>, <T>, <P>\} = \{ <E> \to <E> +<E>, <E> \to (<E>), <E> \to <E> *<E>, <E> \to i \}$ для строки $i+(i+i*i)$.
6. Приведите примеры рекурсивных грамматик.
7. Приведите примеры неоднозначных грамматик.
8. Запишите в нотации Бэкуса-Наура грамматику, описывающую вещественные числа.
9. Постройте синтаксическую диаграмму для грамматики из задания 8.

Глава 6

АВТОМАТНЫЕ ГРАММАТИКИ И КОНЕЧНЫЕ АВТОМАТЫ

Формальные грамматики можно рассматривать как инструмент, позволяющий строить строки языка согласно правилам, которые зафиксированы в схеме грамматики. Другими словами, если задана грамматика, определяющая язык, то всегда можно построить любую строку этого языка.

Для практических применений большое значение имеют другие задачи, а именно задачи распознавания, называемые так же задачами принадлежности. Эта задача формулируется следующим образом: определена формальная грамматика G с терминальным алфавитом A и строка x над алфавитом A. Необходимо определить, принадлежит ли строка x языку, порождаемому грамматикой G.

Не меньшее значение имеют задачи преобразования (по заданной входной строке одного языка получить строку другого языка).

В частности, пример с распознаванием атаки в сети, как раз и является иллюстрацией задачи принадлежности. Последовательность пакетов, посылаемых компьютеру, можно упрощенно представить словом над алфавитом всех возможных пакетов, а атакующую последовательность пакетов - словом языка. Тогда стоит задача распознавания принадлежности полученного сетевым устройством слова к языку атак.

Теория автоматов в целом подразделяется на абстрактную теорию автоматов и структурную теорию автоматов.

Абстрактная теория автоматов рассматривает структуру без привязки к средствам технической реализации. Результатом абстрактного рассмотрения автоматов является выражение в той или иной форме функций переходов и функций выходов. Таким образом, на уровне абстрактной теории понятие “работа автомата” понимается как преобразование входной информации в выходную.

Структурная теория автоматов изучает общие приемы построения структурных схем автоматов на основе элементарных автоматов.

Мы сосредоточимся на абстрактной теории автоматов.

В общем случае задача распознавания строки может быть решена при помощи непосредственного использования правил грамматики. Нужно находить в заданной строке подстроки, совпадающие с правыми частями правил грамматики, и заменять эти подстроки символами левых частей правил. Такую операцию обычно называют сверткой. Если после выполнения последовательности сверток удается получить начальный символ грамматики, то заданная строка принадлежит языку, порождаемому грамматикой.

Доказано, что проблема принадлежности решаема для КЗ и КС грамматик, но в общем случае не разрешима для грамматик типа 0.

Если схема грамматики содержит много правил, и если задана длинная

строка, то процедура сворачивания становится и неоднозначной. Может возникнуть ситуация, когда на очередном шаге нельзя применить ни одно из правил. В этом случае нужно выбрать другое правило на одном из предыдущих этапов и повторить попытку.

Пример.
Допустим, что задана грамматика, схема которой имеет вид:
$R = \{<I> \rightarrow a <B> x, B \rightarrow b <B>, <B> \rightarrow y\}$,
и строка символов *abyx*. Выполнив операцию сворачивания с помощью третьего правила, получим строку $a\ b\ <B>\ x$. Теперь выполним сворачивание с помощью второго правила и получим строку $a\ <B>\ x$. Наконец, выполним сворачивание с использованием первого правила. Получим начальный символ грамматики, следовательно, заданная строка принадлежит языку, порождаемому заданной грамматикой.

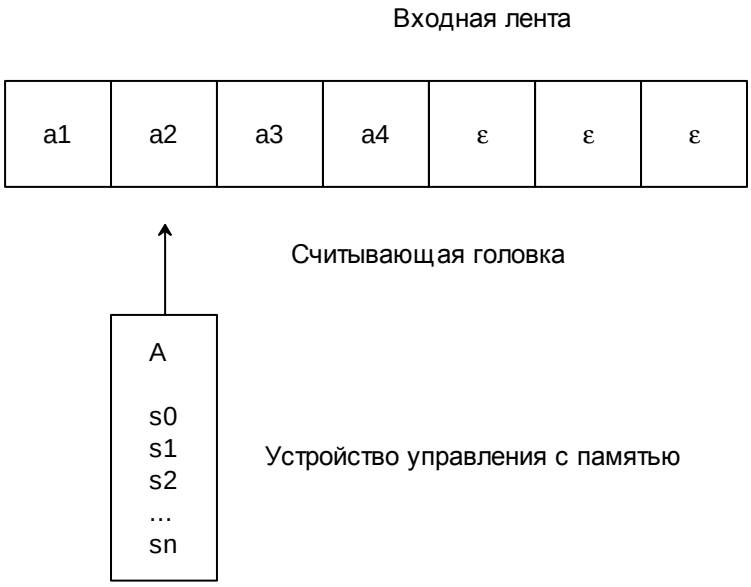
§ 1. Распознаватели

Для решения задачи принадлежности в математике выделяют специальные объекты, называемые распознавателями. Для решения задач преобразования используют преобразователи.
И распознаватели, и преобразователи являются частным случаем более общего класса математических объектов, называемых автоматами.
Известно, что каждому классу грамматик из иерархии Хомского соответствует свой класс распознавателей, определяющий тот же класс языков:

* Язык L праволинейный тогда и только тогда, когда он определяется (односторонним детерминированным) конечным автоматом
* Язык L контекстно-свободный тогда и только тогда, когда он определяется (односторонним недетерминированным) автоматом с магазинной памятью
* Язык L контекстно-зависимый тогда и только тогда, когда он определяется (двусторонним недетерминированным) автоматом с магазинной памятью
* Язык L рекурсивно перечислимый тогда и только тогда, когда он определяется машиной Тьюринга (данный вид математических объектов рассматривается в курсе «Математическая логика и теория алгоритмов»).

Распознаватель Грамматики типа три называются автоматными, потому что существует связь между этими грамматиками и конечными автоматами без выходов или распознавателями. Такой распознаватель включает в себя входную ленту, читающую головку и устройство управления.
Работа распознавателя рассматривается следующим образом:
слово, подаваемое на вход, записано на бесконечную ленту, ограниченную слева. В каждой ячейке ленты расположен один символ входной строки, начиная от крайней левой ячейки;
незаполненные ячейки (по окончании входной строки) заполнены специальным символом, обозначающим конец строки, обычно он совпадает

с символом пустой строки языка ε.



Чтение символов входного алфавита производится читающей головкой, которая после считывания символа сдвигается на одну позицию вправо. Прочитанный таким образом символ подаётся на вход устройства управления, которое может изменять свое состояние.

Распознавателем называют математический объект, являющийся совокупностью пяти компонентов:

$A = \{ P, S, s_0, \varphi, F\}$,

где P – входной алфавит распознавателя, содержащий символы, записываемые на входную ленту,

S - множество состояний распознавателя,

s_0 - начальное состояние, которым является одно из состояний множества S,

φ - функция переходов, задающая отображение

$\varphi : P \times S \rightarrow S$

F - множество конечных состояний, $F \subset S$,

Функция φ каждой паре: состояние и входной символ ставит в соответствие состояние распознавателя. В функциональной форме она записывается так:

$\varphi (s_i, p_k) = s_m$.

В общем случае φ является не функцией, а отношением, поскольку одному сочетанию «состояние распознавателя-входной символ» может ставить в соответствие несколько новых состояний распознавателя.

Работа распознавателя может быть представлена в виде последовательности тактов. Чтобы описать работу распознавателя, введём понятие конфигурации. *Конфигурация* распознавателя определяется состоянием s_i и ещё не прочитанной частью входной строки α' на ленте:

(s_i, α').

В каждом такте происходит *смена конфигурации распознавателя*. Если такту работы соответствует конфигурация $(s_k,p\alpha')$, где p- символ,

расположенный под читающей головкой, и, если определена функция переходов $\varphi(s_k,p)=s_i$, то формируется новая конфигурация (s_i, α'), соответствующая следующему такту работы.

Смену конфигураций обозначают специальным символом и записывают так:

$(s_k,\alpha) \vdash (s_i, \alpha')$.

Если существует последовательность конфигураций, соответствующая последовательности тактов работы распознавателя A

$(s_i, \alpha_1) \vdash (s_{i2}, \alpha_2) \vdash \ldots \vdash (s_{ik}, \alpha_k)$,

то её обозначают следующим образом:

$(s_{i1}, \alpha_1) \vdash^* (s_{ik}, \alpha_k)$

Конфигурация, образованная начальным состоянием s_0 и ещё не прочитанной входной строкой α называется *начальной конфигурацией* и обозначается (s_0, α).

Конфигурация, образованная конечным состоянием из множества F и пустой строкой, называется *конечной или заключительной конфигурацией*.

Используя введённые термины, можно определить два новых понятия.

Входное слово (строка) α называется *допустимым* для распознавателя A, если при последовательном чтении символов α с входной ленты распознаватель переходит из начальной конфигурации в одну из конечных конфигураций.

$(s_0, \alpha) \vdash^* (s_k,\varepsilon)$ *и* $s_k \in F$

Множество строк L, допускаемых распознавателем A, назовём *языком, допускаемым* этим распознавателем L(A).

$L(A) = \{ \alpha \mid \alpha \in P^* \text{ и } (s_0, \alpha) \vdash^* (s_k,\$) \text{ и } s_k \in F\}$

На практике функцию переходов задают в виде двумерной таблицы, которую называют *таблицей переходов* распознавателя, или в виде ориентированного графа, называемого диаграммой переходов распознавателя. При построении таблицы переходов ее строки отмечаются состояниями распознавателя, а столбцы обозначаются входными символами. В каждой клетке таблицы, которая соответствует строке, отмеченной состоянием s_i, и столбцу, отмеченному входным символом p_k, записывается состояние s_m, определяемое функцией переходов $\varphi (s_i, p_k) = s_m$.

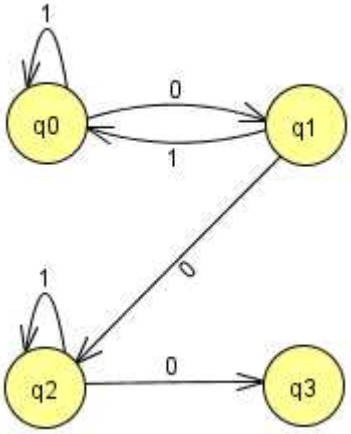
Пример.

В качестве иллюстрации рассмотрим работу распознавателя A_1, который определятся множествами: $P = \{0, 1\}$, $S = \{A, B, C, D\}$, $F = \{D\}$. Функция переходов этого распознавателя задаётся таблицей:

p_k / s_i	0	1
s0	s1	s0
s1	s2	s0
s2	s3	s2

s3	-	-

Этой таблице соответствует диаграмма переходов, изображенная на рисунке



Работа распознавателя для входной строки *01001* может быть представлена в виде последовательности конфигураций

$(q_0,01001)$ |-- $(q_1,1001)$ |-- $(q_0,001)$ |-- $(q_1,01)$ |-- $(q_2,1)$ |-- (q_2,ε),

которая показывает, что строка не допускается распознавателем A_1, поскольку он не достигает конечного состояния после прочтения всех входных символов.

Введенные понятия позволяют определить связь между формальными грамматиками и распознавателями путем установления соответствия языков, порождаемых грамматиками, и языков, допускаемых распознавателями.

Для автоматных грамматик и распознавателей соответствие этих понятий определяется следующим утверждением.

Для каждого распознавателя существует А-грамматика, порождающая язык, совпадающий с множеством строк, допускаемых распознавателем.

Для доказательства этого утверждения достаточно построить алгоритм, для любого произвольного распознавателя формирующий автоматную грамматику. Рассмотрим построение грамматики по заданной диаграмме переходов распознавателя.

Для каждой пары состояний распознавателя X и Y , соединённых дугой, исходящей из X , заходящей в Y и помеченной входным символом a, построим правило грамматики вида $<X> \rightarrow a <Y>$, если состояние Y не является конечным состоянием распознавателя, или - правило $<X> \rightarrow a$, если Y - конечное состояние.

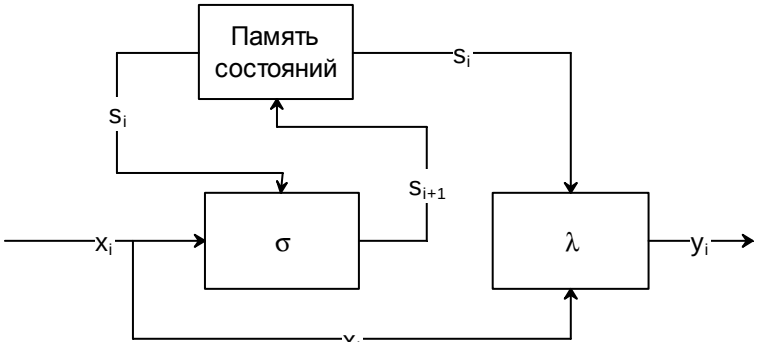
Полученное таким образом множество правил определяет грамматику, которая порождает язык, допускаемый заданным распознавателем. Действительно, каждая порождаемая распознавателем строка α будет вызывать последовательность переходов распознавателя, а каждому переходу соответствует правило грамматики, поэтому последовательность переходов при чтении символов строки α будет соответствовать последовательности применения правил грамматики, то есть выводу этой строки в построенной грамматике.

§ 2. Классификация автоматов

Мы рассмотрели понятие распознавателя. Однако существует более общий класс математических объектов. Основной задачей распознавателя является определение того, принадлежит ли заданная строка языку, соответствующему распознавателю. В то же время, часто возникает задача не распознавания строки, а ее преобразования в другой вид.
Частным случаем может служить и распознавание строки - его можно рассматривать как преобразование произвольного слова в два символа - "да" и "нет", где символ "да" означает, что строка принадлежит заданному языку.
Типичными примерами такого преобразования являются трансляция какой-либо программы с языка высокого уровня на язык машинных кодов, либо задача преобразования математических выражений.

Пример.
При разработке трансляторов (да и в других задачах тоже) возникает задача преобразования математического выражения из обычного вида в так называемую обратную польскую запись.
Что это такое? Обратная польская запись подразумевает использование стека при математических вычислениях и хороша тем, что не учитывает скобки и может быть легко реализована алгоритмически.
Пример записи выражений:
2(2+3)*
*2,3,+,2**
Механизм вычисления состоит в том, что все встреченные в строке числа помещаются в стек, а все операции извлекают из стека два операнда, а результат вновь помещают в стек.
Таким образом, последовательность будет иметь следующий вид (строка; стек):
("2,3,+,2"; "")=>("3,+,2*"; "2")=>("+,2*"; "2,3")=>("2*"; "5")=>("*"; "5,2")=>(""; "10")*

Абстрактный автомат - это математический объект, представляющий собой совокупность пяти элементов
$A=(S, X, Y, \delta, \lambda)$
Где S - конечное множество состояний автомата,
X, Y - конечные входной и выходной алфавиты соответственно, из которых формируются строки, считываемые и выдаваемые автоматом,
$\delta: S \times X \to S$ - переходное отношение (обычно называемое переходной функцией),
$\lambda: S \times X \to Y$ - выходная функция.
Функциональная схема абстрактного автомата:



Здесь s_i - текущее состояние автомата, s_{i+1} - новое состояние автомата, x_i - текущий входной символ, y_i - текущий выходной символ.

