МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Государственное образовательное учреждение
высшего профессионального образования
СЕВЕРО-ЗАПАДНЫЙ ГОСУДАРСТВЕННЫЙ ЗАОЧНЫЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Г.И.Анкудинов
И.Г.Анкудинов
О.А.Петухов

МАТЕМАТИЧЕСКАЯ ЛОГИКА И ТЕОРИЯ АЛГОРИТМОВ

Утверждено редакционно-издательским советом университета
в качестве учебного пособия

ИЗДАНИЕ ВТОРОЕ

Санкт-Петербург
2003

УДК 512

Анкудинов Г.И., Анкудинов И.Г., Петухов О.А. Математическая логика и теория алгоритмов: Учеб. пособие.– 2-е изд. – СПб.: СЗТУ, 2003, 104 с.

Учебное пособие соответствует государственному образовательному стандарту дисциплины “Математическая логика и теория алгоритмов” направления подготовки дипломированных специалистов 654600 – “*Информатика и вычислительная техника”* (Специальность 220100 – “*Вычислительные машины, комплексы, системы и сети”*) и направления подготовки бакалавров 552800 – “*Информатика и вычислительная техника”*.

В пособии излагаются разделы математической логики и теории алгоритмов, необходимые для освоения общепрофессиональных и специальных дисциплин специальности 220100. Достаточно подробно изложены основы логики высказываний и логики предикатов, включая приложение логики предикатов к доказательству правильности алгоритмов. Пособие содержит вводный материал по логическому программированию и клаузальной логике, а также основные понятия нечеткой и модальной логики. Приведены основы теории алгоритмов и алгоритмической разрешимости, доказательство эквивалентности моделей алгоритмов Тьюринга и рекурсивных схем Клини. Пособие содержит также введение в теорию эффективной вычислимости, переборных NP-полных и NP-трудных задач.

Во втором издании исправлены опечатки и неточности.

Рецензенты:

кафедра процессов управления и информационных систем Северо-Западного государственного заочного технического университета (зав.кафедрой **О.И.Золотов**, канд.техн.наук, доц., **А.Б.Шадрин**, д-р техн.наук, проф.);

В.В.Лохмотко, д-р техн.наук, проф., **М.О.Колбанев**, канд.техн.наук, доц. (кафедра информационных управляющих систем Государственного университета телекоммуникаций им. проф. М.А.Бонч-Бруевича).

Глава 1
ЛОГИКА ВЫСКАЗЫВАНИЙ

В основе стандартной (классической) логики лежит *логика высказываний* (*пропозициональная логика*) и *логика предикатов*. *Высказывание* это повествовательное предложение, в отношении которого имеет смысл утверждение об его *истинности* или *ложности*. Пример истинного высказывания: "Земля вращается вокруг Солнца". *Предикат* это повествовательное предложение, содержащее *предметные* (*индивидные* переменные), замена которых на константные значения превращает рассматриваемое предложение в высказывание – истинное или ложное.

1.1. Логические операции над высказываниями

Высказывание – это повествовательное предложение, утверждающее что-то о чем-либо, причем высказывание может быть *истинным* либо *ложным*. Высказывания могут быть *простыми* и *составными*. Составные высказывания образуются из простых с помощью связок НЕ, И, ИЛИ, ЕСЛИ-ТО, ТОГДА-И-ТОЛЬКО-ТОГДА.

В алгебре высказываний все высказывания рассматриваются с точки зрения их *логического значения* или *истинности*. Считается, что каждое высказывание либо *истинно* (И), либо *ложно* (Л). Высказывание не может быть одновременно истинным и ложным. Будем обозначать высказывания простыми латинскими буквами *A,B,C*,.... Составные высказывания образуются из простых с помощью логических операций над высказываниями. Перечислим основные логические операции:

- *отрицание*,
- *конъюнкция,*
- *дизъюнкция,*
- *импликация*,
- эквивалентность.

Отрицание высказывания A образуется с помощью операции отрицания и в данном тексте будет обозначаться $\bar{A}$ или $\neg A$ (читается: "неверно, что A" или, короче "не A"). Логическую функцию, соответствующую зависимости логического значения $\bar{A}$ от логического значения высказывания A, можно представить с помощью таблицы истинности (табл.1.1): если A истинно, то $\bar{A}$ ложно и наоборот.

Таблица 1.1

A	$\bar{A}$
И	Л
Л	И

Конъюнкцией двух высказываний A, B называется новое высказывание, которое обозначается $A\&B$ (читается "A *и* B"). Конъюнкция $A\&B$ истинна тогда и только тогда, когда A и B одновременно истинны, а в остальных случаях ложна (табл.1.2). Конъюнкцию также иногда именуют *логическим произведением*.

Таблица 1.2

A B	$A\&B$
Л Л	Л
Л И	Л
И Л	Л
И И	И

Примечание. Кроме символа & в литературе используются и другие обозначения конъюнкции: $A \wedge B$, $A*B$, AB.

Дизъюнкцией высказываний A и B называется новое высказывание, которое обозначается $A \vee B$ (читается: "A или B"). Дизъюнкция $A \vee B$ ложна тогда и только тогда, когда A и B одновременно ложны, а в остальных случаях истинна (табл.1.3). Дизъюнкцию также иногда именуют *логической суммой*.

Таблица 1.3

A B	$A \vee B$
Л Л	Л
Л И	И
И Л	И
И И	И

Следует обратить внимание на то, что если в таблице истинности для дизъюнкции заменим все Л на И и все И на Л (как для операндов, так и для результата операции), то получим таблицу истинности для конъюнкции).

Импликацией двух высказываний A и B называется новое высказывание, которое обозначается $A \rightarrow B$ (варианты чтения: "Если A, то B"; "То, что A, влечет то, что B"; "A только тогда, когда B"; "То, что A, есть достаточное условие того, что B"; "Чтобы A, необходимо, чтобы B"). Высказывание $A \rightarrow B$ ложно тогда и только тогда, когда A истинно, а B ложно (табл.1.4). В составе импликации $A \rightarrow B$

Таблица 1.4

A B	$A \rightarrow B$
Л Л	И
Л И	И
И Л	Л
И И	И

высказывание A называется *условием* или *посылкой*, а высказывание B – *заключением* или *следствием*.

Примечание. Импликация $A \to B$ может быть записана также как $B \leftarrow A$ (читается: "B при условии, что A", "То, что B, есть необходимое условие того, чтобы A", "Чтобы B, достаточно того, чтобы A".)

Эквивалентностью двух высказываний A и B называется новое высказывание, которое обозначается $A \leftrightarrow B$ (читается: "A эквивалентно B", "A, тогда и только тогда, когда B"; "A, если и только если B", "Чтобы A, необходимо и достаточно, чтобы B"; "То, что A, есть необходимое и достаточное условие для того, чтобы B"). Высказывание $A \leftrightarrow B$ истинно тогда и только тогда, когда значения истинности A и B совпадают (табл.1.5).

Таблица 1.5

A B	$A \leftrightarrow B$
Л Л	И
Л И	Л
И Л	Л
И И	И

Часто в литературе, особенно в технических приложениях, для логического значения "истина" вместо И используется обозначение **1**, а для логического значения "ложь" вместо Л – обозначение **0**.

1.2. Составные высказывания

С помощью логических операций, рассмотренных в п.1.1, можно из простых высказываний строить различные *составные высказывания*. Например, из высказываний A, B и C можно построить составные высказывания

$$\neg(A \,\&\, B) \vee C \qquad \text{и} \qquad A \to [\, B \leftrightarrow (A \vee C)].$$

Логическое значение составного высказывания зависит только от логических значений образующих его элементарных высказываний. Например, если $A = \mathbf{0}$, $B = \mathbf{1}$ и $C = \mathbf{0}$, то

$$\neg(A \,\&\, B) \vee C = \neg(\mathbf{0} \,\&\, \mathbf{1}) \vee \mathbf{0} = \overline{\mathbf{0}} \vee \mathbf{0} = \mathbf{1} \vee \mathbf{0} = \mathbf{1} \text{ и}$$

$$A \to [B \leftrightarrow (A \vee C)] = \mathbf{0} \to [\mathbf{1} \leftrightarrow (\mathbf{0} \vee \mathbf{0})] =$$

$$\mathbf{0} \to [\mathbf{1} \leftrightarrow \mathbf{0}] = \mathbf{0} \to \mathbf{0} = \mathbf{1}.$$

Формулой исчисления высказываний называются

а) отдельные буквы, обозначающие переменные высказывания (P_1, P_2,...,P_N);

б) выражения вида $\neg(\Phi)$, $(\Phi_1)\&(\Phi_2)$, $(\Phi_1)\vee(\Phi_2)$, $(\Phi_1)\rightarrow(\Phi_2)$, $(\Phi_1)\leftrightarrow(\Phi_2)$, где Φ, Φ_1, Φ_2 - некоторые формулы.

Формулу, состоящую из переменных P_1, P_2, ..., P_N, логических символов и скобок, будем обозначать $\Phi(P_1, P_2, ..., P_N)$. Если в формулу Φ вместо переменных P_1, P_2, ..., P_N подставить высказывания A_1, A_2, ..., A_N, то получим составное высказывание $\Phi(A_1, A_2, ..., A_N)$, имеющее конкретное логическое значение. Зависимость логического значения $\Phi(A_1, A_2, ..., A_N)$ от P_1, P_2, ..., P_N можно выразить таблицей истинности. Например, таблица 1.6 выражает такую зависимость для формулы $\neg(P_1 \& P_2) \vee P_3$. Формула исчисления высказываний $\Phi(P_1, P_2,..., P_N)$ называется *тавтологией* или *тождественно истинной*, если ее значение для любых значений P_1, P_2,..., P_N есть истина.

Таблица 1.6

P_1	P_2	P_3	$\neg(P_1 \& P_2) \vee P_3$
0	**0**	**0**	**1**
0	**0**	**1**	**1**
0	**1**	**0**	**1**
0	**1**	**1**	**1**
1	**0**	**0**	**1**
1	**0**	**1**	**1**
1	**1**	**0**	**0**
1	**1**	**1**	**1**

1.3. Основные тавтологии

Закон исключенного третьего:

$$\bar{P} \vee P . \qquad (1.1)$$

Закон отрицания противоречия:

$$\neg(\bar{P} \& P). \qquad (1.2)$$

Закон двойного отрицания:

$$\neg\neg P \leftrightarrow P. \qquad (1.3)$$

Следующие законы выражают свойства конъюнкции и дизъюнкции.

Законы идемпотентности:

$$(P \,\&\, P) \leftrightarrow P, \tag{1.4}$$

$$(P \vee P) \leftrightarrow P. \tag{1.5}$$

Законы упрощения:

$$(P_1 \,\&\, P_2) \rightarrow P_1, \tag{1.6}$$

$$P_1 \rightarrow (P_1 \vee P_2). \tag{1.7}$$

Законы коммутативности:

$$(P_1 \,\&\, P_2) \leftrightarrow (P_2 \,\&\, P_1), \tag{1.8}$$

$$(P_1 \vee P_2) \leftrightarrow (P_2 \vee P_1). \tag{1.9}$$

Законы ассоциативности:

$$[(P_1 \,\&\, P_2) \,\&\, P_3] \leftrightarrow [P_1 \,\&\, (P_2 \,\&\, P_3)], \tag{1.10}$$

$$[(P_1 \vee P_2) \vee P_3] \leftrightarrow [P_1 \vee (P_2 \vee P_3)]. \tag{1.11}$$

Законы дистрибутивности:

$$[(P_1 \vee P_2) \,\&\, P_3] \leftrightarrow [(P_1 \,\&\, P_3) \vee (P_2 \,\&\, P_3)], \tag{1.12}$$

$$[(P_1 \,\&\, P_2) \vee P_3] \leftrightarrow [(P_1 \vee P_3) \,\&\, (P_2 \vee P_3)]. \tag{1.13}$$

Закон де-Моргана:

$$\neg(P_1 \,\&\, P_2) \leftrightarrow (\bar{P}_1 \vee \bar{P}_2), \tag{1.14}$$

$$\neg(P_1 \vee P_2) \leftrightarrow (\bar{P}_1 \,\&\, \bar{P}_2). \tag{1.15}$$

Следующие законы выражают свойства импликации и эквивалентности.

Закон тождества:

$$P \rightarrow P. \tag{1.16}$$

Закон контрапозиции:

$$(P_1 \rightarrow P_2) \leftrightarrow (\bar{P}_2 \rightarrow \bar{P}_1)). \tag{1.17}$$

Правило цепного заключения:

$$[(P_1 \rightarrow P_2) \,\&\, (P_2 \rightarrow P_3)] \rightarrow (P_1 \rightarrow P_3). \tag{1.18}$$

Законы рефлексивности, симметричности и транзитивности:

$$P \leftrightarrow P, \tag{1.19}$$

$$(P_1 \leftrightarrow P_2) \leftrightarrow (P_2 \leftrightarrow P_1), \tag{1.20}$$

$$[(P_1 \leftrightarrow P_2)\ \&\ (P_2 \leftrightarrow P_3)] \rightarrow (P_1 \leftrightarrow P_3). \tag{1.21}$$

Закон противоположности:

$$(P_1 \leftrightarrow P_2) \leftrightarrow (\bar{P}_1 \leftrightarrow \bar{P}_2). \tag{1.22}$$

Следующие законы выражают зависимости между основными логическими операциями.

Выражение конъюнкции через дизъюнкцию и отрицание и дизъюнкции через конъюнкцию и отрицание:

$$(P_1\ \&\ P_2) \leftrightarrow \neg(\bar{P}_1 \vee \bar{P}_2), \tag{1.23}$$

$$(P_1 \vee P_2) \leftrightarrow \neg(\bar{P}_1\ \&\ \bar{P}_2). \tag{1.24}$$

Выражение эквивалентности через конъюнкцию и импликацию:

$$(P_1 \leftrightarrow P_2) \leftrightarrow [(P_1 \rightarrow P_2)\ \&\ (P_2 \rightarrow P_1)]. \tag{1.25}$$

Выражение импликации через конъюнкцию и отрицание и через дизъюнкцию и отрицание:

$$(P_1 \rightarrow P_2) \leftrightarrow \neg(P_1\ \&\ \bar{P}_2), \tag{1.26}$$

$$(P_1 \rightarrow P_2) \leftrightarrow (\bar{P}_1 \vee P_2). \tag{1.27}$$

Выражение конъюнкции и дизъюнкции через отрицание и импликацию:

$$(P_1\ \&\ P_2) \leftrightarrow \neg(P_1 \rightarrow \bar{P}_2), \tag{1.28}$$

$$(P_1 \vee P_2) \leftrightarrow (\bar{P}_1 \rightarrow P_2). \tag{1.29}$$

1.4. Равносильные формулы

Две формулы исчисления высказываний $\Phi_1(P_1,...,P_N)$ и $\Phi_2(P_1,...P_N)$ называются *равносильными*, если они принимают одинаковое значение для любых значений $P_1,...P_N$. Две равносильные формулы имеют одну и ту же таблицу истинности и, наоборот, формулы, имеющие одну и ту же таблицу истинности, равносильны.

Условие равносильности формул выражает

Теорема 1.1. *Формулы* $\Phi_1(P_1,...P_N)$ и $\Phi_2(P_1,...P_N)$ *равносильны тогда и только тогда, когда их эквивалентность*

$$\Phi_1(P_1,...P_N) \leftrightarrow \Phi_2(P_1,...P_N)$$

является тавтологией.

Отношение равносильности обозначается символом $\Leftrightarrow$. Например, из законов (1.12) и (1.13) следуют равносильности:

$$(P_1 \vee P_2) \,\&\, P_3 \Leftrightarrow (P_1 \,\&\, P_3) \vee (P_2 \,\&\, P_3), \tag{1.30}$$

$$(P_1 \,\&\, P_2) \vee P_3 \Leftrightarrow (P_1 \vee P_3) \,\&\, (P_2 \vee P_3). \tag{1.31}$$

Из законов (1.14) и (1.15) следуют равносильности:

$$\neg(P_1 \,\&\, P_2) \Leftrightarrow \bar{P}_1 \vee \bar{P}_2, \tag{1.32}$$

$$\neg(P_1 \vee P_2) \Leftrightarrow \bar{P}_1 \,\&\, \bar{P}_2. \tag{1.33}$$

Из тавтологий (1.23,...,1.29) следуют равносильности:

$$P_1 \,\&\, P_2 \Leftrightarrow \neg(\bar{P}_1 \vee \bar{P}_2), \tag{1.34}$$

$$P_1 \vee P_2 \Leftrightarrow \neg(\bar{P}_1 \,\&\, \bar{P}_2), \tag{1.35}$$

$$P_1 \leftrightarrow P_2 \Leftrightarrow (P_1 \rightarrow P_2) \,\&\, (P_2 \rightarrow P_1), \tag{1.36}$$

$$P_1 \rightarrow P_2 \Leftrightarrow \neg(P_1 \,\&\, \bar{P}_2), \tag{1.37}$$

$$P_1 \rightarrow P_2 \Leftrightarrow \bar{P}_1 \vee P_2, \tag{1.38}$$

$$P_1 \,\&\, P_2 \Leftrightarrow \neg(P_1 \rightarrow \bar{P}_2), \tag{1.39}$$

$$P_1 \vee P_2 \Leftrightarrow \bar{P}_1 \rightarrow P_2. \tag{1.40}$$

Полезно также использовать следующие равносильности с логическими константами:

$$P \vee \mathbf{1} \Leftrightarrow \mathbf{1}, \tag{1.41}$$

$$P \vee \mathbf{0} \Leftrightarrow P, \tag{1.42}$$
$$P \,\&\, \mathbf{1} \Leftrightarrow P, \tag{1.43}$$
$$P \,\&\, \mathbf{0} \Leftrightarrow \mathbf{0}. \tag{1.44}$$

Для упрощения формул исчисления высказываний полезны следующие равносильности, называемые правилами склеивания:

$$(P_1 \,\&\, P_2) \vee (P_1 \,\&\, \bar{P}_2) \Leftrightarrow P_1, \tag{1.45}$$
$$(P_1 \vee P_2) \,\&\, (P_1 \vee \bar{P}_2) \Leftrightarrow P_1; \tag{1.46}$$

правила поглощения:

$$P_1 \vee (P_1 \,\&\, P_2) \Leftrightarrow P_1, \tag{1.47}$$
$$P_1 \,\&\, (P_1 \vee P_2) \Leftrightarrow P_1; \tag{1.48}$$

формулы Блейка - Порецкого:

$$(P_1 \& P_2) \vee (P_3 \& \bar{P}_2) \Leftrightarrow (P_1 \& P_2) \vee (P_3 \& \bar{P}_2) \vee (P_1 \& P_3), \tag{1.49}$$
$$(P_1 \vee P_2) \,\&\, (P_3 \vee \bar{P}_2) \Leftrightarrow (P_1 \vee P_2) \& (P_3 \vee \bar{P}_2) \& (P_1 \vee P_3) \tag{1.50}$$

и формулы равносильности:

$$P_1 \vee (\bar{P}_1 \,\&\, P_2) \Leftrightarrow P_1 \vee P_2, \tag{1.51}$$
$$P_1 \,\&\, (\bar{P}_1 \vee P_2) \Leftrightarrow P_1 \,\&\, P_2, \tag{1.52}$$
$$P \,\&\, \bar{P} \Leftrightarrow \mathbf{0}, \tag{1.53}$$
$$P \vee \bar{P} \Leftrightarrow \mathbf{1}, \tag{1.54}$$
$$P \,\&\, P \Leftrightarrow P, \tag{1.55}$$
$$P \vee P \Leftrightarrow P. \tag{1.56}$$
$$\neg \bar{P} \Leftrightarrow P. \tag{1.57}$$

Часто требуется упростить формулу исчисления высказываний, т.е. получить формулу равносильную исходной, но содержащую по возможности меньшее число пропозициональных букв и символов логических операций. Например, дана формула

$$(X_1 \vee X_2 \vee X_3) \& (X_1 \vee \bar{X}_2 \vee X_3) \& (X_1 \vee \bar{X}_3) \& (X_2 \vee X_3 \vee X_4) \&$$
$$(X_1 \vee \bar{X}_2 \vee \bar{X}_3) \& (X_1 \vee X_3 \vee \bar{X}_4) \& (X_1 \vee X_2).$$

Применим к первым двум подформулам $(X_1 \vee X_2 \vee X_3)\&(X_1 \vee \bar{X}_2 \vee X_3)$ равносильность (1.46), считая $P_1 = X_1 \vee X_3$ и $P_2 = X_2$, тогда их можно заменить одной подформулой $(X_1 \vee X_3)$, что дает более простую формулу, равносильную исходной:

$$(X_1 \vee X_3)\&(X_1 \vee \bar{X}_3)(X_2 \vee X_3 \vee X_4)\&(X_1 \vee \bar{X}_2 \vee \bar{X}_3)\&(X_1 \vee X_3 \vee \bar{X}_4)\&(X_1 \vee X_2).$$

К подформулам $(X_1 \vee X_3)$ и $(X_1 \vee \bar{X}_3)$ снова применим равносильность (1.46) и получаем еще более простую формулу, равносильную исходной:

$$X_1 \& (X_2 \vee X_3 \vee X_4)\&(X_1 \vee \bar{X}_2 \vee \bar{X}_3)\&(X_1 \vee X_3 \vee \bar{X}_4)\& (X_1 \vee X_2).$$

Коммутативность конъюнкции позволяет переписать последнее выражение, а равносильность (1.48) – выполнить дальнейшее упрощение

$$X_1 \&(X_1 \vee \bar{X}_2 \vee \bar{X}_3)\&(X_1 \vee X_3 \vee \bar{X}_4)\& (X_1 \vee X_2) \& (X_2 \vee X_3 \vee X_4) =$$
$$X_1 \& (X_2 \vee X_3 \vee X_4).$$

Если в формуле используются операции импликации и эквивалентности, то, как правило, их следует преобразовать с помощью равносильностей (1.36), (1.37) и (1.38).

Например:

$$((X_1 \to X_2) \vee (X_1 \to X_4)) \to (X_1 \to (X_2 \vee X_3)) =$$
$$(\bar{X}_1 \vee X_2 \vee \bar{X}_1 \vee X_4) \to (\bar{X}_1 \vee X_2 \vee X_3) =$$
$$(\bar{X}_1 \vee X_2 \vee X_4) \to (\bar{X}_1 \vee X_2 \vee X_3) =$$
$$\neg(\bar{X}_1 \vee X_2 \vee X_4) \vee \bar{X}_1 \vee X_2 \vee X_3 =$$
$$(X_1 \& \neg(X_2 \vee X_4)) \vee \bar{X}_1 \vee X_2 \vee X_3 =$$
$$(X_1 \& \bar{X}_2 \& \bar{X}_4) \vee \bar{X}_1 \vee X_2 \vee X_3 =$$
$$(\bar{X}_2 \& \bar{X}_4) \vee \bar{X}_1 \vee X_2 \vee X_3 =$$
$$\bar{X}_1 \vee X_2 \vee X_3 \vee \bar{X}_4.$$

1.5. Логическое следование

Кроме отношения равносильности, между формулами исчисления высказываний на практике приходится рассматривать *отношение логического следования*: формула $\Phi_2(P_1,...,P_N)$ *логически следует* из формулы $\Phi_1(P_1,...,P_N)$, или

$$\Phi_1(P_1,...,P_N) \Rightarrow \Phi_2(P_1,...,P_N),$$

если $\Phi_2(P_1,...,P_N)$ истинна на всех наборах значений $P_1,...,P_N$, на которых $\Phi_1(P_1,...,P_N)$ истинна.

Логическое следование $\Phi_1 \Rightarrow \Phi_2$ означает, что из истинности Φ_1 следует истинность Φ_2, но если Φ_1 ложна, то относительно Φ_2 ничего сказать нельзя. Функция Φ_1 в этом случае называется *импликантой* для Φ_2. Важное значение имеет

Теорема 1.2. $\Phi_1(P_1,...,P_N) \Rightarrow \Phi_2(P_1,...,P_N)$ *тогда и только тогда, когда импликация* $\Phi_1(P_1,...,P_N) \rightarrow \Phi_2(P_1,...,P_N)$ *является тавтологией*.

Пример 1.1. Необходимо выяснить, является ли одно составное высказывание логическим следствием другого. Возьмем высказывание: "Равные треугольники подобны". Это высказывание можно записать символически $\Phi_1 = A \rightarrow B$, где A = "Треугольники равны", B = "Треугольники подобны".

Рассмотрим высказывание: “Треугольники подобны только в случае их равенства", которое можно записать символически как $\Phi_2 = B \rightarrow A$, где A и B определены выше. Является ли Φ_2 логическим следствием Φ_1? Для получения ответа на этот вопрос рассмотрим импликацию $\Phi_3 = (A \rightarrow B) \rightarrow (B \rightarrow A)$ и, чтобы проверить, является ли она тавтологией, упростим эту формулу:

$$(A \rightarrow B) \rightarrow (B \rightarrow A) = (\bar{A} \vee B) \rightarrow (\bar{B} \vee A) = \neg(\bar{A} \vee B) \vee \bar{B} \vee A =$$

$$(\neg \bar{A} \,\&\, \bar{B}) \vee \bar{B} \vee A = \bar{B} \vee A = B \rightarrow A,$$

т.е. Φ_3 не является тавтологией, следовательно, нельзя сказать, что Φ_2 является логическим следствием Φ_1.

1.6. Логические функции

Логической функцией от n аргументов называют функции вида

$$f\colon \{\mathbf{0},\mathbf{1}\}^n \to \{\mathbf{0},\mathbf{1}\},$$

т.е. областью определения логической функции являются всевозможные *n*–местные векторы, компоненты которых принимают значения из множества $\{\mathbf{0},\mathbf{1}\}=\{И,Л\}$, а областью значений – то же множество $\{\mathbf{0},\mathbf{1}\}$. Логические функции называют также *булевыми функциями*, по имени англичанина Дж.Буля, разработавшего в XIX веке основы логики высказываний. Другое название логических функций – *переключательные функции*, поскольку они широко применяются для анализа и синтеза логических устройств из релейных (переключательных) элементов. Общее обозначение логической функции: $f(x_1,x_2,\ldots,x_n)$, где $x_1,x_2,\ldots,x_n$ – логические (булевы) переменные.

Дизъюнктивной нормальной формой (д.н.ф.) называется формула, представляющая логическую функцию $f(x_1,x_2,\ldots,x_n)$ в виде дизъюнкции некоторого числа элементарных конъюнкций и логических переменных с отрицанием или без отрицания, причем под *элементарной конъюнкцией* понимается логическое произведение любого числа неодинаковых логических переменных x_i (i=1,…,n) с отрицанием или без отрицания.

Пример д.н.ф.: $f(x_1,x_2,x_3)= \bar{x}_1\bar{x}_2 \vee x_1\bar{x}_2x_3 \vee x_1x_2\bar{x}_3$.

