ОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ НАУК
КАФЕДРА КИБЕРНЕТИКИ

А.К. Гуц

Математическая логика и теория алгоритмов

Омск 2003

ББК 60
УДК 53:630.11

Гуц А.К. Математическая логика и теория алгоритмов: Учебное пособие. – Омск: Издательство Наследие. Диалог-Сибирь, 2003. – 108 с.

ISBN 5 - 8239 - 0126 - 7

Учебное пособие посвящено изложению основ математической логики и теории алгоритмов. Основу пособия составляют конспекты лекций, которые читались студентам второго курса отделения компьютерных наук Омского государственного университета в 2002 году.

Для студентов, обучающихся по специальности 075200 – «Компьютерная безопасность» и по специальности 220100 – «Вычислительные машины, комплексы, системы и сети».

ISBN 5 - 8239 - 0126 - 7 © Омский госуниверситет, 2003

Оглавление

Часть I

Логика

Глава 1

Классическая логика

Чем занимается наука *логика*? Это теория, которая учит, как нужно правильно рассуждать, правильно делать умозаключения и выводы, получая в результате верные (правильные) высказывания. Поэтому логика как наука должна содержать список правил получения правильных высказываний. Такой набор правил, умозаключений называется *списком силлогизмов.* [1]

1.1. Логика высказываний

1.1.1. Высказывания

Высказывание – это утверждение об изучаемых объектах, имеющее однозначное и точно определенное значение [31, с.15].

В русском языке высказывание представляет собой повествовательное предложение, о котором можно сказать, что оно сообщает нам нечто верное либо нечто совершенно неверное. Следовательно, высказывание может быть либо *истинным*, либо *ложным*. Иначе

[1] *Силлогизм* [гр. syllogismos] – умозаключение. У Аристотеля – умозаключение, состоящее из двух высказываний $\mathcal{A}$ и $\mathcal{B}$ (посылок), из которых следует третье высказывание $\mathcal{C}$ (вывод). Для силлогизма используется символическая запись вида

$$\frac{\mathcal{A},\ \mathcal{B}}{\mathcal{C}}.$$

говоря, каждому высказыванию приписывается *истинностное значение*. В классической (двузначной) логике рассматривается всего два истинностных значения: $\top$ – «истина» и $\bot$ – «ложь».

Если даны два высказывания $\mathcal{A}$ и $\mathcal{B}$, то логика, во-первых, говорит нам, как из них построить новое высказывание, а во-вторых, учит, как найти его истинностное значение. Для построения новых высказываний используются *логические связки* $\&, \vee, \Rightarrow, \Leftrightarrow$ и $\neg$:

$$\mathcal{A}\&\mathcal{B}, \quad \mathcal{A} \vee \mathcal{B}, \quad \mathcal{A} \Rightarrow \mathcal{B}, \quad \mathcal{A} \Leftrightarrow \mathcal{B}, \quad \neg\mathcal{A}, \quad \neg\mathcal{B}.$$

Связка $\&$ интерпретируется как союз «и», связка $\vee$ – «или», связка $\Rightarrow$ как глаголы «следует», «вытекает», связка $\Leftrightarrow$ – «тогда и только тогда, когда» и, наконец, $\neg\mathcal{A}$ понимается как отрицание высказывания $\mathcal{A}$ («не $\mathcal{A}$»).

Для установления истинностного значения нового высказывания применяют *истинностные таблицы*[2]:

$\mathcal{A}$	$\mathcal{B}$	$\mathcal{A}\&\mathcal{B}$	$\mathcal{A} \vee \mathcal{B}$	$\mathcal{A} \Rightarrow \mathcal{B}$	$\mathcal{A} \Leftrightarrow \mathcal{B}$
$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$
$\bot$	$\top$	$\bot$	$\top$	$\top$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\top$	$\top$

$\mathcal{A}$	$\neg\mathcal{A}$
$\top$	$\bot$
$\bot$	$\top$

Логическая наука, занимающаяся высказываниями, называется *алгеброй высказываний*. Название это становится понятным, если связку $\&$ назвать произведением, а $\vee$ – сложением. Мы как бы перемножаем и складываем высказывания.

1.1.2. Основные законы логики

Классическая логика, как наука о правильных умозаключениях, основывается на следующих четырех законах.

Закон тождества (Аристотель)[3].

$$\models (\mathcal{A} \Leftrightarrow \mathcal{A})$$

[2] Таблицы истинности введены австрийским логиком Витгенштейном.

[3] Знак $\models$ следует понимать как фразу «истинно, что». Знак придуман американским логиком С.К.Клини в 1956 году.

Закон непротиворечия[4] (Аристотель)[5].

$$\nvDash (\mathcal{A}\&\neg\mathcal{A})$$

Закон исключенного третьего (Аристотель).

$$\vDash (\mathcal{A} \vee \neg\mathcal{A}).$$

Закон достаточного основания (Лейбниц). *Никакое высказывание $\mathcal{A}$ не может быть принято, если оно не является следствием, полученным в ходе применения силлогизмов из ранее принятых утверждений или строго установленных фактов, выраженных также в форме высказываний.*

1.1.3. Логический парадокс Рассела

В конце XIX века в математике, а точнее, в теории множеств была доказана теорема, которая опровергает закон непротиворечия Аристотеля. Покажем, как это было сделано.