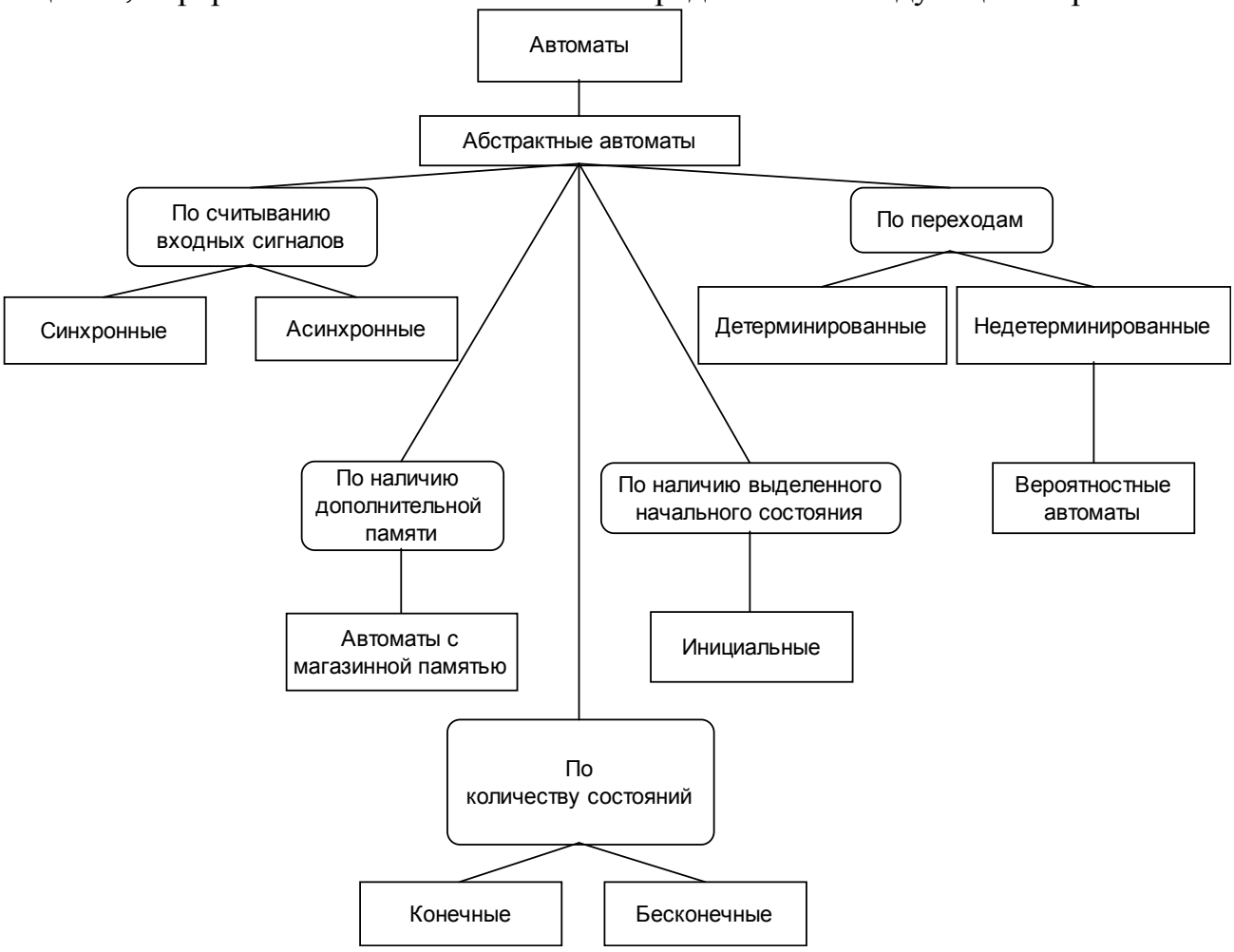
Порядок работы такого автомата следующий:

- при поступлении очередного символа на вход автомата переходная функция σ на основании поступившего символа xi и текущего состояния si определяет новое состояние автомата si+1;
- выходная функция на основе текущего состояния автомата si и текущего входного символа x_i определяет текущий выходной символ.

Говорят о объеме памяти абстрактного автомата. *Объемом памяти абстрактного автомата* называют количество его внутренних состояний.

Наиболее общим является понятие абстрактного автомата. Фактически, большинство видов автоматов являются частным случаем абстрактного автомата.

В целом, иерархия автоматов может быть представлена следующим образом:



Следует отметить, что способы задания автоматов аналогичны способам задания распознавателей - это таблица переходов и диаграмма переходов.

Инициальный автомат Абстрактный автомат – понятие довольно общее, используемое для описания большого количества объектов.

Среди них отдельно выделяют *инициальный автомат* - абстрактный автомат с выделенным начальным состоянием.

Таким образом, множество инициальных автоматов описывается одним абстрактным автоматом.

Синхронные и асинхронные автоматы По характеру отсчёта дискретного времени автоматы делятся на синхронные и асинхронные. В синхронных конечных автоматах моменты времени, в которые автомат считывает входные сигналы, определяются принудительно синхронизирующими сигналами. После очередного синхронизирующего сигнала с учётом «считанного» и в соответствии с соотношениями для функционирования автомата происходит переход в новое состояние и выдача сигнала на выходе, после чего автомат может воспринимать следующее значение входного сигнала. Асинхронный конечный автомат считывает входной сигнал непрерывно, и поэтому, реагирую на достаточно длинный входной сигнал постоянной величины x, он может, как следует из соотношений для функционирования автомата, несколько раз изменять состояние, выдавая соответствующее число выходных сигналов, пока не перейдёт в устойчивое состояние, которое уже не может быть изменено данным входным сигналом.

Вероятностный автомат Если функция переходов и/или функция выходов являются случайными, то автомат называют вероятностным.

Для моделирования поведения отдельных личностей или социальных групп, организаций, можно применять вероятностные автоматы.

Вероятностным автоматом называется система, описываемая конечными наборами входных сигналов $X=\{x_1, x_2,\ldots, x_m\}$, состояний $S=\{s_1, s_2,\ldots, s_k\}$, выходных сигналов $Y=\{y_1, y_2,\ldots, y_n\}$. Кроме этого, задан набор условных вероятностей того, что вероятностный автомат, находясь в состоянии a_j и получив входной сигнал x, перейдет в состояние a_i и выдаст сигнал y.

Вероятностный автомат очень удобен как модель, поскольку адекватно отображает неполноту наших знаний о некоторой, например, социальной, системе и позволяет повышать точность модели по мере получения новых знаний о моделируемой системе. Функционирование вероятностного автомата легко имитируется на компьютере. Такой важный для исследования живых систем аспект как процесс самообучения - отображается вероятностным автоматом через изменение вероятностей правильных реакций.

Абстрактные автоматы Для описания поведения и проектирования цифровых устройств применяют модели конечных автоматов (А - преобразователей) без входной и выходной лент. Входные символы подаются на вход таких автоматов из внешней среды последовательно в дискретные моменты времени, которые определяются генератором тактовых сигналов. Предполагается, что такой генератор существует во внешней среде (вне автомата). Если на вход автомата пода-

ется входной сигнал в момент времени t, то к моменту подачи следующего входного сигнала t+1, автомат изменяет свое состояние и вырабатывает новый выходной сигнал.

Одним из заметных отличий абстрактных автоматов от распознавателей является наличие выходной функции.

Конечным детерминированным автоматом типа Мили называется совокупность пяти объектов

$A=(S, X, Y, \delta, \lambda)$,

где S, X и Y - конечные непустые множества, а δ и λ - отображения вида:

$\delta: S \times X \to S$ - переходное отношение (обычно называемое переходной функцией),

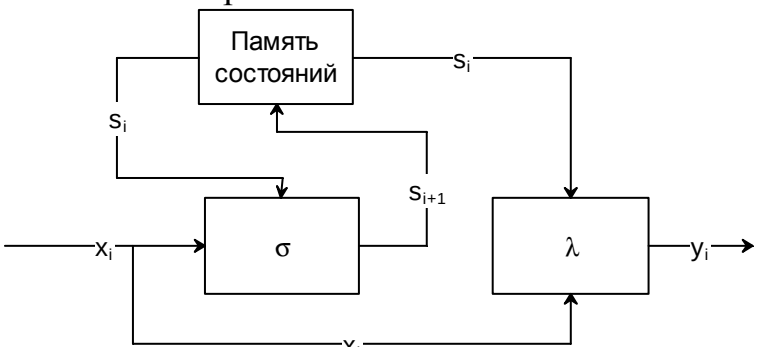
$\lambda: S \times X \to Y$ - выходная функция.

со связью элементов множеств S, X и Y в абстрактном времени $T = \{0, 1, 2, \ldots\}$ уравнениями:

$s(t+1)= \delta(s(t), x(t))$

$y(t)= \lambda(s(t), x(t)), t \in T$

Особенностью автомата Мили является то, что функция выходов является двухаргументной и символ в выходном канале $y(t)$ обнаруживается только при наличии символа во входном канале $x(t)$. Функциональная схема не отличается от схемы абстрактного автомата.



Зависимость выходного сигнала только от состояния представлена в автоматах типа Мура. В автомате Мура функция выходов определяет значение выходного символа только по одному аргументу - состоянию автомата. Эту функцию называют также функцией меток, так как она каждому состоянию автомата ставит метку на выходе.

Конечным детерминированным автоматом типа Мура называется совокупность пяти объектов:

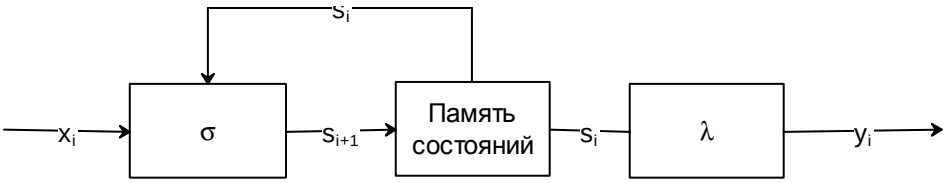
$A=(S, X, Y, \delta, \mu)$,

где S, X, Y и δ - соответствуют определению *автомата типа Мили*, а μ является отображением вида: $\mu : S \to Y$,

с зависимостью состояний и выходных сигналов во времени уравнением:

$y(t)= \lambda(s(t)), t \in T$.

Функциональная схема автомата Мура представлена на рисунке



Особенностью автомата Мура является то, что символ *y(t)* в выходном канале существует все время пока автомат находится в состоянии *s(t)*.

Сравнение автоматов Мили и Мура Может возникнуть ошибочное представление о том, что автомат Мура – это просто автомат Мили, в котором выход не зависит от входного символа. Однако, это не так.

На самом деле, в автоматах Мура выходной сигнал существует всегда, и возникает даже раньше, чем на вход будет подан первый символ.

В автоматах Мили выходной сигнал возникает в момент перехода автомата в новое состояние, и сохраняется вплоть до получения нового входного символа.

Однако особенность, которую мы только что рассмотрели, позволяет нам ввести новое определение.

Входно-независимым автоматом Мили называется автомат, в котором выходной сигнал не зависит от входного символа, то есть выполняется условие

$\forall s \in S \ \forall x,x' \in X \ \delta(s, x) = \delta(s, x')$.

§ 3. Конечные автоматы

Ограничение числа параметров абстрактного автомата определило такое понятие как конечный автомат.

Конечный автомат - в теории алгоритмов математическая абстракция, позволяющая описывать пути изменения состояния объекта в зависимости от его текущего состояния и входных данных, при условии что общее возможное количество состояний конечно. Конечный автомат является частным случаем абстрактного автомата.

$M=(Q, A, \delta, q_0, F)$, где:

Q - конечное *множество состояний* автомата;

q_0 - *начальное состояние* автомата $q_0 \in Q$;

F - множество *заключительных* (или *допускающих*) состояний, таких что $F \subset Q$. При достижении одного из этих состояний работа автомата прекращается;

A - допустимый *входной алфавит* (конечное множество допустимых входных символов), из которого формируются строки, считываемые автоматом;

δ - заданное отображение множества $Q \times A$ во множество подмножеств $P(Q)$ $\delta : Q \times A \to Q$ (иногда δ называют *функцией переходов автомата*).

Автомат начинает работу в состоянии q_0, считывая по одному символу входной строки. Считанный символ переводит автомат в новое состояние из Q в соответствии с функцией переходов.

Конечный автомат называют *полностью определенным*, если из каждого его состояния по каждому входному символу возможен переход в некоторое состояние

$\forall q \in Q \ \forall a \in A \ \delta(q,a) \neq \varnothing$

Конечный автомат распознает регулярные языки.

Детерминированные и недетерминированные конечные автоматы

Детерминированным конечный автоматом называется такой автомат, в котором при любой данной последовательности входных символов существует лишь одно состояние, в которое автомат может перейти из текущего.

Переформулируя, получим следующее определение: если функции переходов и выходов однозначно определены для каждой пары $(q,a) \in Q \times A$, то автомат называют детерминированным.

Более точно это можно сформулировать следующим образом:

Конечный автомат называется *детерминированным*, если в нем отсутствуют переходы по пустому символу ε, и из любого состояния по любому допустимому входному символу возможен переход строго в одно состояние:

$\forall q \in Q, \ \forall a \in A \ |\delta(q, a)| = 1$

Иногда такой автомат называют строго определенным.

Конечный автомат называется *квазидетерминированным* или *частично определенным*, если в нем отсутствуют переходы по пустому символу ε, и из любого состояния по любому допустимому входному символу возможен переход не более чем в одно состояние:

$\forall q \in Q, \ \forall a \in A \ |\delta(q, a)| \leq 1$

В противном случае автомат называют недетерминированным.

Определение недетерминированного конечного автомата (НКА) практически совпадает с определением ДКА. Отличий два:

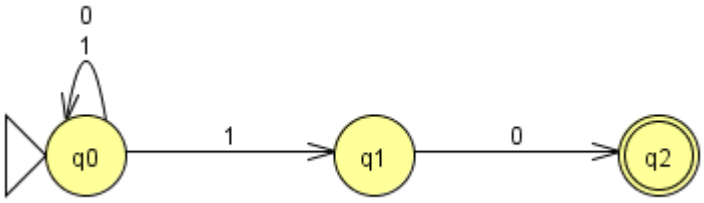
- Выходным значением δ - функции перехода является множество (возможно - пустое). Фактически, δ - не функция, а отношение.
- Если автомат находится в состоянии q_i, а на вход поступает символ a, то автомат переходит во множество состояний $\delta(q_i \ a)$. Если автомат находится во множестве состояний $\{q_i\}$, то он переходит во множество состояний, получаемое объединением множеств $\delta(q_i, a)$:

$$\{q_i\}^{next} = \bigcup_{q_j \in \{q_j\}^{current}} \delta(q_i, a)$$

НКА распознаёт строки символов аналогично ДКА, строка считается допустимой, если после её обработки множество состояний, в котором оказался автомат, содержит хотя бы одно допустимое. Таким образом, НКА также задаёт некоторый язык.

Конечные автоматы удобно иллюстрировать с помощью *диаграмм переходов*, аналогичных диаграммам переходов распознавателей (двойным кружком обозначены конечные состояния).

На рисунке изображён НКА, допускающий строки из 0 и 1, заканчивающиеся на 10.

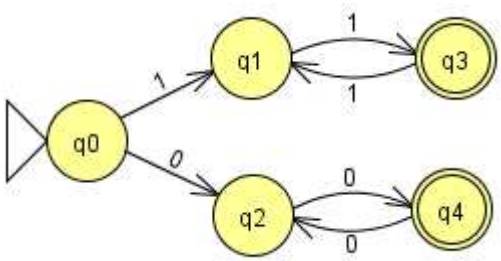


Этот же автомат можно представить в виде таблицы:

	0	*1*
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\varnothing$	$\{q_2\}$
q_2	$\varnothing$	$\varnothing$

Слово *w* над алфавитом *A допускается конечным автоматом* $M=(Q, A, \delta, q_0, F)$, если существует последовательность конфигураций автомата, такая, что $(q_0, w) \vdash^* (q_n, \varepsilon)$, $q_n \in F$.
Язык L распознается конечным автоматом, если каждое слово языка *L* допускается этим конечным автоматом.
В реальных системах НКА достаточно неудобны и поэтому практически не применяются. Однако, во многих случаях описание языка удобнее делать именно с помощью недетерминированного автомата.

Пример.
Пусть имеется язык $L=\{11^*|00^*\}$.
Этот язык распознается конечным автоматом, определенным следующей диаграммой переходов:



§ 4. Преобразования автоматов

Два *автомата* $M_1=(Q_1, A, \delta_1, q_{10}, F_1)$ и $M_2=(Q_2, A, \delta_2, q_{20}, F_2)$ называются *эквивалентными*, если они распознают один и тот же язык над алфавитом *A*.
Два *состояния* q_i и q_j называются *эквивалентными*, если переходы из этих состояний под действием любого входного слова $\alpha \in P^*$ происходят в одно и то же состояние или в эквивалентные состояния.
Такое условие эквивалентности справедливо как для двух разных автоматов,

так и для одного автомата. Таким образом, появляется возможность выделить эквивалентные состояния в одном и том же автомате.

Приняв, что заключительные и начальные состояния двух разных автоматов эквивалентны, можно сформулировать условие эквивалентности двух автоматов по-другом: автоматы M_1 и M_2 называются эквивалентными тогда и только тогда, когда для любого состояния автомата M_1 существует эквивалентное ему состояние автомата M_2, и для каждого состояния автомата M_2 существует эквивалентное ему состояние автомата M_1.

Эквивалентность двух состояний конечного автомата означает, что с точки зрения распознавания (или преобразования) входной строки они неразличимы. Следовательно, существует возможность заменить два или более эквивалентных состояний одним, получив автомат, эквивалентный исходному.