Каждую элементарную конъюнкцию можно представить в общем виде

$$C = x_1^{\sigma_1} x_2^{\sigma_2} \ldots x_n^{\sigma_n},\ \text{где}$$

$$x_i^{\sigma_i} = \begin{cases} x_i, \text{если } \sigma_i = 1; \\ \bar{x}_i, \text{если } \sigma_i = 0. \end{cases}$$

Элементарная конъюнкция $C = x_1^{\sigma_1} x_2^{\sigma_2} \ldots x_n^{\sigma_n}$ называется *конституентой единицы* функции $f(x_1, x_2,\ldots, x_n)$, если она содержит все *n* переменных функции и $C \Rightarrow f(x_1, x_2,\ldots, x_n)$, т.е. $f(\sigma_1,\sigma_2,\ldots,\sigma_n)=1$.

Конституента единицы $x_1^{\sigma_1} x_2^{\sigma_2} \ldots x_n^{\sigma_n}$ принимает значение 1 только на одном наборе $(x_1, x_2, \ldots, x_n)=(\sigma_1, \sigma_2, \ldots, \sigma_n)$.

Совершенной дизъюнктивной нормальной формой (с.д.н.ф.) называется формула, представляющая логическую функцию $f(x_1,x_2,\ldots,x_n)$ в виде дизъюнкции некоторого числа конституент единицы.

Конъюнктивной нормальной формой (к.н.ф.) называется формула, представляющая логическую функцию $f(x_1,x_2,\ldots,x_n)$ в виде конъюнкции некоторого числа элементарных дизъюнкций и логических переменных с отрицанием или без отрицания. Под *элементарной дизъюнкцией* понимается логическая сумма любого числа неодинаковых логических переменных x_i (i=1,…,n) с отрицанием или без отрицания.

Пример к.н.ф.: $f(x_1,x_2,x_3)=(x_1 \vee \bar{x}_2)(\bar{x}_1 \vee \bar{x}_2 \vee x_3)(x_1 \vee x_2 \vee x_3)$.

Совершенной конъюнктивной нормальной формой (с.к.н.ф.) называется формула, представляющая логическую функцию $f(x_1,x_2,\ldots,x_n)$ в виде конъюнкции некоторого числа *конституент нуля* $f(x_1,x_2,\ldots,x_n)$. Конституента нуля $f(x_1,x_2,\ldots,x_n)$ принимает значение **0** только на одном наборе $(x_1,x_2,\ldots,x_n)$.

1.7. Формальные теории и исчисление высказываний

Формальная теория это

а) Множество *правильно построенных формул* (ППФ), или выражений, определяющих язык теории.

б) Подмножество формул множества ППФ, называемых *аксиомами* теории.

в) *Правила вывода*, т.е. конечное множество отношений между формулами.

Доказательством называется конечная последовательность формул $\Phi_1,...,\Phi_n$, такая, что каждая Φ_i есть либо аксиома, либо получена из предыдущих формул по одному из правил вывода.

Теоремой называется такая формула теории Φ, что существует доказательство $\Phi_1,...,\Phi_n$, где $\Phi_n=\Phi$.

Аксиоматическая теория *полна*, если присоединение к ее аксиомам формулы, не являющейся теоремой, делает теорию *противоречивой*, т.е. могут быть доказаны как Φ, так и "не Φ".

Интерпретацией формальной теории в содержательную теорию называется соответствие теорем формальной теории истинным утверждениям содержательной теории.

Пример формальной теории. Исчисление высказываний имеет

а) множество ППФ, определенное выше;

б) множество аксиом:

1. $A\to(B\to A)$
2. $(A\to (B\to C))\to ((A\to B)\to (A\to C))$
3. $(\neg A\to \neg B)\to ((\neg A\to B)\to A)$;

в) правила вывода:

1. $\dfrac{\Phi(A)}{\Phi(B)}$ (правило подстановки).
2. $\dfrac{\Phi,\ \Phi\to B}{B}$ (правило Modus Ponens).

В записи правил вывода над чертой располагаются формулы, называемые посылкой правила, из которых непосредственно следуют формулы, стоящие под чертой и называемые заключением правила.

Правило подстановки позволяет заменять в ППФ все вхождения некоторой буквы на другую букву. Правило Modus Ponens (MP) позволяет выводить формулу B из формул Φ и $\Phi\to B$.

Пример 1.2. Требуется доказать, что из A следует $B \to A$, т.е. $A \vdash (B \to A)$.

Доказательство. Имеем A. Тогда в соответствии с аксиомой 1 по правилу MP получаем

$$\frac{A, A \to (B \to A)}{B \to A} \quad (\text{MP}) .$$

Интерпретацией исчисления высказываний является логика высказываний.

В качестве дополнительной литературы по алгебре высказываний и основам формальных теорий можно рекомендовать [7, 8, 11, 13, 14].

Глава 2
ЛОГИКА ПРЕДИКАТОВ

2.1. Основные понятия теории множеств

Поскольку логика предикатов базируется на понятиях *теории множеств*, приведем определения основных понятий этой теории. Немецкий математик Георг Кантор, разработавший начала теории множеств в 70-х годах XIX века, определял множество, как объединение отдельных объектов в единое целое. Группа математиков Н.Бурбаки дает такое определение: "Множество образуется из элементов, обладающих некоторыми свойствами и находящихся в некоторых отношениях между собой или с элементами других множеств". Вместо термина "множество" могут использоваться также наименования: "совокупность", "класс", "система". Множество, не имеющее ни одного элемента, называется *пустым* и обозначается символом в виде перечеркнутого кружка $\varnothing$.

Множество из n элементов обозначается $\{a_1,a_2,\ldots,a_n\}$, а множество из одного элемента a_1 обозначается $\{a_1\}$.

Конечное множество M можно задать перечислением всех его элементов, например, $M = \{a,b,c,d,e,f\}$, при этом порядок записи элементов не существенен, т.е. $\{a,b,c,d,e,f\}=\{b,a,f,c,d,e\}= \{f,c,a,d,e,b\}$ и т.д.

Любое (конечное или бесконечное) множество можно задать указанием общего свойства C всех его и только его элементов:

$M = \{x$: x обладает свойством $C\}$ или
$M = \{x \mid x$ обладает свойством $C\}$.

Символ x в этом случае называется *предметной переменной* (в дальнейшем мы узнаем, что утверждение "x обладает свойством C" называется предикатом). Пример задания множества всех действительных чисел, обладающих свойством C= 'x больше единицы':

$M = \{ x \mid x$ – действительное число, $x>1\}$.

Пусть $M = \{a,b,c,d,e,f\}$. Мы говорим, например, что элемент b *принадлежит* M или используем общепринятый символ принадлежности $\in$, например, $b \in M$.

Пусть $M=\{a,b,c,d,e,f\}$ и $L=\{b,d,e\}$. Мы говорим, что L является *подмножеством* M или, что каждый элемент L является элементом M, или используя общепринятый символ включения множеств: $L \subset M$.

Множество всех подмножеств множества M называется *булеаном* M и обозначается как 2 в степени M: 2^M. Пусть $M=\{a,b,c\}$, тогда $2^M = \{\varnothing, \{a\}, \{b\}, \{c\}, \{a,b\}, \{a,c\}, \{b,c\}, \{a,b,c\}\}$.

Число элементов, или *мощность* множества M обозначается $|M|$.

Пример 2.1. Пусть $M=\{a,b,c,d,e\}$, тогда $|M|=5$. Для булеана $\left|2^M\right| = 2^{|M|}$.

Операции над множествами. *Дополнением* подмножества L множества M называется подмножество $\overline{L}$, содержащее все элементы M, не принадлежащие L:

$$\overline{L} =\{x \mid x \in M,\ x \text{ не принадлежит } L\}.$$

Пересечением подмножеств M_1 и M_2 множества M называется совокупность L всех элементов M, принадлежащих одновременно M_1 и M_2:

$$L = M_1 \cap M_2 = \{ x \mid x \in M_1, x \in M_2 \}.$$

Объединением подмножеств M_1 и M_2 множества M называется совокупность L всех элементов M, принадлежащих первому или второму подмножеству:

$$L = M_1 \cup M_2 = \{ x \mid x \in M_1 \text{ или } x \in M_2 \}.$$

Разность множеств M_1 и M_2:

$$L = M_1 \setminus M_2 = \{ x \mid x \in M_1 \text{ и неверно, что } x \in M_2) \}.$$

Декартовым произведением множеств M и L называется множество всех упорядоченных пар элементов $\{(x,y) \mid x \in M, y \in L\}$. В качестве символа декартова произведения обычно используется символ ×, тогда

$$M \times L = \{(x,y) \mid x \in M, y \in L \}.$$

Пример 2.2. Пусть $M = \{a,b,c\}$ и $L = \{f,g,k\}$, тогда декартово произведение $M \times L = \{(a,f),(a,g),(a,k),(b,f),(b,g),(b,k),(c,f),(c,g),(c,k)\}$.

Бинарным отношением между элементами множеств M и L называется подмножество $M \times L$, например, $R = \{(a,f),(b,f),(b,k)\}$.

Декартово произведение n множеств:

$$M_1 \times M_2 \times ... \times M_n = \{(x_1,x_2,...,x_n) \mid x_1 \in M_1, x_2 \in M_2, ..., x_n \in M_n \}.$$

Можно ввести также n-ю *декартову степень* множества M :

$M^n = M \times M \times ... \times M$, где M в правой части повторяется n раз.

Пример 2.3. Пусть $M = \{ a,b,c \}$, тогда $M^2 = \{(a,a), (a,b), (a,c), (b,a), (b,b), (b,c), (c,a), (c,b), (c,c)\}$. Пусть $M = \{a,b\}$, тогда $M^3 = \{(a,a,a), (a,a,b), (a,b,a), (a,b,b), (b,a,a), (b,a,b), (b,b,a), (b,b,b)\}$.

2.2. Определение предиката

Предикатом называется выражение, имеющее грамматическую форму высказывания, но содержащее предметные переменные некоторых множеств. Например, предложение "8 - четное число" является высказыванием. Заменим "8" на "x", тогда предложение "x-четное число", где x принадлежит множеству натуральных чисел, будет предикатом.

Выражение $A(x_1,x_2,...,x_n)$, содержащее предметные переменные $x_1 \in M_1$, $x_2 \in M_2$,..., $x_n \in M_n$, называется *n-местным предикатом*, определенным на множествах M_1, M_2,..., M_n.

При замене предметных переменных константами из соответствующих множеств предикат $A(x_1,x_2,...,x_n)$ превращается в высказывание, которое может быть либо истинным, либо ложным.

Пример 2.4 . Пусть N = { 1,2,3,4,5,6 }, $A(x)$ = "x - четное число", тогда $A(1)$ = Л, $A(2)$ = И, $A(3)$ = Л и т.д., т.е. значения аргументов 2, 4, 6 *удовлетворяют* предикату $A(x)$, а значения 1, 3, 5 *не удовлетворяют*.

Множество истинности, или интенсионал, предиката $A(x_1,x_2,...,x_n)$ – это подмножество его области определения $M_1 \times M_2 \times ... \times M_n$, на котором этот предикат истинен:

$$\{(x_1, x_2,..., x_n) \mid A(x_1, x_2,..., x_n)=\text{И}\}.$$

В дальнейшем будем в этом случае писать $\{(x_1,x_2,...,x_n) \mid A(x_1,x_2,..., x_n)\}$, подразумевая, как это обычно и принято, что берутся значения $(x_1,x_2,..., x_n)$, на которых $A(x_1,x_2,..., x_n)$=И.

Пример 2.5. Для рассмотренного в примере 2.4 предиката множество истинности $\{ x \mid A(x)\}$ = { 2,4,6 }.

Пример 2.6. Пусть N = { 1,2,3,4 } и $B(x_1,x_2)$ = "$x_1 > x_2$", тогда множество истинности предиката $B(x1,x2)$: $\{(x_1,x_2) \mid x_1 > x_2\}$ = {(2,1), (3,1), (3,2), (4,1), (4,2), (4,3)}.

Два n-местных предиката, определенных на одних и тех же множествах M_1, M_2,..., M_n, называются *равносильными*, если значения их для любых аргументов совпадают, т.е. они имеют одно и то же множество истинности. Например, предикаты "$x > 2$" и "$x - 2 > 0$" равносильны.

Предикат $B(x_1,x_2,...,x_n)$ называется *следствием* предиката $A(x_1,x_2,...,x_n)$, если $B(x_1,x_2,...,x_n)$ удовлетворяется любыми аргументами, удовлетворяющими $A(x_1, x_2,...,x_n)$.

Пример 2.7. Предикат "n делится на 3" есть следствие предиката "n делится на 6", где n - целое число.

Два n-местных предиката, определенные на одних и тех же множествах, равносильны тогда и только тогда, когда каждый из них является следствием другого.

Предикат называется

а) *тождественно истинным*, если значение его для любых аргументов есть "истина";

б) *тождественно ложным*, если значение его для любых аргументов есть "ложь";

в) *выполнимым*, если существует, по крайней мере, одна n-система его аргументов, для которой значение предиката есть "истина".

Пример 2.8. Предикат "$x + y = y + x$" является тождественно истинным, предикат "$x + 1 = x$" – тождественно ложным, предикат "$x + y = 5$" – выполнимым.

Каждый тождественно истинный предикат является выполнимым, но обратное неверно. Каждый выполнимый предикат будет не тождественно ложным и обратно, каждый не тождественно ложный предикат будет выполнимым.

Каждому n-местному предикату $A(x_1, x_2,...,x_n)$, определенному на множествах $M_1,M_2,...,M_n$ соответствует n-отношение R, являющееся подмножеством декартова произведения $M_1 \times M_2 \times ... \times M_n$ и равное множеству истинности этого предиката

$$R = \{ (x_1, x_2,...,x_n) \mid A(x_1, x_2,...,x_n) \}.$$

Обратно, каждому n-отношению R между элементами множеств $M_1,M_2,...,M_n$ соответствует предикат, множество истинности которого есть R.

2.3. Операции над предикатами

Предикат $A(x_1,x_2,...,x_n)$ можно рассматривать как логическую функцию, определенную на $M_1 \times M_2 \times ... \times M_n$ и принимающую значения на множестве {И,Л}. Простейшими логическими операциями над предикатами, так же как и для высказываний являются: отрицание, конъюнкция, дизъюнкция, импликация и эквивалентность.

Рассмотрим предикаты $A(x_1,x_2,...,x_n)$ и $B(x_1,x_2,...,x_n)$, определенные на $M_1 \times M_2 \times ... \times M_n$.

Отрицанием предиката $A(x_1,x_2,...,x_n)$ называется новый n-местный предикат $\overline{A}(x_1,x_2,... x_n)$, множество истинности которого является дополнением множества истинности предиката $A(x_1,x_2,...,x_n)$, т.е.

$$\{(x_1,x_2,...,x_n) \mid \overline{A}(x_1,x_2,...,x_n)\} =$$

$$M_1 \times M_2 \times ... \times M_n \setminus \{(x_1,x_2,...,x_n) \mid A(x_1,x_2,...,x_n \}.$$

Конъюнкцией предикатов $A(x_1,x_2,...,x_n)$ и $B(x_1,x_2,...,x_n)$ называется новый n-местный предикат

$$C(x_1,x_2,...,x_n) = A(x_1,x_2,...,x_n) \,\&\, B(x_1,x_2,...,x_n),$$

множество истинности которого есть пересечение множеств истинности $A(x_1,x_2,..., x_n)$ и $B(x_1,x_2,..., x_n)$, т.е.

$$\{(x_1,x_2,...,x_n) \mid C(x_1,x_2,...,x_n)\} =$$
$$\{(x_1,x_2,...,x_n) \mid A(x_1,x_2,...,x_n)\} \cap \{(x_1,x_2,...,x_n) \mid B(x_1,x_2,...,x_n)\}.$$

Дизъюнкцией предикатов $A(x_1,x_2,...,x_n)$ и $B(x_1,x_2,...,x_n)$ называется n-местный предикат

$$D(x_1,x_2,...,x_n) = A(x_1,x_2,...,x_n) \vee B(x_1,x_2,...,x_n),$$

множество истинности которого есть объединение множеств истинности $A(x_1,x_2,...,x_n)$ и $B(x_1,x_2,...,x_n)$, т.е.

$$\{(x_1,x_2,...,x_n) \mid D(x_1,x_2,...,x_n)\}=$$
$$\{(x_1,x_2,..., x_n) \mid A(x_1,x_2,..., x_n)\} \cup \{(x_1,x_2,..., x_n) \mid B(x_1,x_2,..., x_n)\}.$$

Импликацией предикатов $A(x_1,x_2,...,x_n)$ и $B(x_1,x_2,...,x_n)$ называется предикат

$$I(x_1,x_2,...,x_n) = A(x_1,x_2,...,x_n) \to B(x_1,x_2,...,x_n),$$

который имеет значение ЛОЖЬ на тех и только на тех наборах аргументов $x_1,x_2,...,x_n$, на которых $A(x_1,x_2,...,x_n)$ имеет значение ИСТИНА, а $B(x_1,x_2,...,x_n)$ - значение ЛОЖЬ.

Пусть $M(A)$, $M(B)$ и $M(I)$ множества истинности $A(x_1,x_2,...,x_n)$, $B(x_1,x_2,...,x_n)$ и $I(x_1,x_2,...,x_n)$, тогда $M(I) = \overline{M(A)} \cup M(B)$, где $\overline{M(A)}$ - дополнение множества $M(A)$.

Эквивалентностью предикатов $A(x_1,x_2,...,x_n)$ и $B(x_1,x_2,...,x_n)$ называется предикат

$$E(x_1,x_2,...,x_n) = A(x_1,x_2,...,x_n) \leftrightarrow B(x_1,x_2,...,x_n),$$

который имеет значение *истина* на тех и только на тех наборах аргументов $x_1,x_2,...,x_n$, на которых значения истинности $A(x_1,x_2,...,x_n)$ и $B(x_1,x_2,...,x_n)$ совпадают. Множество истинности $M(E)$ для эквивалентности предикатов $A(x_1,x_2,...,x_n) \leftrightarrow B(x_1,x_2,...,x_n)$ есть

$$M(E) = [\overline{M(A)} \cap \overline{M(B)}\ ] \cup [M(A) \cap M(B)].$$

Пример 2.9. Даны универсальное множество $M = \{a,b,c,d,e,f\}$ и два подмножества $L = \{b,c,d\}$ и $K = \{d,e,f\}$, и два предиката $A(x)$ и $B(x)$, причем $\{x \mid A(x)=И\}=L$ и $\{x \mid B(x)=И\}=K$, т.е. L и K являются множествами истинности предикатов $A(x)$ и $B(x)$ соответственно. Требуется найти множество истинности эквивалентности $E(x)=A(x)\leftrightarrow \neg B(x)$.

Для решение этой задачи используем определение эквивалентности предикатов:

$$\{x \mid E(x)=И\} = (\ \bar{L} \cap K) \cup (L \cap\ \bar{K}) =$$
$$(\{a,e,f\} \cap \{d,e,f\}) \cup (\{b,c,d\} \cap \{a,b,c\}) =$$
$$\{e,f\} \cup \{b,c\} = \{e,f,b,c\}.$$

2.4. Логические операции квантификации

Пусть $A(x)$ – одноместный предикат, определенный на множестве M. *Универсальным высказыванием*, соответствующим

$A(x)$, называется высказывание "каждый элемент множества M удовлетворяет предикату $A(x)$", которое будем обозначать $\forall x\, A(x)$.

Высказывание $\forall x\, A(x)$ считается истинным, если предикат $A(x)$ тождественно истинный, и ложным - в противном случае.

Символ $\forall x$ называется *квантором всеобщности* по переменной x, его читают: "для всех x" или "для каждого x" или "для любого x". Выражение $\forall x\, A(x)$ читается: "для всех x, A (x)" или "для каждого x, $A(x)$".

Например, $\forall x(x=x)$ – это истинное универсальное высказывание, а $\forall x(x > 2)$ – ложное универсальное высказывание.

Если $A(x)$ – одноместный предикат, определенный на конечном множестве $\{a_1,a_2,...,a_m\}$, то $\forall x\, A(x) \leftrightarrow [A(a_1)\ \&\ A(a_2)\ \&\ ...\ \&\ A(a_m)]$. Таким образом, квантор всеобщности можно понимать как оператор конъюнкции по квантифицируемой переменной.

Экзистенциональным высказыванием, соответствующим предикату $A(x)$, называется высказывание "существует элемент множества M, удовлетворяющий предикату $A(x)$", которое обозначается $\exists x\, A(x)$, и считается истинным, если предикат $A(x)$ выполнимый, и ложным - в противном случае.

Символ $\exists$ называют *квантором существования*, а выражение $\exists x,$ в котором этот квантор предшествует переменной x, читают: "существует x такой, что ..." или "для некоторого x, ...". Например, выражение $\exists x\, A(x)$ читается: "существует x такой, что $A(x)$ " или "для некоторого x, $A(x)$".

Например, $\exists x(x>2)$ – истинное экзистенциональное высказывание, а $\exists x(x=x+1)$ – ложное экзистенциональное высказывание.

Если $A(x)$ - одноместный предикат, определенный на конечном множестве $\{a_1,a_2,...,a_m\}$, то $\exists x\, A(x) \leftrightarrow [A(a_1) \vee A(a_2) \vee ... \vee A(a_m)]$. Таким образом, квантор существования можно понимать как оператор дизъюнкции по квантифицируемой переменной.

Применение кванторов к *n*-местным предикатам. К n-местному предикату можно применить n кванторов. Применение квантора к n-местному предикату ($n\geq 1$) дает (n-1)- местный предикат.

Пример 2.10. $\exists x_1\ A(x_1,x_2,...,x_n)$ - $(n\text{-}1)$-местный предикат, у которого x_1 - *связанная* переменная, а x_2,x_3, ..., x_n - *свободные* переменные. Если все переменные n-местного предиката связаны, получаем 0-местный предикат, или высказывание, например:

$$(\exists x_1,x_2,...,x_n)\ A(x_1,x_2,...,x_n);$$

$$(\exists x_1,x_2)\ (\forall x_3)\ B(x_1,x_2,x_3).$$

2.5. Исчисление предикатов

Введем символы двух видов:

предметные переменные $(x,y,z,x_1,x_2...)$ и

предикатные буквы $(P,Q,R,P_1,P_2,...)$.

Из предикатных букв, предметных переменных, логических символов и скобок можно сформировать различные выражения, некоторые из которых называются *формулами*.

Формулами исчисления предикатов являются

а) каждая предикатная буква и предикатная буква со следующими за ней в скобках предметными переменными;

б) выражения $\neg(\Phi)$, $(\Phi_1)\&(\Phi_2)$, $(\Phi_1)\vee(\Phi_2)$,$(\Phi_1)\to(\Phi_2)$, $(\Phi_1)\leftrightarrow(\Phi_2)$, $\forall x\ (\Phi)$ и $\exists x\ \Phi(x)$, где Φ,Φ_1,Φ_2 – некоторые формулы; x – некоторая индивидная переменная, причем $\neg(\Phi)$ называется *отрицанием* формулы Φ, а $(\Phi_1)\&(\Phi_2)$, $(\Phi_1)\vee(\Phi_2)$, $(\Phi_1)\to(\Phi_2)$, $(\Phi_1)\leftrightarrow(\Phi_2)$ называются *конъюнкцией*, *дизъюнкцией*, *импликацией* и *эквивалентностью* формул Φ_1 и Φ_2 соответственно; $\forall x(\Phi)$ и $\exists x(\Phi)$ называются *квантификацией* формулы Φ по переменной x квантором общности и существования соответственно.

Предметная переменная называется *свободной*, если она не следует непосредственно за квантором и не входит в область действия квантора по этой переменной, все другие переменные, входящие в формулу, называются *связанными*.

Примечание. В общем случае формула исчисления предикатов может иметь вид предикатной буквы, за которой в скобках следуют предикатные переменные или функциональные буквы.

Интерпретацией формулы исчисления предикатов называется конкретизация множеств, из которых принимают значения предметные переменные и конкретизация отношений и соответствующих множеств истинности для каждой предикатной буквы.

Пример 2.11. 0-местная формула $\forall x(P(x) \to Q(x)) \leftrightarrow \neg\exists x\ (P(x)\ \&\ Q(x))$ ложна для интерпретации, при которой $x \in \{a,b,c,d,e\}$, $\{x\,|P(x)\} = \{a,b\}$, $\{x\,|Q(x)\} = \{a,b,c\}$.

Действительно, универсальное высказывание $\forall x(P(x) \to Q(x))$ истинно, поскольку $\{x\,|P(x)\} \subset \{x\,|Q(x)\}$. В то же время, $\exists x(P(x)\ \&\ Q(x))$ истинно, т.к. пересечение $\{x\,|P(x)\}$ и $\{x\,|Q(x)\}$ равно $\{a,b\}$, т.е. не пусто и, следовательно, высказывание $\neg\exists x(P(x)\ \&\ Q(x))$ ложно .

Таким образом, эквивалентность приводится к виду И↔Л, т.е. ложна.

Эта же 0-местная формула истинна для интерпретации, при которой $x \in \{a,b,c,d,e\}$, $\{x\,|P(x)\}=\{a,b\}$, $\{x\,|Q(x)\}=\{c,b\}$,

поскольку в этом случае формулы $\forall x(P(x) \to Q(x))$ и $\neg\exists x(P(x)\ \&\ Q(x))$ ложны и, следовательно, эквивалентность приводится к виду Л↔Л, т.е. истинна.

Пример 2.11 показывает, что существуют формулы исчисления высказываний, истинность которых зависит от интерпретации.

Формула исчисления предикатов называется *общезначимой*, если она тождественно истинна при любой интерпретации.

Общезначимая формула исчисления предикатов получается из тавтологии исчисления высказываний при замене входящих в нее пропозициональных букв предикатными буквами с произвольным числом приданных предметных переменных.

Общезначимость формул, содержащих кванторы, требует особого доказательства.

Приведем некоторые общезначимые формулы.