Пусть дан объект

$$u = \{x : x \notin x\}.$$

Теорема 1.1. $\vDash (u \in u)\&(u \notin u)$.

Доказательство

Докажем, что $u \in u$. Доказательство поведем методом от противного. Предположим, что $u \notin u$. Тогда из определения объекта u следует, что $u \in u$. Получаем противоречие с исходным предположением. Значит, предположение неверно, и верно, что $u \in u$. Утверждение доказано.

А теперь докажем, что $u \notin u$. Снова применим рассуждение от противного. Допустим, что в действительности $u \in u$. Тогда из определения объекта u следует, что $u \notin u$. Это противоречит нашему исходному допущению, поэтому в действительности $u \notin u$. Требуемое доказано.

Теорема 1.1 доказана.

[4] Чаще этот закон называется законом противоречия.

[5] Знак $\nvDash$ следует понимать как фразу «ложно, что».

1.1.4. Алгебра (логика) высказываний

В повседневной жизни можно найти примеры утверждений, содержащие *переменные параметры*, значения которых не фиксируется. Например, утверждение

$$X^3 + 1 > 0$$

содержит параметр X. Пока переменной X не придано значение, скажем, 2, мы не можем ничего сказать о справедливости приведенного утверждения.

В логике переменным параметрам, входящим в утверждения, естественно приписывать значение $\top$ или $\bot$, получая при этом высказывания. Назовем такие переменные параметры, следуя традиции, *пропозициональными переменными*. Как правило, для краткости пропозициональные переменные именуют просто *переменными*. Будем обозначать их как $X_1, X_2, ..., X_n, ...$

Поскольку логика под влиянием математики превратилась в *математическую логику*, то естественным было использование в логике терминологии, принятой в математике. В частности, утверждение $\mathcal{A}$, содержащее переменные, стали называть *формулой*. Для того чтобы быть *точным*, логик обязан был дать строгое определение того утверждения, которое он хотел бы назвать формулой. Отсюда следующее индуктивное **определение формулы**:

1) *пропозициональная переменная есть (атомарная) формула;*
2) *если* $\mathcal{A}$ *и* $\mathcal{B}$ *формулы, то* $(\mathcal{A}\&\mathcal{B})$, $(\mathcal{A} \vee \mathcal{B})$, $(\mathcal{A} \Rightarrow \mathcal{B})$ *формулы;*
3) *если* $\mathcal{A}$ *формула, то* $\neg\mathcal{A}$ *формула.*

Как видим, при построении формул используются скобки (и). Однако, как часто делается, лишние скобки опускают. К примеру, формулу $((\mathcal{A} \vee \mathcal{B})\&\mathcal{C})$ пишут как $(\mathcal{A} \vee \mathcal{B})\&\mathcal{C}$, а $(\mathcal{A}\&\mathcal{B})$ как $\mathcal{A}\&\mathcal{B}$. Существуют и другие правила, позволяющие не ставить лишних скобок.

Формулу $\mathcal{A}$, содержащую пропозициональные переменные $X_1, X_2, ..., X_n$, будем обозначать как $\mathcal{A}(X_1, X_2, ..., X_n)$.

Теория, которая изучает формулы, определенные выше, называется *алгеброй высказываний*. В алгебре высказываний каждая пропозициональная переменная, каждая формула принимает одно из двух *значений* – $\top$ (истина) или $\bot$ (ложь).

Функция $f(X_1, ..., X_n)$ такая, что ее переменные и она сама могут принимать только два значения – $\top$ или $\bot$ – называется *булевой* функцией. Иначе говоря, *булева функция* – это отображение

$$f : \underbrace{\{\bot, \top\} \times ... \times \{\bot, \top\}}_{n-\text{раз}} \to \{\bot, \top\}.$$

Теорема 1.2. *Пусть $f(X_1, ..., X_n)$ – произвольная булева функция. Тогда*

$$f(X_1, ..., X_n) \equiv [f(\top, ..., \top)\&X_1\&...\&X_n] \vee$$
$$\vee [f(\top, ..., \top, \bot)\&X_1\&...\&X_{n-1}\&\neg X_n] \vee ...$$
$$... \vee [f(\bot, ..., \bot)\&\neg X_1\&...\&\neg X_n].$$

Доказательство. См. [32, с.56].

Таким образом, каждая (двузначная) булева функция является формулой логики (алгебры) высказываний.

1.1.5. Релейно-контактные схемы

Каждой булевой функции можно поставить в соответствие так называемую *релейно-контактную схему*.

Релейно-контактная схема строится в предположении, что пропозициональная переменная X – это *замыкающий контакт* в электрической схеме, который замкнут при подаче (управляющего) тока X, т.е. $X = \top$, и разомкнут при его отсутствии – $X = \bot$. Далее, формуле $\neg X$ отвечает *размыкающий контакт*, который замкнут, т.е. $\neg X = \top$, пока нет тока ($X = \bot$), и размыкается – $\neg X = \bot$, когда ток есть ($X = \top$).