Такое преобразование называется минимизацией конечного автомата.

Задачу минимизации числа состояний автомата можно сформулировать следующим образом. Для заданного конечного автомата без выходов, имеющего m состояний, требуется построить автомат, выполняющий те же функции, но имеющий меньшее число состояний. Эта задача решается с помощью объединения избыточных состояний, для выявления которых на множестве состояний автомата определяется отношение эквивалентности.

Предположив, что начальные состояния автоматов эквивалентны, то подавая на вход автомата различные символы входного автомата, мы получим другие пары эквивалентных состояний.

На основании данного утверждения можно составить алгоритм разбиения множества состояний на классы эквивалентности.

Состояния, входящие в каждый класс эквивалентности, реагируют на входные воздействия одинаковым образом, поэтому каждому классу эквивалентности можно поставить в соответствие одно состояние и построить новый автомат с меньшим числом состояний.

Для выявления эквивалентных состояний используют входные строки ограниченной длины $k = 0,1,2,\ldots,r-1$, а состояния эквивалентные для цепочек длины k называют k-эквивалентными состояниями.

Два состояния s_i и s_j называются *k-эквивалентными*, если переходы из этих состояний под действием каждой входной строки длины k происходят в одно и то же состояние или в k-эквивалентные состояния.

С помощью отношения k-эквивалентности множество состояний автомата разбивается на множества k-эквивалентных классов $E^1,E^2,\ldots E^q$, а каждое такое множество в свою очередь разделяется на классы:

$$E^k = E_1^k \cup E_2^k \cup \ldots \cup E_r^k$$

Каждую входную последовательность произвольной длины можно представить как совокупность последовательностей длины k, поэтому можно считать, что k-эквивалентные состояния являются эквивалентными состояниями.

Алгоритм минимизации числа состояний ДКА По конечному автомату часто можно построить автомат с меньшим числом состояний, эквивалентный исходному. Соответствующий процесс называется минимизацией конечного

автомата. Вначале удалим из автомата все состояния, недостижимые из начального. Затем разобьем все состояния КА на классы эквивалентности следующим способом:

1. Разбить множества состояний автомата Q на 0-эквивалентные классы: в первый класс включаются все состояния автомата, не являющиеся конечными: $E_1^0 = Q - F$, а в остальные классы включить все заключительные состояния $E_2^0 = Z_1$, $E_3^0 = Z_2$ и так далее.

2. В каждом классе E_i^k найти группы таких состояний, из которых под действием одинаковых входных символов выполняется переход в k-эквивалентные состояния. Каждая такая группа представляет собой класс $k+1$-эквивалентных состояний. В результате получаем разбиение множества состояний на классы $k+1$-эквивалентности:

$$Q = E_1^{k+1} \cup E_2^{k+1} \cup \ldots \cup E_n^{k+1}$$

3. Увеличить значение $k = k+1$.

4. Если при выполнении k-го шага новых классов эквивалентности не появилось $E^{k-1} = E^k$, то это означает, что классы k-эквивалентности у автомата отсутсвуют, и построение закончено.

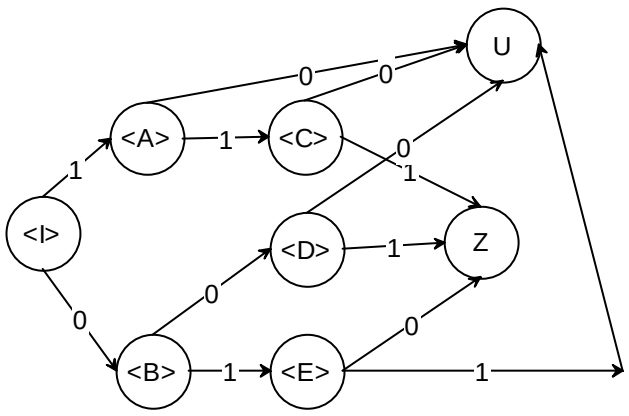
5. Если же появились новые классы $E^{k-1} \neq E^k$, то перейти к пункту 2.

6. После построения классов эквивалентности $E^k = E_1^k \cup E_2^k \cup \ldots \cup E_r^k$ каждому классу эквивалентности E_i^k нужно поставить в соответствие новое состояние автомата q_i'

7. Функции переходов для новых состояний $Q=(s_1', s_2', \ldots s_r')$ определяются следующим образом: если из состояний класса E_i^k под действием входного символа a_j автомат в состояния класса E_o^k, то строится функция переходов $\varphi(a_j, s_i') = s_j'$.

Пример.

В качестве примера минимизации числа состояний рассмотрим построение классов эквивалентности для автомата, изображенного на рис.



1. В качестве начального разбиения для k = 0 примем классы:

E_1^0={<I>, <A>, <B>, <C>, <D>, <E>}, E_2^0= {Z}, E_3^0 = {U}.

2. Классы 0-эквивалентности обозначены цифрами 1, 2, 3. Для каждого состояния определим, в какие классы переходит автомат под действием входных символов 0 и 1. Запишем номера классов под каждым состоянием.

E_1^0={	<I>,	<A>,	<B>,	<C>,	<D>,	<E>}
	1 1	3 1	1 1	3 2	3 2	3 2

, E_2^0= {Z}, E_3^0 = {U}.

В классе E_1^0 состояния разделяются на три группы. Из всех состояний первой группы выполняется переход в состояния группы E_1^0, а из состояний второй группы в состояния классов E_2^0 и E_3^0. Следовательно, получаем множество 1-эквивалентных классов в виде:

E_1^1={<I>, <B> }, E_2^1= { <A> }, E_3^1 ={ <C>, <D>, <E> }, E_4^1 = { <Z> }, E_4^1 = {U}.

3. Снова обозначим новые 1-эквивалентные классы цифрами и определим, в какие классы происходят переходы из различных состояний. В результате получаем:

E_1^1={	<I>,	<B> }	E_2^1= {	<C>,	<D>,	<E> }
	2 1	3 3		5 4	5 4	4 5

E_3^1 = { Z}, E_4^1 = {U}.

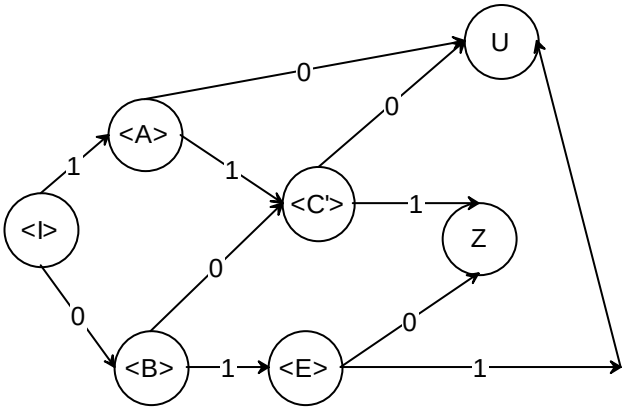
После разделения класса E_1^1 и E_2^1, получаем два новых класса эквивалентности: E_1^2={<I>}, E_2^2= {<A>}, E_3^2 = {<B>}, E_4^2 = {<C>, <D>}, E_5^2 = {<E>}, E_6^2 = {Z}, E_7^2= {U}.

4. Попытка дальнейшего разделения классов не приводит к успеху.

E_1^2={<I>}, E_2^2= {<A>}, E_3^2 = {<B>}, E_4^2 = {<C>(5 4), <D>(5 4)}, E_5^2 = {<E>}, E_6^2 = {Z}, E_7^2= {U}.

Получаем $E^2 = E^3$, что говорит о том, что классы эквивалентности построены.

5. Обозначим класс эквивалентности E_4^2 символом нового состояния <C'> и построим диаграмму переходов искомого автомата, заменяя состояния C и D новым состоянием C'. Окончательно получаем диаграмму переходов автомата в виде, показанном на рис.



В результате минимизации нам удалось сократить число состояний автомата на 1. В более сложных случаях, при большем числе состояний автомата, выигрыш будет более заметным.

Справедлива следующая теорема (о ДКА, эквивалентном НКА):
Пусть $L=L(M)$ – язык, распознаваемый некоторым недетерминированным автоматом M, тогда $L=L(M')$ для некоторого детерминированного автомата M'.
Другими словами: если недетерминированный автомат M распознает некоторый язык L, тогда найдется такой детерминированный автомат M', такой, что распознает тот же самый язык L.
Эта теорема доказывается конструктивным образом, т.е. путем указания общего алгоритма построения детерминированного автомата M', определяющего тот же язык, что и M. Пусть $M=(Q, A, \delta, q_0, F)$, тогда мы определим $M'=(Q', A, \delta', q'_0, F')$, следующим образом:

- Q'совпадает с множеством состояний автомата M'
- $q'_0=q_0$
- $F'=\{S\in Q: S\cap F\neq\varnothing\}$
- $\delta'(S, a)=S'$ для всех $S\subseteq Q$, где $S'=\{p: \delta'(q, a)$ содержит p для некоторого $q\in S\}$

Можно показать, что M' задает тот же язык, что и M. Таким образом, классы языков, задаваемых детерминированными и недетерминированными конечными автоматами, полностью совпадают. Естественно, детерминированные конечные автоматы удобнее, и в дальнейшем мы будем иметь дело только с ними
Любая регулярная грамматика может быть преобразована в конечный автомат (возможно, недетерминированный)

Преобразование НКА без переходов по пустой строке в ДКА

Рассмотрим произвольный НКА с тремя состояниями - q_0, q_1, q_2.
Независимо от своей внутренней структуры, в каждый конкретный момент этот НКА может находиться в одном из следующих множеств состояний:

- $\varnothing$ (пустое множество)
- $\{q_0\}$
- $\{q_1\}$
- $\{q_2\}$
- $\{q_0, q_1\}$

- $\{q_0, q_2\}$
- $\{q_1, q_2\}$
- $\{q_0, q_1, q_2\}$

При этом функция перехода между множествами состояний строго определена - это объединение значений функции перехода для каждого из состояний, и сами значения и их объединения тоже находятся в этом списке.

Для произвольного НКА с любым количеством состояний, мы можем определить следующий ДКА:

- его состояниями будут множества состояний НКА, определенные выше:

$$q_j^{дка} = \{q_i^{нка}\}.$$

- входной алфавит совпадает с алфавитом НКА
- функция перехода будет сопоставлять состоянию ДКА, которое соответствует множеству состояний НКА, и состояние ДКА, соответствующее множество состояний НКА и входному символу.

$$\delta^{дка}(q_j^{дка}, a) = \bigcup_{q_i^{нка} \in q_j^{дка}} \delta^{нка}(q_i^{нка}, a)$$

- начальным состоянием будет множество, состоящее из начального состояние НКА.

$$q_0^{дка} = \{q_0^{нка}\}$$

- допускающими будут те состояния ДКА, которые содержат хотя бы одно допускающее состояние НКА.

$$F^{дка} = \{q_j^{дка} : q_j^{дка} \cap F^{нка} \neq 0\}$$

Преобразование НКА с эпсилон-переходами в ДКА Введём понятие ε-замыкания.

ε-замыканием состояния q_i называется множество состояний ε-НКА, в которые из q_i можно попасть по цепочке ε-переходов. Как минимум, в это множество входит само q_i.

Функцию, аргументом которой является состояние, а значением - соответствующее ε-замыкание, назовём *eclose*.

Функцию *eclose* можно определить так:

eclose(q_i)=$\{q_i\} \cup \{ q_j: q_j \in Q$ *и* $\exists$ *ε-переход от* q_i *к* $q_j \}$

Преобразование ε-НКА в соответствующий ДКА несколько отличается:

- функция перехода (отношение перехода) ДКА будет сопоставлять множеству и входному символу другое множество:

$$\delta^{дка}(q_i^{дка}, a) = \bigcup_{q_j^{нка} \in q_i^{дка}} eclose\left[\delta^{нка}(q_j^{нка}, a)\right]$$

- начальным состоянием ДКА будет ε-замыкание начального состояние ε-НКА:

$$q_0^{дка} = eclose(q_0^{нка})$$

Доказательство идентичности описываемых автоматами языков аналогично доказательству для НКА без ε-переходов.

Общий алгоритм преобразования НКА в ДКА

Возможны два подхода:

- Сначала сгенерировать все состояния ДКА, которые только могут потребоваться, установить между ними связи, а потом избавляться от недостижимых состояний.
- Сразу же генерировать только достижимые состояния.

Мы пойдём вторым путём. Основная идея алгоритма заключается в следующем:

- Начнем с начального состояние ДКА - $eclose(q_0)$. Оно достижимо по определению.
- Если состояние достижимо, то все состояния, в которые из него можно попасть, тоже достижимы.
- Из состояния q_i можно попасть в те состояния, которые являются значением $\delta(q_i, x)$, где δ – функция перехода ДКА, а x принадлежит множеству входных символов. Перебрав все входные символы, получим все такие состояния.
- После чего применяем к полученным состояниям тот же принцип. Остановка произойдет, когда все сгенерированные состояния рассмотрены подобным образом.

Эквивалентность праволинейных грамматик и конечных автоматов

Как упоминалось выше, любой конечно-автоматный язык может быть определен праволинейной грамматикой.

На практике часто возникают задачи преобразования формальной грамматики в конечный автомат и наоборот, восстановления формальной грамматики по конечному автомату. Рассмотрим, каким образом можно построить грамматику по конечному автомату.

Для конструирования праволинейной грамматики, соответствующей данному конечному автомату, достаточно включить в грамматику правила вида $q \to ar$ для всех переходов вида $\delta(q, a)=r$ и правила вида $z \to \varepsilon$ для всех заключительных состояний z.

Класс языков, задаваемых праволинейными грамматиками, очень удобен в задачах компиляции, т.к. ему соответствуют конечные автоматы, для которых алгоритмически разрешимы такие проблемы, как эквивалентность двух языков, пустота определяемого языка и проверка входной строки на принадлежность данному языку. Многие элементы языков программирования, такие как константы, слова языка и строки, могут быть определены с помощью праволинейных грамматик.

К сожалению, не все особенности современных языков программирования могут быть описаны конечными автоматами. Например, в большинстве языков существуют составные операторы (описываемые, например, скобками {} в языках C/C++).

Можно показать, что не существует праволинейной грамматики, описывающий данный язык, но зато его легко записать с помощью следующей контекстно-свободной грамматики: $\langle A \rangle \to \{\langle A \rangle\}$, $\langle A \rangle \to \langle A \rangle \langle A \rangle$, $\langle A \rangle \to \varepsilon$.

По этой причине контекстно-свободные грамматики получили значительно

большее распространение в практических задачах.

§ 5. Синтез и анализ конечного автомата

Для конечного автомата могут возникать две задачи: задача синтеза конечного автомата, то есть построения автомата по заданному языку, и задача анализа КА, то есть вычисления языка заданного КА.

На практике могут встречаться задачи, когда устройство должно реагировать только на определенные последовательности сигналов, подаваемых на вход. Например, большинство протоколов обмена информацией и представляют собой описание таких последовательностей. Если набор таких последовательностей конечен, то задача построения устройства может быть сформулирована в следующем виде. Задан язык L над алфавитом A. Требуется построить конечный распознаватель этого языка.

Эта задача может быть решена путем построения автоматной грамматики для заданного конечного языка, и последующего построения автомата без выходов для этой грамматики.

Имеет смысл говорить о двух типах распознавателей-автоматов. Первый тип служит лишь для определения, принадлежит ли данное слово языку, заданному распознавателем.

Второй тип распознавателей не только определяет принадлежность слова языку, но и идентифицирует это слово.

В первом случае достаточно двух заключительных состояний автомата (строка принадлежит языку/строка не принадлежит языку), или даже одного (строка принадлежит языку), а во втором случае число состояний будет равно количеству распознаваемых строк +1 (распознается каждая строка и еще одно состояние выделяется для всех строк, не принадлежащих заданному языку).

В общем случае заданный конечный язык может содержать слова разной длины. Возможность построения детерминированного распознавателя для такого языка можно сформулировать следующим образом.

Если заданный конечный язык не содержит слов, являющихся начальными отрезками другого слова этого языка, то для него можно построить детерминированный конечный распознаватель.

Пример.

Рассмотрим пример задачи, для решения которой может быть использован конечный распознаватель.

Существует несколько методов сжатия информации, различающихся как степенью сжатия, так применимостью к различным типам данных. Одним из наиболее эффективных алгоритмов является алгоритм Хаффмана.

Его использование основано на том, что в тексте на естественном языке различные буквы встречаются с различной частотой.