Законы де-Моргана для кванторов:

$$\neg\forall x\ P(x) \leftrightarrow \exists x\ \neg P(x), \tag{2.1}$$

$$\neg\exists x\ P(x) \leftrightarrow \forall x\ \neg P(x). \tag{2.2}$$

Законы пронесения кванторов через конъюнкцию и дизъюнкцию:

$$\forall x[P(x)\ \&\ Q(x)] \leftrightarrow [\forall x\ P(x)\ \&\ \forall x\ Q(x)], \tag{2.3}$$

$$\forall x[P(x) \vee Q] \leftrightarrow [\forall x\ P(x) \vee Q], \tag{2.4}$$

$$\exists x[P(x) \vee Q(x)] \leftrightarrow [\exists x\, P(x) \vee \exists x\, Q(x)], \quad (2.5)$$

$$\exists x[P(x) \,\&\, P] \leftrightarrow [\exists x\, P(x) \,\&\, P]. \quad (2.6)$$

Законы пронесения кванторов через импликацию:

$$\forall x[P(x) \to Q(x)] \to [\forall x\, P(x) \to \forall x\, Q(x)], \quad (2.7)$$

$$\forall x[P(x) \to Q(x)] \to [\exists x\, P(x) \to \exists x\, Q(x)], \quad (2.8)$$

$$\forall x[P(x) \to Q] \leftrightarrow [\exists x\, P(x) \to Q]. \quad (2.9)$$

Законы удаления квантора общности и введения квантора существования:

$$\forall x\, P(x) \to P(x), \quad (2.10)$$

$$P(x) \to \exists x\, P(x). \quad (2.11)$$

Законы преобразования категорических высказываний:

$$\forall x[P(x) \to Q(x)] \leftrightarrow \neg\exists x[P(x) \,\&\, \neg Q(x)], \quad (2.12)$$

$$\forall x[P(x) \to \neg Q(x)] \leftrightarrow \neg\exists x[P(x) \,\&\, Q(x)]. \quad (2.13)$$

Все формулы исчисления предикатов можно разделить на три типа:

- *истинные при любой интерпретации*, т.е. общезначимые;
- *ложные при любой интерпретации*, т.е. противоречивые;
- формулы, *истинность которых зависит от интерпретации*.

Чтобы определить тип формулы, то сначала следует попытаться установить общезначимость или противоречивость заданной формулы (тем более, что для одноместных предикатов это алгоритмически разрешимая задача) и, если это не удается сделать, то перейти к установлению значения истинности формулы для заданной интерпретации, например так, как было показано выше в примере 2.11.

Пример 2.12. Требуется установит тип формулы

$$\forall x(\neg P(x) \to \neg Q(x)) \leftrightarrow \neg\exists x(\neg P(x) \,\&\, Q(x)).$$

Попытаемся установить общезначимость или противоречивость данной формулы. Для этого левую часть заданной эквивалентности заменим равносильной формулой $\forall x(P(x) \vee \neg Q(x)$ на основе (1.38), а правую часть - формулой $\forall x \neg(\neg P(x) \,\&\, Q(x))$ на основе (2.2). Далее, с учетом (1.14), правая часть примет вид $\forall x(P(x) \vee \neg Q(x))$, т.е. исходная формула преобразована к виду

$\forall x(P(x) \vee \neg Q(x)) \leftrightarrow \forall x(P(x) \vee \neg Q(x))$, где левая часть равна правой. Следовательно, заданная формула является общезначимой.

2.6. Логика доказательства правильности алгоритмов и программ

Цель данного раздела – познакомить читателя с принципами *верификации* (*доказательства правильности*) алгоритмов и программ. Создание и весь последующий жизненный цикл надежного программного обеспечения для современных информационно-вычислительных систем – многоэтапный и трудоемкий процесс, который упрощенно можно охарактеризовать как *перевод требований технического* задания сначала в точные спецификации и, наконец, в текст программы.

Для описания алгоритмов используются различные методы, отличающиеся степенью детализации и формализации. Теоретическое описание обычно дается в повествовательно-формальном изложении, цель которого – обосновать без лишних подробностей процедуру, предлагаемую в качестве алгоритма. Для наглядного представления структуры алгоритмов широко применяются графические средства: графы, блок-схемы, сети. Формальное и полное описание алгоритмов осуществляется на алгоритмических языках; оно содержит всю необходимую для реализации алгоритма информацию.

Сложность программного продукта как объекта проектирования – основная причина ошибок перевода спецификаций в текст программы и, следовательно, ненадежности программного обеспечения. Для снижения сложности проекта используют технологию модульного проектирования и объектно-ориентированный подход.

Распространенный подход к обеспечению надежности проектируемого программного обеспечения – это *тестирование*. Цель тестирования – выявление ошибок, вкравшихся в программу на

разных стадиях проектирования. При таком подходе при написании программ акцент делается на их *тестируемость*, т.е. на создание программ, которые удобно тестировать, а безошибочность и корректность программы в значительной степени зависят от творческих способностей и интуиции разработчика.

В отличие от интуитивного подхода, который мы охарактеризовали выше, рассматриваемый далее подход трактует программирование как точную математическую науку. Этот подход основан на том, что спецификация программ выражается средствами логики, причем связь программ с их спецификацией осуществляется путем определения семантики программ также средствами логики (алгоритмическая логика Хоара[*]). Этот подход открывает путь к верификации (доказательству правильности) алгоритмов и программ средствами логики.

В основе верификации программ (алгоритмов) – анализ действия программ (алгоритмов) над данными. Для каждого исходного состояния данных X, для которого выполнение завершается, результирующее состояние данных Y является определенным. Это значение Y единственно для данного X, поэтому множество всех упорядоченных пар (X,Y) определяет функцию, которую будем называть *программной функцией*.

Мы будем использовать *символьные вычисления*, чтобы получить аналитическое выражение программной функции для исследуемой программы. Программную функцию f для *простой* программы P обозначим через $[P]$ и определим выражением

$$[P]=\{(X,Y) \mid Y=f(X)\},$$

где X - состояние поля данных до выполнения P,

Y - состояние поля данных после выполнения P,

$\{(X,Y) \mid Y=f(X)\}$ - множество пар (X,Y), таких, что $Y=f(X)$.

Последовательные (линейные) структуры

Рассмотрим *последовательность двух блоков* $g=\{(X,Y)\}$ и $h=\{(Y,Z)\}$, изображенную на рис.2.1. Программная функция такой

[*] Дал У., Дейкстра Э., Хоор[illegible] X [illegible]уктурное пр[illegible] Y [illegible]мирование. – [illegible] Z [illegible]ир, 1975.

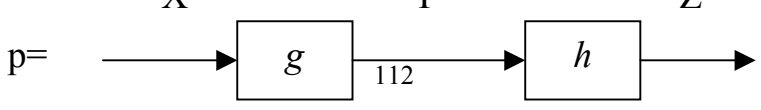

Рис.2.1.

последовательности является композицией $h \circ g$ двух частных функций h и g: $[P]=\{(X,Z) \mid Z=h(g(X))\}$.

Пример 2.13. Задана программа в виде последовательности операторов присваивания на алгоритмическом языке PASCAL (рис.2.2). Требуется определить функцию, реализуемую этим алгоритмом.

Рассмотрим поле данных (x,y,z), которое используется в данной программе. Начальное значение поля данных (x_0,y_0,z_0).

```
x:= y + z;   y:= x - z;   z:= -x + y;
```
Рис.2.2.

Процесс прослеживания состояния поля данных после выполнения каждого оператора называется *трассировкой*. Его удобно представить таблицей 2.1.

Таблица 2.1

Оператор	x	y	z
$x:=y+z$	$x_1=y_0+z_0$	$y_1=y_0$	$z_1=z_0$
$y:=x-z$	$x_2=x_1$	$y_2=x_1-z_1$	$z_2=z_1$
$z:=-x+y$	$x_3=x_2$	$y_3=y_2$	$z_3=-x_2+y_2$

После выполнения первого оператора присваивания первоначальное значение x_0 стирается и заменяется новым значением, $x_1=y_0+z_0$, причем значения y и z не меняются, т.е. $y_1=y_0$, $z_1=z_0$. После выполнения второго оператора присваивания получаем:

$$x_2 = x_1 = y_0 + z_0,$$
$$y_2 = x_1 - z_1 = y_0 + z_0 - z_0 = y_0,$$
$$z_2 = z_1 = z_0.$$

После выполнения третьего оператора присваивания получаем:

$$x_3 = x_2 = x_1 = y_0 + z_0,$$
$$y_3 = y_2 = y_0,$$
$$z_3 = -x_2 + y_2 = -y_0 - z_0 + y_0 = -z_0.$$

Таким образом, функцию реализуемую алгоритмом, можно представить как одновременное присваивание $(x,y,z):= (y_0+z_0 \ , \ y_0 \ , \ -z_0)$, в результате которого первоначальное значение поля данных (x_0,y_0,z_0) изменится на $(y_0+z_0 \ , \ y_0 \ , \ -z_0)$.

Структуры с ветвлениями

Для *программы с ветвлениями* все возможные пути выполнения определяются так называемым *деревом выполнения* (*E-деревом*).

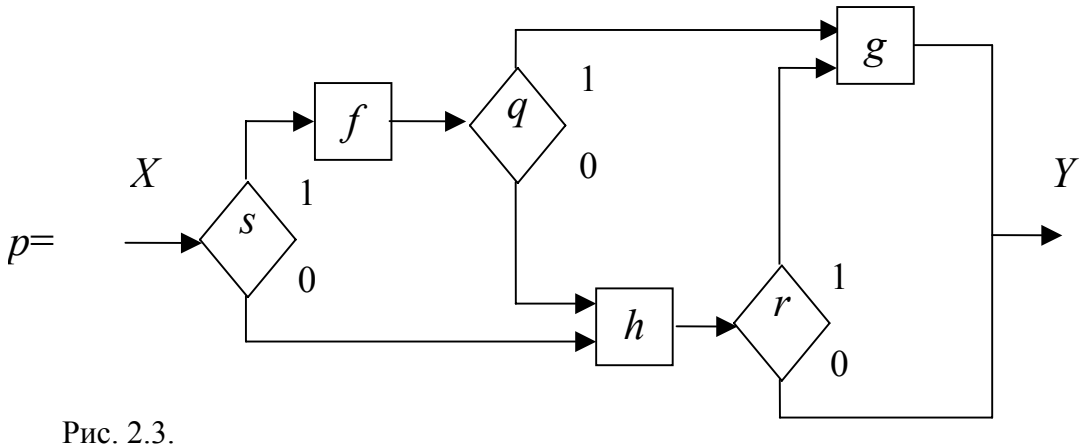

Рис. 2.3.

Пример 2.14. Рассмотрим блок-схему программы на рис.2.3. Эта программа имеет дерево выполнения, представленное на рис.2.4. Условия ветвления выражаются предикатами *s, q, r*. Программная функция программы с ветвлениями без циклов определяется как *объединение композиций программных функций*, которые получаются непосредственно из *E*-дерева. Необходимое и достаточное условие выполнения конкретной ветви определяется композицией каждого предиката с предшествующей функцией пути.

В рассматриваемом примере имеется пять путей выполнения, пронумерованных как показано на рис. 2.4. Выпишем программную функцию каждого пути:

1. $\{(X,Y)\colon s(X) \,\&\, q(f(X)) \,\&\, Y=g(f(X))\}$.
2. $\{(X,Y)\colon s(X) \,\&\, \neg q(f(X)) \,\&\, r(h(f(X))) \,\&\, Y=g(h(f(X)))\}$.
3. $\{(X,Y)\colon s(X) \,\&\, \neg q(f(X)) \,\&\, \neg r(h(f(X))) \,\&\, Y=h(f(X))\}$.
4. $\{(X,Y)\colon \neg s(X) \,\&\, r(h(X)) \,\&\, Y=g(h(X))\}$.
5. $\{(X,Y)\colon \neg s(X) \,\&\, \neg r(h(X)) \,\&\, Y=h(X)\}$.

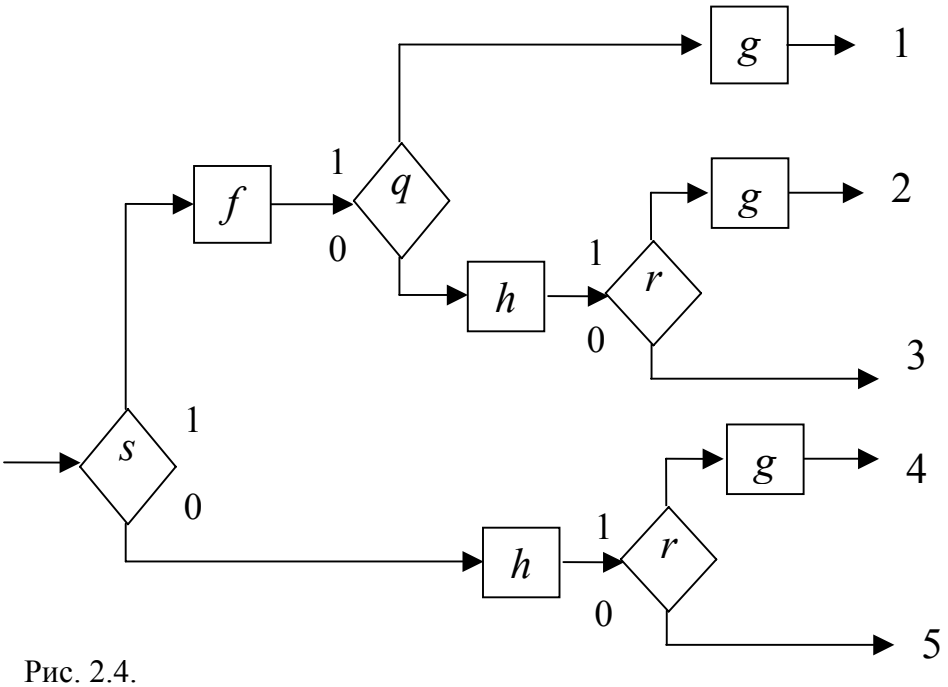

Рис. 2.4.

Результирующая программная функция может быть определена как условное правило:

$$[P]=\{(X,Y) \mid s(X) \,\&\, q(f(X)) \to Y=g(f(X));$$

$$s(X) \,\&\, \neg(q(f(X)) \,\&\, r(h(f(X))) \to Y=g(h(f(X)));$$

$$s(X) \,\&\, \neg(q(f(X))) \,\&\, \neg(r(h(f(X)))) \to Y=h(f(X));$$

$$\neg s(X) \,\&\, r(h(X)) \to Y=g(h(X));$$

$$\neg s(X) \,\&\, \neg(r(h(X))) \to Y=h(X)\}.$$

Пример 2.15. Задан алгоритм с использованием операторов ветвления (Рис.2.5).

```
IF x<-1 THEN x:=x+y  ELSE  IF y<0 THEN  y:=x-y;
IF y>-2 THEN y:=x+y  ELSE  x:=x-y
```

Рис.2.5

Требуется определить функцию, реализуемую этим алгоритмом.

В данном алгоритме используется поле данных (x,y), начальное значение этого поля (x_0,y_0). Для установления функции данного алгоритма требуется проследить все ветви его выполнения.

Для этого введем обозначение предикатов, описывающих условия ветвления:

$P_1(x)$="x<-1" , $P_2(y)$="y<0" , $P_3(y)$="y>-2",
а также обозначения операторов присваивания:

A_1="x:=x+y", A_2="y:=x-y", A_3="y:=x+y", A_4="x:=x-y".
Тогда можно записать схему алгоритма (рис.2.6).

IF P_1 THEN A_1 ELSE IF P_2 THEN A_2;
IF P_3 THEN A_3 ELSE A_4

Рис.2.6.

Этой схеме соответствует граф-схема алгоритма (рис.2.7).

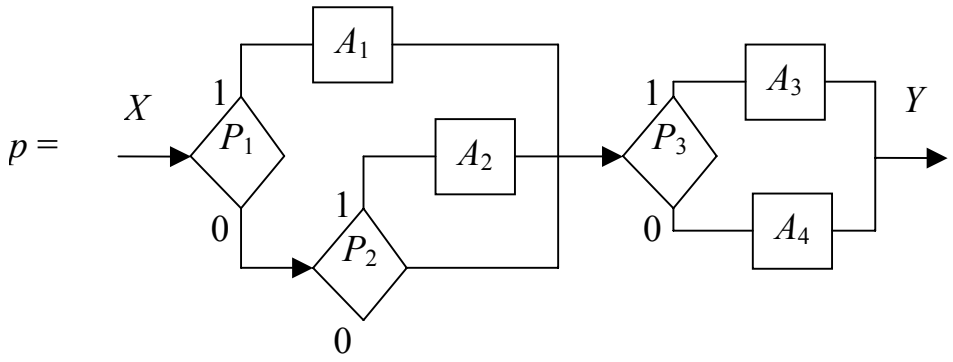

Рис. 2.7.

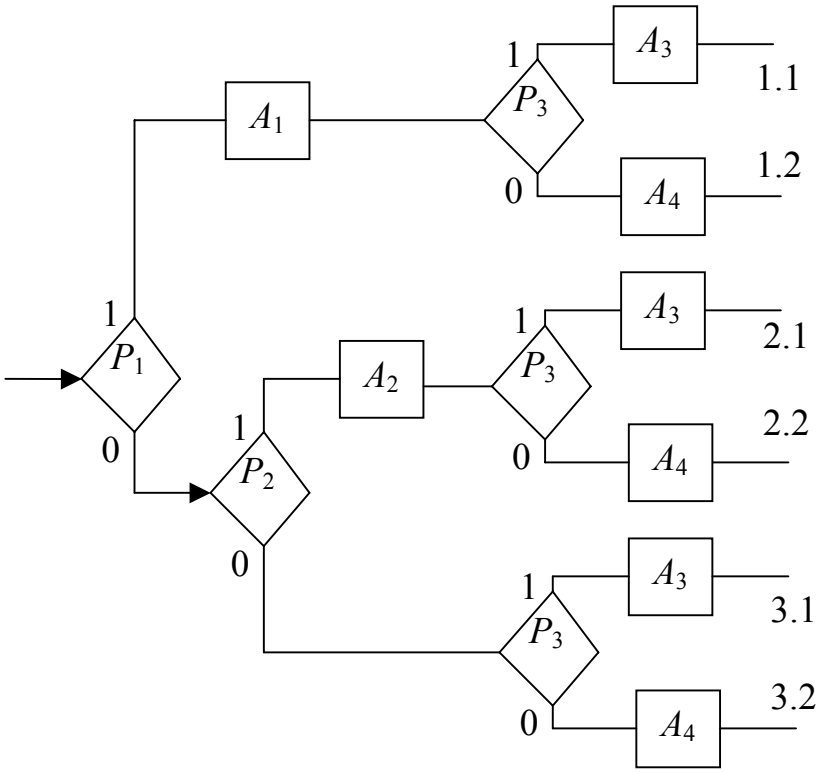

Рис. 2.8.

Таблица 2.2

Путь 1.1				Условие для пути 1.1: $(x_0<-1)\ \&\ (y_1>-2) = (x_0<-1)\ \&\ (y_0>-2)$
Оператор	Условие	x	y	
$x:=x+y$	$x_0<-1$	$x_1=x_0+y_0$	$y_1=y_0$	Функция пути 1.1: $x_2=x_1=x_0+y_0$, $y_2=x_1+y_1=x_0+y_0+y_0=x_0+2y_0$, т.е. $(x,y):=(x_0+y_0, x_0+2y_0)$
$y:=x+y$	$y_1>-2$	$x_2=x_1$	$y_2=x_1+y_1$	

Таблица 2.3

Путь 1.2				Условие пути 1.2: $(x_0<-1)\&(y_1\le-2)= (x_0<-1)\&(y_0\le-2)$
Оператор	Условие	x	y	
$x:=x+y$	$x_0<-1$	$x_1=x_0+y_0$	$y_1=y_0$	Функция пути 1.2: $x_2=x_1-y_1=x_0+y_0-y_0=x_0$; $y_2=y_1=y_0$, т.е. $(x,y):=(x_0, y_0)$
$x:=x-y$	$y_1\le-2$	$x_2=x_1-y_1$	$y_2=y_1$	

Таблица 2.4

Путь 2.1				Условие пути 2.1: $(x_0\ge-1)\&(y_0<0)\&(y_1>-2) = (x_0\ge-1)\&(x_0-y_0>-2)\&(y_0<0)= (x_0\ge-1)\&(y_0<0)\&(y_0<x_0+2)= (x_0\ge-1)\&(y_0<0)$
Оператор	Условие	x	y	
$y:=x-y$	$(x_0\ge-1)\ \&\ (y_0<0)$	$x_1=x_0$	$y_1=x_0-y_0$	Функция пути 2.1: $x_2=x_1=x_0$; $y_2=x_1+y_1= x_0+x_0-y_0=2x_0-y_0$, т.е. $(x,y):=(x_0, 2x_0-y_0)$
$y:=x+y$	$y_1>-2$	$x_2=x_1$	$y_2=x_1+y_1$	

Таблица 2.5

Путь 2.2				Условие пути 2.2: $(x_0\ge-1)\&(y_0<0)\&(y_1\le-2)= (x_0\ge-1)\&(x_0-y_0\le-2)\&(y_0<0)= (x_0\ge-1)\ \&\ (y_0<0)\ \&\ (y_0>x_0+2) =$ **ЛОЖЬ, т.е. путь 2.2 не реализуется**
Опера-тор	Условие	x	y	
$y:=x-y$	$(x_0\ge-1)\ \&\ (y_0<0)$	$x_1=x_0$	$y_1=x_0-y_0$	—
$x:=x-y$	$y_1\le-2$	$x_2=x_1-y_1$	$y_2=y_1$	

Таблица 2.6

Путь 3.1				Условие пути 3.1: $(x_0 \geq -1)\&(y_0 \geq 0)\&(y_1 > -2) = (x_0 \geq -1)\&(y_0 \geq -2)\&(y_0 \geq 0) = (x_0 \geq -1)\&(y_0 \geq 0)$
Оператор	Условие	x	y	
-	$(x_0 \geq -1)$ & $(y_0 \geq 0)$	$x_1 = x_0$	$y_1 = y_0$	Функция пути 3.1: $x_2 = x_1 = x_0$; $y_2 = x_1 + y_1 = x_0 + y_0$, т.е. $(x,y) := (x_0, x_0 + y_0)$
$y := x + y$	$y_1 > -2$	$x_2 = x_1$	$y_2 = x_1 + y_1$	

Таблица 2.7

Путь 3.2				Условие пути 3.2: $(x_0 \geq -1)\&(y_0 \geq 0)\&(y_1 \leq -2) = (x_0 \geq -1)\&(y_0 \leq -)\&(y_0 \geq 0) =$ **ЛОЖЬ, т.е. путь 3.2 не реализуется**
Оператор	Условие	x	y	
-	$(x_0 \geq -1)$ & $(y_0 \geq 0)$	$x_1 = x_0$	$y_1 = y_0$	—
$x := x - y$	$y_1 \leq -2$	$x_2 = x_1 - y_1$	$y_2 = y_1$	

Граф-схема реализации алгоритма (рис.2.8) показывает все шесть возможных путей, по которым может пойти вычислительный процесс: 1.1, 1.2, 2.1, 2.2, 3.1, 3.3. Для выявления функции, реализуемой алгоритмом, составим шесть таблиц трассировки, а именно таблицы 2.2, 2.3, 2.4, 2.5, 2.6, 2.7 соответственно. Из этих таблиц выписываем результирующую функцию алгоритма

$$(x,y) = \begin{cases} (x_0+y_0,\ x_0+2y_0), & \text{если } (x_0<-1)\&(y_0>-2); \\ (x_0,\ y_0), & \text{если } (x_0<-1)\&(y_0\leq -2); \\ (x_0,\ 2x_0-y_0), & \text{если } (x_0\geq -1)\&(y_0<0); \\ (x_0,\ x_0+y_0), & \text{если } (x_0\geq -1)\&(y_0\geq 0). \end{cases}$$

Циклические структуры

Для определения программной функции *циклических программ* возможны два подхода.

Первый подход основан на выявлении *узлов слияния* в блок-схеме программы. Для каждого узла слияния записывается выражение программной функции. Рассмотрим пример циклической программы на рис. 2.9.

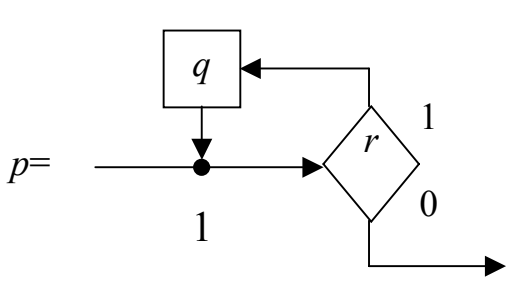

Рис. 2.9.

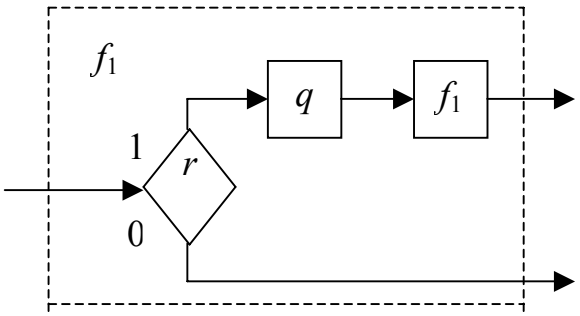

Рис. 2.10.

В этом примере один узел 1, для которого определяем функцию f_1, исходя из *E*-схемы на рис.2.10:

$$f_1=\{(X,Y) \mid r(X) \to f_1 \circ q(X)\ ;\ \ \overline{r}(X) \to Y=X\}. \qquad (2.14)$$

Очевидно, что выполняется соотношение $[P]=f_1$. Однако определение $[P]$ таким образом не всегда возможно, т.к. не удается разрешить выражение типа (2.14) относительно f_1.

Второй подход основан на использовании *инвариантов цикла*. Пусть в предыдущем примере (рис.2.8) имеем поле данных (u,v), предикат p="$v\neq 0$", а функцию блока q конкретизируем как $u,v := u+1, v-1$. Предикат $u+v= u_0+v_0$, где u_0,v_0 - начальные значения u и v, принимает значение "истинно" в трех случаях:

- при входе в цикл,
- при каждой итерации,
- при выходе из цикла.

Следовательно, предикат $u+v= u_0+v_0$ является инвариантом. При выходе из цикла $v=0$, откуда $u+0=u_0+v_0$ и $u=u_0+v_0$. Следовательно, функция программы

$[P]=(u,v\geq 0 \to u,v := u+v,0)$.

Дополнительные сведения по логике предикатов можно найти в [5, 7, 10, 11]. Методика верификации алгоритмов и программ рассмотрена подробно в [6].

Глава 3

ВАРИАНТЫ ЛОГИКИ И ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

3.1. Стандартная логика

Логическое программирование основано на *клаузальной логике*. Клаузальная логика – это форма *стандартной логики* (классической) и отличается от нее системой обозначений. В основе стандартной формы логики лежит логика высказываний (пропозициональная логика) и логика предикатов, рассмотренные в предыдущих главах.

Для изложения клаузальной логики и принципов логического программирования повторим основные определения и приведем дополнительные сведения из логики высказываний и логики предикатов.

Напомним, что высказывание – это повествовательное предложение, в отношении которого имеет смысл утверждение об его истинности или ложности. Пример истинного высказывания: "Земля вращается вокруг Солнца". Предикат – это повествовательное предложение, содержащее предметные (индивидные переменные), замена которых на константные значения превращает рассматриваемое предложение в высказывание – истинное или ложное.