В связи с этим, можно использовать для кодирования различных символов тек-

ста битовые последовательности разной длины - для часто встречаемых символов более короткие, для редко встречаемых - более длинные.
Тогда для разжатия информации потребуется определить последовательность символов в исходном тексте по их сокращенным кодам.
Алгоритм состоит в следующем:
1. Для блока информации составляется таблица частоты символов.
2. В зависимости от частоты символов каждому из них присваивается свой код состоящий из нескольких бит. Длинна кода, зависит от количества символов с данной частотой. Чем меньше частота, тем больше число бит.

Символ	Частота	До сжатия	После сжатия
a	*0.2*	*0000*	*00*
b	*0.2*	*0001*	*01*
c	*0.2*	*0010*	*10*
d	*0.15*	*0011*	*1100*
e	*0.15*	*0100*	*1101*
f	*0.15*	*0101*	*1110*
g	*0.1*	*0110*	*111100*

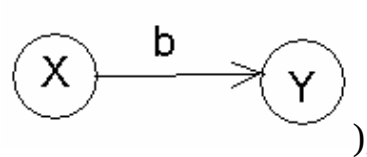
Коды символов выбираются таким образом, чтобы соблюдалось два условия**:**
Коды символов с меньшей частотой имеют длину большую длину.
Каждый символ определяется однозначно.
Пример сжатия строки выглядит следующим образом:

Текст	*abbacfgefab*
Несжатое представление	*0000 0001 0001 0000 0010 0101 0100 0110 0100 0101 0000 0001*
Сжатое представление	*00 01 01 00 10 1110 111101 1101 1110 00 01*

Экономия составляет 48-32=16 бит.
Коэффициент сжатия составляет 32/48=0,67
Разархивация как раз и даст пример использования распознавателя – необходимо перевести текст из сжатого вида в исходный.
Алгоритм преобразования регулярной грамматики в конечный автомат
Рассмотрим преобразование регулярной грамматики в конечный автомат.
Напомним особенности регулярной грамматики:
В них допустимы только правила следующих видов видов:

$<X> \to b<Y>$ цепочка терминалов - нетерминал
$<X> \to b$ цепочка терминалов
$<X> \to \varepsilon$ аннулирующее правило
$<X> \to <Y>$ нетерминал

Существуют следующие правила преобразования регулярной грамматики в диаграмму состояний-переходов КА:
1. нетерминальный левой части правила соответствует исходному состоянию КА, помеченному данным символом;
2. нетерминальный символ правой части соответствует состоянию перехода, обозначенному данным символом (т.е. $<X> \rightarrow b<Y>$ будет соответствовать



);

3. терминальные символы правой части правила соответствует последовательности входных символов, по которому производится переход из состояния, обозначенного нетерминальным символом левой части правила, в состояние, обозначенное нетерминальным символом правой части правила через цепочку промежуточных состояний;
4. аннулирующее правило соответствует переходу в заключительное состояние по пустому символу ε (в КА для лексического анализа это соответствует возвращению в начальное состояние перед получением следующей лексемы);
5. правило с цепочкой терминальных символов соответствует переходу в заключительное состояние по последовательности символов этой цепочки через промежуточные состояния;
6. правило с единственным нетерминальным символом соответствует переходу из одного состояния в другое по пустому символу ε.
Для построения полностью определенного автомата необходимо выполнение еще нескольких пунктов:
1. добавить еще одно состояние *U*, соответствующее ситуации, когда анализируемое слово не принадлежит языку;
2. для каждого состояния, у которого отсутствуют переходы по некоторым символам терминального алфавита, добавить переходы по этим символам в состояние *U*;
3. Из состояния *U* добавить переходы по всем символам терминального алфавита, ведущие в *U*.
Если необходимо распознавать слова, следует добавить по одному состоянию на каждое слово, и модифицировать пятое правило:
5. правило с цепочкой терминальных символов соответствует переходу в соответствующее заключительное состояние по последовательности символов этой цепочки через промежуточные состояния.

Пример.
В качестве примера построим распознаватель, определяющий принадлежность слов к заданному языку (подмножество слов языка из примера алгоритма Хаффмана), который состоит из следующих слов двоичного алфавита $V_m = \{0,1\}$,:

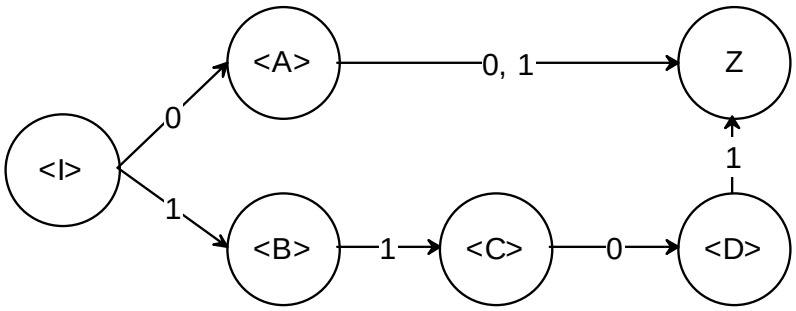
L_1= { 00, 01, 1101 }

Построим А-грамматику, порождающую этот язык. В качестве нетерминальных символов будем использовать прописные буквы латинского алфавита. Начальный символ грамматики обозначим <*I*>. Все слова начинаются либо с 0, либо с 1, поэтому построим правила <*I*> → *0* <*A*>, <*I*> → *1* <*B*>. Первое слово начинается с нуля и заканчивается нулем. Конечные правила должны иметь вид, определяемый последней буквой каждого слова . Следовательно, для первого слова имеем два правила: <*I*> → *0* <*A*> и <*A*> → *0*. Для второго слова необходимо правило <*A*> → *1*. Единственное слово, начинающееся с единицы, будет соответствовать правилам <*B*> → *1* <*C*>, <*C*> → *0* <*D*>, <*D*> → *1*.

В результате получаем схему искомой грамматики в виде:

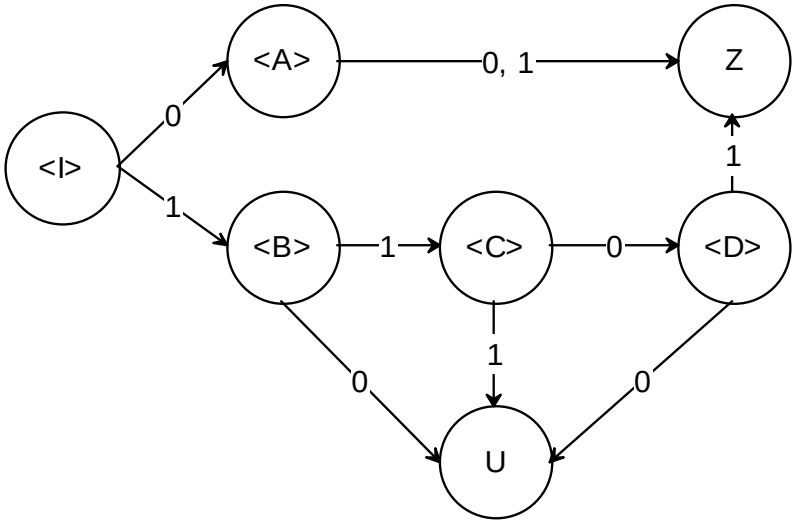
R = {<*I*> → *0* <*A*>, <*I*> → *1* <*B*>, <*A*> → *0*, <*A*> → *1*, <*B*> → *1* <*C*>, <*C*> → *0* <*D*>, <*D*> → *1*}.

Применяя вышеперечисленные правила, получим следующий конечный автомат:

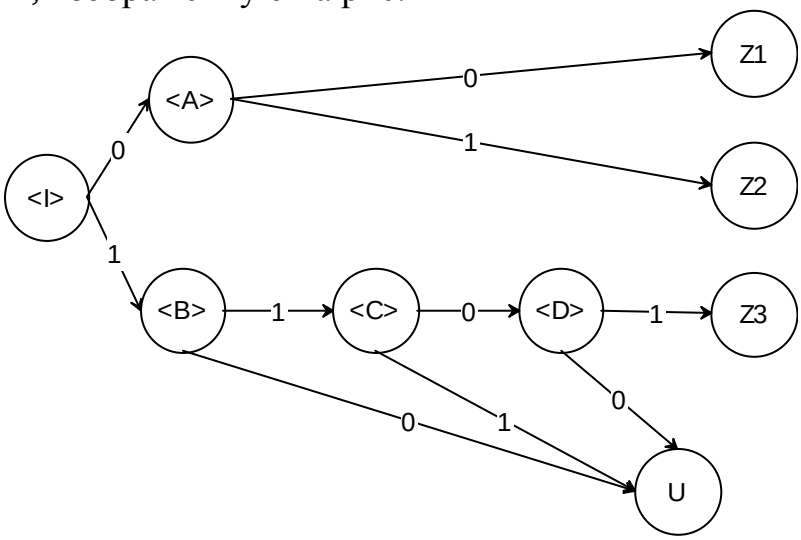


Следует отметить, что этот автомат является квазидетерминированным. Признаком того, что данное слово принадлежит языку, заданному автоматом, является то, что по окончании работы автомат находится в состоянии Z.

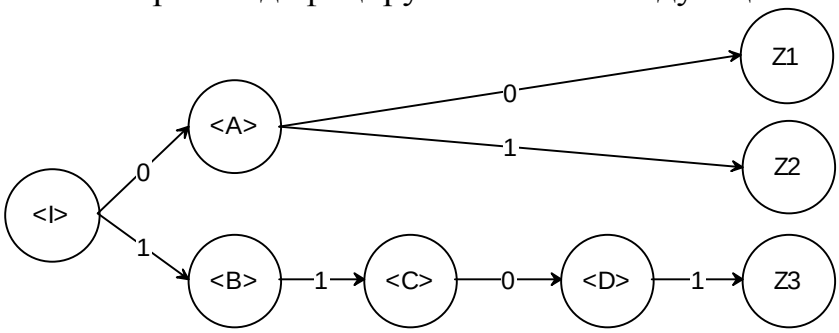
Для данной задачи можно построить полностью определенный ДКА.



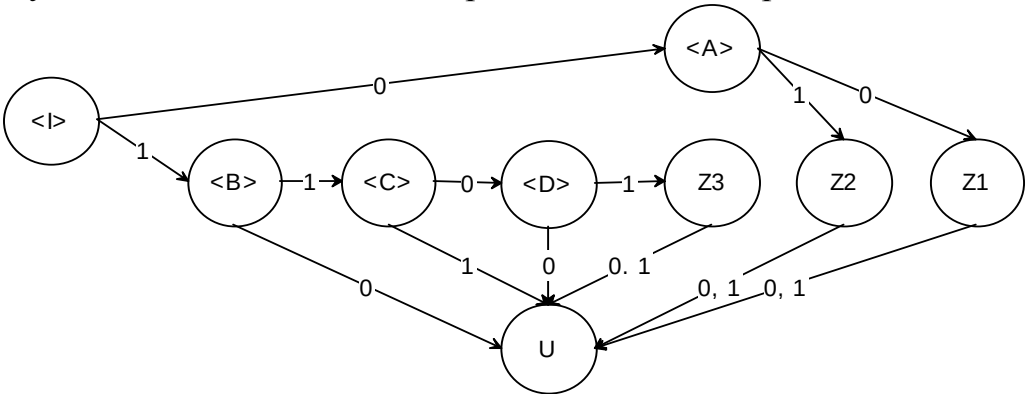
Для того чтобы построить распознаватель, определяющий, какое из слов входного языка подано на вход, заключительные правила построенной грамматики нужно дополнить не одним символом Z , а тремя различными символами: Z1, Z2, Z3. В результате получаем диаграмму переходов распознавателя, изображенную на рис.



Для распознавания строк модифицируем автомат следующим образом:



Ну, и наконец, полностью определенный ДКА, распознающий слова языка.



Распознаватели, построенные описанным способом, как правило, обладают избыточными состояниями. Такие состояния можно объединить с другими состояниями, не нарушая заданных функций распознавателя. Правила выявления и объединения состояний были рассмотрены нами ранее.

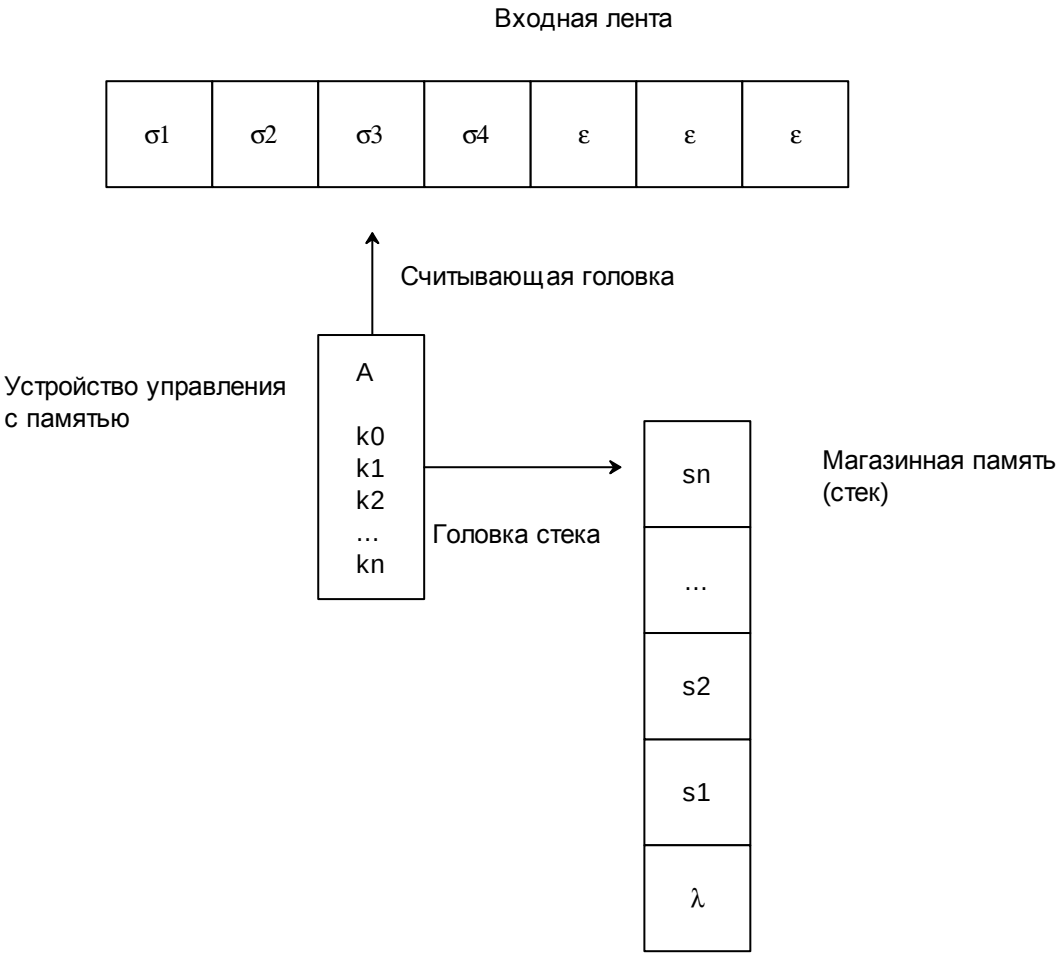
§ 6. Контекстно-свободные грамматики и МП-автоматы

КС-грамматики обладают несколькими важными особенностями:

1. Не существует в общем случае способов эквивалентного преобразования КС-

грамматики из одного вида в другой (например, невозможно преобразовать произвольно взятую КС-грамматику к какому-то стандартному виду).
2. Невозможно в общем случае доказать (и даже установить) эквивалентность двух и более произвольных КС-грамматик.
3. Невозможно определить, пересекаются ли языки, порождаемые двумя произвольными КС-грамматиками.
4. Невозможно определить, является ли данная КС-грамматика однозначной.
5. Невозможно определить, можно ли для языка, порожденного неоднозначной грамматикой, построить однозначную грамматику.
Эти особенности существенно затрудняют построение соответствующих распознавателей.
Более того, если для автоматных языков построение распознавателя по заданной грамматике однозначно, то для КС-грамматик это не так. Фактически, однозначно построить МП-автомат по заданной КС-грамматике невозможно.
Автоматы с магазинной памятью Языки, распознаваемые конечными автоматами – это языки, определяемые КЗ-грамматиками. Однако класс языков конекстно-зависимых грамматик не так широк, как хотелось бы. Для многоих практических задач гораздо большее значение имеют языки, определяемые КС-грамматиками. Для решения задач распознавания языков, определенных КС-грамматиками, используется особый вид автоматом, называемых *автоматами с магазинной памятью*, или *магазинными автоматами*.
Автомат с магазинной памятью (или МП-автомат) определяется набором:
$M=(K, \Sigma, \delta, s_0, F, S, \lambda)$
Где K - конечное множество состояний автомата,
$s_0 \in K$ - единственно допустимое начальное состояние автомата,
$F \subset K$ - множество конечных состояний, причём допустимо $F=\varnothing$, и $F=K$,
Σ - допустимый входной алфавит, из которого формируются строки, считываемые автоматом,
S - алфавит памяти (магазина),
$\lambda \in S$ - нулевой символ памяти.
δ - переходная функция, $\delta: K \times \Sigma \times S \to K \times S$
Память работает как стек, то есть для чтения доступен последний записанный в неё элемент.
По комбинации текущего состояния, входного символа и символа на вершине магазина автомат выбирает следующее состояние и, возможно, символ для записи в магазин. В случае, когда в правой части автоматного правила присутствует ε, в магазин ничего не добавляется, а элемент с вершины стирается. Если магазин пуст, то срабатывают правила с ε в левой части.
В работах, связанных с формальными языками и грамматиками, используется модель магазинного автомата, состоящая из входной ленты, устройства управления и вспомогательной ленты, называемой магазином или стеком. Входная лента разделяется на клетки (позиции), в каждой из которых может быть
записан символ входного алфавита. При этом предполагается, что в неиспольз-

зуемых клетках входной ленты расположены пустые символы ε. Вспомогательная лента также разделена на клетки, в которых могут располагаться символы магазинного алфавита. Начало вспомогательной ленты называется дном магазина. Связь устройства управления с лентами осуществляется двумя головками, которые могут перемещаться вдоль лент.



Головка входной ленты может перемещаться только в одну сторону – вправо или
оставаться на месте. Она может выполнять только чтение. Головка вспомогательной ленты способна выполнять как чтение, так и запись, но эти операции связаны с перемещением головки определенным образом:
- при записи головка предварительно сдвигается на одну позицию вверх, а затем символ заносится на ленту,
- при чтении символ, находящийся под головкой считывается с ленты, а затем головка сдвигается на одну позицию вниз,т.о.головка всегда установлена против последнего записанного символа. Позицию, находящуюся в рассматриваемый момент времени под головкой, называют вершиной магазина.

Автоматы с магазинной памятью распознают КС-языки.

Функции перехода магазинного автомата В дальнейшем при построении магазинных автоматов потребуются две разновидности функций переходов, изменяющих состояние автомата без перемещения входной головки: 1) функция

переходов с пустым символом в качестве входного символа:
$\delta^0(s, \varepsilon, h) = (s', \beta)$,
которая, независимо от того какой символ находится под читающей головкой входной ленты, предписывает прочитать символ h из вершины магазина, изменить состояние автомата на s' и записать строк β в магазин.
2)функция переходов с определенным входным символом:
$\delta^*(s, a, h) = (s', \beta)$,
которая предписывает изменение состояния и запись строки в магазин при условии, что символ a читается входной головкой, а в вершине магазина находится символ h.

Алгоритм построения недетерминированного МП-автомата по КС-грамматике В общем случае имеется возможность построить недетерминированный МП-автомат по произвольной КС-грамматике.

На практике обычно этой возможностью не пользуются, поскольку не существует однозначного способа свести НМП-автомат к ДМП-автомату, а алгоритмическая реализация НМП-автомата – весьма сложная задача.

Рассмотрим преобразование произвольной КС-грамматики $G= \{ V_m, V_A, <I> \in V_A, R \}$ в недетерминированный МП-автомат $M=(K, \Sigma, \delta, s_0, F, S, \lambda)$.

Множество состояний автомата K будет включать в себя 3 состояния - начальное, рабочее и конечное, $K=\{s_0, s, s_e\}$.

Входной алфавит Σ совпадает с терминальным алфавитом грамматики V_m, $\Sigma= V_m$.

Множество символов магазина представляет собой объединение множеств нетерминальных символов грамматики G и специального множества символов, в котором каждый символ соответствует символу терминального алфавита: $S= V_a \cup V_m^c$, V_m^c: $\exists f: V_m \to V_m^c$, и f - биекция. Будем так же считать, что функция f определена и на множестве V_a, причем возвращает она на нем те же нетерминальные символы.

Функция переходов δ определяется следующим образом:

1. $\delta(s0, \varepsilon, \lambda)=(s, <I>)$, в стек помещаем начальный символ $<I>$, не учитывая ни входную строку, ни содержимое стека.
2. $(\forall a \in \Sigma)\ \delta(s, a, f(a))=(s, \lambda)$, для любого терминального символа на входе и соответствующего ему символа в стеке выбрасываем символ из стека.
3. $(\forall <A> \in V_a)\ (\forall (<A> \to \alpha) \in R)\ \delta(s, \varepsilon, <A>)=(s, f(\alpha))$, для любого нетерминального символа на вершине стека происходит его замена на одно из слов в соответствии с правилами грамматики, при этом входной символ не читается.
4. $\delta(s, \varepsilon, \lambda)=(s_e, \lambda)$, если стек пуст, не читая входной символ переходим в заключительное состояние.

Работа данного автомата приводит к построению слова, принадлежащего языку, порожденному грамматикой G. Соответственно, данный НМП-автомат способен и распознавать такие слова.

Пример.

Рассмотрим грамматику, описывающую правильные скобочные выражения.

$G=\{\{(,)\},\{<I>\},<I>,\{<I>\to<I><I>,<I>\to(<I>),<I>\to\varepsilon\}\}$.
Соответствующий ей МП-автомат будет иметь следующий вид
$M=(\{s_0, s, s_e\}, \{(,)\}, \delta, s_0, \{s_e\}, \{<I>, (^c,)^s\}, \lambda)$,
$\delta(s0, \varepsilon, \lambda)=(s, <I>)$, в стек помещаем начальный символ $<I>$, не учитывая ни входную строку, ни содержимое стека.
$\delta(s, (, (^c)=(s, \lambda)$, для терминального символа (на входе и соответствующего ему символа в стеке выбрасываем символ из стека.
$\delta(s,),)^s)=(s, \lambda)$, для терминального символа (на входе и соответствующего ему символа в стеке выбрасываем символ из стека.
$\delta(s, \varepsilon, <I>)=(s, <I><I>)$, для нетерминального символа $<I>$ на вершине стека происходит его замена на одно из слов в соответствии с правилами грамматики, при этом входной символ не читается.
$\delta(s, \varepsilon, <I>)=(s, (<I>))$, аналогично.
$\delta(s, \varepsilon, <I>)=(s, \varepsilon)$, аналогично.
$\delta(s, \varepsilon, \lambda)=(s_e, \lambda)$, если стек пуст, не читая входной символ переходим в заключительное состояние.
При анализе строки этим МП-автоматом проблемы начинают возникать при выборе 4, 5 и 6 правил, поскольку неизвестно, какое из них необходимо использовать. Одновременное использование всех этих правил (автомат недетерминированный!) приводит к степенному росту используемого объема памяти, либо использованию возвратов.

§ 7. Синтаксический анализ

Подход, используемый для распознавания автоматных языков, для КС-языков неприменим, то есть в общем случае невозможно по заданной грамматике построить детерминированный МП-автомат, успешно реализуемый алгоритмически. Поэтому для распознавания языков, порождаемых КС-грамматиками, используют особые приемы.
При построении грамматики может возникнуть ситуация, когда некоторые нетерминальные символы использованы для описания грамматики, но не участвуют в выводе слов языка. Естественно, при анализе строки языка такие символы учитываться не должны. В таком случае, имеет смысл преобразовать заданную грамматику таким образом, чтобы в ней были использованы только реально задействованные в выводе нетерминальные символы.
Рассмотрим несколько определений.
Нетерминальный символ $<X>$ в КС-грамматике $G=(V_m, V_a, <I>\in V_a, R)$ называется *выводимым* если он:
1. является начальным символом грамматики, $<X>=<I>$
2. существует какая-либо цепочка вывода, хотя бы одна строка которой содержит данный символ, $\exists\alpha, \beta\in(Va\cup Vm): <I>=>*\alpha<X>\beta$
Символ $<X>$ КС-грамматики $G=(V_t, V_a, <I>\in V_a, R)$ называется *производящим*, если существует вывод строки терминальных символов α из этого символа, $\exists\alpha\in V_m*: <X>=>*\alpha$.

Нетерминальный символ $<X>$ грамматики $G=(V_t, V_a, <I>\in V_a, R)$ будем называть *существенным*, если он выводимый и производящий.

Алгоритм выделения существенных символов

Поскольку существенными символами мы назвали те, которые являются одновременно выводимыми и производящими, очевидно, что все нетерминальные существенные символы КС-грамматики можно найти как пересечение множеств выводимых символов и производящих символов.

Найдем множество выводимых символов.

Пусть дана КС-грамматика $G=(V_m, V_a, <I>\in V_a, R)$.

Построим множество V_i выводимых символов следующим образом:

1. $i=0, V_i=\{<I>\}$
2. $V_{i+1}=V_i \cup \{<X>: (<A> \to \alpha <X> \beta) \in R, A \in V_i\}$
3. если $|V_{i+1}|>|V_i|$, перейти к шагу 2.

Построим множество W_i производящих символов следующим образом:

1. $i=0, W_i=\{V_m\}$
2. $V_{i+1}=V_i \cup \{<X>: (<X> \to \alpha) \in R, \alpha \in W_i^*\}$
3. если $|W_{i+1}|>|W_i|$, перейти к шагу 2.

Отбросив все правила, в которых встречаются несущественные символы, мы получим грамматику, эквивалентную исходной.

КС-грамматика, в правила которой входят только существенные символы, называется *редуцированной*.

Разбор строки

Как уже было сказано, задача принадлежности строки к языку может быть решена путем построения дерева вывода для данной строки. Если такое построение возможно, строка принадлежит заданному языку.

Для автоматных грамматик данная задача может быть решена путем использования автомата-распознавателя, для КС-грамматик эта задача сложнее.

В общем случае для контекстной грамматики эта задача может быть решена с использованием двух походов - нисходящего и восходящего разбора.

В случае нисходящего разбора задача сводится к выбору правил, позволяющих из начального символа получить заданную строку.

В случае восходящего разбора мы пытаемся подобрать правила, на основе которых могла быть получена заданная строка.

Другими словами, в случае нисходящего разбора мы из начального символа, подбирая правила, пытаемся получить исходную строку, а в случае восходящего разбора, на основе строки, подбирая правила, пытаемся получить начальный символ.

Не все виды грамматик одинаково пригодны к различным типам разборов.

Среди КС-грамматик различают LL(k)-грамматики (от Left-to-Right Leftmost derivation, разбор строки слева направо с использованием левостороннего вывода). К этому типу относятся такие грамматики, в которых из правил с одинаковыми левыми частями можно однозначно выбрать любое правило по k-1 начальным нетерминальным символам правой части правила.

Помимо LL(k)-грамматик выделяют так же LR(k)-грамматики (от Left-to-Right

Rightmost derivation - разбор слева направо по правостороннему выводу) схожи с LL(k)-грамматиками.
Самый простой способ синтаксического разбора в соответствии с грамматикой, используемый в настоящее время, называется LL-парсингом (аналогично названию LL-грамматик, в русскоязычной литературе такой анализ часто называют разбором сверху-вниз).
Для того, чтобы грамматика допускала разбор по методу LL, необходимо выполнение нескольких условий:
1. Правая часть каждого правила начинается терминалом.
2. Если два правила имеют одинаковые левые части, то правые части этих правил должны начинаться различными терминальными символами.
Грамматика, правила которой удовлетворяют этим требованиям, и не содержащая аннулирующих правил, называется *простой*, или *разделенной*.

Алгоритм построения детерминированного нисходящего МП-распознавателя для разделенной грамматики

1. Множество состояний распознавателя примем $K=\{s_0, s_k\}$, где s_k - конечное состояние.
2. Входной алфавит распознавателя совпадает с терминальным алфавитом грамматики: $\Sigma=Vm$.
3. Алфавит стека есть объединение терминального и нетерминального алфавитов грамматики $S=Va \cup Nm \cup \{\lambda\}$.
4. Функция перехода автомата δ строится следующим образом:

4.1. каждому правилу разделенной грамматики $<A> \rightarrow a\alpha$ поставим в соответствие команду $f(s_0, a, <A>) = (s_0, \alpha')$, где α' - строка α в зеркальном отражении;
4.2. для всех терминальных символов, которые присутствуют в правой части какого-либо правила не на первой позиции, строятся правила $f(s_0, b, b) = (s_0, \varepsilon)$;
4.3. переход в заключительное состояние осуществляется по команде $f(s_0, \varepsilon, h_0) = (s_1, \varepsilon)$.

5. Начальная конфигурация автомата имеет следующий вид: (s0 , a , λ<I>).

Пример.
Рассмотрим грамматику

$R = \{$
$<I> \rightarrow a,$
$<I> \rightarrow b+(<B>),$
$<B> \rightarrow a+b,$
$<B> \rightarrow b+<B>$
$\}$

Данная грамматика является разделенной LL(1)-грамматикой. Построим для нее детерминированный МП-распознаватель.

1. $K=\{s_0, s_k\}$.
2. $\Sigma=\{a, b, +, (,)\}$.
3. $S=\{ a, b, +, (,), <I>, <B>, \lambda\}$.
4.

$f(s_0, a, <I>) = (s_0, \varepsilon)$
$f(s_0, b, <I>) = (s_0,)<B>(+)$
$f(s_0, a, <B>) = (s_0, b+)$
$f(s_0, b, <B>) = (s_0, <B>+)$
$f(s_0,),)) = (s_0, \varepsilon)$
$f(s_0, +, +) = (s_0, \varepsilon)$
$f(s_0, (, ( ) = (s_0, \varepsilon)(s_0 , \varepsilon , \lambda) = (s_k , \varepsilon)$.

5. Начальная конфигурация автомата имеет следующий вид: $(s_0 , (b+(a)) , \lambda<I>)$.

Пример вывода выглядит следующим образом:

$(s_0$, b+(a+b) , $\lambda<I>)$ |-
$(s_0$, +(a+b) , $\lambda)<B>(+)$ |-
$(s_0$, (a+b) , $\lambda)<B>( )$ |-
$(s_0$, a+b) , $\lambda)<B>)$ |-
$(s_0$, +b) , $\lambda)b+)$ |-
$(s_0$, b) , $\lambda)b)$ |-
$(s_0$,) , $\lambda))$ |-
$(s_0 , \varepsilon , \lambda)$ |-
$(s_k , \varepsilon , \varepsilon)$

LL(k)-грамматика

Разделенная грамматика описывает слишком узкий класс языков, чего недостаточно для решения практических задач. Например, в разделенных грамматиках отсутствуют аннулирующие правила.

Более широким классом являются LL(k)-грамматики.

Функции ПЕРВ, СЛЕД и ВЫБОР При работе МП-автомата возникает вопрос, в каких случаях применять то или иное правило. Для решения этой проблемы используется специальная функция *ВЫБОР*. Эта функция возвращает множество терминальных символов, при появлении которых под читающей головкой распознавателя должно быть применено данное правило.

Основой функции *ВЫБОР* служат функции *ПЕРВ* и *СЛЕД*.

Аргументом функции *ПЕРВ* является строка символов терминального и нетерминального алфавитов, а возвращает она множество терминальных символов, которые могут стоять на первом месте в строках, выводимых из строки, являющейся аргументом функции *ПЕРВ*.