В теоретических рассмотрениях используются формулы на основе *префиксного обозначения* предикатов (*префиксной нотации*), например,

$$P(x_1,...,x_n),$$

где P – предикатная буква (предикатный символ), $x_1,...,x_n$ – предметные переменные, принимающие значения из множеств $A_1,...,A_n$.

Каждому набору (кортежу) $(a_1,...,a_n)$ из n элементов $a_1 \in A_1,...,a_n \in A_n$ ставится в соответствие истинное или ложное значение предиката $P(x_1,...,x_n)$. Напомним, что множество кортежей $(a_1,...,a_n)$, на которых предикат P принимает истинное значение, называется множеством истинности предиката $P(x_1,...,x_n)$.

Множество истинности предиката $P(x_1,...,x_n)$ – это некоторое подмножество декартова произведения $A_1 \times A_2 \times ... \times A_n$ множеств, на которых определены предметные переменные $x_1,...,x_n$. Тем самым, множество кортежей, принадлежащих множеству истинности предиката, является n-арным отношением, и мы можем говорить о предикате $P(x_1,...,x_n)$ отношения P, используя один и тот же символ для обозначения предикатов и отношений.

В естественных языках используется *инфиксная нотация* предикатов. Это означает, что предметные переменные в той или иной степени рассредоточены по предложению, выражающему предикат. В таблице 3.1 приведены примеры инфиксной и префиксной нотации предикатов

Таблица 3.1

Инфиксная нотация	Префиксная нотация
X вращается вокруг Y	*вращается_вокруг* (*X,Y*)
X является звездой	*звезда*(*X*)
X это планета	*планета*(*X*)
X является спутником Y	*спутник*(*X,Y*)

Логика высказываний и предикатов основана на пропозициональных связках:

- И (конъюнкция),
- ИЛИ (дизъюнкция),
- ЕСЛИ-ТО (импликация),
- ЕСЛИ-И-ТОЛЬКО-ЕСЛИ (эквивалентность),
- НЕ (операции отрицания, инверсия).

С помощью этих связок из простых предложений, выражающих высказывания или предикаты, строятся составные предложения. Сводка определений этих связок приведена в таблице 3.2, в которой *X* и *Y* это переменные высказывания или предикаты.

Таблица 3.2

X *Y*	НЕ ¬	И &	ИЛИ ∨	ЕСЛИ-ТО →	ЭКВИВА-ЛЕНТНОСТЬ ↔
Л Л	И	Л	Л	И	И
Л И	И	Л	И	И	Л
И Л	Л	Л	И	Л	Л
И И	Л	И	И	И	И

При построении предложений (формул) в стандартной логике применяются также специальные обозначения, называемые кванторами. Квантор всеобщности записывается как "$\forall x$" и читается как "для всех x". Квантор существования записывается как "$\exists x$" и читается как "существует x". В логике предикатов подразумевается, что задано множество A, на котором определена переменная, входящая в квантор всеобщности или существования, причем квантифицированные предикаты понимаются следующим образом:

$$\forall x\, P(x) = \&_{x \in A}\, P(x) \qquad \text{и} \qquad \exists x\, P(x) = \vee_{x \in A}\, P(x).$$

С помощью пропозициональных связок и кванторов из предикатных символов строятся формулы и сложные высказывания.

Для дальнейшего изложения требуется определить основные понятия, связанные с представлением предикатов и высказываний в виде формул.

Атом (*атомарная формула*) – это выражение вида $P(t_1,...,t_n)$, где P это n-местный предикатный символ, $t_1,...,t_n$ – это термы ($n \geq 1$). Каждый *терм* – это либо предметная переменная, константа или n-местный функциональный символ $f_i(t_1,...,t_n)$, представляющий некоторую функцию.

Формула – это выражение вида $(X \& Y)$, $(X \vee Y)$, $(X \rightarrow Y)$ или $(Y \leftarrow X)$, $(X \leftrightarrow Y)$, $\neg X$, $\forall vX$, $\exists vX$, где X,Y суть формулы, а v - переменная.

Вхождение некоторой переменной v в формулу Z является свободным (или несвязанным), если оно не принадлежит ни одной подформуле Z, имеющей вид $\forall vX$ или $\exists vX$. Формула является высказыванием, если и только если она не содержит никаких свободных вхождений предметных переменных.

Рассмотрим предложение "Если y планета и x вращается вокруг y, то x является спутником y". Подразумевается, что переменные x и y

определены на множестве небесных тел и вхождения этих переменных в рассматриваемое предложение связаны квантором всеобщности. Поэтому перевод данного предложения в стандартную форму логики имеет вид:

$$\forall x \forall y\ [планета(y)\ \&\ вращается\_вокруг(x,y) \rightarrow спутник(x,y)].$$

Это предложение не содержит свободных переменных, поэтому оно является высказыванием.

3.2. Клаузальная логика

При записи предложений в клаузальной форме кванторы в явном виде не указываются, но предполагается, что все переменные связаны квантором всеобщности. *Клауза* (от англ. *clause* – предложение) общего вида записывается следующим образом:

$$B_1,...,B_m \leftarrow A_1,...,A_n,$$

где $B_1,...,B_m$, $A_1,...,A_n$ – это атомарные формулы ($n \geq 0$, $m \geq 0$), $A_1,...,A_n$ – это совместные *посылки* клаузы, а $B_1,...,B_m$ – это альтернативные *заключения*.

В стандартной форме клауза равносильна записи

$$A_1\ \&\ ...\&\ A_n \rightarrow B_1 \vee ... \vee B_m.$$

Таким образом, *запятые в посылке клаузы означают конъюнктивные связки* между предложениями, составляющими посылку, а *запятые в заключении клаузы символизируют дизъюнктивные связки* между предложениями, составляющими посылку. Множество клауз *невыполнимо*, если из него можно вывести *пустое предложение* (обозначаемое □).

Преобразование предложений из стандартной формы в клаузальную

Любое предложение в стандартной форме может быть преобразовано в клаузальную форму, причем результирующее

множество клауз совместно *если и только если* совместно исходное множество предложений в стандартной форме.

Для упрощения в записи формул будем опускать внешние скобки. Также для упрощения записи в дальнейшем будем считать, что символы отрицания $\neg$ и кванторов $\forall$ и $\exists$ задают более тесную связь, чем другие пропозициональные связки, а символы конъюнкции & и дизъюнкции $\vee$ связывают теснее, чем импликация $\rightarrow$ и эквивалентность. С учетом ассоциативности $\vee$ и & можно, например, писать

$$A \vee B \vee C \leftarrow D \,\&\, E \,\&\, F$$

вместо

$$[A \vee [B \vee C\,]] \leftarrow [[D \,\&\, E\,] \,\&\, F].$$

Кроме того, если из контекста ясно, что формула является высказыванием, можно опускать квантор всеобщности и, например, вместо

$$\forall x \forall y\, [\textit{планета}(y) \,\&\, \textit{вращается\_вокруг}(x,y) \rightarrow \textit{спутник}(x,y)]$$

писать

$$\textit{планета}(y) \,\&\, \textit{вращается\_вокруг}(x,y) \rightarrow \textit{спутник}(x,y).$$

Для преобразования формул (предложений) из стандартной формы в клаузальную используется следующая система правил эквивалентности:

$$\forall uX \leftrightarrow \forall v\, X' \text{ и } \exists uX \leftrightarrow \exists vX', \tag{3.1}$$

где X' получается из X заменой всех вхождений u на v, причем v не встречается в X;

$$\forall vX \leftrightarrow X \text{ и } \exists vX \leftrightarrow X, \tag{3.2}$$

где переменная v не встречается в X;

$$[X \rightarrow Y] \leftrightarrow \neg X \vee Y, \tag{3.3}$$

$$[X \leftrightarrow Y] \leftrightarrow [X \rightarrow Y] \& [Y \rightarrow X], \text{ т.е.} \tag{3.4}$$

$$[X \leftrightarrow Y] \leftrightarrow [\neg X \vee Y] \& [\neg Y \vee X]; \tag{3.5}$$

$$\neg [X \& Y] \leftrightarrow \neg X \vee \neg Y, \tag{3.6}$$

$$\neg [X \vee Y] \leftrightarrow \neg X \,\&\, \neg Y, \tag{3.7}$$

$$\neg\exists vX \leftrightarrow \forall v\neg X, \tag{3.8}$$

$$\neg\forall vX \leftrightarrow \exists v\neg X, \tag{3.9}$$

$$\neg\neg X \leftrightarrow X, \tag{3.10}$$

где X и Y суть формулы, а v – произвольная переменная;

$$[Y \,\&\, Z] \vee X \leftrightarrow [Y \vee X]\&[Z \vee X], \tag{3.11}$$

$$\exists vY \vee X \leftrightarrow \exists v[Y \vee X], \tag{3.12}$$

$$\forall vY \vee X \leftrightarrow \forall v[Y \vee X], \tag{3.13}$$

где переменная v не встречается в X;

$$\forall v[X \,\&\, Y] \leftrightarrow \forall vX \,\&\, \forall vY. \tag{3.14}$$

Порядок преобразования формул (предложений) из стандартной формы в клаузальную следующий.

1. Выполнить стандартизацию переменных, т.е. устранить случаи, когда одна и та же переменная связана разными вхождениями одного квантора, например: $\exists x[P(x) \,\&\, \exists xQ(x)]$. Применяя правило (3.1), получаем $\exists x[P(x) \,\&\, \exists zQ(z)]$.

2. Выразить импликацию $\rightarrow$ и эквивалентность $\leftrightarrow$ через $\vee$, $\&$ и $\neg$ с помощью правил (3.3-3.5).

3. Переместить отрицания $\neg$ внутрь предложений за знаки $\&$, $\vee$ и кванторы так, чтобы они оказались непосредственно перед атомарными формулами. Для этого используются правила (3.6-3.13).

4. Переместить знаки дизъюнкции $\vee$ внутрь предложений за знаки $\&$ и кванторы так, чтобы знаки $\vee$ связывали только атомы и атомы с отрицаниями. Для этого используются правила (3.13).

5. Устранить кванторы существования введением так называемых функций Сколема (см. ниже) и продвинуть все кванторы A внутрь конъюнкций с помощью правила (3.14).

6. Перезаписать полученные дизъюнкции атомов и их отрицаний (иногда называемых дизъюнктами)

$$A_1 \vee\ldots\vee A_m \vee \neg B_1 \vee\ldots \neg B_n$$

в виде клауз

$$A_1,\ldots,A_m \leftarrow B_1,\ldots,B_n. \tag{3.15}$$

Поясним пункт 5. Для исключения квантора существования $\exists$ вводится константа или функциональный символ (функция Сколема) для того, чтобы дать имя значению переменной, на которую неявно ссылается $\exists$. Пусть дано множество (конъюнкция) предложений S. Для исключения $\exists$ из предложения вида $\forall v_1 \forall v_2 ... \forall v_n \exists u X$, входящего в S, заменяем это предложение на новое $\forall v_1 \forall v_2 ... \forall v_n X'$, где X' получено из X заменой всех свободных вхождений u в X на терм $f(v_1,...,v_n)$, где f - некоторый функциональный символ, не встречающийся в S. Если n=0, то терм $f(v_1,...,v_n)$ сводится к константному символу.

Отметим, что замена не является эквивалентной и применима только к высказываниям, но не к формулам со свободными переменными. Из нового предложения следует исходное, но не наоборот. Тем не менее, исключение квантора $\exists$ не оказывает влияния на совместность результирующего множества клауз.

Приведенные правила (3.1-3.14) являются логической основой семейства алгоритмов приведения стандартной формы к клаузальной. Для повышения эффективности алгоритма следует применять эти правила прежде всего к самым внешним пропозициональным связкам и заменять формулу с левой стороны правила эквивалентной формулой с правой стороны.

Число шагов преобразований сокращается, если в процессе преобразований не заменять импликации $\rightarrow$. Для этого полезны следующие эквивалентности:

$$[X \rightarrow Y \,\&\, Z] \leftrightarrow [X \rightarrow Y] \& [X \rightarrow Y]. \tag{3.16}$$

$$[X \vee Y \rightarrow Z] \leftrightarrow [X \rightarrow Y] \& [X \rightarrow Y]. \tag{3.17}$$

$$[X \,\&\, \neg Y \rightarrow Z] \leftrightarrow [X \rightarrow Y \vee Z]. \tag{3.18}$$

$$[X \rightarrow \neg Y \vee Z] \leftrightarrow [X \,\&\, Y \rightarrow Z]. \tag{3.19}$$

$$[X \rightarrow [Y \rightarrow Z]] \leftrightarrow [X \,\&\, Y \rightarrow Z]. \tag{3.20}$$

$$[[X \rightarrow Y] \rightarrow Z] \leftrightarrow [X \vee Y] \& [Y \rightarrow Z]. \tag{3.21}$$

$$X \rightarrow \forall v Y \leftrightarrow \forall v [X \rightarrow Y]. \tag{3.22}$$

$$X \rightarrow \exists v Y \leftrightarrow \exists v [X \rightarrow Y]. \tag{3.23}$$

$$\forall v Y \to X \leftrightarrow \exists v[Y \to X]. \quad (3.24)$$

$$\exists v Y \to X \leftrightarrow \forall v[Y \to X]. \quad (3.25)$$

$$[U \,\&[X \vee Y] \to Z] \leftrightarrow [U \,\& X \to Z] \& [U \,\& Y \to Z]. \quad (3.26)$$

$$[U \,\&[X \to Y] \to Z] \leftrightarrow [U \to X \vee Z] \& [U \,\& Y \to Z]. \quad (3.27)$$

Пример 3.1. Дана стандартная форма:

$\forall x \, \forall y$ *Планета*(y) & *вращается_вокруг*(x,y) $\to$ *спутник*(x,y).

Перевод в клаузальную форму (в скобках указаны номера используемых правил):

(п3) $\neg$[*планета*(y) & *вращается_вокруг*(x,y)] $\vee$ *спутник*(x,y);

(п6) $\neg$*планета*(y) $\vee$ $\neg$*вращается_вокруг*(x,y) $\vee$ *спутник*(x,y);

(п15) *спутник*(x,y) $\leftarrow$ *планета*(y), *вращается_вокруг*(x,y).

Пример 3.2. Дано предложение на русском языке:

Каждый совершает ошибки.

Введем функцию Сколема *ошибка*(y) и запишем это предложение в стандартной форме:

$\forall x \exists y$[*человек*(x) $\to$ *совершает*(x,y) & *ошибка*(y)].

Перевод в клаузальную форму:

(функция Сколема)

человек(x) $\to$ *совершает*(x,*ош*(x))& *ошибка*(*ош*(x)),

где *ош*(x) - функциональный символ;

(п3) $\neg$*человек*(x) $\vee$ [*совершает*(x,*ош*(x)) & *ошибка*(*ош*(x))]

(п11) [$\neg$*человек*(x) $\vee$ *совершает*(x,*ош*(x))] &

[$\neg$*человек*(x) $\vee$ *ошибка*(*ош*(x))];

(п15) *совершает*(x,*ош*(x)) $\leftarrow$ *человек*(x),

ошибка(*ош*(x)) $\leftarrow$ *человек*(x).

3.3. Логическое программирование

Основные трудности, с которыми постоянно сталкиваются разработчики программного обеспечения, связаны с построением систем, осуществляющих простую и быструю работу с данными. Это прежде всего задачи из области искусственного интеллекта: синтаксический и семантический анализ текста, организация баз данных и знаний, распознавание образов, диагностика

неисправностей в технике и заболеваний в медицине, управление роботами и т.д.

В общем случае везде, где требуется анализ данных и возможны различные варианты решения, а сам ход решения не очень ясен для программиста или же желательно избежать детального описания своих действий, которые и так всем интуитивно понятны, во всех этих случаях целесообразно использовать логическое программирование. Логическое программирование позволяет обойтись без традиционной алгоритмизации решаемой задачи и сконцентрировать усилия на *формализации знаний о предметной области* решаемой задачи.

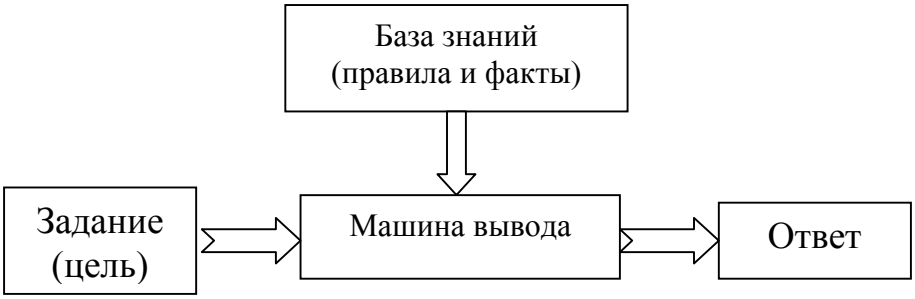

Рис.3.1.

Наибольшее распространение получила система логического программирования, основанная на представлении знаний в виде так называемых *клауз Хорна*. Это система и язык Prolog (PROgramming in LOGic), предложенная в начале 70-х годов Аланом Колмероем. Общая идея логического программирования представлена схемой на рис. 3.1. База знаний содержит описание *правил вывода* и известных *фактов* в форме клауз Хорна. Задание на работу с системой логического программирования имеет вид клаузы или набора клауз, в которых фигурируют предметные переменные, значения которых необходимо определить.

Машина вывода предназначена для *процедурной интерпретации* логической программы, т.е. всей совокупности клауз Хорна (запроса и базы знаний). Машина вывода реализует заложенный в нее алгоритм обработки клауз (утверждений) логической программы, называемый *методом резолюций*, для поиска возможных решений.

Клаузы Хорна и метод резолюций

Клаузой Хорна называется частный случай клаузы $B_1,...,B_m \leftarrow A_1,...,A_n$, получающийся при m=1:

$$B \leftarrow A_1,...,A_n.$$

Метод резолюций основан на использовании *правил резолюции*. Правило резолюции – это правило вывода, которое позволяет из двух предложений

$$B_1,...,B_m \leftarrow A_1,...,A_i,...,A_n \quad \text{и}$$

$$B'_1,...,B'_j,...,B'_r \leftarrow A'_1,...,A'_k\ ,$$

для которых существует так называемая *согласующая подстановка* σ такая, что $A_i\sigma=B'_j\sigma$, вывести третье

$$B_1\sigma,..., B_m\sigma, B'_1\sigma,..., B'_{j-1}\sigma, B'_{j+1}\sigma,..., B'_r\sigma \leftarrow$$

$$A_1\sigma,..., A_{j-1}\sigma, A_{j+1}\sigma,..., A_n\sigma,\ A'_1\sigma,..., A'_k\sigma.$$

Понятие согласующей подстановки σ определяется как множество пар $\{(x_q,t_q)\}$, означающих, что переменная x_q всюду в A_i и B'_j заменяется термом t_q так, что $A_i\sigma=B'_j\sigma$, причем любая другая унифицирующая подстановка θ носит менее общий характер.

Для клауз Хорна правило резолюции можно записать в общем виде

$$\frac{B \leftarrow A_1,A_2,...,A_n \mid B' \leftarrow A'_1,...,A'_k}{B\sigma \leftarrow A'_1\sigma,...,A'_k\sigma, A_2\sigma,...,A_n\sigma}\ A_1\sigma=B'_1\sigma,$$

где σ - согласующая подстановка, | - разделитель клауз.

Сущность метода резолюций – это *процедурная интерпретация клауз Хорна*, которая заключается в следующем. Множество клауз Хорна *S*, описывающих постановку задачи, дополняется отрицанием C← целевого утверждения C, содержащего в общем случае некоторые предметные переменные, значения которых необходимо определить. *С помощью правил резолюции находим решение задачи, а именно значения предметных переменных, для которых множество* $S \cup \{C\leftarrow\}$ *невыполнимо*. Это значения предметных

переменных, при которых из множества предложений $S \cup \{C\leftarrow\}$ удается вывести пустое предложение □.

Пример 3.3. Дана формулировка задачи на естественном языке:
Робот находится там, где находится тележка. Определить, где находится робот, если известно, что тележка находится на складе.

Запись условий задачи в стандартной форме:

где(*робот*, *x*) ← *где*(*тележка*, *x*).
где(*тележка, склад*) ←.
← *где*(*робот*, *y*).

В соответствии с методом резолюций запрос формируется как отрицание предложения "*где*(*робот*, *y*)". Последовательность резолюций:

← *где*(*робот*, *y*) | *где*(*робот*, *x*) ← *где*(*тележка*, *x*)
——————————————————————— *y*=*x* ;
← *где*(*тележка*, *x*)

← *где*(*тележка*, *x*) | *где*(*тележка*, *склад*) ←
——————————————————————— *x*=*склад* .
□

Таким образом, на втором шаге при использовании согласующей подстановки *x*=*склад* получено пустое предложение □ и ответ *где*(*робот*, *склад*), что означает, что робот находится на складе.

3.4. Prolog – язык логического программирования

Программа на языке Prolog состоит из разделов, имеющих специальные названия:

constants /*объявление значений констант*/
domains /*объявление доменов, описывающих классы объектов*/
database /*предикаты динамической базы данных*/
predicates /*описание имен и структур предикатов, определяемых пользователем */
goal /*целевое утверждение (цель), описывает желаемый результат */
clauses /*содержит описание предметной области задачи: факты и правила логического вывода решения */.

Не все разделы обязательно должны присутствовать в Prolog-программе. Возможности языка Prolog продемонстрируем на примерах Prolog-программ.

Пример 3.4. Для клаузальной формулировки задачи, рассмотренной в примере 3.2, можно составить следующую Prolog-программу:

```
domains
        being=symbol
        what=symbol; err(being)
predicates
        person(being)
        makes(being,what)
        error(what)
clauses
        person("Иван").
        person("Таня").
        makes(X,err(X)):-person(X).
        error(err(X)):-person(X).
```

Дадим пояснения к этой программе.

В разделе **domains** используется стандартный тип данных **symbol** – строка символов, а также типы данных, определенные пользователем: **being**, **what** – пользовательские варианты стандартного типа **symbol**; **err(being)** – пользовательский структурированный тип (так называемый *функтор*, базирующийся на уже определенном типе **being**), соответствующий функции *ош*(*x*) из примера 3.2. Обратите внимание, что пользовательский тип **what** может трактоваться либо как **symbol**, либо как **err(being)** – это одна из особенностей языка Prolog.

В разделе **predicates** описаны предикаты, определяемые пользователем. Соответствие обозначениям примера 3.2:

человек(*x*)	**person(being)**
совершает(*x*,*y*)	**makes(being,what)**
ошибка(*y*)	**error(what) .**

Обратите внимание на синтаксис записи фактов и правил в разделе **clauses**.

Цель может быть задана в программе в разделе **goal** или введена непосредственно с клавиатуры в режиме диалога. Введя цель **makes(X,Y)**, получим

```
X=Иван, Y=err("Иван")
X=Таня Y=err("Таня")
2 solutions .
```

Процедурная интерпретация клауз Хорна заключается в том, что каждое целевое предложение рассматривается как вызов соответствующей логической программы или подпрограммы. В отличие от обычных программ, в которых логика управления, как правило, тесно связана выбором той или иной ветви реализации алгоритма (программы), логические программы, как правило, не управляют порядком выбора варианта правила или подцели при обработке некоторого целевого утверждения. Данные программ на языке клауз Хорна могут быть представлены либо посредством отношений, либо посредством термов. Списочные структуры это один из примеров возможностей представления данных на языке Prolog. Рассмотрим возможность использования функторов для представления древовидных структур данных.

Пример 3.5. На рис. 3.2 изображено перевернутое дерево, листьями которого являются вершины *a*,*b* и *c*. Программа подсчета числа листьев дерева:

```
domains
        ttree=tree(ttree,ttree); l(symbol)
predicates
        is_leaf(ttree)
        leaves(ttree,integer)
clauses
        is_leaf(l(a)).
        is_leaf(l(b)).
        is_leaf(l(c)).
        leaves(X,1):-is_leaf(X).
        leaves(tree(X,Y),W):-
        leaves(X,U),leaves(Y,V),U+V=W.
```

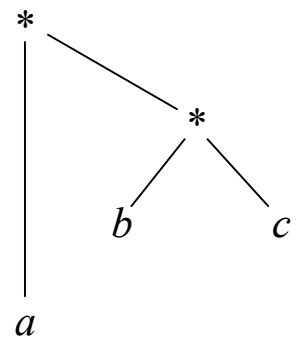

Рис. 3.2.

В этой программе используется стандартный тип данных **integer.** Обратите внимание, что для встроенного предиката + (суммирование) используется инфиксная нотация.

Вводим цель

leaves(tree(l(a),tree(l(b),l(c))),X)

и получаем ответ

X=3
1 solution .

Важная особенность логических программ – это возможность выполнения одной и той же процедуры при различном распределении ролей входов и выходов между параметрами процедуры. Например, стандартный предикат PDC Prolog'a **char_int(char,integer)**, устанавливающий соответствие между символом кода ASCII (**char**) и номером байта **(integer)**, реализует следующие варианты параметров: (*i,i*), (*i,o*), (*o,i*), где *i* означает, что предикат воспринимает параметр как входной (*input*), а *o* означает, что предикат воспринимает параметр как выходной (*output*).

Кроме того, логические программы характеризуются недетерминизмом первого и второго рода. *Недетерминизм первого рода* связан с тем, что с данным процедурным вызовом могут быть согласованы несколько вариантов процедур и имеется свобода выбора порядка их выполнения. *Недетерминизм второго рода* связан с тем, что в теле вызываемой процедуры есть более одной подпроцедуры и, следовательно, также возникает свобода выбора.

Основной принцип, используемый Prolog-интерпретатором в отношении процедур для разрешения недетерминизма первого и второго рода, заключается в том, что порядок исполнения таких процедур совпадает с порядком их записи в тексте Prolog-программы. Поскольку порядок записи зависит от составителя программы, этот фактор следует учитывать при составлении логических программ.

Что касается недетерминизма первого рода, то число просматриваемых вариантов может быть ограничено с помощью так называемого "зеленого" отсечения или "красного" отсечения. Разработчики Prolog-интерпретатора предусмотрели встроенный *предикат отсечения* (!).

Пример 3.6. Программа

p(X):-*c*(X).

p(X):-*a*(X),!,*b*(X).

имеет декларативный смысл $p(X)=a(X)\&b(X) \vee c(X)$.

Если убрать отсечение

p(X):-*c*(X).

p(X):-*a*(X),*b*(X). ,

то декларативный смысл программы не изменится. Следовательно, отсечение в данном примере – “зеленое”.

Пример 3.7. В предыдущем примере поменяем местами предикаты:

p(X):-*a*(X),!,*b*(X).

p(X):-*c*(X).

Декларативный смысл изменился: $p(X)=a(X)\&b(X) \vee \neg a(X)\&c(X)$. Следовательно, отсечение в данном примере – “красное ”.

При выборе порядка записи подпроцедур в теле процедуры следует учитывать распределение входов и выходов процедур. Более эффективным,

как правило, является порядок, когда сначала выполняются процедуры, имеющие большее отношение числа входных параметров к числу выходных.

Отрицание как неудача. Язык клауз Хорна, на котором основан Prolog, дополненный отрицанием **not**, обладает значительно большими возможностями. Однако, предикат **not** в языке Prolog не полностью соответствует оператору отрицания в стандартной логике. Например, в стандартной логике из $P \leftarrow \neg P$ следует $P \leftarrow$. Если же мы запустим программу

```
predicates
        p
clauses
        p:- not(p)
```

с внешней целью **p**, то получим зацикливание. Определение отрицания связано с отсечением:

```
not_p(X):- p(X),!,fail.
not_p(X).
```

Если-и-только-если **определения.** Prolog базируется на клаузах Хорна и, следовательно, использует **если**-определения. Однако, полная **если-и-только-если** форма необходима для получения ответов на вопросы, содержащие кванторы общности и отрицания.

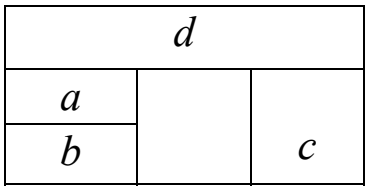

Рис. 3.3.

Пример 3.8. Рассмотрим сцену на рис.3.3. На множестве блоков $\{a,b,c,d\}$ необходимо определить отношение *свободен*(x), описывающее свободные блоки, через отношение *на*(x,y), означающее, что блок x расположен на блоке y. Эту сцену можно описать на языке стандартной логики, используя эквивалентность:

$\forall y[\textit{свободен}(y) \leftrightarrow \neg\exists x\ \textit{на}(x,y)]$,

$\forall x \forall y[\textit{на}(x,y) \leftrightarrow [x=a \& y=b] \vee [x=d \& y=a] \vee [x=d \& y=c]]$.

В соответствии с приведенной формулировкой, свободны все блоки, кроме a, b и c, потому что множество, из которого принимают значения x и y, явно не определено.

Переведем вышеприведенные определения в клаузальную форму. Для отношения *свободен* можно ограничиться **если-то-**определением:

$\forall y\ [\textit{свободен}(y) \leftarrow \neg\exists x\ \textit{на}(x,y)]$.

$\textit{свободен}(y) \leftarrow \neg \textit{на}(x,y)$.

Для отношения *на* можно ограничиться **только-если-**определением:

$$\forall x \forall y [на(x,y) \rightarrow [x=a \& y=b] \vee [x=d \& y=a] \vee [x=d \& y=c]]$$

или

$$\forall x \forall y \, [\neg на(x,y) \leftarrow [x \neq a \vee y \neq b] \, \& \, [x \neq d \vee y \neq a] \, \& \, [x \neq d \vee y \neq c]] \; . \quad (3.28)$$

Заметим, что выражение

$$[x \neq a \vee y \neq b] \, \& \, [x \neq d \vee y \neq a] \, \& \, [x \neq d \vee y \neq c]]$$

эквивалентно дизъюнкции 2^3=8 конъюнкций

$$x \neq a \, \& \, x \neq d \, \& \, x \neq d \vee x \neq a \, \& \, x \neq d \, \& \, y \neq c \vee ...$$

$$...\vee y \neq b \, \& \, y \neq a \, \& \, y \neq c \; .$$

Тогда (3.28) можно с учетом правила (3.17) переписать в виде 8 клауз:

$$\neg на(x,y) \leftarrow x \neq a, x \neq d \; .$$

$$\neg на(x,y) \leftarrow x \neq a, x \neq d, y \neq c \; .$$

...

$$\neg на(x,y) \leftarrow y \neq b, y \neq a, y \neq c \; .$$

Полезной оказывается только последняя клауза. Докажем, что блок *d* свободен:

$$\leftarrow свободен(d) \mid свободен(y) \leftarrow \neg на(x,y)$$

$$\text{---------------------------------------} \; y=d \; ;$$

$$\leftarrow \neg на(x,d)$$

$$\leftarrow \neg на(x,d) \mid \neg на(x,y) \leftarrow y \neq b, y \neq a, y \neq c$$

$$\text{---------------------------------------} \quad y=d \; .$$

$$\leftarrow d \neq b, d \neq a, d \neq c$$

Далее потребуются утверждения $d \neq b \leftarrow$, $d \neq a \leftarrow$ и $d \neq c \leftarrow$:

$$\leftarrow d \neq b, d \neq a, d \neq c \mid d \neq b \leftarrow$$

$$\text{---------------------} \quad ;$$

$$\leftarrow d \neq a, d \neq c$$

$$\leftarrow d \neq a, d \neq c \mid d \neq a \leftarrow$$

$$\text{---------------------} \quad ;$$

$$\leftarrow d \neq c$$

$$\leftarrow d \neq c \mid d \neq c \leftarrow$$

$$\text{---------} \quad .$$

□

На языке Prolog задача, рассмотренная выше, может быть решена, если определить предикат *свободен* (**is_free**) как отрицание предиката *покрыт* (**cov**):

```
predicates
        on(symbol,symbol)
```

```
        is_free(symbol)
        block(symbol)
        cov(symbol)
    clauses
        on(a,b).
        on(d,a).
        on(d,c).
        on(e,f).
        block(X):-on(X,_).
        block(X):-on(_,X).
        cov(Y):- block(X),block(Y),X<>Y, on(X,Y).
        is_free(X):-block(X),not(cov(X)).
```

Введя цель **is_free(X)**, получим

```
X=d
X=d
X=e
3 solutions .
```

3.5. Другие варианты логики

Модальная логика

Модальная логика [13] изучает структуру и законы построения рассуждений, содержащих так называемые модальности, выражаемые в естественном языке словами *“необходимо”,* “возможно”, “мочь” и т.д. Наряду с пропозициональными связками классического исчисления высказываний (¬, ∨, &), в модальной логике для выражения модальностей используются *модальные операторы*:

- *оператор необходимости*, который будем обозначать символом ■ (читается: “необходимо, что …”);
- *оператор возможности*, который будем обозначать символом ◆ (читается: “возможно, что …”).

Логическое значение сложного высказывания, образованного с помощью модального оператора, *чувствительно к смыслу* и, поэтому, не определяется однозначно *логическим значением* того высказывания, к которому применяется

модальный оператор. Например, мы обнаружили в стручке 10 горошин. Тогда высказывание: *В данном стручке 10 горошин* истинно, но высказывание: *Необходимо, что в данном стручке 10 горошин* ложно. Вместе с тем, высказывания: *Данная горошина содержит белок* и *Необходимо, что данная горошина содержит белок* – оба истинны.

Некоторые основные законы модальной логики известны еще с античных времен:

$\blacksquare A \rightarrow A$ ("Если необходимо, что *A*, то *A*"); (3.29)

$A \rightarrow \blacklozenge A$ ("Если *A,* то возможно, что *A*, "); (3.30)

$\blacksquare A \leftrightarrow \neg \blacklozenge \neg A$ ("Необходимо, что *A*, тогда и только тогда, когда невозможно, что не-*A*"); (3.31)

$\blacklozenge A \leftrightarrow \neg \blacksquare \neg A$ ("Возможно, что *A*, тогда и только тогда, когда не необходимо, что не-*A* "). (3.32)

В естественном языке слова "необходимо" и "возможно" имеют несколько смысловых вариантов, которые порождают соответствующие варианты модальной логики.

- *Логика алетических модальностей*. Эта логика имеет два варианта:

1. "Необходимо" означает объективную значимость содержания высказывания. В этом случае, если "необходимо" относится ко всему высказыванию, то это означает, что высказывание формулирует объективно необходимый закон для некоторой предметной области.

"Возможно" в этом случае указывает на объективную возможность того, о чем говорится в высказывании.

2. "Необходимо" понимается как "доказуемо", а "возможно" как "неопровержимо" в некоторой научной теории.

- *Деонтическая логика.*

"Необходимо" означает "должно", "обязательно", а "возможно" означает "допустимо" (в нормативном смысле). Таким образом, в рассматриваемом случае "необходимо" (соответственно, "возможно") указывает на некоторое, например, моральное или юридическое долженствование (соответственно, некоторую, например, моральную или юридическую допустимость).

- *Логика временных модальностей*.

"Необходимо" употребляется в смысле "всегда", а "возможно" означает, что "иногда". Часто в этом случае содержание высказывания относится к будущему.

Нечеткие множества и нечеткая логика

Пусть M некоторое множество и рассмотрим какое-либо подмножество A этого множества. Для любого $x \in M$ имеем $x \in A$ или $x \notin A$. Можно ввести *характеристическую функцию* подмножества A:

$$\mu_A(x) = \begin{cases} 1, \text{если } x \in A; \\ 0, \text{если } x \notin A. \end{cases}$$

Тогда, если $M=\{a,b,c,d,e,f,g,h\}$ и $A=\{a,b,c,d\}$, имеем:

x	...	a	b	c	d	e	f	g	h
$\mu_A(x)$	…	1	1	1	1	0	0	0	0.

Для *универсального* множества M: $\forall x\ \mu_M(x)=1$.

Множества, для которых характеристическая функция принимает только значения 0 или 1 будем называть *четкими*. Классическая логика оперирует с четкими множествами. Однако, при описании сложных систем часто приходится использовать нечеткие понятия и рассуждения и классическая логика оказывается в таких случаях неадекватной.

Для формализации таких задач Заде разработал основы математического аппарата, опирающегося на понятия *нечетких множеств* и *нечеткой логики*. Для нечетких множеств характеристическая функция может принимать любые значения из интервала [0,1]. Например,

x	...	a	b	c	d	e	f	g	h
$\mu_A(x)$	…	1,0	0,9	0,8	0,7	0,5	0,2	0,1	0,0;

x	...	a	b	c	d	e	f	g	h
$\mu_B(x)$	…	0,1	0,3	0,8	0,9	0,5	0,1	0,0	0,0.

Для записи нечетких множеств будем использовать обозначение:

$A=\{(x, \mu_A(x)) \mid \forall x \in M\}$.

Будем также использовать запись $x \in_{\mu_A(x)} A$, характеризующую степень принадлежности элемента x множеству A. Например, $a \in_{1,0} A$, $b \in_{0,9} A$ и т. д.

Для нечетких множеств определены отношение включения ($\subset$) и равенства (=):

A$\subset$B тогда и только тогда, когда $\forall x \in M$: $\mu_A(x) \le \mu_B(x)$;
A=B тогда и только тогда, когда $\forall x \in M$: $\mu_A(x) = \mu_B(x)$.

Операции над нечеткими множествами

Дополнение A

$$\bar{A} = \{(x, 1- \mu_A(x)) \mid \forall x \in M\}.$$

Очевидно, что выполняется соотношение $\overline{\bar{A}} = A$.

Пересечение

$$A \cap B = \{(x, \min\{\mu_A(x), \mu_B(x)\}) \mid \forall x \in M\}.$$

Объединение

$$A \cup B = \{(x, \max\{\mu_A(x), \mu_B(x)\}) \mid \forall x \in M\}.$$

Очевидно, что для нечетких множеств выполняется соотношение $A \cup \bar{A} = M$, где M – универсальное множество.

Нечеткая логика. Можно считать, что введенная выше функция $\mu_A(x)$ характеризует "степень истинности" утверждений типа "$x \in A$ для $\forall x \in M$". Пусть A и B – нечеткие множества, тогда для истинности нечеткого высказывания "$x \in A$ и $x \in B$" вводится нечеткая конъюнкция:

$$a \,\&\, b = \min\{a,b\}, \text{ где } a=\mu_A(x),\ b=\mu_B(x).$$

Нечеткая дизъюнкция вводится соотношением:

$$a \vee b = \max\{a,b\}.$$

Нечеткое отрицание определяется формулой: $\bar{a} = 1 - a$.

Нечеткая импликация:

$$a \to b = \max\{1- a, b\}.$$

Для введенных нечетких логических операций выполняются все соотношения, аналогичные соотношениям 1.30 – 1.57.

Темпоральная логика. Темпоральная логика (логика времени*) используется для описания временных отношений между объектами (событиями) предметной области. Введем в рассмотрение множество $T=\{t_1, t_2, t_3\ldots\}$ моментов времени и множество $E=\{e_1, e_2, \ldots, e_n\}$ событий. Предполагаются естественные свойства времени: однонаправленность, линейность, непрерывность, бесконечность, гомогенность (однородность).

Ограничимся рассмотрением *точечных* событий, т.е. событий, которые существуют только в один единственный момент времени $t \in T$. На множестве точечных событий зададим временные отношения: r_0 – происходить одновременно, r_1 – быть раньше, r_2 – быть позже, $r_3^{n,L}$ – быть раньше на n единиц времени по шкале L, $r_4^L t$ – происходить в момент t по шкале L.

Используем в качестве аксиом множество продукций вида $\alpha, \beta \vdash \gamma$ (пример отношения γ выводится из примеров α и β).

$$(e_i\ r_3^{n,L}\ e_j) \vdash (e_i\ r_1\ e_j), \tag{3.33}$$

$$(e_i\ r_1\ e_j), (e_j\ r_1\ e_k) \vdash (e_i\ r_1\ e_k), \tag{3.34}$$

$$(e_i\ r_3^{n,L}\ e_j), (e_j\ r_3^{m,L}\ e_k) \vdash (e_i\ r_3^{n+m,L}\ e_k), \tag{3.35}$$

$$(e_i\ r_4^L\ t), (e_i\ r_3^{n,L}\ e_j) \vdash (e_j\ r_4^L\ (t \oplus n)), \tag{3.36}$$

где $\oplus$ – операция суммирования по шкале L.

Используем также множество правил вывода

$$\frac{\Phi_1, \Phi_2 \vdash \Phi_3}{\Phi_3}, \quad \frac{\Phi_2 \quad \Phi_1}{\Phi_1}, \quad \frac{\Phi_2}{\Phi_2}.$$

Рассмотрим описание ситуации на естественном языке: “29 апреля группа экскурсантов прибыла на вокзал, а затем заняла места в поезде Санкт-Петербург – Москва. Поезд отправился в 23 часа 55 мин. Поезд находился в пути 8 часов и прибыл в Москву”. Выделим точечные события: e_1 – “прибытие на вокзал”, e_2 – “посадка на поезд”, e_3 – “отправление поезда”, e_4 – “прибытие поезда”.

* Литвинцева Л.В., Поспелов Д.А. Пополнение знаний // Искусственный интеллект. — В 3-х кн. Кн. 2. Модели и методы. – М.: Радио и связь, 1990. – С. 76 – 81.

На языке темпоральной логики эта ситуация может быть описана формулой

$(e_1\ r_1\ e_2\ )\ \&\ (\ e_2\ r_1\ e_3\ )\ \&\ (\ e_3\ r_4^{L}\ t_1)\ \&\ \&\ (e_3\ r_3^{L,T1}\ e_4)$,

где t_1= 29 апреля 23 часа 55 мин, T_1 = 8 часов. Используя схемы аксиом и правила вывода получим новые факты:

$(e_3\ r_3^{L,T1}\ e_4) \vdash (e_3\ r_1\ e_4\ )$ (по схеме 3.33);

$(e_1\ r_1\ e_2\ ),\ (\ e_2\ r_1\ e_3\ ) \vdash (e_1\ r_1\ e_3\ )$ (по схеме 3.34);

$(e_1\ r_1\ e_3\ ),\ (\ e_3\ r_1\ e_4\ ) \vdash (e_1\ r_1\ e_4\ )$ (по схеме 3.34);

$(\ e_3\ r_4^{L}\ t_1),\ (\ e_3\ r_3^{L,T1}\ e_4) \vdash (\ e_4\ r_4^{L}\ (t_1 \oplus T1))$ (по схеме 3.36),

где $t_1 \oplus T_1$=30 апреля 7 часов 55 мин.

Дополнительные сведения вариантам логики и логическому программированию можно найти в [1, 2, 4, 5, 7, 9, 13].

Глава 4
ЭЛЕМЕНТЫ ТЕОРИИ АЛГОРИТМОВ

4.1. Понятие алгоритма

Интуитивное представление об алгоритме как о формальном предписании, которое определяет совокупность операций и порядок их выполнения для решения всех задач какого-либо типа, существует в математике с давних времен.

Термин "алгоритм" происходит от имени средневекового узбекского математика Аль-Хорезми, который еще в IX веке сформировал правила выполнения арифметических действий, которые мы изучаем в школе. Под алгоритмом обычно понимается точное предписание, определяющее процесс переработки исходных данных в требуемый результат.

При этом требуется:

- чтобы исходные данные были заданы в конкретном алфавите и могли принимать значения из некоторого множества, т.е. носили *массовый* характер;

- чтобы процесс переработки данных состоял из отдельных *дискретных* шагов и был *детерминированным*;
- чтобы были четко указаны условия остановки процесса, и что следует считать *результатом* процесса.

Таким образом, любой алгоритм характеризуется *массовостью, детерминированностью и результативностью*. Для решения любой массовой задачи требуется построить соответствующий алгоритм, поэтому важность понятия алгоритм трудно переоценить.

Математика накопила большое число алгоритмов для решения разнообразных задач. В то же время попытки решения ряда задач натолкнулись на трудности, которые не удалось преодолеть. Возникла необходимость доказать *существование алгоритма* (*алгоритмическую разрешимость*) или *принципиальную невозможность построения алгоритма* (*алгоритмическую неразрешимость*) для ряда важных задач.

Между тем, сформулированное выше понятие алгоритма нельзя считать точным определением, моделью алгоритма. Поэтому были предложены точные математические модели, уточняющие понятие алгоритма: машина Тьюринга, система рекурсивных функций Клини, нормальный алгоритм А.А.Маркова, схема Колмогорова-Успенского, лямбда-конверсии Черча, финитные комбинаторные процессы Поста.

А.Черч впервые обосновал положение о том, что все уточнения понятия алгоритма эквивалентны ("тезис Черча"), т.е. правильно отражают интуитивное представление об алгоритмах, сложившееся в математике.

С помощью этих моделей алгоритма доказана алгоритмическая неразрешимость ряда важных задач математики и вычислительной техники и, в частности, неразрешимость *проблемы останова* универсальной вычислительной машины, реализующей любой алгоритм. Например, из алгоритмической неразрешимости проблемы останова следует важный практический вывод: *невозможно создать универсальную отладочную программу для обнаружения возможности зацикливания отлаживаемой программы*.

Алгоритмически неразрешимой оказалась проблема выводимости в исчислении предикатов: для любых формул P и Q узнать, существует ли дедуктивная цепочка, ведущая от P к Q.

Однако для исчисления высказываний и одноместных предикатов проблема выводимости разрешима.

Алгоритмическая неразрешимость некоторой задачи означает, что не существует общего алгоритма, решающего любую задачу рассматриваемого класса, однако для отдельных подклассов алгоритмически неразрешимой задачи может существовать алгоритм. Не следует смешивать алгоритмическую неразрешимость какой-либо проблемы с положением, когда некоторая проблема еще не решена. Для нерешенных проблем остается надежда найти разрешающий алгоритм, тогда как для алгоритмически неразрешимых проблем любые попытки поиска алгоритма бесполезны.

Прикладная теория алгоритмов базируется на выводах теории алгоритмов об алгоритмической разрешимости тех или иных проблем, но занимается, главным образом, разработкой наиболее эффективных с точки зрения практики алгоритмов, способов их описания, преобразования и реализации на современных ЭВМ.

Алгоритм рассматривается как совокупность определенным образом связанных между собой операторов, представляющих элементарные операции, которые производятся над множеством подвергающихся переработке объектов. Способы реализации операторов предполагаются известными (как правило, операторы сами являются некоторыми стандартными алгоритмами). При решении конкретной задачи задаются также значения исходных данных и параметров, входящих в описание алгоритма.

Для описания алгоритмов используются различные методы, отличающиеся степенью детализации и формализации. Теоретическое описание обычно дается в повествовательно-формальном изложении, цель которого – обосновать без лишних подробностей процедуру, предлагаемую в качестве алгоритма. Для наглядного представления структуры алгоритмов широко применяются графические средства: графы, блок-схемы, сети. Формальное и полное описание алгоритмов осуществляется на специально разработанных алгоритмических языках (BASIC, FORTRAN, PASCAL, C и др.); такое описание содержит всю необходимую для реализации алгоритма информацию, но не связано

непосредственно со специфическими особенностями вычислительных машин.

4.2. Машина Тьюринга

Модель алгоритма, называемая машиной Тьюринга [16], состоит из бесконечной ленты (БЛ), разделенной на ячейки, и управляющей головки (УГ), которая перемещается по ленте и способна считывать символ в ячейке, против которой она находится, а также замещать обозреваемый символ новым (рис.4.1).

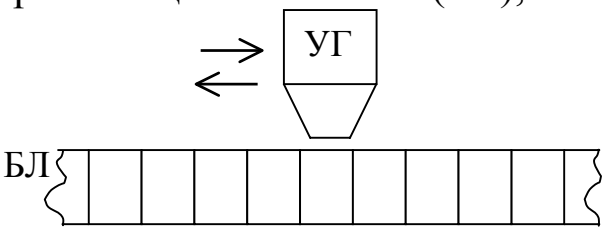

Рис.4.1.

В каждой ячейке может быть записан один символов из *ленточного алфавита A*. Головка может находиться в одном из внутренних состояний, принадлежащих конечному множеству (*алфавиту состояний*) *Q*. Работа машины происходит в дискретном времени в соответствии с программой, задаваемой набором команд вида

$$qa \to a^{+}Dq^{+}.$$

В зависимости от состояния головки $q \in Q$ и символа $a \in A$, против которого она стоит, головка записывает на ленте новый символ a^{+} (или оставляет старый), переходит в новое состояние q^{+} (или остается в старом) и передвигается: вправо (П), влево (Л) или остается в прежнем положении (Н).

Назовем *конфигурацией машины Тьюринга* K_t в момент t содержимое ее ленты, состояние головки $q \in Q$ и обозреваемый ею символ a. Пусть на ленте записана цепочка символов $\ldots\Lambda a_1 a_3 a_1 a_2 a_3 a_1 \Lambda\ldots$, слева и справа от которой свободные ячейки (содержат символ Λ), причем головка, находясь в состоянии q_i, обозревает символ a_2. Соответствующую конфигурацию будем записывать, помещая обозначение состояния головки перед обозреваемым символом: $a_1 a_3 a_1 q_i\, a_2 a_3 a_1$.

Конфигурация машины Тьюринга называется *заключительной*, если головка машины Тьюринга находится в состоянии останова q_0.

Работу машины Тьюринга можно описать как последовательную смену ее конфигураций, причем машина переходит от конфигурации K_t к конфигурации K_{t+1} в соответствии со своей программой. Любая начальная конфигурация K_0, которой соответствует состояние q_1, порождает последовательность конфигураций $K_0, K_1, K_2, \ldots, K_t, \ldots$.

Эта последовательность обрывается, если машина оказывается в заключительной конфигурации. В этом случае будем говорить, что машина Тьюринга *применима к конфигурации* K_0.

Если последовательность конфигураций $K_0, K_1, K_2, \ldots, K_t, \ldots$ никогда не обрывается, т.е. машина работает вечно (“зацикливается”), будем говорить, что машина Тьюринга *неприменима к конфигурации* K_0.

Для решения задачи исходные данные должны быть закодированы некоторым “естественным” образом символами некоторого алфавита A и записаны в виде слова X на ленте машины, причем головка в начальном состоянии $q_1 \in Q$ обозревает самый левый символ слова X, т.е. начальная конфигурация имеет вид q_1X. Результирующая конфигурация имеет вид $q_0 f(X)$.

В этом случае будем говорить, что машина Тьюринга *вычисляет словарную функцию* $f(\cdot)$, причем слово $f(X)$ есть значение этой функции для аргумента X. Числовые функции – это частный случай словарных, поскольку конкретный вид символов, которыми оперирует машина, несуществен, также как и тип данных: цифровых, алфавитно-цифровых и т.д.

Примеры машин Тьюринга

Рассмотрим несколько примеров специализированных машин Тьюринга с ленточным алфавитом $A=\{\Lambda,+,1\}$, алфавитом состояний $\{q_0,q_1,q_2, \ldots, q_n\}$ и алфавитом перемещений $D \in \{\text{П,Л,Н}\}$. Символ Λ

играет роль разделителя. Символы q_1, q_0 – соответственно начальное и заключительное состояние машины (останов).

Рассматриваемые машины выполняют арифметичекие операции над неотрицательными целыми числами, для представления которых используется унитарнй код. Число x представляется (x+1)-й единицей, причем отдельно записанная единица представляет нулевое значение x.

Таблица 4.1

a	q	
	q_1	q_2
Λ	-	1Hq_0
1	1Пq_2	-

Пример 4.1. Прибавление единицы. В таблице 4.1 приведен пример программы машины с алфавитом состояний $\{q_1,q_2,q_0\}$. Машина увеличивает значение числа на единицу.

Например, увеличение числа три на единицу машина осуществляет за два шага:

$$\ldots\Lambda\Lambda q_1 1111\Lambda\ldots \Rightarrow \ldots\Lambda q_2\Lambda 1111\Lambda\ldots \Rightarrow \ldots\Lambda\, q_0 11111\Lambda\ldots .$$

Для исходной конфигурации …Λq_1Λ11Λ… поведение машины не определено. Это означает, что рассматриваемая машина реализует *частичную словарную функцию*.

Таблица 4.2

a	q	
	q_1	q_2
Λ	ΛHq_0	1Hq_0
1	ΛПq_2	1Hq_0

Пример 4.2. Усеченное вычитание единицы. В таблице 4.2 приведен пример программы машины, выполняющей *усеченное вычитание* единицы. Вычитание единицы для неотрицательных целых чисел – частичная функция, поскольку значение 0-1 не определено. Введем усеченное вычитание ¬, доопределив обычное вычитание: $x\neg y = x\text{-}y$ для $x \geq y$ и $x\neg y = 0$ для $x < y$.

Например, усеченное вычитание единицы из четырех машина осуществляет за два шага:

$$\ldots\Lambda q_1 11111\Lambda\ldots \Rightarrow \ldots\Lambda\Lambda q_2 1111\Lambda\ldots \Rightarrow \ldots\Lambda\Lambda q_0 1111\Lambda\ldots .$$

Усеченное вычитание из нуля:

$$\ldots\Lambda q_1 1\Lambda\Lambda\ldots \Rightarrow \ldots\Lambda\Lambda q_2\Lambda\Lambda\ldots \Rightarrow \ldots\Lambda\Lambda q_0 1\Lambda\ldots .$$

Пример 4.3. Сложение. В таблице 4.3 приведен пример программы машины с алфавитом состояний $\{q_1,q_2,q_3,q_0\}$.