Значение функции *ПЕРВ*(μ) можно определить пользуясь следующими правилами:

1) Если строка μ начинается терминальным символом и имеет вид bμ', то функция

ПЕРВ(μ) = {b},
2) Если строка μ является пустой строкой, $\mu = \varepsilon$, то функция
ПЕРВ(μ) = ε,
3) Если строка μ начинается нетерминальным символом <*B*> и имеет вид <*B*>μ', а в схеме грамматики имеется *n* правил, в левой части которых находится символ <*B*>:
$<B> \rightarrow \alpha_1 \mid \alpha_2 \mid ... \mid \alpha_n$,
и, если не существует вывода <*B*> =>* ε, то функция *ПЕРВ(<B>μ')* представляет собой объединение множеств:
$ПЕРВ(<B>\mu') = ПЕРВ(\alpha_1) \cup ПЕРВ(\alpha_2) \cup ... \cup ПЕРВ(\alpha_n)$,
4) Если цепочка μ начинается нетерминальным символом и имеет вид <*B*>μ', в схему грамматики входят *n* правил вида
$<B> \rightarrow \alpha_1 \mid \alpha_2 \mid ... \mid \alpha_n$,
и <*B*> является аннулирующим нетерминалом, т.е. существует <*B*> => *ε, то функция
$ПЕРВ(<B>\mu') = ПЕРВ(\mu') \cup ПЕРВ(\alpha_1) \cup ПЕРВ(\alpha_2) \cup ... \cup ПЕРВ(\alpha_n)$.
Аргументом функции *СЛЕД* является нетерминальный символ, а значение функции *СЛЕД(<B>)* представляет собой множество терминалов, которые могут следовать непосредственно за нетерминалом <*B*> в строках, выводимых из начального символа грамматики.
Вычисление значения функции *СЛЕД(<B>)* должно выполняться по следующим правилам:
1) Если в схеме грамматики имеются правила вида
$<X_1> \rightarrow \mu_1 <B> \alpha_1$, $<X_2> \rightarrow \mu_2 <B> \alpha_2$, ... , $<X_n> \rightarrow \mu_n <B> \alpha_n$,
и все цепочки $\alpha_i \neq \varepsilon$, то
$СЛЕД(<B>) = ПЕРВ(\alpha_1) \cup ПЕРВ(\alpha_2) \cup ... \cup ПЕРВ(\alpha_n)$.
2) Если же среди приведенных выше правил имеется хотя бы одна строка $\alpha_i = \varepsilon$, например пусть $\alpha_1 = \varepsilon$, то функция вычисляется так:
$СЛЕД(<B>) = СЛЕД(<X_1>) \cup ПЕРВ(\alpha_2) \cup ... \cup ПЕРВ(\alpha_n)$.
Функция *ВЫБОР*, которая потребуется нам для построения переходов магазинных автоматов,можно определить с помощью функций *ПЕРВ* и *СЛЕД* следующим образом:
1) Если правило грамматики имеет вид $<B> \rightarrow \alpha$ и α не является аннулирующей строкой, другими словами не существует вывод α =>*ε, то
$ВЫБОР(<B> \rightarrow \alpha) = ПЕРВ(\alpha)$.
2) Для аннулирующих правил грамматики вида <B> $\rightarrow$ ε, множество выбора определяется так
$ВЫБОР(<B> \rightarrow \varepsilon) = СЛЕД(<B>)$.
3) Если правило грамматики имеет вид $<B> \rightarrow \mu$ и μ является аннулирующей строкой, то
$ВЫБОР(<B> \rightarrow \mu) = ПЕРВ(\mu) \cup СЛЕД(<B>)$.

КС-грамматика называется *слаборазделенной*, если выполняются следующие

условия:

- правая часть каждого правила представляет собой либо пустую строку ε, либо начинается с терминального символа,
- если два правила имеют одинаковые левые части, то правые части правил должны начинаться разными символами,
- для каждого нетерминала <A>, такого что <A> =>* ε множество начальных символов не должно пересекаться с множеством символов, следующих за <A>: $ПЕРВ(A) \cap СЛЕД(A) = \varnothing$

КС-грамматика является LL(1)-грамматикой тогда и только тогда, когда выполняются следующие условия:

Для каждого нетерминала, являющегося левой частью нескольких правил:

$<A> \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$,

необходимо, чтобы пересечение функций $ПЕРВ(\alpha_i)$ и $ПЕРВ(\alpha_j)$ было пусто для всех $i \neq j$.

Для каждого аннулирующего нетерминала <A>,такого что <A> =>* ε, необходимо, чтобы пересечение множеств $ПЕРВ(<A>)$ и $СЛЕД(<A>)$ было пустым.

Из определения следует, что грамматики LL(1), в отличие от разделенных грамматик и слаборазделенных, могут содержать правила, начинающиеся нетерминальными символами

КС-грамматика называется LL(1) грамматикой тогда и только тогда, когда множества ВЫБОР, построенные для правил с одинаковой левой частью, не содержат одинаковых элементов.

Алгоритм построения детерминированного МП-распознавателя по LL(1)-грамматике

Известно следующее: для каждой LL(1)-грамматики G можно построить МП-распознаватель M, допускающий язык, порождаемый грамматикой G.

Задача построения магазинного автомата для заданной LL(1)-грамматики формулируется следующим образом.

Задана грамматика $G = \{V_m, V_a, <I>, R\}$, и требуется определить МП-автомат $M=(K, \Sigma, \delta, s_0, F, S, \lambda)$.

Порядок построения МП-автомата следующий:

1. Входной алфавит автомата совпадает с терминальным алфавитом грамматики: $\Sigma = V_m$.
2. Для упрощения автомата будем рассматривать только два состояния, $K=\{s_0, s_к\}$, $F= K=\{s_к\}$.
3. Алфавит стека является объединением терминального и нетерминального алфавитов, а так же включает символ конца стека: $S=\{V_m \cup V_a \cup \{\lambda\}\}$.
4. Функция переходов строится по следующим правилам

4.1. Для каждого правила грамматики, правая часть которого начинается терминальным символом $<A> \rightarrow b\alpha$, функция перехода имеет вид $f(s_0 , b , <A>) = (s_0 , \alpha')$, где α' - зеркальное отражение строки α.

4.2. Для каждого правила грамматики, правая часть которого начинается с

нетерминального символа $<A> \rightarrow <B>\alpha$, функция перехода имеет вид $f^*(s_0 , x , <A>) = (s_0 , \alpha' <B>)$, где f^* – команда чтения входного символа без сдвига читающей головки, α' - зеркальное отражение строки α, x - множество символов, полученных с использованием функции ВЫБОР($<A> \rightarrow <B>\alpha$). Количество команд определяется мощностью множества ВЫБОР($<A> \rightarrow <B>\alpha$).

4.3. Для каждого аннулирующего правила грамматики $<A> \rightarrow \varepsilon$ строятся команды вида $f^*(s_0 , x , <A>) = (s_0 , \varepsilon)$ где f^* – команда чтения входного символа без сдвига читающей головки, x - множество символов, полученных с использованием функции ВЫБОР($<A> \rightarrow \varepsilon$).

4.4. Для каждого терминального символа b, расположенного в середине или на конце правых частей правил грамматики, построим команду $f(s_0 , b , b) = (s_0 , \varepsilon)$.

4.5. Для перехода в заключительное состояние используем команду $f^*(s_0 , \varepsilon , \lambda) = (s_к , \varepsilon , \varepsilon)$.

5. Начальная конфигурация традиционно имеет вид $(s_0, \mu, \lambda<I>)$

Пример.

Рассмотрим грамматику

$R=\{<A> \rightarrow x,$
$<A> \rightarrow (<B>),$
$<B> \rightarrow <A><C>,$
$<C> \rightarrow +<A><C>,$
$<C> \rightarrow \varepsilon\}$

Построим множества ВЫБОР для каждого из правил для проверки принадлежности данной грамматики к классу LL(1)-грамматик.

ВЫБОР($<A> \rightarrow x$) = *ПЕРВ*(x) = $\{x\}$
ВЫБОР($<A> \rightarrow (<B>)$) = *ПЕРВ*($(<B>)$) = $\{(\}$
ВЫБОР($<B> \rightarrow <A><C>$) = *ПЕРВ*($<A><C>$) = $\{x, (\}$
ВЫБОР($<C> \rightarrow +<A><C>$) = *ПЕРВ*($+<A><C>$) = $\{+\}$
ВЫБОР($<C> \rightarrow \varepsilon$) = *СЛЕД*($<C>$) = *СЛЕД*($<B>$) = $\{) \}$.

Множества выбора для правил с символом $<A>$ в левой части и для правил с символом $<C>$ в левой части не пересекаются, следовательно, данная грамматика относится к классу LL(1)-грамматик.

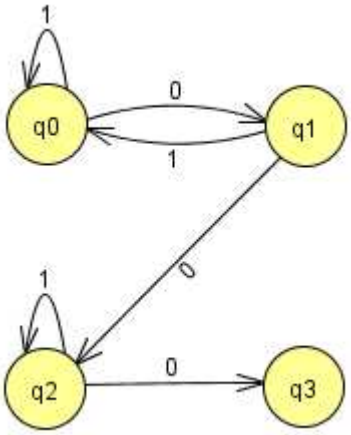
Используя функции ВЫБОР, построим соответствующие им команды искомого автомата.

$f(s_0 , x , <A>) = (s_0 , \varepsilon)$ - по п.4.1;
$f(s_0 , (, <A>) = (s_0 ,)<B>)$ - по п.4.1;
$f(s_0 , + , <C>) = (s_0 , <C><A>)$ - по п.4.1;
$f^*(s_0 , (, <B>) = (s_0 , <C><A>)$ - по п.4.2;
$f^*(s_0 , x , <B>) = (s_0 , <C><A>)$ - по п.4.2;
$f^*(s_0 ,) , <C>) = (s_0 , \varepsilon)$ - по п.4.3;

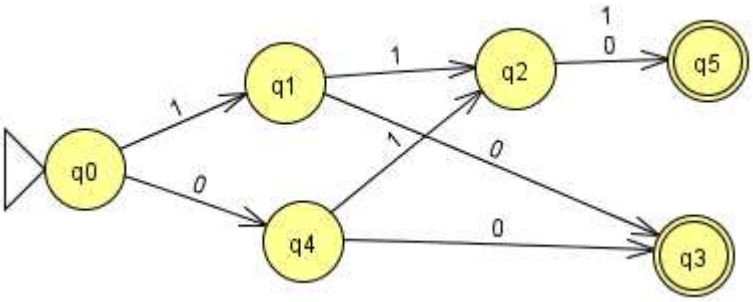
$f(s_0,),)) = (s_0, \varepsilon)$ - по п.4.4;
$f(s_0, \varepsilon, \lambda) = (s_к, \varepsilon)$ - по п.4.5.
Учитывая, что закрывающиеся скобки расположены на последнем месте второго правила, построим команду
Кроме того, построим команду для перехода в заключительное состояние:
Рассмотрим работу полученного автомата на примере строки (x+x+x)
(s_0 , (x+x+x) , λ<A>) |-
(s_0 , x+x+x) , λ)<B>) |-
(s_0 , x+x+x) , λ)<C><A>) |-
(s_0 , +x+x) , λ)<C>) |-
(s_0 , x+x) , λ)<C><A>) |-
(s_0 , +x) , λ)<C>) |-
(s_0 , x) , λ)<C><A>) |-
(s_0 ,) , λ)<C>) |-
(s_0 ,) , λ)) |-
(s_0 , ε , λ) |-
($s_г$, ε , ε)

Задачи и упражнения

1. Приведите пример конечного автомата, заданного в виде диаграммы переходов.
2. По диаграмме переходов определите, является ли данный автомат детерминированным, квазидетерминированным или недетерминированным



.
3. Минимизируйте детерминированный конечный автомат, описанный диаграммой переходов



4. Постройте редуцированную грамматику для заданной
5. Определите тип грамматики R = {*<I> → 0 <A>, <I> → 1 <B>, <A> → 0, <A> → 1, <B> → 1 <C>, <C> → 0 <D>, <D> → 1*}.
6. Постройте детерминированный автомат с магазинной памятью для следующей грамматики R = {*<I> → 0 <A>, <I> → 1 <B>, <A> → 0, <A> → 1, <B> → 1 (<B>), <B> → 0 <D>, <D> → 1*}.
7. Найдите значение функции *ПЕРВ(<A>)* для грамматики R = {*<I> → 0 <A>, <I> → 1 <B>, <A> → 0, <A> → 1, <B> → 1 <C>, <C> → 0 <D>, <D> → 1*}.

Глава 7

ТРАНСЛЯЦИЯ И ТРАНСЛЯТОРЫ

В общем случае перевод или трансляцию можно представить как соответствие между словами алфавита *P* и алфавита *W*. Если слово $\alpha \in P^*$, слово $\beta \in W^*$, и слово β соответствует слову α, они называются соответственно входным и выходным словами.
Если заданы входной алфавит *P* и выходной алфавит *W*, то *переводом* с языка $L_{вх}$, состоящего из слов множества *P**, на язык $L_{вых}$, состоящий из слов множества *W**, называется множество *C* пар слов (α,β) таких, что $\alpha \in L_{вх}$ и $\beta \in L_{вых}$.
$C=\{(\alpha,\beta) : \alpha \in L_{вх} \text{ и } \beta \in L_{вых}\}$.

Таким образом, перевод устанавливает соответствие слов языка $L_{вх}$ словам языка $L_{вых}$, а следовательно, является отношением из $L_{вх}$ в $L_{вых}$.
Конечные переводы с небольшим числом элементов можно задавать в виде перечислений. Однако, для задания больших конечных переводов и бесконечных переводов нужны такие же средства определения, как и для формальных языков.

Пример.
Вернемся к алгоритму Хаффмана. Его тоже можно рассматривать как перевод:

Символ	Частота	До сжатия	После сжатия
a	*0.2*	*0000*	*00*
b	*0.2*	*0001*	*01*
c	*0.2*	*0010*	*10*
d	*0.15*	*0011*	*1100*
e	*0.15*	*0100*	*1101*
f	*0.15*	*0101*	*1110*
g	*0.1*	*0110*	*111100*

Алгоритм Хаффмана можно представить как отношение между двумя множествами *{a, b, c, d, e, f, g}* и *{00, 01, 10, 1100, 1101, 1110, 111100}*, и это отношение будет выглядеть следующим образом
{(a, 00), (b, 01), (c, 10), (d, 1100), (e, 1101), (f, 1110), (g, 111100)}.

§ 1. Перевод с использованием грамматик

Пусть входной язык перевода *C* можно задать с помощью грамматики *G* и вы-

ходной язык перевода C – с помощью грамматики G'. Примером такого задания может служить перевод, определяющий для каждой входной строки α ее зеркальное отображение α'. Если входной и выходной алфавиты состоят из двух символов - {0,1}, то правила построения такого перевода имеют вид:

Входные строки	Выходные строки
$<I> \rightarrow 0<I>$	$<I> \rightarrow <I>0$
$<I> \rightarrow 1<I>$	$<I> \rightarrow <I>1$
$<I> \rightarrow \varepsilon$	$<I> \rightarrow \varepsilon$

Применяя одновременно соответствующие правила построения к входной и выходной строкам, получаем:
(<I>,<I>) ==> (0<I>,<I>0) ==> (00<I>,<I>00) ==> (001<I>,<I>100) ==> (001,100)
При таком построении существенное значение имеет заданное соответствие между правилами построения входных и выходных цепочек.
Синтаксически управляемые схемы В общем случае можно описать подобный перевод при помощи так называемой СУ-схемы.
Схемой синтаксически управляемого перевода (СУ-схемой) называется совокупность $T = \{V_a, V_{твх}, V_{твых}, Q, <I>\}$,
где V_a - множество нетерминальных символов,
$V_{твх}$ - множество терминальных символов, используемых для построения входных цепочек,
$V_{твых}$ - множество терминальных символов, используемых для построения выходных цепочек,
$<I>$ - начальный символ, $<I> \in Va$,
Q - множество правил вида $<A> \rightarrow \alpha, \beta$,
где $<A>$ принадлежит V_a, $\alpha \in (V_a \cup V_{твх})^*$, $\beta \in (V_a \cup V_{твых})^*$ и нетерминалы, входящие в цепочку β образуют перестановку нетерминалов цепочки α.
Фактически, это означает, что каждое правило СУ-схемы одновременно определяет строку из нетерминалов и терминальных символов входного алфавита и строку из нетерминалов и терминальных символов выходного алфавита. СУ-схема позволяет получить грамматики входного и выходного языков.
Пусть $T = \{V_a, V_{твх}, V_{твых}, Q, I\}$ – СУ-схема, тогда грамматика $G = \{V_a, V_{твх}, R, I\}$, где $R = \{<A> \rightarrow \alpha : (<A> \rightarrow \alpha, \beta) \in Q\}$, называется входной грамматикой СУ-схемы T, а грамматика $G' = \{V_a, V_{твых}, R', I\}$, где $R' = \{<A> \rightarrow \beta : (<A> \rightarrow \alpha, \beta) \in Q\}$, называется выходной грамматикой СУ-схемы T.
С помощью СУ - схемы можно строить пары соответствующих строк. Такое построение называется *выводом СУ-схемы*, а получаемые пары строк - *выводимыми парами*.
Переводом $C(T)$, определяемым СУ-схемой T назовем множество пар, состоящих из входной и выходной цепочек, выводимых из пары, включающей два начальных символа. $C(T) = \{(\alpha, \beta) : (<I>,<I>) =>^* (\alpha, \beta)$ и $\alpha \in V_{твх}^*, \beta \in$

$V_{твых}$*}
В качестве примера рассмотрим СУ-схему, заданную следующим образом:
V_a = {<I>,<A>}, $V_{твх}$ = {0,1},$V_{твых}$ = {a,b}
Q = { <I> → 0<I>, <I>a;
<I> → 1, b;
}.
Эта схема определяет перевод, входные слова которого состоят из нулей и единиц, а выходные из букв *a, b*. Нулям входной цепочки должны соответствовать буквы *a*, а единицам – буквы *b* выходной цепочки, причем расположение символов в выходной цепочке должно быть зеркальным по отношению к соответствующим символам входной цепочки. Вывод в рассматриваемой СУ - схеме может, например, иметь вид:

(<I>,<I>) => (0<I>,<I>a) ==> (00<I>,<I>aa) ==>(0001,baaa).*

Определение СУ - схемы не накладывает никаких ограничений на правила, кроме необходимости совпадения нетерминалов во входной и выходной частях правила. На основе таких СУ-схем построить детерминированное устройство, осуществляющее перевод, не представляется возможным. Для таких задач пригоден более узкий класс СУ-схем, называемых простыми.
СУ-схема $T = \{V_a, V_{твх}, V_{твых}, Q, <I>\}$ называется *простой*, если для каждого правила $<A> \to \alpha, \beta$ из Q соответствующих друг другу вхождения нетерминалов встречаются в α и β в одном и том же порядке.
Перевод называется *простым СУ-переводом*, если он определяется простой СУ-схемой

Пример.
В качестве примера простой СУ - схемы, приведем СУ - схему, которая задает перевод в виде множества постфиксных и соответствующих им инфиксных выражений.