Таблица 4.3

a	q			
	q_1	q_2	q_3	q_4
Λ	Λ Пq_1	Λ Пq_1	1 Лq_2	Λ Нq_0
1	Λ Пq_3	1 Лq_2	1 Пq_3	Λ Пq_0
+	Λ Пq_4	+ Лq_2	+ Пq_3	Λ Нq_0

Например, сложение чисел 11+111 (один плюс два) осуществляется за два цикла. Начальное состояние головки q_1. Состояние q_3 поддерживает перемещение головки вправо с одновременным "перетаскиванием" единицы:

$\ldots\Lambda q_1 11{+}111\Lambda\Lambda\Lambda\ldots \Rightarrow \ldots\Lambda\Lambda q_3 1{+}111\Lambda\Lambda\Lambda\ldots \Rightarrow \ldots\Lambda\Lambda 1 q_3{+}111\Lambda\Lambda\Lambda\ldots \Rightarrow$

$\ldots\Lambda\Lambda 1{+}q_3 111\Lambda\Lambda\Lambda\ldots \Rightarrow \ldots\Lambda\Lambda 1{+}1 q_3 11\Lambda\Lambda\Lambda\ldots \Rightarrow \ldots\Lambda\Lambda 1{+}11 q_3 1\Lambda\Lambda\Lambda\ldots \Rightarrow$

$\ldots\Lambda\Lambda 1{+}111 q_3\Lambda\Lambda\Lambda\ldots \Rightarrow \ldots\Lambda\Lambda 1{+}11 q_2 11\Lambda\Lambda\ldots$.

Состояние q_2 соответствует перемещению головки влево:

$\Lambda\Lambda 1{+}11 q_2 11\Lambda\Lambda\ldots \Rightarrow \ldots\Lambda\Lambda 1{+}1 q_2 111\Lambda\Lambda\ldots \Rightarrow \ldots\Lambda\Lambda 1{+}q_2 1111\Lambda\Lambda\ldots \Rightarrow$

$\ldots\Lambda\Lambda 1 q_2{+}1111\Lambda\Lambda\ldots \Rightarrow \ldots\Lambda\Lambda q_2 1{+}1111\Lambda\Lambda\ldots \Rightarrow \ldots\Lambda q_2\Lambda 1{+}1111\Lambda\Lambda\ldots \Rightarrow$

$\ldots\Lambda\Lambda\, q_1 1{+}1111\Lambda\Lambda\ldots$.

Первый цикл закончен. Второй цикл:

$\ldots\Lambda\Lambda\, q_1 1{+}1111\Lambda\Lambda\ldots \Rightarrow \ldots\Lambda\Lambda\Lambda q_3{+}1111\Lambda\Lambda\ldots \Rightarrow \ldots\Lambda\Lambda\Lambda{+}q_3 1111\Lambda\Lambda\ldots \Rightarrow$

. . .

$\ldots\Lambda\Lambda\Lambda{+}\, 1111 q_3\Lambda\Lambda\ldots \Rightarrow \ldots\Lambda\Lambda\Lambda{+}\, 111 q_2 11\Lambda\ldots \Rightarrow \ldots\Lambda\Lambda\Lambda{+}\, 11 q_2 111\Lambda\ldots \Rightarrow$

. . .

$\ldots\Lambda\Lambda q_2\Lambda{+}\, 11111\Lambda\ldots \Rightarrow \ldots\Lambda\Lambda\Lambda q_1{+}\, 11111\Lambda\ldots$.

Второй цикл закончен. Машина удаляет лишнюю единицу

$\ldots\Lambda\Lambda\Lambda q_1{+}\, 11111\Lambda\ldots \Rightarrow \ldots\Lambda\Lambda\Lambda\Lambda q_4 11111\Lambda \ldots \Rightarrow \ldots\Lambda\Lambda\Lambda\Lambda q_0 1111\Lambda\ldots$

и, достигнув состояния q_0, останавливается. Конфигурация $\ldots\Lambda\Lambda\Lambda\Lambda q_0 11111\Lambda\ldots$ дает решение задачи.

Пример 4.4. Усеченное вычитание. В таблице 4.4 приведен пример программы вычисления усеченной разности.

Таблица 4.4

a	q

	q_1	q_2	q_3	q_4	q_5	q_6
Λ	Λ Л q_2	—	Λ П q_4	—	Λ П q_0	1 Н q_0
1	1 П q_1	Λ Л q_3	1 Л q_3	Λ П q_1	1 Л q_5	Λ П q_6
–	– П q_1	1 Л q_5	– Л q_3	Λ П q_6	—	—

Например, вычитание 1111–111 (три минус два) начинается тремя циклами, в результате каждого из которых стираются крайние единицы содержимого ленты. Начальное состояние q_1 поддерживает перемещение головки вправо до разделителя Λ:

$\ldots\Lambda q_1 1111{-}111\Lambda\ldots \Rightarrow \ \ldots \quad \ldots\Lambda 1111{-}111 q_1\Lambda\ldots$.

Затем в состоянии q_2 стирается самая правая единица и в состоянии q_3 происходит перемещение головки влево до разделителя Λ:

$\ldots\Lambda 1111{-}111 q_1\Lambda\ldots \Rightarrow \ldots\Lambda 1111{-}11 q_2\Lambda\ldots \Rightarrow \ldots\Lambda 1111{-}1 q_3 1\Lambda\ldots \Rightarrow$

$\ldots\Lambda 1111{-} q_3 11\Lambda\ldots \Rightarrow \ldots\ldots\Lambda q_3\Lambda 1111{-}11\Lambda\ldots$.

Затем в состоянии q_4 стирается самая левая единица и начинается второй цикл: в состоянии q_1 повторяется перемещение головки вправо до разделителя Λ и т.д.:

$\ldots q_3\Lambda 1111{-}11\Lambda\ldots \Rightarrow \ldots\Lambda q_4 1111{-}11\Lambda\ldots \Rightarrow \ldots\Lambda q_1 111{-}11\Lambda\ldots \Rightarrow \ldots$.

Четвертый и последний цикл отличается тем, что, обозревая символ "–" в состоянии q_2, головка заменяет его на "1" и переходит в состояние q_5, обеспечивающее корректное завершение программы:

$\ldots\Lambda q_1 1{-}\Lambda\ldots \Rightarrow \ldots\Lambda 1 q_1{-}\Lambda\ldots \Rightarrow \ldots\Lambda 1{-} q_1\Lambda\ldots \Rightarrow \ldots\Lambda 1 q_2{-}\Lambda\ldots \Rightarrow$

$\ldots\Lambda\, q_5 11\Lambda\ldots \Rightarrow \ldots q_5\Lambda 11\Lambda\ldots \Rightarrow \ldots q_0 11\Lambda\ldots$.

Состояние q_6 предусмотрено для случая, когда вычитаемое больше уменьшаемого. В этом состоянии головка очищает ленту: перемещается вправо и стирает все единицы. Встретив разделитель Λ в состоянии q_6, головка заменяет его единицей, остается на месте и переходит в заключительное состояние q_0.

Приемы программирования машины Тьюринга

Цель настоящего раздела – продемонстрировать универсальные вычислительные возможности машин Тьюринга. Для этого покажем,

как можно реализовать на этих машинах основные программные структуры:

- суперпозицию программ;
- композицию программ;
- ветвление и циклические структуры.

Суперпозиция программ. Предположим, что машины M_1 и M_2 вычисляют соответственно функции $f_1(X)$ и $f_2(X)$ в одном и том же алфавите A. Построим машину $M=M_1 \circ M_2$ для вычисления суперпозиции $f_2(f_1(X))$ (рис.4.2). Обозначения состояний машин M_1 и M_2 должны различаться за исключением того, что заключительное состояние M_1 имеет то же обозначение, что и начальное состояние M_2. В таком случае программа машины M получается простым объединением программ M_1 и M_2. Работу машины $M=M_1 \circ M_2$ можно представить схемой

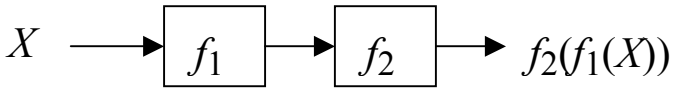

Рис.4.2.

$$X \underset{M_1}{\longrightarrow} f_1(X) \underset{M_2}{\longrightarrow} f_2(f_1(X)).$$

Композиция программ. Пусть машины M_1 и M_2 вычисляют функции $f_1(X)$ и $f_2(X)$ соответственно. Построим машину $M= M_1 * M_2$ для вычисления композиции $f_1(X)*f_2(X)$, где * - символ, не встречающийся в алфавитах M_1 и M_2. Машина M использует "двухэтажную" ленту, на которую записываются пары символов вида $\begin{pmatrix} b \\ a \end{pmatrix}$. Слово X записывается нижними элементами пары вида $\begin{pmatrix} \Lambda \\ a \end{pmatrix}$, где Λ - пустой символ. После запуска M оно переписывается в верхний этаж и оказывается записанным символами вида $\begin{pmatrix} a \\ a \end{pmatrix}$. Далее M работает на нижнем этаже как M_1 и вычисляет $f_1(X)$, затем M работает на верхнем этаже как M_2 и вычисляет $f_2(X)$. В завершение M

приписывает $*f_2(X)$ в конце записи $f_1(X)$ на нижнем этаже и стирает $f_2(X)$ на верхнем этаже.

Ветвление программ. Пусть машины M_1 и M_2 вычисляют словарные функции $f_1(X)$ и $f_2(X)$ соответственно в одном и том же алфавите A. Введем символы $и$ (истина) и $л$ (ложь), причем $и \notin A$ и $л \notin A$. Построим машину $M = M_1 \underline{\vee} M_2$, которая преобразует слово $\sigma * X$, где $\sigma \in \{и, л\}$, в слово $f_1(X)$, если $\sigma = и$, и в слово $f_2(X)$, если $\sigma = л$ (рис.4.3). Для этого, полагая что алфавиты состояний M_1 и M_2 различны, в основу M положим объединение их программы. Введем для M начальное состояние q_1 и, полагая, что $q_{1,1}$ и $q_{2,1}$ – начальные состояния M_1 и M_2 соответственно, дополним объединение программ M_1 и M_2 командами

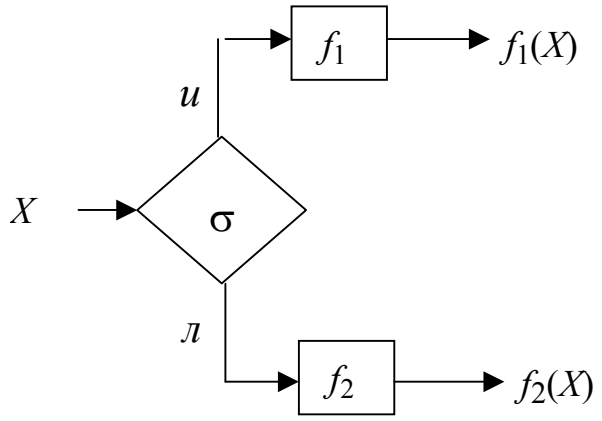

Рис.4.3.

$$
\begin{aligned}
иq_1 &\to \Lambda\Pi q_{1,1},\\
лq_1 &\to \Lambda\Pi q_{2,1},\\
*q_{1,1} &\to \Lambda\Pi q_{1,1},\\
*q_{2,1} &\to \Lambda\Pi q_{2,1}.
\end{aligned}
$$

Заключительные состояния $q_{1,0}$ и $q_{2,0}$ машин M_1 и M_2 обозначим как заключительное состояние q_0 машины $M = M_1 \underline{\vee} M_2$.

Машину $M = M_1 \underline{\vee} M_2$ можно представить схемой на рис.4.4.

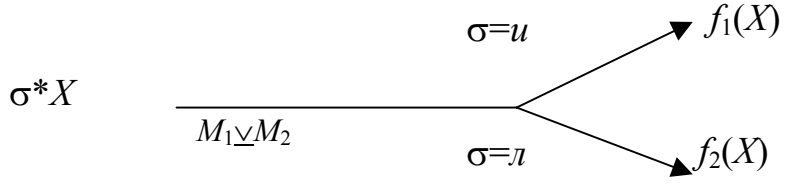

Рис.4.4.

Циклическая программа. Рассмотрим циклическую программу на рис.4.5. Если значение предиката $P(X)$ *истина* ($и$), то программа выдает значение $f_1(X)$. Если же значение предиката $P(X)$ ($л$), то программа вычисляет значение $X^{(1)} = f_2(X)$ и, если $P(X^{(1)}) = и$, то программа выдает значение $f_1(X^{(1)})$, в противном случае вычисляет значение $X^{(2)} = f_2(X^{(1)})$ и т.д.

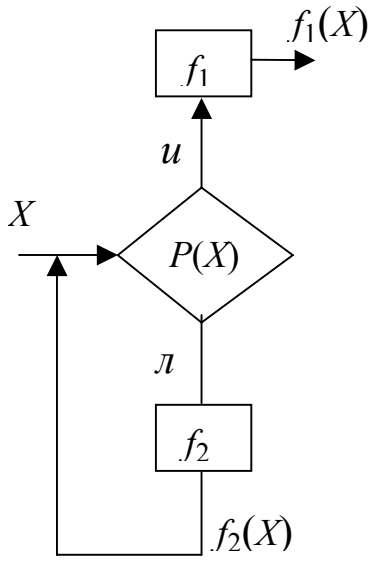

Рис. 4.5.

Приведем схему машины

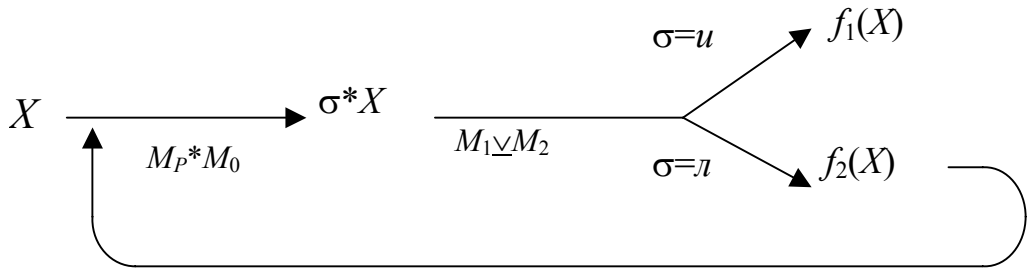

Рис. 4.6.

Тьюринга, реализующей циклическую программу (рис.4.6). На этой схеме машина M_P вычисляет значение σ предиката $P(X)$, машина M_0 передает X без изменения, поэтому композиция $M_P * M_0$ формирует $\sigma * X$. Машины M_1 и M_2 вычисляют $f_1(X)$ и $f_2(X)$ соответственно. Машина $M_1 \underline{\vee} M_2$ обеспечивает ветвление.

4.3. Элементы теории рекурсивных функций

Рекурсия - способ определения функций, являющийся одним из основных объектом изучения в теории алгоритмов. Смысл рекурсии

в том, что значение функции f в точке x определяется через значения этой функции в "предшествующих" точках.

Оператор примитивной рекурсии

Оператор примитивной рекурсии определяет (n+1)-местную функцию $f(x_1,...,x_n;y)$ через n-местную функцию $g(x_1,...,x_n)$ и (n+2)-местную функцию $h(x_1,...,x_n; y; z)$ следующим образом:

$f(x_1,...,x_n;0)=g(x_1,...,x_n)$,

$f(x_1,...,x_n; 1)=h(x_1,...,x_n; 0; f(x_1,...,x_n; 0))$,

$f(x_1,...,x_n; 2)=h(x_1,...,x_n; 1; f(x_1,...,x_n; 1))$,

...

$f(x_1,...,x_n; m+1)=h(x_1,...,x_n; m; f(x_1,...,x_n; m))$.

Пример 4.5. Пусть переменные $x_1,...,x_n$ отсутствуют, причем рекурсивная схема задана соотношениями

$$\begin{cases} g=0, \\ h(y; z)=2*y+z. \end{cases}$$

Тогда

$f(0)=0$,

$f(1)=2*0+0=0$,

$f(2)=2*1+0=2$,

$f(3)=2*2+2=6$,

$f(4)=2*3+6=12$,

$f(5)=2*4+12=20$,

...

$f(x+1)=2*x+f(x)$.

Функцию определенную данной рекурсивной схемой можно выразить аналитически:

$f(x+1)=2*x+f(x)=2*x+2*(x-1)+f(x-1)=...=$

$2*x+2*(x-1)+2*(x-2)+...+2*2+2*1=$

$2*[x+(x-1)+(x-2)+...+2+1]$.

Выражение в прямоугольных скобках [ ] представляет сумму чисел от 1 до x, равную по формуле арифметической прогрессии $(x+1)*x/2$, поэтому получаем $f(x+1)=(x+1)*x$, или $f(x)=x*(x-1)$.

Определение4.1. *Примитивно-рекурсивная функция* (ПР-функция) это одна из простейших функций:

- *следования* $S(x)=x+1=x'$,
- *константы ноль* $0(x)=0$,
- *тождества* $Im(x_1,...,x_n)=x_m$ $(1\le m\le n)$,
- либо функция, которая может быть получена из простейших функций с помощью применения конечного числа операторов *подстановки и примитивной рекурсии*.

Все ПР-функции всюду определенные. В предыдущем примере мы имели ПР-функцию $f(x)=x*(x-1)$. Ниже мы рассмотрим и другие примеры ПР-функций. В дальнейшем изложении и примерах мы предполагаем, что области определения и значения ПР-функций – множества неотрицательных целых чисел (что не уменьшает общности выводов).

Примеры ПР-функций

***Функция сложения*:**

$$f_c(x; 0)=x,$$
$$f_c(x; 1)=h(x; 0; f_c(x; 0))=S(f_c(x; 0)),$$
$$...$$
$$f_c(x; m+1)=S(f_c(x; m)).$$

***Функция умножения*:**

$$f_y(x; 0)=0,$$
$$f_y(x; 1)=h(x; 0; f_y(x; 0))=f_c(x; f_y(x; 0)),$$
$$...$$
$$f_y(x; m+1)=f_c(x; f_y(x; m)).$$

***Функции сигнум и антисигнум*.** Поскольку мы не используем отрицательных чисел, определим функцию сигнум следующим образом

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

и антисигнум как

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

Обе эти функции рекурсивны и могут быть заданы схемами:

$$\begin{cases} \text{sgn}\,(0) = 0, \\ \text{sgn}\,(x+1) = 1; \end{cases}$$

$$\begin{cases} \overline{\text{sgn}}\,(0) = 1, \\ \overline{\text{sgn}}\,(1) = 0. \end{cases}$$

Усеченная разность. Разность $x - y$ не определена для неотрицательных целых x и y, т.е. является частичной функцией. Введем у*сеченную разность* $x \dot{-} y$, доопределив равную $x - y$ следующим образом

$$x \dot{-} y = \begin{cases} x - y, \text{ если } x - y \geq 0, \\ 0 \text{ в противном случае.} \end{cases}$$

Усеченная разность определяется рекурсивной схемой

$$\begin{cases} x \dot{-} y = 0, \\ x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1. \end{cases}$$

Модуль разности $|x - y|$ можно выразить через усеченную разность:

$$|x - y| = (x \dot{-} y) + (y \dot{-} x).$$

Оператор минимизации

Примитивной рекурсии недостаточно, чтобы задать любую вычислимую функцию. В общем случае, требуется еще *оператор минимизации*

$$\mu_t[f(x_1,...,x_{n-1}; t)=x_n],$$

который обозначает вычисление минимального значения t, при котором выполняется равенство

$$f(x_1,...,x_{n-1}; t)=x_n.$$

Могут быть следующие случаи, когда значение оператора минимизации не определено:

1) значение $f(x_1,...,x_{n-1}; 0)$ не определено;

2) значения $f(x_1,...,x_{n-1}; y)$ при $y=0,1,2,...,k$ определены, но

отличны от x_n, а при $y=k+1$ не определено;

3) значения $f(x_1,...,x_{n-1}; y)$ при всех $y=0,1,2,3,...$ определены,

но отличны от x_n.

Определение 4.2. Функция называется *частично-рекурсивной* (ЧР), если она может быть получена из $S(x)$, $0(x)$ и Im с помощью применения конечного числа операторов подстановки, примитивной рекурсии и минимизации.

Определение 4.3. ЧР-функция называется *общерекурсивной* (ОР), если она всюду определена.

Примеры ЧР-функций

***Целая часть* [*x/y*]**. Функция x/y является частичной, поскольку для $y=0$ не определена. Введем в рассмотрение функцию $[x/y]$, полученную доопределением x/y так, что $[x/0]=x$. Можно показать, что $[x/y]= \mu_z[(x \dot{-} z)((x + 1) \dot{-} y(z + 1))=0]$, следовательно, $[x/y]$ – ОР-функция [14].

***Функция* [*x* mod *y*].** Функция $x \bmod y$ (остаток от деления x на y) является частичной, поскольку для $y=0$ не определена. Введем в рассмотрение функцию $[x \bmod y]$, полученную доопределением $x \bmod y$ так, что $[x \bmod 0]=x$.

Можно показать, что $[x \bmod y]= x \dot{-} y[x/y]$, следовательно, $[x \bmod y]$ – ОР-функция [14].

4.4. Эквивалентность алгоритмических систем

Как было сказано в начале данной главы, А.Черч впервые обосновал положение о том, что все уточнения понятия алгоритма эквивалентны. Продемонстрируем подход к доказательству эквивалентности вычислимости по Тьюрингу, и класса функций, определяемых ЧР-схемой.

Теорема 4.1. *Класс функций, вычислимых по Тьюрингу, эквивалентен классу ЧР-функций.*

Наша цель – дать читателю схематичное представление о подходе к доказательству сформулированной теоремы, поэтому дальнейшее изложение не претендует на полноту.

Общий план изложения заключается в доказательстве двух утверждений:

- *о возможности вычисления любой ЧР-функции на машине Тьюринга*;
- *о возможности воспроизведения любого вычисления на машине Тьюринга с помощью некоторой ЧР-схемы.*

Порядок доказательства утверждений несущественен.

Утверждение 4.1. *Любая ЧР-функция может быть вычислена на машине Тьюринга.*

Для доказательства этого утверждения докажем, что операции суперпозиции, рекурсии, минимизации, а также функции следования, константы ноль и тождества, могут быть запрограммированы на машине Тьюринга.

Программирование операции суперпозиции. Рассмотрим эту операцию на примере

$$h(x_1, \ldots, x_m)=g(f_1(x_1, \ldots, x_m), \ldots, f_n(x_1, \ldots, x_m)).$$

Если функции g, $f_1, \ldots, f_n$ вычисляются машинами $M_g, M_1, \ldots, M_n$, то функция h может быть вычислена машиной $M_g \circ (M_1 * \ldots * M_n)$.

Программирование операции рекурсии. Рассмотрим эту операцию на упрощенном примере:

$$\begin{cases} f(0)=a, \\ f(x+1)=h(x, f(x)). \end{cases}$$

Рекурсивная схема реализуется машиной Тьюринга, а именно циклической программой на рис. 4.7.

На этом рисунке:

- машина A перерабатывает слово x в слово $x*0*a$;
- машина B получает слово $x_1*x_2*x_3$ и выдает $\sigma=и$, если $x_1=0$, и $\sigma=л$ в противном случае;
- машина M_0 оставляет входное слово без изменения;
- машина M_1 по слову $x_1*x_2*x_3$ выдает $x_1 \neg 1$;
- машина M_2 по слову $x_1*x_2*x_3$ выдает $x_2 + 1$;

- машина M_3 по слову $x_1*x_2*x_3$ выдает x_3;
- машина M_4 по слову $x_1*x_2*x_3$ выдает $h(x_2, x_3)$.

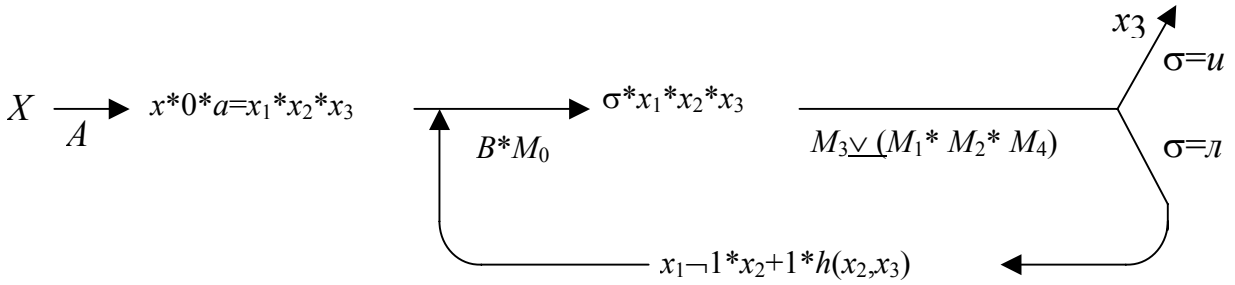

Рис. 4.7.

Программирование операции минимизации. Рассмотрим эту операцию на упрощенном примере:

$$f(x)= \mu_y\,(h(x, y)=0).$$

Операция минимизации реализуется на машине Тьюринга циклической программой на рис. 4.8.

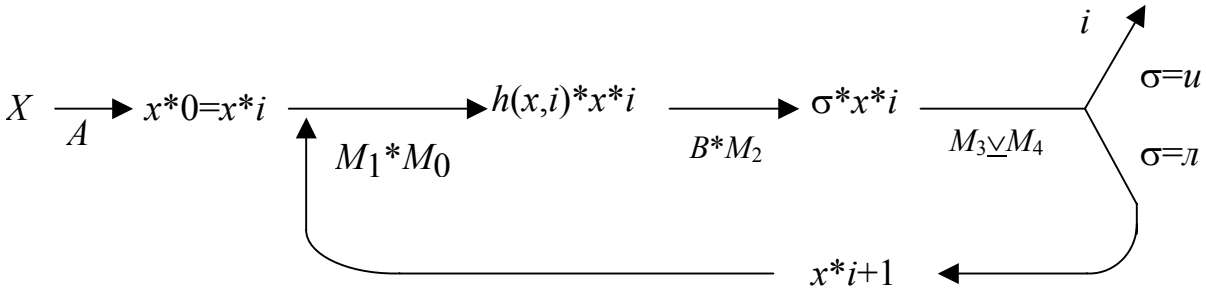

Рис. 4.8.

На этом рисунке

- машина A перерабатывает слово x в слово $x*0$;
- машина B получает слово $x_1*x_2*x_3$ и выдает $\sigma=u$, если $x_1=0$, и $\sigma=л$ в противном случае;
- машина M_0 оставляет входное слово без изменения;
- машина M_1 по слову x_1*x_2 вычисляет $h(x_1,x_2)$;
- машина M_2 преобразует слово $x_1*x_2*x_3$ в слово x_2*x_3;
- машина M_3 по слову x_1*i выдает i;
- машина M_4 преобразует слово x_1*i в слово x_1*i+1.

Программирование функций следования, константы ноль и тождества. Функции следования $S(x)=x+1=x'$ и константы ноль $0(x)=0$ реализуются достаточно просто [14]. Функция тождества $I_m^n(x_1,...,x_n)=x_m$ реализуется машиной Тьюринга, способной вести счет до n, для этого головка машины должна иметь достаточное число состояний, превышающее n.

Утверждение 4.2. *Любая функция, вычислимая на машине Тьюринга, частично-рекурсивна.*

Докажем, что всякая функция, вычислимая по Тьюрингу, может быть смоделирована ЧР-схемой.

Для этого будем использовать прием, называемый *арифметизацией* машины Тьюринга. Арифметизация состоит в том, что нечисловые параметры (объекты, состояние головки) нумеруются натуральными числами, что позволяет выразить преобразования конфигураций машины Тьюринга через арифметические операции.

Рассмотрим машину Тьюринга с ленточным алфавитом $A=\{a_0, a_1, \ldots, a_{k-1}\}$ и состояниями $Q=\{q_0, q_1, \ldots, q_{r-1}\}$. Будем считать, что $a_0=\Lambda$ (пробел). Букве a_s (s = 0, 1, …, k-1) сопоставим цифру i k-ичной системы счисления. Будем также считать, что q_1 – начальное состояние, q_0 – заключительное состояние. Состоянию q_j сопоставим число j.

Рассмотрим произвольную конфигурацию

$$\ldots\Lambda b_3 b_2 b_1 b_0 q_j a_s c_0 c_1 c_2 c_3 \Lambda\ldots . \qquad (4.1)$$

Содержимое ленты для рассматриваемой конфигурации можно представить тройкой чисел $<s,m,n>$, где

s – номер обозреваемого символа a_s;

$m = \sum_{i=0}^{\infty} b_i k^i$, $n = \sum_{i=0}^{\infty} c_i k^i$ – k-ичная запись чисел, представляющих содержимое ленты соответственно слева и справа от обозреваемого символа.

Числа m и n содержат конечное число цифр, отличных от нуля, поэтому арифметизированное представление содержимого бесконечной ленты заполнено нулями за пределами активной зоны.

Конфигурацию машины Тьюринга представим четверкой $<j,s,m,n>$, где j – k-ичная запись номера состояния, а $<s,m,n>$ - содержимое ленты.

Рассмотрим три варианта команд: с перемещением головки вправо, влево и с неподвижной головкой.

Выполнение команды $q_j a_s \to a_s^{+} \Pi q_j^{+}$ преобразует конфигурацию (4.1) к виду

$$\ldots \Lambda b_3 b_2 b_1 b_0 a_s^{+} q_j^{+} c_0 c_1 c_2 c_3 \Lambda \ldots . \qquad (4.2)$$

Значения j,s,m,n для конфигурации (2) обозначим через j',s',m',n'.
Введем обозначения:

s^{+} - номер замещающего символа a_s^{+};

j^{+} - номер нового состояния головки q_j^{+};

$[n \bmod k]$ – значение n по модулю k;

$[n/k]$ – результат деления n на k нацело.

Тогда можно записать:

$s'=[n \bmod k]$, поскольку головка обозревает младшую цифру n;

$m'=mk + s^{+}$, поскольку сдвиг m влево относительно головки равносилен умножению на k;

$n'=[n/k]$.

Таким образом, четверка $<j,s,m,n>$ преобразуется командой, перемещающей головку направо, к виду $< j^{+}, [n \bmod k], mk+s^{+}, [n/k]>$. Предполагаем здесь и далее, что все арифметические операции выполняются в k-ичной системе счисления.

Выполнение команды $q_j a_s \to a_s^{+} \Lambda q_j^{+}$ преобразует конфигурацию (4.1) к виду

$$\ldots \Lambda b_3 b_2 b_1 b_0 q_j^{+} a_s^{+} c_0 c_1 c_2 c_3 \Lambda \ldots . \qquad (4.3)$$

Для рассматриваемой команды можно записать:

$$s'=[n \bmod k],$$
$$m'=[m/k],$$
$$n'=nk + s^{+}.$$

Таким образом, четверка $<j,s,m,n>$ преобразуется командой, перемещающей головку влево, к виду

$$< j^{+},\ [n \bmod k],\ [m/k],\ nk + s^{+}. >.$$

Выполнение команды $q_j a_s \to a_s^{+} \mathrm{H} q_j^{+}$, не меняющей положение головки, не изменяет значения m' и n': $s' = s^{+}$, $m' = m$, $n' = n$. Эта команда преобразует конфигурацию (1) к виду $< j^{+}, s^{+}, m, n >$.

Введем числовую функцию $D(j,s)$, характеризующую перемещение головки для команды $q_j a_s \to a_s^{+} D\, q_j^{+}$:

$$D(j,s) = \begin{cases} 0, \text{ если } D = \Pi, \\ 1, \text{ если } D = \Л, \\ 2, \text{ если } D = \mathrm{H}. \end{cases}$$

Введем также функции $Q(j,s)=j^{+}$ и $A(j,s)=s^{+}$. Если функции $D(j,s)$, $Q(j,s)=j^{+}$ и $A(j,s)=s^{+}$ частичные, доопределим их равными нулю. Таким образом, функции $D(j,s)$, $Q(j,s)=j^{+}$ и $A(j,s)=s^{+}$ - рекурсивные.

Введем, наконец, в рассмотрение рекурсивные числовые функции, характеризующие перемещение головки:

$$\delta_{\Pi}(j,s) = \overline{\operatorname{sgn}}\ D(j,s),$$
$$\delta_{\Л}(j,s) = \overline{\operatorname{sgn}}\ \left| D(j,s) - 1 \right|,$$
$$\delta_{\mathrm{H}}(j,s) = \overline{\operatorname{sgn}}\ \left| D(j,s) - 2 \right|.$$

Функция $\delta_{\Pi}(j,s)$ принимает значение 1 только если $D(j,s)=0$, т.е. $D=\Pi$, в противном случае $\delta_{\Pi}(j,s)=0$. Функция $\delta_{\Л}(j,s)$ принимает значение 1 только если $D(j,s)=1$, т.е. $D=\Л$, в противном случае $\delta_{\Л}(j,s)=0$. Наконец, функция $\delta_{\mathrm{H}}(j,s)$ принимает значение 1 только если $D(j,s)=2$, т.е. $D=\mathrm{H}$, в противном случае $\delta_{\mathrm{H}}(j,s)=0$.

Теперь мы можем обобщить преобразование четверки $<j,s,m,n>$ командой $q_j a_s \to q_j^{+} a_s^{+} D$ в виде числовых функций

$$j' = j'(j,s,m,n) = Q(j,s);$$
$$s' = s'(j,s,m,n) = \delta_{\Pi}(j,s)[n \bmod k] + \delta_{\Л}(j,s)[m \bmod k] + \delta_{\mathrm{H}}(j,s)A(j,s);$$
$$m' = m'(j,s,m,n) = \delta_{\Pi}(j,s)(mk + A(j,s)) + \delta_{\Л}(j,s)[m/k] + \delta_{\mathrm{H}}(j,s)m;$$
$$n' = n'(j,s,m,n) = \delta_{\Pi}(j,s)[n/k] + \delta_{\Л}(j,s)(nk + A(j,s)) + \delta_{\mathrm{H}}(j,s)n.$$

Функции $j'(j,s,m,n)$, $s'(j,s,m,n)$, $m'(j,s,m,n)$, $n'(j,s,m,n)$ получены суперпозицией рекурсивных функций, поэтому они рекурсивны.

Теперь мы можем ввести в рассмотрение функции $\tilde{Q}(t,j,s,m,n)$, дающие номер состояния, в которое перейдет машина из начальной конфигурации $<j,s,m,n>$ через t тактов. Введем также функции $\tilde{S}(t,j,s,m,n)$, $\tilde{M}(t,j,s,m,n)$, $\tilde{N}(t,j,s,m,n)$, дающие номера остальных элементов четверки через t тактов. Введенные функции удовлетворяют рекурсивным соотношениям:

$$\tilde{Q}(0,j,s,m,n)=j,$$

$$\tilde{S}(0,j,s,m,n)=s,$$

$$\tilde{M}(0,j,s,m,n)=m,$$

$$\tilde{N}(0,j,s,m,n)=n,$$

$$\tilde{Q}(t+1,j,s,m,n)=Q(\tilde{Q}(t,j,s,m,n),\tilde{A}(t,j,s,m,n)),$$

$$\tilde{S}(t+1,j,s,m,n)=s'(\tilde{Q}(t,j,s,m,n),\tilde{S}(t,j,s,m,n),\tilde{M}(t,j,s,m,n),\tilde{N}(t,j,s,m,n)),$$

$$\tilde{M}(t+1,j,s,m,n)=m'(\tilde{Q}(t,j,s,m,n),\tilde{S}(t,j,s,m,n),\tilde{M}(t,j,s,m,n),\tilde{N}(t,j,s,m,n)),$$

$$\tilde{N}(t+1,j,s,m,n)=n'(\tilde{Q}(t,j,s,m,n),\tilde{S}(t,j,s,m,n),\tilde{M}(t,j,s,m,n),\tilde{N}(t,j,s,m,n)).$$

Выписанные формулы определяют схему совместной рекурсии функций $\tilde{Q}$, $\tilde{S}$, $\tilde{M}$, $\tilde{N}$. Совместная рекурсия равносильна простой рекурсии [9, 14]. Поскольку функции Q, s', m', n' рекурсивны, функции $\tilde{Q}$, $\tilde{S}$, $\tilde{M}$, $\tilde{N}$ также рекурсивны. Таким образом, получена система рекурсивных функций, позволяющих найти конфигурацию, в которую переходит машина Тьюринга из начальной конфигурации через t тактов. Остается только доказать частичную рекурсивность функции $f(x)$, вычисляемой машиной Тьюринга.

Пусть функция $f(x)$ вычисляется машиной Тьюринга с начальной конфигурацией $\ldots\Lambda q_1 a_s c_0 c_1 c_2 c_3 \Lambda\ldots$, где цепочка $a_s c_0 c_1 c_2 c_3$ представляет значение аргумента x. Для арифметизированной записи начальной конфигурации $<1, s(x), 0, n(x)>$ имеем $m(x)=0$, а номера обозреваемого символа $s(x)$ и $n(x)$ определяются значением аргумента x. Тогда можно записать выражение для номера состояния через t тактов

$$q(t,x)= \tilde{Q}(t, 1, s(x), 0, n(x)).$$

Аналогично определим функции, задающие остальные элементы четверки через t тактов:

$$s(t,x)= \tilde{S}(t, 1, s(x), 0, n(x)),$$

$$m(t,x)= \tilde{M}(t, 1, s(x), 0, n(x)),$$

$$n(t,x)= \tilde{N}(t, 1, s(x), 0, n(x)).$$

Функции $q(t,x)$, $s(t,x)$, $m(t,x)$, $n(t,x)$ рекурсивны, поскольку получены суперпозицией рекурсивных функций.

Машина останавливается, когда приходит в заключительное состояние q_0. Поэтому момент остановки $t_0(x)$ можно найти, если он определен, из условия, что q=0:

$$t_0(x)=\mu_t(q(t,x)).$$

Если значение $f(x)$ не определено, машина никогда не останавливается и $t_0(x)$ имеет неопределенное значение. Следовательно, функция $t_0(x)$ является частично-рекурсивной.

Если значение $t_0(x)$ определено, можно найти остальные элементы четверки для заключительной конфигурации:

$$s_0(x)=s(t_0(x),x),$$

$$m_0(x)=m(t_0(x),x),$$

$$n_0(x)=n(t_0(x),x).$$

Полагаем, что $m_0(x)=m(t_0(x),x)=0$. Тогда заключительная конфигурация может быть представлена четверкой $<0, s(f(x)), 0, n(f(x))>$. Рассмотрим операции преобразования этой конфигурации к символьному виду, а именно

$$\ldots\Lambda q_0 a_s c_{i_0} c_{i_1} c_{i_2} c_{i_3} \Lambda\ldots,$$

где $a_s c_{i_0} c_{i_1} c_{i_2} c_{i_3} \ldots$ - представление значения функции $f(x)$. Напомним, что все арифметические операции выполняются в k-ичной системе счисления. Поскольку в k-ичной записи $n(f(x))= \ldots i_3 i_2 i_1 i_0$, для получения соответствующих цифр $f(x)$, а именно $\ldots a_s c_{i_0} c_{i_1} c_{i_2} c_{i_3} \ldots$, необходимо взять запись $n(f(x))$ в обратном порядке. Очевидно, что

это последнее преобразование не выходит за рамки рекурсивной схемы.

Следовательно, функции, вычисляемые на машине Тьюринга, являются частично-рекурсивными. *Доказательство закончено.*

4.5. Универсальные машины Тьюринга и алгоритмическая разрешимость

Метод Геделя. Курт Гедель [3, 8, 9, 15] предложил числовое кодирование математических объектов. Рассмотрим один из возможных способов такого кодирования - нумерации машин Тьюринга.

Команды $qa \rightarrow a^{+}Dq^{+}$ машины будем кодировать цифрами восьмеричной системы счисления. Рассмотрим машину Тьюринга с алфавитом состояний $Q=\{q_0, q_1, \ldots, q_{r-1}\}$ и ленточным алфавитом $A=\{a_0, a_1, \ldots, a_{k-1}\}$, где $a_0 = \Lambda$ - символ разделителя. Для кодирования индексов используем двоичную систему счисления с алфавитом $\{0_2,1_2\}$, где 0_2 – двоичный ноль, 1_2 –двоичная единица. Поставим в соответствие символам Λ, *a*, *q*, , *u*, Л, П, Н восьмеричные цифры:

символ	. . .	Λ	*a*	*q*	0_2	1_2	Л	П	Н
цифра	. . .	0	1	2	3	4	5	6	7.

Например, команда $q_3a_5 \rightarrow a_4\Pi q_7$ будет закодирована как восьмеричное число, не содержащее нулей:

2441434143362444.

Роль разделителей между кодами q_i , a_j и D играют цифры 1, 2, 5, 6 и 7. Каждой команде машины Q_k поставим в соответствие ее геделевский номер (шифр), а затем расположим эти номера в порядке возрастания

$$Ш(Q_1)\ Ш(Q_2)\ \ldots\ Ш(Q_n).$$

Таким образом построенная последовательность (геделевский номер машины) z=Ш(Q_1)Ш(Q_2)…Ш(Q_n) обеспечивает однозначность кодирования и декодирования машин Тьюринга. Процедура

нумерации может быть выполнена некоторой машиной, которую обозначим через M_1. Очевидно, что машина M_1 может вычислить и свой собственный геделевский номер. Очевидно также, что существует машина M_2, которая для любого восьмеричного числа X выдаст один из двух ответов:

- "число X не является геделевским номером какой-либо машины";
- "число X является геделевским номером машины, имеющей следующие команды …".

Определим $Q_z(X)$ следующим образом:

$$Q_z(X) = \begin{cases} f(X), \text{ где } f(X) - \text{значение функции } f(X), \\ \text{вычисляемой машиной с номером } z \text{ для задания } X; \\ 0, \text{ если } z \text{ не является номером какой-либо машины.} \end{cases}$$

Теорема 4.2. *Функция $Q_z(X)$ частично-вычислима*.

Доказательство. Последовательность $Q_0(X)$, $Q_1(X)$,…, $Q_n(X)$,… перечисляет, быть может с повторениями, множество частично-вычислимых функций. Тем самым мы можем также перечислять области определения таких функций.

Пусть заданы конкретные значения z и X. Чтобы вычислить $Q_z(X)$, необходимо сначала с помощью машины M_2 узнать, является ли z геделевским номером какой-либо машины. Если ответ положительный, то определив по z набор команд соответствующей машины Тьюринга Z, можно предложить этой машине X в качестве входного задания. Таким образом, соединяя вместе M_2 и Z, получим машину, вычисляющую $Q_z(X)$. *Теорема доказана*.

Машина, вычисляющая $Q_z(X)$, называется *универсальной* машиной Тьюринга.

Рекурсивные множества

Пусть A – некоторое множество натуральных чисел. Функцию

$$\chi_A(x) = \begin{cases} 1, \text{ если } x \in A; \\ 0, \text{ если } x \notin A, \end{cases}$$

называют *характеристической функцией* множества A.

Множество A называют *рекурсивным* (разрешимым), если его характеристическая функция рекурсивна.

Множество называют *рекурсивно-перечислимым*, если оно является областью определения некоторой ЧР-функции.

Теорема 4.3. *Множество рекурсивно тогда и только тогда, когда оно само и его дополнение рекурсивно-перечислимы.*

Доказательство. Пусть множество R рекурсивно. Тогда существует машина Тьюринга M, которая вычисляет его характеристическую функцию. Изменим M так, чтобы получить две машины, вычисляющие функции

$$f(x) = \begin{cases} 1, \text{ если } x \in R; \\ \text{не определена, если } x \notin R. \end{cases}$$

$$\bar{f}(x) = \begin{cases} 0, \text{ если } x \in \bar{R}; \\ \text{не определена, если } x \notin \bar{R}. \end{cases}$$

Следовательно, из рекурсивности R следует, что R и $\bar{R}$ рекурсивно-перечислимые множества.

Предположим теперь, что R и $\bar{R}$ рекурсивно-перечислимые множества. В таком случае они являются областями определения частично-вычислимых функций $f(x)$ и $\bar{f}(x)$, для которых имеются вычисляющие их машины Тьюринга Z и $\bar{Z}$ соответственно. Если предложить этим машинам некоторое задание x, то одна из них обязательно даст результат, а другая нет. Если это машина Z, то $\chi_R(x)=1$, а если это машина $\bar{Z}$, то $\chi_R(x)=0$. Пара машин $\bar{Z}$ и Z эквивалентна одной машине Тьюринга. Следовательно, $\chi_R(x)$ вычислима, а R и $\bar{R}$ – рекурсивные множества. *Доказательство закончено*.

Теорема 4.4. *Проблема останова алгоритмически неразрешима.*

Доказательство. Предположим, что существует машина Тьюринга T, которая, получив задание z^*X, даст ответ

- “да”, если машина Z с геделевским номером z, получив задание X, отработает, остановится и выдаст некоторый результат;
- “нет”, если машина Z с входным заданием X никогда не остановится.

Пусть E - рекурсивно-перечислимое, но не рекурсивное множество. Тогда E является областью определения некоторой ЧР-функции f_m, вычисляемой машиной M. Машина T для любого натурального x дала бы ответ

- “да, $x \in E$ и M остановится ”;
- “нет, $x \notin E$ и M не остановится ”.

Это означало бы, что E – рекурсивное множество, что противоречит предположению о свойствах этого множества. Следовательно, машина Тьюринга T не может существовать.

Остается показать, что существуют рекурсивно-перечислимые, но не рекурсивные множества. Для этого рассмотрим множество машин Тьюринга, вычисляющих функции одного аргумента. Каждой такой машине предложим в качестве исходного задания ее собственный геделевский номер z. Если машина Z, начав вычисления с z=Ш(Z), когда-либо остановится, то она называется *самоприменимой*.

Таким образом, мы можем построить функцию $Q_Z(Z)$. Очевидно, что $Q_Z(Z)$ – частично вычислимая функция. Следовательно, множество G геделевских номеров самоприменимых машин рекурсивно-перечислимо.

Предположим, что это множество является также и рекурсивным. Это значит, что $\overline{G}$ также рекурсивно-перечислимо. Тогда должна существовать машина Тьюринга с номером λ, перечисляющая $\overline{G}$ (т.е. вычисляющая некоторую функцию с областью определения $\overline{G}$).

В перечислении рекурсивно-перечислимых множеств множество $\overline{G}$ имеет именно этот конкретный номер: $\overline{G} = R_\lambda$. Тогда для любого натурального x мы имели бы $x \in R_\lambda \Leftrightarrow x \in R_x$. Если

положить x=λ, то $\lambda \in R_\lambda \Leftrightarrow \lambda \notin\in R_\lambda$, т.е. приходим к противоречию. *Теорема доказана*.

Дополнительные сведения по теории алгоритмов и алгоритмической разрешимости можно найти в [3, 7, 8, 9, 11, 12, 15, 16].

Глава 5
ЭФФЕКТИВНОСТЬ АЛГОРИТМОВ

Сложность вычислений и эффективность алгоритмов составляют одну из важнейших проблем современной теории вычислительных систем. В области теории и практики программирования выработано много различных подходов к проблеме сложности вычислений. Задача настоящей главы – познакомить читателя с основами современных точных методов оценки сложности задач и эффективности алгоритмов. Изучение таких методов полезно для развития интуитивных представлений об эффективном (с точки зрения стоимости) использовании вычислительных машин и для выработки практических навыков оценки потребности в вычислительных ресурсах (таких, как память и время), необходимых при наиболее благоприятных условиях для вычисления функции данной сложности на универсальных вычислительных машинах.

5.1. Переборные задачи и сложность вычислений

Задача распознавания П – это множество $Z_{\text{П}}$ всевозможных индивидуальных задач и подмножество $Z_{\text{П.да}} \subset Z_{\text{П}}$ задач с ответом “да”. Задача распознавания П называется *переборной,* если каждая индивидуальная задача z формулируется следующим образом: *существует ли такой объект y, что выполняется свойство, выражаемое предикатом R(z,y)?* При этом предполагается, что проверка истинности *R(z,y)* имеет полиномиальную сложность.

Например, задача 3-выполнимости к.н.ф.: дано множество дизъюнкций $D=\{d_1,d_2,\ldots,d_m\}$ на конечном множестве булевых переменных $X=\{x_0,x_1,\ldots,x_n\}$, таких, что число $|d_i|$ переменных каждой дизъюнкции равно 3. Требуется ответить на вопрос: существует ли на X набор значений истинности, при котором выполняются все дизъюнкции из D?

Предполагается, что для представления исходных данных используется алфавит Σ и некоторый естественный способ кодирования K, причем длина кода исходных данных задачи z равна $l(z)$.

Введем обозначение Σ^* для множества всевозможных цепочек символов (слов) в алфавите Σ. Любое подмножество множества Σ^* цепочек называется *языком над алфавитом* Σ. Множество текстов задачи П с ответом из множества $Z_{\text{П.да}}$ при выбранном способе кодирования K будем рассматривать как язык L(П,K).

Сложность вычислений на машине Тьюринга

Рассмотрим детерминированную машину Тьюринга (ДМТ) M, вычисляющую рекурсивную функцию

$$f_M\colon \Sigma^* \to \Gamma^*,$$

где Σ^* и Γ^* - множество всевозможных цепочек над алфавитами Σ и Γ соответственно. При начальной конфигурации $q_1\sigma$, где $\sigma\in\Sigma^*$, машина, если когда-либо остановится, завершит работу в конфигурации $q_0\gamma$, где $\gamma=f_M(\sigma)\in\Gamma^*$.

Число тактов работы для получения $\gamma=f_M(\sigma)$ назовем *временно́й сложностью* машины M и обозначим через $t_M(\sigma)$. Если значение $f_M(\sigma)$ не определено, то временная сложность $t_M(\sigma)$ также не определена. *Активной зоной* машины M при работе со входом σ называют множество всех ячеек ленты, участвующих в вычислении $\gamma=f_M(\sigma)$. Длину активной зоны обозначим через $s_M(\sigma)$.

Теорема 5.1. *Если ленточный алфавит машины M содержит k*

символов, а алфавит состояний головки - r символов, то для сложности вычислений справедливы оценки:

$$s_M(\sigma) \le |\sigma| + t_M(\sigma),$$
$$t_M(\sigma) \le rs_M^2(\sigma)k^{s_M(\sigma)},$$

где $|\sigma|$ *- длина цепочки* σ.

Доказательство [14].

1. В начальной ситуации на ленте записана цепочка σ, занимающая $|\sigma|$ ячеек. На каждом шаге вычислений добавляется не более одной активной ячейки, поэтому $s_M(\sigma) \le |\sigma| + t_M(\sigma)$.
2. Выполним подсчет числа всевозможных конфигураций
$$K = a^{(1)} \ldots a^{(i-1)} q_j a^{(i)} \ldots a^{(s')} \quad (s' \le s)$$
с длиной активной зоны, не превышающей s. Имеется $k^{s'} \le k^s$ вариантов записи символов на ленте, r вариантов состояния и $s' \le s$ вариантов положения головки, а также s вариантов длины s' конфигурации K. Поэтому общее число конфигураций не превосходит rs^2k^s. Повторение конфигураций возможно только в случае зацикливания машины. Следовательно, для числа тактов можно записать $t_M(\sigma) \le rs_M^2(\sigma)k^{s_M(\sigma)}$.

Теорема доказана.

Частный случай – машина, вычисляющая характеристическую функцию множества $L_M \subseteq \Sigma^*$:

$$\mu_M: \Sigma^* \to \{0,1\}.$$

Множество L_M называется языком, распознаваемым машиной M:

$$L_M = \{\sigma \mid \sigma \in \Sigma^*, \mu(\sigma) = 1\}.$$

Временна́я сложность вычисления T_M: $N \to N$:

$$T_M(n) = \max\left\{ m \,\middle|\, \begin{array}{l} \text{существует такая цепочка } \sigma \in \Sigma^*, |\sigma| = n, \text{ что} \\ \text{вычисление со входом } \sigma \text{ требует } m \text{ шагов} \end{array} \right\}.$$

5.2. Классы задач P и NP

Детерминированная машина M называется *полиномиальной*, если существует полином p такой, что для всех $n \in N$, $T_M \le p(n)$.

Класс P-*языков* определяется следующим образом:

$$P = \left\{ L \left| \begin{array}{l} \text{существует полиномиальная машина } M, \\ \text{для которой } L = L_M \end{array} \right. \right\}.$$

Задача распознавания П при выбранном способе кодирования K принадлежит классу P, если $L(\Pi,K) \in P$.

Недетерминированные вычисления

Рассмотрим *недетерминированную машину Тьюринга* (НДМТ) $M_у$, которая для любой формулировки $z \in Z_\Pi$ задачи распознавания П выдает следующий результат:

если $z \in Z_{П.да}$, то машина угадывает значение y, удовлетворяющее $R(z,y)$, и записывает код y на ленту рядом с z;

если $z \notin Z_{П.да}$, машина $M_у$ сообщает об этом.

Обратите внимание, что машина $M_у$ не читает ленту, а только угадывает ответ и пишет его на ленту.

Детерминированная машина $M_{пр}$ по входу (z,y) проверяет истинность $R(z,y)$.

Рассмотрим НДМТ M в виде суперпозиции $M_у$ и $M_{пр}$. Эта машина решает задачу П за полиномиальное время, если найдется такой полином p, что для любой задачи $z \in Z_{П.да}$ машина $M_у$ найдет такое значение y, что детерминированная машина $M_{пр}$ по значению (z,y) проверит истинность $R(z,y)$ за время $p(l(z))$. Это означает, что размер y ограничен полиномом от $l(z)$.

Класс NP - это все задачи распознавания, которые (при разумном кодировании) могут быть решены недетерминированным алгоритмом (N - non-deterministic) за полиномиальное время (P).