V_a = {E,T,F}, $V_{твх}$ = {+,*,a}, $V_{твых}$ = {+,*,a,(,)}
R= {E → ET+ , E+T;
E→T , T;
T→TF* , T*F;
T→F , F;
F→E , (E);
F→a , a; }.

Вывод в этой СУ - схеме, выполняемый путем применения правил к самым левым нетерминалам промежуточных цепочек, имеет вид:

(E,E) => (T,T) => (TF,T*F) => (FF*,F*F) =>(aF*,a*F) => (aE*,a*(E)) =>*
(aET+,a*(E+T))=> => (aTT+*,a*(T+T)) => (aFT+*,a*(F+T)) =>*

=> (aaT+,a*(a+T)) => (aaF+*,a*(a+F)) => (aaa+*,a*(a+a))*

Алгоритм построения простой СУ-схемы В общем случае построение СУ-схемы для заданного перевода – более сложная задача, чем построение двух грамматик, поскольку при этом необходимо учитывать соответствие между этими грамматиками. Однако, для простых СУ-схем задача построения существенно упрощается, поскольку расположение нетерминалов во входной и выходной строках правил должно быть одинаковым. Вначале строится грамматика, описывающая входной язык. Построение выходной грамматики имеет смысл совместить с построением правил СУ-схемы, так как порядок нетерминальных символов в грамматиках совпадает. Правила, состоящие только из нетерминальных символов, во входной и выходной грамматиках совпадают.

Пример.
В качестве примера рассмотрим построение перевода арифметических выражений, задаваемых следующей грамматикой, в постфиксные польские выражения.
$G:V_m = \{x,+,(.)\}\ V_a = \{A,B,C\}$
$R = \{$
$<A> \rightarrow x,$
$<A> \rightarrow (<B>),$
$<B> \rightarrow <A><C>,$
$<C> \rightarrow +<A><C>,$
$<C> \rightarrow \varepsilon$
$\}$
Поскольку постфиксные польские выражения не содержат скобок, $V_{твых} = \{\underline{x}, \underline{+}\}$. Выходные символы будем помечать подчеркиванием.
Для правил, содержащих только терминалы, достаточно заменить терминальный символ входной грамматики:
$<A> \rightarrow x\ ,\ \underline{x}\ .$
Третье правило грамматики не содержит терминалов:
$<B> \rightarrow <A><C>,<A><C>\ .$
Пятое правило является аннулирующим, поэтому оно должно сохраниться в выходной грамматике
$<C> \rightarrow \varepsilon\ ,\ \varepsilon\ .$
Второе правило грамматики содержит скобки, которые, согласно правилам построения, должны отсутствовать в постфиксной польской записи:
$<A> \rightarrow (<B>),<B>\ .$
Наибольший интерес представляет четвертое правило, поскольку именно в нем должен быть изменен порядок терминальных символов:
$<A> \rightarrow +<A><C>,\ <A>\underline{+}<C>\ .$
Построим множество правил искомой СУ - схемы:
$Q = \{<A> \rightarrow x,\underline{x},$
$<A> \rightarrow (<B>),<B>,$
$<B> \rightarrow <A><C>,\ <A><C>,$

<C> → +<A><C>, <A>+<C>,
<C> → ε, ε}.
Чтобы в первом приближении убедиться в правильности построения СУ-схемы, выполним вывод входной цепочки ((x+x)+(x+x)) и соответствующей ей выходной цепочки, используя
построенные правила.
(<A>,<A>) => ((<B>),<B>) => ((<A><C>),<A><C>) =>
*((<A>+<A><C>),<A><A>+<C>) => ((<A>+<A>),<A><A>+) =>**
*(((<B>)+(<B>)),<B><B>+) =>**
*(((<A><C>)+(<A><C>)),<A><C><A><C>+) =>**
*(((<A>+<A><C>)+(<A>+<A><C>)),<A><A>+<C><A><A>+<C>+) =>**
*(((<A>+<A>)+(<A>+<A>)),<A><A>+<A><A>++) =>**
(((x+x)+(x+x)),xx+xx++)

Транслирующие грамматики Помимо СУ-схем, существует еще один способ описать перевод из одной грамматики в другую. Этот способ называется построением транслирующей грамматики. Суть транслирующей грамматики заключается в одновременном построении строк входного и выходного языка. Соответственно, транслирующая грамматика включает одновременно и входной, и выходной алфавиты.
Транслирующей грамматикой (Т-грамматикой) называется КС-грамматика, множество терминальных символов которой разбито на множество входных символов и множество выходных символов, которые называются также символами действия.
Для того, чтобы избежать путаницы во входных и выходных символах, символы выходного алфавита принято выделять каким-либо образом.

Пример.
Рассмотрим пример Т-грамматики
$V_{твх}$ = {a,b,c}, $V_{твых}$ = {1,2,3}, V_a = {<I>,<A>}
R = {
<I> → a<I>1<A>,
<I> → 3,
<A> → <A><C>,
<A> → b2
}.
Вывод в транслирующих грамматиках выполняется аналогично выводу в КС - грамматиках. В данной грамматике может быть выведена следующая строка:

<I> => a<I>1<A> => a31<A> => a31b2

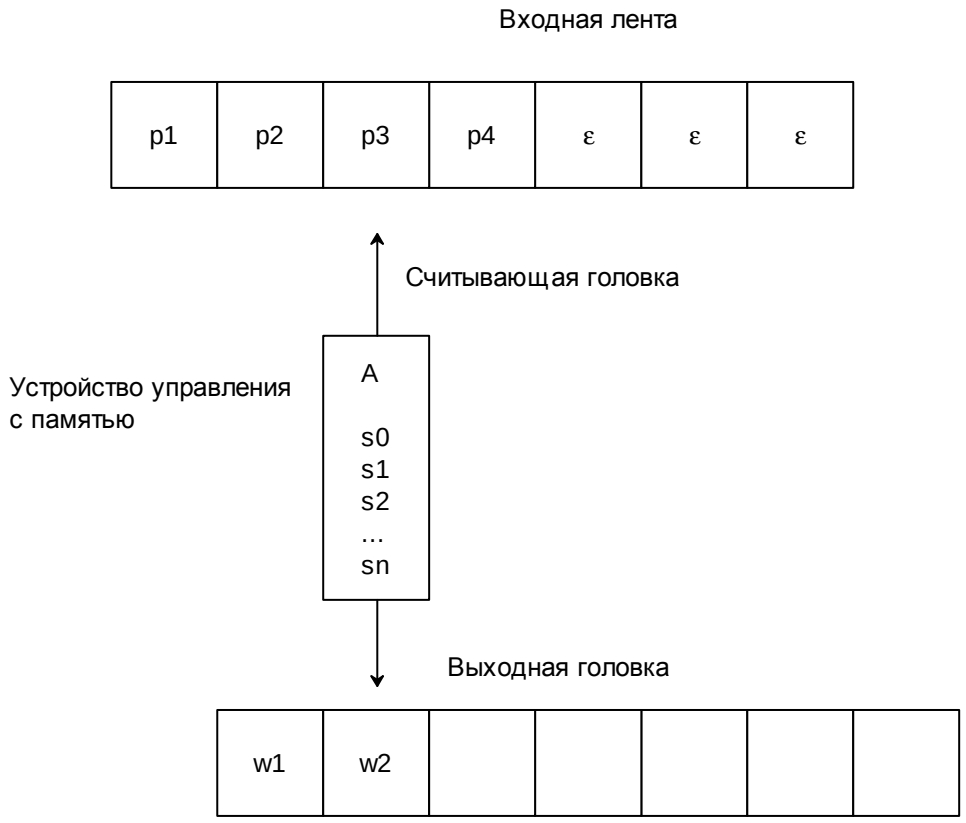
Выделенные символы или строки символов, рассматриваются как единый символ, называемый символом действия. В общем случае выделенные строки символов, рассматриваются как имена процедур, выполнение которых производит

требуемый эффект на выходе. Когда нужно подчеркнуть, что используется такая интерпретация символов действия, то Т-грамматику называют *грамматикой цепочного перевода*.

Получение входной и выходной грамматик из транслирующей Из каждой Т-грамматики можно получить две обычных грамматики, одна из которых позволяет строить входные строки, а другая - выходные.
Для получения грамматики входного языка достаточно из правил вывода удалить символы действия, или символы выходного языка $V_{твых}$.
Наоборот, для получения грамматики выходного языка достаточно удалить из правил транслирующей грамматики символы входного языка $V_{твх}$.
Соответствующие языки называются *входным и выходным языками транслирующей грамматики*.
Строки символов, получаемые выводом из начального символа Т-грамматики, можно преобразовать в строки входного и выходного языков. Для этого необходимо удалить из полученной строки соответственно символы выходного и входного терминальных алфавитов.

§ 2. Автоматные преобразователи

Применение автоматных преобразователей имеет смысл не только с точки зрения преобразования программного кода, например, с одного языка на другой, но и в некоторых других случаях.
Например, часто встречается ситуация, когда настройки той или иной системы хранятся в текстовом виде в конфигурационных файл (что характерно для Unix-систем, например).
Естественно, форма записи в таких файлах должна быть человекочитаемой, но внутреннее представление той же информации может существенно отличаться. Именно для преобразования информации из удобной для человека во внутреннюю форму и может быть использован А-преобразователь.
Пусть задан перевод C, тогда *преобразователем или транслятором* назовем устройство, которое по заданной строке $\alpha \in L_{вх}$, находит строку $\rho \in L_{вх}$, такую что $(\alpha, \rho) \in C$.
Простейшим преобразователем является конечный автоматный преобразователь (А- преобразователь). Он отличается от автомата без выходов тем, что в каждом такте работы он может создавать на выходной ленте в общем случае цепочку выходных символов. Схему такого преобразователя можно изобразить так:



A-преобразователь A определяется следующей совокупностью объектов:
$A = \{P, S, s_0, F, W, \varphi\}$,
где P - входной алфавит, состоящий из символов, записываемых на входную ленту,
S - множество состояний автомата,
s_0- начальное состояние из множества, $s_0 \in S$,
F - множество конечных или заключительных состояний, представляющих собой подмножество, $F \subset S$,
W - выходной алфавит, содержащий символы, записываемые на выходную ленту,
φ - функция переходов, которая в общем случае задает отображение.
$\varphi: P \times S \rightarrow S \times W^*$,
которое в функциональной форме можно записать так:
$\varphi(s, p)=(s', \gamma)$,
где $p \in P$, $s,s' \in S$, $\gamma \in W^*$.
В частном случае, когда входному символу соответствует отдельный выходной символ,
$\varphi: P \times S \rightarrow S \times W$.
В функциональной форме отображение имеет вид:
$\varphi\ q(s, p)=(s',w)$,
где $p \in P$, $s,s' \in S$, $w \in W$.
Этот частный случай соответствует определению конечного автомата с выходами.
Работу КП удобно описывать с помощью смены конфигураций. Как обычно,

конфигурация представляет собой совокупность
(s, α, γ),
где $s \in S$, $\alpha \in P^*$ и $\gamma \in W^*$.
Смену конфигураций обозначим так:
$(s, \alpha, \gamma\mu) \vdash (s', \beta, \gamma'\mu)$.
Последовательности сменяющих друг друга конфигураций, как обычно, будем обозначать символом $\vdash^*$.
Строку γ назовем *выходом входной строки* α, если существует последовательность конфигураций $(s_0, \alpha, \varepsilon) \vdash^* (s', \varepsilon, \gamma)$
для некоторого $s' \in F$.
Множество пар, состоящих из входных и соответствующих им выходных строк, назовем *переводом, определяемым преобразователем A* и обозначим *C(A)*.
$C(A) = \{ (\alpha, \gamma):(s_0, \alpha, \varepsilon) \vdash^* (s', \varepsilon, \gamma) \ \& \ s \in F \}$.
Известно, что для каждой простой СУ-схемы $T=\{V_A, V_{твх}, V_{твых}, I, Q\}$, у которой входная и выходная грамматики являются автоматными, можно построить конечный преобразователь *A*, такой что $C(A)=C(T)$.
К сожалению (и это очевидно при рассмотрении переходов а-преобразователя), автоматный преобразователь способен преобразовывать язык, порожденный правосторонней грамматикой, в язык, порожденный правосторонней грамматикой же.
Невозможно преобразовать правило Т-грамматики следующего вида $<A> \rightarrow a<A>\underline{b}$ во фрагмент а-преобразователя, если $a \in V_{твх}$, а $\underline{b} \in V_{твых}$.
Алгоритм построения А-преобразователя Построение преобразователя *A* по заданной СУ-схеме *T* начнем с того, что примем $P = V_{твх}$, $W = V_{твых}$, $S = V_A$, $s_0 = <I>$.
Далее вводится нетерминальный символ <Z> для обозначения конечного состояния. Множество допустимых заключительных состояний примем $F = <Z>$.
Функцию переходов преобразователя определим следующим образом:
1. для каждого правила СУ-схемы $<A> \rightarrow a<B>, \underline{b}<B>$, построим команду $\varphi(<A>,a)=(<B>,\underline{b})$.
2. для каждого правила $<A> \rightarrow a,\underline{b}$ построим правило вида
$<A> \rightarrow a<Z>,\underline{b}<Z>$, где Z - заключительное состояние и
команду $\varphi\ (<A>,a)=(<Z>,\underline{b})$.
На диаграмме состояний преобразователя правилу
$<A> \rightarrow a<B>,\underline{b}<B>$ ставится в соответствие следующий фрагмент графа автомата.