Для распознавания свойства $R(z,y)$ можно сформировать пару задач.

- *Прямая задача*: верно ли, что для заданного z существует такое y, что выполняется $R(z,y)$?
- *Обратная задача*: верно ли, что для заданного z не существует такое y, что выполняется $R(z,y)$?

Если прямая задача принадлежит классу P, то и дополнительная задача принадлежит классу P. Если же прямая задача принадлежит классу NP, то не известно, принадлежит ли дополнительная задача классу NP.

Недетерминированная машина M *принимает* x, если, по крайней мере, одно вычисление для x является принимающим. Язык, распознаваемый программой M:

$$L_M=\{x\in\Sigma^* \mid M \text{ принимает } x\}.$$

Время, за которое принимается $x\in L_M$, – это минимальное число шагов по всем вычислениям при входе x.

Временна́я сложность программы НДМТ M – это функция T_M: $N\to N$ (N – множество целых чисел):

$$T_M(n)=\max\left(\{1\}\cup\left\{m \;\middle|\; \begin{array}{l}\text{существует такое значение } x\in L_M,\ |x|=n,\\ \text{что время принятия } x \text{ программой}\\ \text{машины } M \text{ равно } m\end{array}\right\}\right).$$

$T_M(n)=1$, если нет ни одного входа длины m, принимаемого программой M.

НДМТ имеет полиномиальную временную сложность, если найдется такой полином p, что $T_M(n)\le p(n)$ для всех $n\ge 1$.

Формальное определение класса NP:

$$NP=\left\{L \;\middle|\; \begin{array}{l}\text{существует НДМТ } M \text{ с полиномиальным}\\ \text{временем работы, такая, что } L=L_M\end{array}\right\}.$$

Хотя недетерминированный алгоритм угадывает y, зависящий некоторым образом от z, угадывающий модуль M_y при этом полностью игнорирует вход z.

Теорема 5.2. *Если* $\Pi\in$NP, *то существует такой полином* p, *что* Π *может быть решена детерминированным алгоритмом с временной сложностью* $O(2^{p(n)})$.

Доказательство можно найти, например, в [3].

5.3. Класс NP-полных задач

Полиномиальная сводимость NP-полных задач

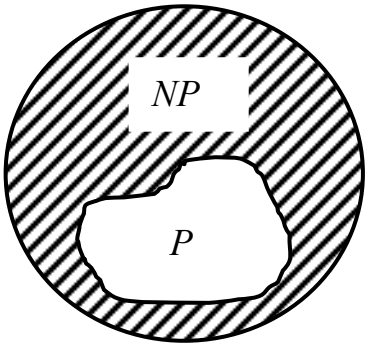

Рис. 5.1.

Если P≠NP, то между P и NPC=NP\P имеется существенное различие. *Многолетняя мировая практика разработки алгоритмов показывает, что еще никому не удалось построить алгоритм с полиномиальной временно́й сложностью для некоторых классов задач.* Цель теории NP-полных задач в доказательстве результатов вида: **Если P≠NP, то Π∈ NP\P.**

Важную роль в теории NP-полных задач играет полиномиальная сводимость. Язык $L_1 \subseteq \Sigma_1^*$ *полиномиально сводится* к языку $L_2 \subseteq \Sigma_2^*$, если существует функция f: $\Sigma_1^* \to \Sigma_2^*$, удовлетворяющая двум условиям.

1. Существует ДМТ-программа, вычисляющая f с полиномиальной временной сложностью.
2. Для любого $x \in \Sigma_1^*$, $x \in L_1$ в том и только том случае, если $f(x) \in L_2$.

Будем говорить "L_1 сводится к L_2" (опуская слово "полиномиально") и писать $L_1 \propto L_2$.

Утверждение 5.1. *Если* $L_1 \propto L_2$, *то из* $L_2 \in$ P *следует, что* $L_1 \in$ P.

Доказательство можно найти, например, в [3].

Если Π_1 и Π_2 задачи распознавания, а K_1 и K_2 их схемы кодирования, то будем писать $\Pi_1 \propto \Pi_2$, если $L_1(\Pi_1,K_1) \propto L_2(\Pi_2,K_2)$.

Утверждение 5.2. *Если* $L_1 \propto L_2$ и $L_2 \propto L_3$, *то* $L_1 \propto L_3$ (*транзитивность*). Доказательство можно найти, например, в [3].

Языки L_1 и L_2 *полиномиально эквивалентны*, если $L_1 \propto L_2$ и $L_2 \propto L_1$. Язык L называется NP-*полным* ($L \in$NPC), если $L \in$NP и любой другой $L' \in$NP сводится к L.

По современным представлениям более точное соотношение между P, NP и NPC показано на рис. 5.2.

Языки L_1 и L_2 *полиномиально эквивалентны*, если $L_1 \propto L_2$ и $L_2 \propto L_1$. Язык L называется NP-*полным* ($L \in$NPC), если $L \in$NP и любой другой $L' \in$NP сводится к L.

По современным представлениям более точное соотношение между P, NP и NPC показано на рис. 5.2.

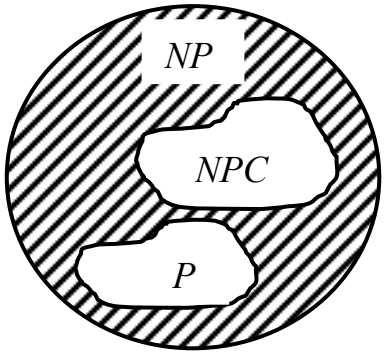

Рис. 5.2.

Утверждение 5.3. *Если* L_1 *и* $L_2 \in$NP, $L_1 \in$NPC *и* $L_1 \propto L_2$, *то* $L_2 \in$NPC. Доказательство можно найти, например, в [3].

Теорема 5.2. *Задача о выполнимости к.н.ф. есть* NP-*полная задача.*

Доказательство [14]. Доказательство основано на описании работы машин Тьюринга с помощью к.н.ф.. Рассматриваем произвольную переборную задачу $Z_{\text{П}}=\{z\}$ распознавания свойства $R(z,y)$. Пусть машина Тьюринга M проверяет истинность $R(z,y)$ за время, ограниченное полиномом от суммы размерностей записи z и y на ленте, т.е. $l(z) + l(y)$. Длина $l(y)$ не превосходит некоторого полинома от $l(z)$, поэтому время распознавания свойства $R(z,y)$ и требуемый размер активной зоны ограничены некоторым полиномом

$P(l(z))$.

Пусть $n= l(z)$, $T=P(n)$, ячейки ленты занумерованы числами …,-2,-1,0,1,2,… , а исходные данные z размещены в ячейках 1,…,n. В этом случае вычисление $R(z,y)$ для любого y происходит в зоне ячеек $–T,…,T$ за время, не превышающее T тактов.

Машина начинает работу в конфигурации q_1z*y и завершает ее в конфигурации q_01, если свойство $R(z,y)$ выполнено, и в конфигурации q_00 в противном случае. Для удобства изложения модифицируем машину M так, чтобы при попадании в состояние q_0 она работала вечно: тогда в момент T машина будет в состоянии q_0.

Доказательство основано на том, что работы машины Тьюринга, вычисляющей $R(z,y)$, может быть описана с помощью формулы, имеющей вид к.н.ф.

$$Ф=B\&C\&D\&E\&F\&G,$$

которая выполняется только тогда, когда $R(z,y)=1$, причем

B,C,D – условия того, что ДМТ работает правильно;

E - условие того, что начальная конфигурация есть q_1z*y;

F - условия того, что ДМТ работает в соответствии с программой;

G - условие того, что заключительная конфигурация есть q_01.

Сложность формулы Ф будем оценивать числом булевых переменных, использованных в записи формулы, и обозначать через compl(Ф).

Пусть машина использует ленточный алфавит $A=\{a_0,a_1,…,a_{k-1}\}$ и алфавит состояний $Q=\{q_0,q_1,…,q_{r-1}\}$. Введем булевы переменные $u^i_{s,t}$, v^j_t и $w_{s,t}$ ($0\le i\le k-1$, $0\le j\le r-1$, $-T\le s\le T$, $0\le t\le T$):

$u^i_{s,t}$ =1 тогда и только тогда, когда на шаге t ячейка s содержит символ a_i;

v^j_t = 1 тогда и только тогда, когда на шаге t машина находится в состоянии j;

$w_{s,t}$ = 1 тогда и только тогда, когда на шаге t головка обозревает ячейку s.

Высказывание B обозначает, что на каждом такте t ($0\le t\le T$) обозревается ровно одна ячейка:

$$B=\&_{0\le t\le T} B_t,$$

где

$B_t = (\vee_{-T\le s\le T} w_{s,t})(\&_{-T\le s<q\le T} (\bar{w}_{s,t} \vee \bar{w}_{q,t}))$ – условие того, что на такте t обозревается ровно одна ячейка ленты.

В выражении для B_t дизъюнкция $\vee_{-T\le s\le T} w_{s,t}$ означает, что на такте t обозревается некоторая ячейка. Отметим, что сложность этой формулы равна $2T+1$. Формула $\&_{-T\le s<q\le T} (\bar{w}_{s,t} \vee \bar{w}_{q,t})$ - это условие того, что на такте t обозревается не более одной ячейки. Действительно, если хотя бы две переменные $w_{s,t}$ и $w_{q,t}$ одновременно равны 1, эта формула принимает значение 0. Сложность рассматриваемой формулы равна числу всевозможных пар $w_{s,t}$ и $w_{q,t}$, т.е. числу сочетаний $C^2_{2T+1}=T(2T+1)$. Поскольку формула для B содержит $T+1$ подформул вида B_t, сложность формулы B оценивается полиномом

$$\mathrm{compl}(B)=(T+1)(2T+1+ T(2T+1))=2T^3+5T^2+4T+1.$$

Высказывание C обозначает, что на каждом такте t ($0\le t\le T$) в любой ячейке s ($-T\le s\le T$) находится ровно 1 символ:

$$C=\&_{0\le t\le T} \&_{-T\le s\le T} C_{s,t},$$

где

$C_{s,t} = (\vee_{0\le i\le k-1} u^i_{s,t})(\&_{0\le i<m\le k-1} (\bar{u}^i_{s,t} \vee \bar{u}^m_{q,t}))$ – высказывание о том, что на такте t в ячейке s находится ровно 1 символ.

Формула для C строится аналогично формуле A и имеет полиномиальную сложность:

$$\mathrm{compl}(C)=(T+1)(2T+1)(k+C^2_k)=k(k+1)(2T^2+3T+1)/2.$$

Высказывание D обозначает, что на каждом такте t ($0\le t\le T$) машина находится ровно в одном состоянии:

$$D=\&_{0\le t\le T} D_t,$$

где

$D_t = (\vee_{0\le j\le r-1} v^j_t)(\&_{0\le j<m\le r-1} (\bar{v}^j_t \vee \bar{v}^m_t))$ – обозначает, что на такте t машина находится ровно в одном состоянии.

Формула для D строится аналогично формулам для A, B и имеет полиномиальную сложность:

$$\mathrm{compl}(D)=(T+1)(r+\mathrm{C}^2_r)=r(r+1)(T+1)/2.$$

Высказывание E состоит из двух высказываний: $E = E_1 \& E_2$. Высказывание E_1 утверждает, что в начальной конфигурации (при t=0) головка обозревает ячейку 1 в состоянии q_1, а в ячейках 1,…,n+1записано задание $z = z(1)\ldots z(n)$ с разделителем * (для упрощения записи вместо $u^{i}_{s,t}$ пишем $u^{a_i}_{s,t}$):

$$E_1=v^{1}_{0} w_{1,0} \,(\&_{-T\le s\le 0}\, u^{\Lambda}_{s,0})(\&_{1\le m\le n}\, u^{z(m)}_{m,0})\, u^{*}_{n+1,0}\,.$$

Сложность этой формулы $\mathrm{compl}(E_1)=2+(T+1)+n+1=T+n+3$.

Пусть $A' \subseteq A \setminus \{\Lambda\}$ – алфавит, в котором представляются допустимые слова y. Для упрощения будем считать, что допустимыми словами являются все слова в алфавите A', располагающиеся правее z*, т.е. такие, что $1 \le l(y) \le T-n-1$. Высказывание E_2 означает, что, начиная с ячейки n+2, записано некоторое слово в алфавите A', за которым идут подряд пробелы:

$$E_2=[\&_{n+2\le s\le T}\,(\vee_{a\in A'\cup\{\Lambda\}}\, u^{a}_{s,0})]\;\bar{u}^{\Lambda}_{n+2,0}\;\{\&_{n+2\le s\le T-1}(\;\bar{u}^{\Lambda}_{s,0} \vee u^{\Lambda}_{s+1,0})\}.$$

В формуле для E_2 член в квадратных скобках означает условие заполнения ячеек s (n+2≤s≤T) символами алфавита A' или пробелами Λ, сомножитель $\bar{u}^{\Lambda}_{n+2,0}$ указывает на то, что символ в ячейке n+2 не является пробелом. Часть формулы для E_2, заключенная в фигурные скобки, выражает условие того, что после записи y идут ячейки, содержащие только пробелы ($u^{\Lambda}_{s,0} \to u^{\Lambda}_{s+1,0} = \bar{u}^{\Lambda}_{s,0} \vee u^{\Lambda}_{s+1,0}$).

Сложность формулы E_2: $\mathrm{compl}(E_2)=(T-n-1)(k-1)+1+(T-n-2)*2$.

Следовательно, сложность $\mathrm{compl}(E)= \mathrm{compl}(E_1)+ \mathrm{compl}(E_2)$.

Высказывание F утверждает, что машина работает в соответствии с программой:

$$F=\&_{-T\le s\le T}\,\&_{0\le t\le T}\,\&_{\;0\le i\le k-1}\,\&_{\;0\le j\le r-1}\, F^{i,j}_{s,t}\,,$$

где $F^{i,j}_{s,t}$ – высказывание, утверждающее, что если в момент t в ячейке s находится символ a_i, машина находится в состоянии q_j, а головка обозревает ячейку s, то конфигурация машины меняется в соответствии с программой, причем содержимое остальных ячеек не

меняется.

Пусть команды машины имеют вид:

$$q_j a_i \to a_{\alpha(j,i)}\, D_{j,i}\, q_{\beta(j,i)} \qquad (0 \le i \le k\text{-}1,\ 0 \le j \le r\text{-}1).$$

Обозначим через $\delta(j,i)$ перемещение головки для команды с левой частью $q_j a_i$:

$$\delta(j,i) = \begin{cases} -1, \text{ если } D_{j,i}=\text{Л}, \\ 0, \text{ если } D_{j,i}=\text{Н}, \\ 1, \text{ если } D_{j,i}=\text{П}. \end{cases}$$

Тогда

$$F^{i,j}_{s,t} = (v^j_t u^i_{s,t} w_{s,t} \to v^{\beta(j,i)}_{t+1} u^{\alpha(j,i)}_{s,t+1} w_{s+\delta(j,i),t+1}) \& (u^i_{s,t}\ \bar{w}_{s,t} \to u^i_{s,t+1}).$$

Выразим импликацию через дизъюнкцию и отрицание, используем закон де-Моргана и дистрибутивный закон, чтобы получить окончательное выражение:

$$F^{i,j}_{s,t} = (\bar{v}^j_t \vee \bar{u}^i_{s,t} \vee \bar{w}_{s,t} \vee v^{\beta(j,i)}_{t+1} u^{\alpha(j,i)}_{s,t+1} w_{s+\delta(j,i),t+1}) \&$$
$$\&(\bar{u}^i_{s,t} \vee w_{s,t} \vee u^i_{s,t+1}) =$$
$$= (\bar{v}^j_t \vee \bar{u}^i_{s,t} \vee \bar{w}_{s,t} \vee v^{\beta(j,i)}_{t+1}) \&(\bar{v}^j_t \vee \bar{u}^i_{s,t} \vee \bar{w}_{s,t} \vee u^{\alpha(j,i)}_{s,t+1}) \&$$
$$\&(\bar{v}^j_t \vee \bar{u}^i_{s,t} \vee \bar{w}_{s,t} \vee w_{s+\delta(j,i),t+1}) \&(\bar{u}^i_{s,t} \vee w_{s,t} \vee u^i_{s,t+1}).$$

Сложность $F^{i,j}_{s,t}$ равна 15, поэтому сложность F выражается полиномом $\mathrm{compl}(F)=15(2T+1)(T+1)kr$.

Высказывание G означает, что на шаге T машина окажется в конфигурации $q_0 1$:

$$G = [(\&_{-T \le s \le T} (u^{\Lambda}_{s,t} \vee u^1_{s,t}))(\&_{-T \le s < q \le T} (\bar{u}^1_{s,t} \vee \bar{u}^1_{q,t}))] \&$$
$$\& v^0_T \{\&_{-T \le s \le T} (\bar{w}_{s,T} \vee u^1_{s,T})\}.$$

В формуле для G:

- выражение в квадратных скобках означает, что в момент времени T на ленте записаны символы Λ и 1, причем 1 – не более чем в одной ячейке;
- v^0_T означает, что машина находится в состоянии q_0;
- выражение в фигурных скобках обеспечивает наличие 1 в обозреваемой ячейке.

Сложность формулы для G можно оценить выражением

$$\mathrm{compl}(G)=2(2T+1)+C^{2}_{2T+1}+1+2(2T+1)=4(2T+1)+C^{2}_{2T+1}+1.$$

Из приведенных выражений видно, что можно построить к.н.ф. $\Phi=B\&C\&D\&E\&F\&G$, которая утверждает, что существует такое допустимое y, что машина, начав работу в конфигурации q_1z^*y, перейдет в заключительную конфигурацию q_01.

Если задача $z\in Z_\Pi$ имеет положительный ответ, то для некоторого допустимого y выполняется $R(z,y)=1$. В этом случае, начав работу в конфигурации q_1z^*y, получим заключительную конфигурацию q_01, и, если назначить переменным $u^i_{s,t}$, v^j_t и $w_{s,t}$ значения, соответствующие у, то получим набор, на котором $\Phi=1$. Обратно, если существует набор, обращающий Ф в 1, то значения переменных $u^i_{s,0}$ ($0\le i\le k$-1, $n+2\le s\le T$) определяют слово y такое, что машина переводит q_1z^*y в q_01 и, следовательно, задача z имеет положительный ответ. Тем самым задача Z_Π сведена к задаче выполнимости к.н.ф.

Сложность формулы $\Phi=B\&C\&D\&E\&F\&G$ равна сумме сложностей ее компонентов, для которых выше были приведены полиномиальные выражения. Поэтому сложность Ф ограничена полиномом от $n=l(z)$, причем при записи этой формулы в некотором алфавите ее длина возрастет примерно в $\log(\mathrm{compl}(\Phi))$ раз и останется полиномиально ограниченной. Рассмотрение формулы Ф показывает, что от индивидуальной задачи z зависит лишь сомножитель $\&_{1\le m\le n}\, u^{z(m)}_{m,0}$ в выражении для E_1 и величины $n=l(z)$ и $T=P(n)$. В остальном формула Ф определяется машиной M и, следовательно, массовой задачей Z_Π. Таким образом, сведение осуществлено с полиномиальной сложностью. Теорема доказана.

5.4. Труднорешаемые задачи

NP-полная задача П называется *универсальной*, если к ней полиномиально сводится любая NP-полная задача. Выше была доказана принадлежность задачи выполнимости к.н.ф. к классу NP-полных задач.

Приведем другие примеры задач этого класса: о полном подграфе, о вершинном покрытии, о сложности д.н.ф. частичной булевой функции и о трехмерном сочетании.

Задача о полном подграфе. ***Граф называется* полным*, если любые две его вершины соединены ребром. По заданному неориентированному графу* G *и числу* k *требуется определить, имеется ли в* G *полный подграф с* k *вершинами.***

Задача о вершинном покрытии. Некоторое подмножество V', содержащее $k=|V'|$ вершин, образует *вершинное покрытие* мощности k графа G, если для любого ребра найдется инцидентная ему вершина в подмножестве V'.

По заданному графу G и числу k требуется определить, имеет ли G вершинное покрытие мощности k.

Задача о сложности д.н.ф. частичной булевой функции. По частичной булевой функции f и числу k необходимо установить, возможно ли построить д.н.ф. для f, содержащую не более k букв.

Трехмерное сочетание. Дано множество $M \subseteq W \times X \times Y$, где W,X,Y – непересекающиеся множества, содержащие одинаковое число элементов q. Необходимо установить, что M содержит *трехмерное сочетание*, т.е. подмножество $M' \subseteq M$ такое, что $|M'|=q$ и никакие два разных элемента M' не имеют ни одной равной координаты.

Например, $W=\{a,b,c\}$, $X=\{d,e,f\}$, $Y=\{j,h,i\}$ и $M = \{(a,d,i), (a,e,i), (a,f,j), (b,e,i), (b,f,j), (c,e,h), (c,d,i)\}$. Данный пример имеет положительное решение: $M'=\{(a,d,i), (b,f,j), (c,e,h)\} \subseteq M$.

Пусть $П_1$ – задача о выполнимости к.н.ф. Тогда, чтобы доказать,

что задача $П_q$ является NP-полной, достаточно доказать, что

$$П_1 \propto П_2 \propto \ldots \propto П_q.$$

Например, в [15] показано, что задача выполнимости к.н.ф сводится к задаче о полном подграфе, которая в свою очередь сводится к задаче о вершинном покрытии, а последняя - к задаче о сложности д.н.ф. частичной булевой функции.

NP-трудные задачи

Задача П называется NP-*трудной*, если П - произвольная задача (т.е. не обязательно, что $П \in NP$) распознавания свойств, к которой сводится некоторая NP-полная задача. К NP-трудным могут относиться не только задачи распознавания свойств, но и другие переборные задачи.

Точное решение NP-трудной (так же как и NP-полной) задачи в общем случае может быть получено только за экспоненциальное время. Следовательно, решать ее переборным алгоритмом неэффективно при большом количестве вариантов перебора. Одним из подходов к решению NP-трудных задач является сокращение перебора по *схеме ветвей и границ*. Эта схема перебора позволяет опознать бесперспективные частичные решения, в результате чего от дерева поиска на одном шаге отсекается целая ветвь. Однако в общем случае схема ветвей и границ не выводит рассматриваемые задачи из класса NP-трудных.

Приведем примеры NP-трудных задач: задача коммивояжера и задача построения сингулярной скобочной формы.

Задача коммивояжера. Эта задача, поставленная еще в 1934 г., является одной из самых важнейших задач в теории графов. В области оптимизации дискретных задач задача коммивояжера служит своеобразным полигоном, на котором испытываются все новые методы.

Постановка задачи. Коммивояжер (бродячий торговец) должен выйти из первого города, посетить по разу в неизвестном порядке города 2, 3, …, *n* и вернуться в первый город. Расстояния между всеми городами известны. В каком порядке следует обходить города, чтобы замкнутый путь коммивояжера был кратчайшим? В терминах теории графов: найти *гамильтонов цикл* в графе минимальной длины.

Задача построения сингулярной скобочной нормальной

формы. Назовем *скобочной нормальной формой* (с.н.ф.) булевой функции f такое представление f в базисе {&, ∨, ¬}, при котором операция инверсии ¬ применяется только к отдельным переменным.

Назовем *сингулярной* (или особенной) такую с.н.ф., в которой никакие две подформулы, входящие в одну конъюнкцию, не содержат одинаковых переменных [1, 2, 17, 18].

В [18] показано, что задача выполнимости к.н.ф. сводится к построению сингулярной с.н.ф. для исследуемой к.н.ф.

Литература

1. Анкудинов Г.И. Синтез структуры сложных объектов: логико-комбинаторный подход. – Л.: Изд-во Ленингр. ун-та, 1986.- 260с.
2. Анкудинов Г.И., Золотов О.А., Петухов О.А. Логическое программирование на языке Prolog: Учеб.пособие. – СПб.: СЗТУ, 2001. – 172с.
3. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. – М.: Мир, 1982. – 416с.
4. Заде Л. Понятие лингвистической переменной и его применение к принятию приближенных решений. – М.: Мир, 1976. – 150с.
5. Ковальски Р. Логика в решении проблем: Пер. с англ.- М.: Наука, 1990.– 280с.
6. Лингер Р., Миллс Х., Уитт Б. Теория и практика структурного программирования: Пер. с англ. – М.: Мир, 1982.– 406с.
7. Мендельсон Э. Введение в математическую логику. – М.: Наука, 1984. – 320с.
8. Минский М. Вычисления и автоматы. – М.: Мир, 1971. – 368с.
9. Нильсон Н. Искусственный интеллект: Пер. с англ. – М.: Мир, 1973.– 270с.
10. Новиков П.С. Элементы математической логики. – М.: Физматгиз, 1959. – 400с.
11. Основы кибернетики. Математические основы кибернетики: Учеб.пособие. Под ред. К.А.Пупкова. - М.: Высш.шк., 1974. – 413с.
12. Сигорский В.П. Математический аппарат инженера. – Киев: Техніка, 1977. – 768с.
13. Формальная логика: Учеб.пособие / Под ред. И.Я.Чупахина, И.Н.Бродского. - Л.: Изд-во Ленингр. ун-та, 1977. – 360с.
14. Шоломов Л.А. Основы теории дискретных логических и вычислительных устройств: Учеб.пособие. – М.: Наука, 1980. – 400с.
15. Turing, A.M. On computable numbers, with an application to the Entscheidungsproblem // Proc. London Math. Soc., ser.2, 42, 1936. – P.230–265.

16. Ankoudinov, G.I. On one general approach to structural synthesis of algorithms, devices and systems // Kibernetika.- Kiev, No.1, 1982.– P.65-68 (English translation by The Plenum Publishing Corporation, USA).
17. Ankoudinov, G.I. Symbolic-numerical methods in discrete programming problems with logical constraints // Kibernetika.- Kiev, No.3, 1989. (English translation by The Plenum Publishig Corporation, USA).– P.51–55.
18. Ankoudinov, G. Advanced morphological approach to systems structural modelling // the Fourth St.Petersburg Workshop on Simulation, St.Petersburg State University, June 18-23, 2001.– P.145–150.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

ОГЛАВЛЕНИЕ

Анкудинов Георгий Иванович
Анкудинов Иван Георгиевич
Петухов Олег Александрович

МАТЕМАТИЧЕСКАЯ ЛОГИКА И ТЕОРИЯ АЛГОРИТМОВ

Учебное пособие

Редактор И.Н.Кочугина

Сводный темплан 2003 г.
Лицензия ЛР №020308 от 14.02.97

Подписано в печать **Формат 60 x 84 1/16**

Б. кн.-журн. **П.л. 5,75** **Б.л. 2,875** **РТП РИО СЗТУ.**

Тираж 200 **Заказ**

Северо-Западный государственный заочный технический университет
РИО СЗТУ, член Издательско-полиграфической ассоциации вузов Санкт-Петербурга
191186, Санкт-Петербург, ул.Миллионная, 5