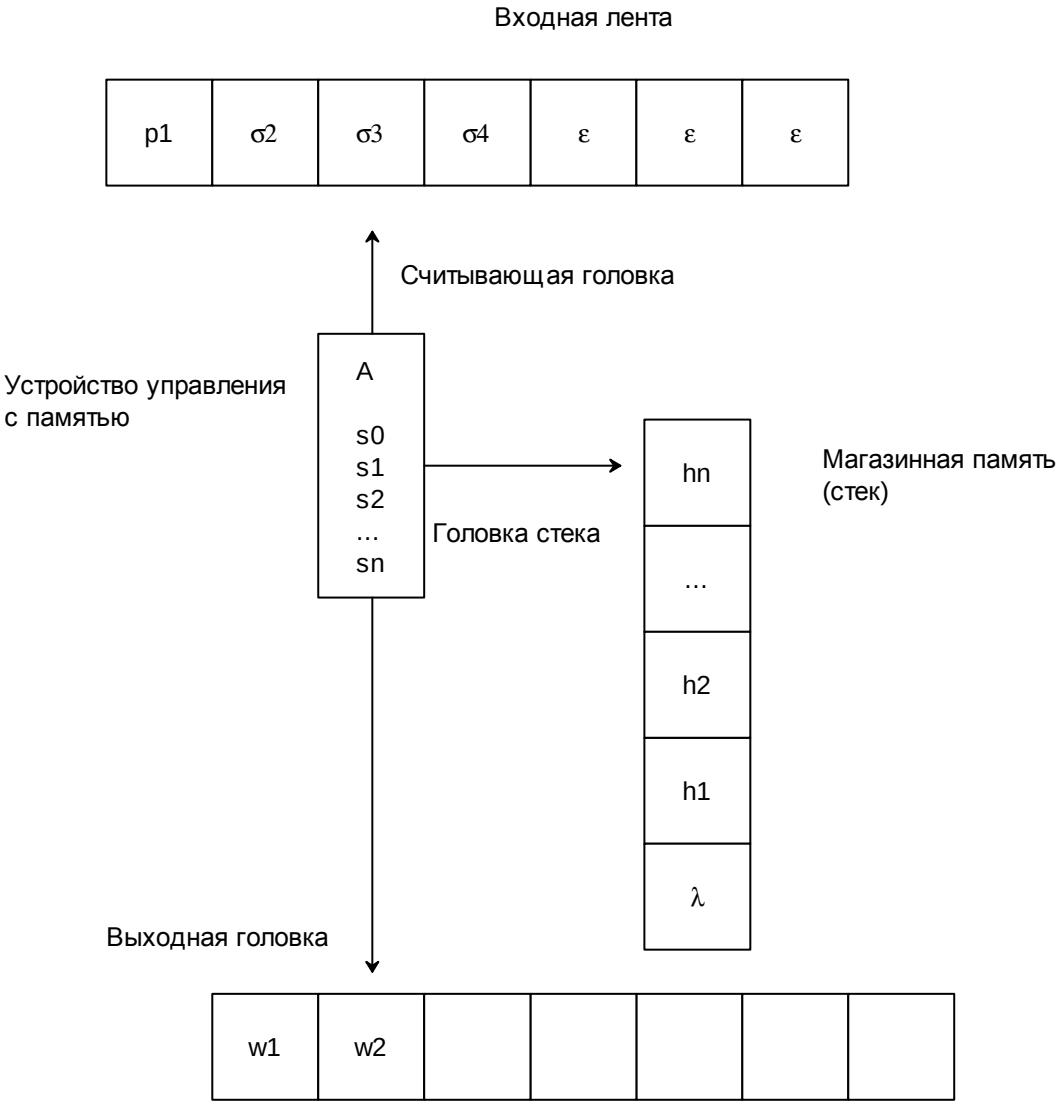
Построенный преобразователь по строке на входной ленте строит соответствующую ей выходную строку на выходной ленте, если входная строка является допустимой. Если же входная строка не является допустимой, преобразователь

останавливается, не достигнув конечного состояния.

§ 3. Магазинные преобразователи

Магазинные автоматы, рассмотренные в предыдущем разделе, позволяют определить для строки, заданной на входе, ее принадлежность к языку, допускаемому автоматом. Настоящий раздел посвящен другому типу моделей, называемому магазинными преобразователями**.** Подобные устройства позволяют строить по заданной входной строке соответствующую ей выходную строку. В отличие от автоматных преобразователей, способных обрабатывать лишь автоматные языки, МП-преобразователи способны работать с более широким классом языков.

Аналогично А-преобразователю, магазинный преобразователь отличается от соответствующего автомата наличием дополнительной выходной ленты, на которую записывается выходная строка. Схему магазинного преобразователя можно изобразить следующим образом:



Здесь входная головка в каждом такте работы может оставаться на месте или

двигаться на одну позицию вправо. Выходная головка также может быть неподвижной или смещаться вправо, а головка магазинной ленты может перемещаться в обоих направлениях, причем такие перемещения связаны с операциями записи и чтения.

Преобразователем с магазинной памятью (МП-преобразователем) называется совокупность:

$Mn=\{P, S, s_0, H, \lambda, F, W, q\}$,

где P - входной алфавит, состоящий из символов, записываемых на входную ленту;

W - выходной алфавит, содержащий символы, записываемые на выходную ленту;

H - магазинный алфавит, содержащий символы, записываемые и считываемые из магазина;

λ - маркер дна магазина, $\lambda \in H$;

S - множество состояний преобразователя;

s_0- начальное состояние, из множества $s_0 \in S$;

F - множество конечных состояний, представляющих собой подмножество $F \subset S$;

φ - функция переходов преобразователя, которая задает отображение,

$\varphi: S \times \{P \cup \{\varepsilon\}\} \times H \Rightarrow S \times H^* \times W^*$.

Она может быть записана в функциональном виде:

$\varphi(s, p, h) = (s', \beta, \gamma)$,

где $h \in H$, $p \in P$, $\beta \in H^*$, $\gamma \in W^*$ *и* $s,s' \in S$.

Конфигурацией МП-автомата назовем совокупность

$(s, \alpha, \eta, \omega)$,

где $s \in S$, $\alpha \in P^*$, $\eta \in H^*$ *и* $\omega \in W^*$.

В качестве начальной примем конфигурацию, которой соответствует заданная входная строка α на ленте, строка $\lambda<I>$ в магазине, начальное состояние s_0 и пустая строка на выходной ленте:

$(s_0, \alpha, \lambda<I>, \varepsilon)$.

Конечной или заключительной конфигурацией назовем четверку $(s_k, \varepsilon, \varepsilon, \omega)$, где s_k - одно из заключительных состояний из множества F, а ω - выходная строка.

Алгоритм построения МП-преобразователя Примем во внимание, что преобразование входной строки возможно только в том случае, если возможно ее распознавание.

Для каждой простой СУ-схемы перевода $T = \{Va, Vтвх, Vтвых, Q, I\}$ можно построить такой МП-преобразователь, что $D(T) = D(Mn)$.

Для каждой простой СУ - схемы перевода T, входная грамматика которой принадлежит классу LL(1) – грамматик, можно построить такой детерминированный магазинный преобразователь Мп, что перевод, определяемый преобразователем, совпадает с переводом, задаваемым СУ-схемой T.

Будем строить преобразователь по правилам транслирующей грамматики, используя правила построения распознавателя.

Рассмотрим построение для грамматики класса LL(1).
В отличие от распознавателя, в котором в магазин будут записываться выходные символы, содержащиеся в правилах грамматики, в Мп-преобразователе могут возникать неопределенные ситуации при появлении выходного символа в вершине стека. Чтобы этого избежать, необходимо выполнение следующего правила: при появлении выходного символа в вершине магазина он должен передаваться на выход независимо от того, какой символ находится под входной головкой.
Порядок построения функции переходов МП-преобразователя:
1. Для всех правил вида $<A> \to b\alpha$, где $b \in V_{твх}$ и $\alpha \in (V_{твх} \cup V_{твых} \cup V_a)^*$, строим команды:
$\varphi(s_0, b, <A>)=(s_0, \alpha', \varepsilon)$,
где α' - зеркальное отображение цепочки α.
2. Для всех правил вида $<A> \to <B>\alpha$, где $B \in Va$ и $\alpha \in (V_{твх} \cup V_{твых} \cup V_a)^*$, строим команды:
$\varphi^*(s_0, u, <A>) = (s_0, \alpha'<B> , \varepsilon)$,
где $u \in$ *ВЫБОР*$(<A> \to <B>\alpha_{вх})$ и $\alpha_{вх}$ - строка, полученная из α путем удаления из нее всех выходных символов, φ^* - не смещает читающую головку, α' – зеркальное отражение строки α.
3. Для всех правил вида $<A> \to \varepsilon$ строим команды:
$\varphi^*(s_0, u, <A>) = (s_0, \varepsilon, \varepsilon)$,
где $u \in$ *ВЫБОР*$(<A> \to \varepsilon)$, φ^* не смещает читающую головку.
4. Для всех символов b, принадлежащих, $V_{твх}$, т.е. символов, заносимых в магазин, строим правила:
$\varphi(s_0, b, b) = (s_0, \varepsilon, \varepsilon)$.
5. Для всех выходных символов $\underline{u}$, таких что $\underline{u} \in V_{твых} \cup \{\varepsilon\}$, строим команды:
$\varphi^*(s_0, z, \underline{u}) = (s_0, \varepsilon, \underline{u})$, где φ^* не смещает читающую головку, $z \in V_{твх}$ (перебираем все возможные сочетания входных и выходных символов)
6. Заключительное правило строим так:
$\varphi(s_0, \varepsilon, \lambda) = (s_k, \varepsilon, \varepsilon)$.

Пример.
Рассмотрим транслирующую грамматику, преобразующую инфиксные скобочные выражения в постфиксную польскую запись.
$Q = \{<A> \to x\underline{x}$,
$<A> \to (<B>)$,
$<B> \to <A><C>$,
$<C> \to +<A>\underline{+}<C>$,
$<C> \to \varepsilon\underline{\varepsilon}\}$.

Построим множества ВЫБОР для каждого из правил для проверки принадлежности данной грамматики к классу LL(1)-грамматик.
ВЫБОР$(<A> \to x) =$ *ПЕРВ*$(x) = \{x\}$

ВЫБОР(<A> → (<B>)) = ПЕРВ((<B>)) = {(}
ВЫБОР(<B> → <A><C>) = ПЕРВ(<A><C>) = {x, (}
ВЫБОР(<C> → +<A><C>) = ПЕРВ(+<A><C>) = {+}
ВЫБОР(<C> → ε) = СЛЕД(<C>) = СЛЕД(<B>) = {) }.

Построим правила вывода в соответствии с пп. 1-6:
1.1. $\varphi(s_0, x, <A>) = (s_0, \underline{x}, \varepsilon)$
1.2. $\varphi(s_0, (, <A>) = (s_0,)<B>, \varepsilon)$
1.3. $\varphi(s_0, +, <C>) = (s_0, <C>\underline{+}<A>, \varepsilon)$
2.1. $\varphi^*(s_0, x, <B>) = (s_0, <C><A>, \varepsilon)$
2.2. $\varphi^*(s_0, (, <B>) = (s_0, <C><A>, \varepsilon)$
3.1. $\varphi^*(s_0,), <C>) = (s_0, \varepsilon, \varepsilon)$
4.1. $\varphi(s_0,x, x) = (s_0, \varepsilon, \varepsilon)$
4.2. $\varphi(s_0,(, ( ) = (s_0, \varepsilon, \varepsilon)$
4.3. $\varphi(s_0,+, +) = (s_0, \varepsilon, \varepsilon)$
4.4. $\varphi(s_0 ,),)) = (s_0, \varepsilon, \varepsilon)$
5.1. $\varphi^*(s_0, x, \underline{x}) = (s_0, \varepsilon, \underline{x})$
5.2. $\varphi^*(s_0, (, \underline{x}) = (s_0, \varepsilon, \underline{x})$
5.3. $\varphi^*(s_0,), \underline{x}) = (s_0, \varepsilon, \underline{x})$
5.4. $\varphi^*(s_0, +, \underline{x}) = (s_0, \varepsilon, \underline{x})$
5.5. $\varphi^*(s_0, x, \underline{+}) = (s_0, \varepsilon, \underline{+})$
5.6. $\varphi^*(s_0, (, \underline{+}) = (s_0, \varepsilon, \underline{+})$
5.7. $\varphi^*(s_0,), \underline{+}) = (s_0, \varepsilon, \underline{+})$
5.8. $\varphi^*(s_0, +, \underline{+}) = (s_0, \varepsilon, \underline{+})$
5.9. $\varphi^*(s_0, x, \varepsilon) = (s_0, \varepsilon, \varepsilon)$
5.10. $\varphi^*(s_0, (,\varepsilon) = (s_0, \varepsilon,\varepsilon)$
5.11. $\varphi^*(s_0,), \varepsilon) = (s_0, \varepsilon, \varepsilon)$
5.12. $\varphi^*(s_0, +, \varepsilon) = (s_0, \varepsilon, \varepsilon)$
6.1. $\varphi(s_0, \varepsilon, \lambda) = (s_k, \varepsilon, \varepsilon)$

Порядок преобразования строки
(s₀, (x+x+x), λ<A>, ε) ⊢
(s₀, x+x+x), λ)<B>, ε) ⊢
(s₀, x+x+x), λ)<C><A>, ε) ⊢
(s₀, +x+x), λ)<C>x, ε) ⊢
(s₀, +x+x), λ)<C>, x) ⊢
(s₀, x+x), λ)<C>+<A>, x) ⊢
(s₀, +x), λ)<C>+x, x) ⊢
(s₀, +x), λ)<C>+, xx) ⊢
(s₀, +x), λ)<C>, xx+) ⊢
(s₀, x), λ)<C>+<A>, xx+) ⊢
(s₀,), λ)<C>+x, xx+) ⊢

$(s_0,), \lambda)<C>\underline{+}, \underline{xx+x}) \vdash$
$(s_0,), \lambda)<C>, \underline{xx+x+}) \vdash$
$(s_0,), \lambda), \underline{xx+x+}) \vdash$
$(s_0, \varepsilon, \lambda, \underline{xx+x+}) \vdash$
$(s_0, \varepsilon, \varepsilon, \underline{xx+x+})$

Алгоритм построения детерминированного магазинного преобразователя В общем случае, если заданы входной и выходной языки, то порядок построения детерминированного магазинного преобразователя можно представить следующим образом:

1. Построить грамматику, описывающую цепочки входного языка.
2. Проверить принадлежность этой грамматики классу LL(1)- грамматик. Если условия LL(1) - грамматики не выполняются, то попытаться выполнить преобразование или вернуться к п.1 и построить другую грамматику.
3. Построить простую СУ-схему, используя построенную грамматику в качестве входной грамматики СУ - схемы.
4. Построить транслирующую грамматику для полученной СУ-схемы.
5. Используя правила построения, найти команды преобразования для разных групп правил транслирующей грамматики.
6. Убедиться, что построенный преобразователь реализует заданный перевод, выполняя несколько примеров построения выходных цепочек с помощью команд преобразователя.

Задачи и упражнения

1. Постройте СУ-схему для перевода

0000	*00*
0001	*01*
0010	*10*
0011	*1100*
0100	*1101*
0101	*1110*
0110	*111100*

2. Является ли построенная для п. 1 СУ-схема простой?
3. Постройте транслирующую грамматику для перевода из п. 1.
4. В чем заключается отличие переходной функции распознавателя и автоматного преобразователя?
5. Какой из типов автоматных преобразователей наиболее удобен для реализации перевода из п.1?
6. Постройте автоматный преобразователь для транслирующей грамматики из п. 3.

ЗАКЛЮЧЕНИЕ

В данном учебном пособии изложен базовый материал теории множеств, математической логики, теории графов, комбинаторики и теории формальных грамматик и автоматов. Знание основ можно использовать для дальнейшего развития знаний и навыков, например, по следующим направлениям:

— логическая теория графов;
— алгебраическая теория графов;
— рекуррентные свойства графов;
— вероятностные графы;
— бесконечные графы
— теория трансляции

и др.

Авторы понимают, что в данном учебном пособии раскрыта лишь небольшую часть материала по данной тематике. Отчасти это вызвано небольшим объемом изучаемого курса.

Своей целью авторы видели дать студенту основы для дальнейшей самостоятельной работы, а так же продемонстрировать, по мере возможности, на примерах области применения дискретной математике в информационных технологиях вообще и в области информационной безопасности в частности.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Лихтарников, Л. М. Математическая логика: Курс лекций / Л. М. Лихтарников, Т. Г. Сукачева. СПб.: Лань, 1999.
2. Ерусалимский, Я. М. Дискретная математика: теория, задачи, приложения / Я. М. Ерусалимский. М.: Вуз. кн., 2000.
3. Новиков, Ф. А. Дискретная математика для программистов / Ф. А. Новиков. СПб: Питер, 2000.
4. Кук, Д. Компьютерная математика / Д. Кук, Г. Бейз. М.: Наука, 1990.
5. Зыков, А. А. Основы теории графов / А. А. Зыков. М.: Наука, 1987.
6. Горбатов, В. А Основы дискретной математики: Учеб. пособие для студентов вузов / В. А Горбатов. М.: Высш. шк., 1986.
7. Фомичев В. С. Формальные языки, грамматики и автоматы / В. С. Фомичев, СПбГЭУ "ЛЭТИ", 2006.
8. Эвнин А.Ю. Дискретная математика: конспект лекций / А. Ю. Эвнин, Челябинск: ЮУрГУ, 1998.
9. Кристофидес Н. Теория графов. Алгоритмический подход / Н. Кристофидес, М.: Мир, 1978.

Учебное издание

САЯПИН Александр Владимирович
СЛИВИНА Татьяна Анатольевна

ДИСКРЕТНАЯ МАТЕМАТИКА

Учебное пособие

Редактор
Компьютерная верстка

Подписано в печать. Уч.-изд. л. 21,5. С 159.

Редакционно-издательский отдел СибГАУ.
660014, Красноярск, просп. им. газ. «Красноярский рабочий», 31.