Удобно вместо символа $\top$ писать символ 1 (ток проходит), а вместо $\bot$ – 0 (ток не проходит). Дизъюнкции $X \vee Y$ соответствует параллельное соединение (замыкающих) контактов (рис.1.1, b)), а конъюнкции – последовательное соединение (рис.1.1, a)). На рис.1.1 символ E означает «вход», а S – ток на «выходе».

Из простейших схем легко собираются произвольные релейно-контактные схемы. Например, на рис.1.2 приведена схема для булевой функции $f(X, Y, Z, U, W) = (X\&W\&(\neg Z \vee Y)) \vee (U\&\neg X)$.

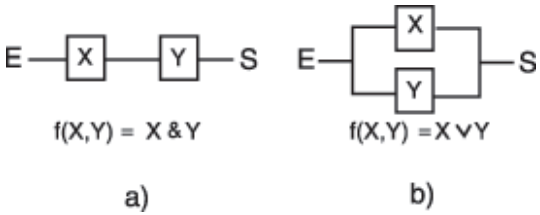

Рис. 1.1: Простейшие релейно-контактные схемы.

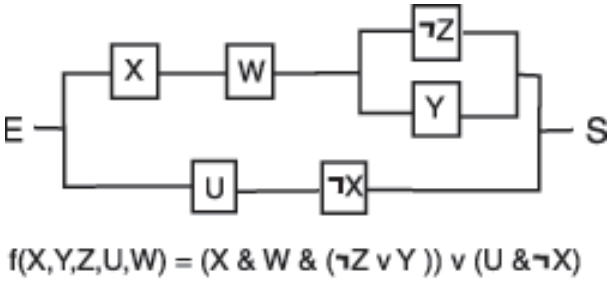

Рис. 1.2: Релейно-контактная схема для булевой функции $f(X, Y, Z, U, W) = (X\&W\&(\neg Z \vee Y)) \vee (U\&\neg X)$.

1.1.6. Равносильные формулы

Две формулы $\mathcal{A}(X_1, X_2, ..., X_n)$ и $\mathcal{B}(X_1, X_2, ..., X_n)$ называются *равносильными*, если их значения совпадают при любых значениях, входящих в них переменных $X_1, X_2, ..., X_n$. Для равносильных формул используется обозначение $\mathcal{A} \equiv \mathcal{B}$.

В логике выделяют следующие равносильные формулы:

$$
\begin{aligned}
\mathcal{A}\&\mathcal{B} &\equiv \mathcal{B}\&\mathcal{A}\\
\mathcal{A} \vee \mathcal{B} &\equiv \mathcal{B} \vee \mathcal{A}\\
(\mathcal{A}\&\mathcal{B})\&\mathcal{C} &\equiv \mathcal{A}\&(\mathcal{B}\&\mathcal{C})\\
(\mathcal{A} \vee \mathcal{B}) \vee \mathcal{C} &\equiv \mathcal{A} \vee (\mathcal{B} \vee \mathcal{C})\\
\mathcal{A}\&\mathcal{A} &\equiv \mathcal{A}\\
\mathcal{A} \vee \mathcal{A} &\equiv \mathcal{A}\\
\mathcal{A}\&(\mathcal{B} \vee \mathcal{C}) &\equiv (\mathcal{A}\&\mathcal{B}) \vee (\mathcal{A}\&\mathcal{C})\\
\mathcal{A} \vee (\mathcal{B}\&\mathcal{C}) &\equiv (\mathcal{A} \vee \mathcal{B})\&(\mathcal{A} \vee \mathcal{C})\\
\mathcal{A}\&\bot &\equiv \bot\\
\mathcal{A} \vee \bot &\equiv \mathcal{A}\\
\mathcal{A}\&\top &\equiv \mathcal{A}\\
\mathcal{A} \vee \top &\equiv \top\\
\neg(\neg\mathcal{A}) &\equiv \mathcal{A}\\
\neg(\mathcal{A}\&\mathcal{B}) &\equiv \neg\mathcal{A} \vee \neg\mathcal{B}\\
\neg(\mathcal{A} \vee \mathcal{B}) &\equiv \neg\mathcal{A}\&\neg\mathcal{B}\\
\mathcal{A}\&\neg\mathcal{A} &\equiv \bot\\
\mathcal{A} \vee \neg\mathcal{A} &\equiv \top\\
\mathcal{A} \vee (\mathcal{A}\&\mathcal{B}) &\equiv \mathcal{A}\\
\mathcal{A}\&(\mathcal{A} \vee \mathcal{B}) &\equiv \mathcal{A}
\end{aligned}
\tag{1.1}
$$

и дополнительно к этим формулам еще две

$$
\begin{aligned}
(\mathcal{A} \Rightarrow \mathcal{B}) &\equiv (\neg\mathcal{A} \vee \mathcal{B})\\
(\mathcal{A} \Rightarrow \mathcal{B}) &\equiv \neg(\mathcal{A}\&\neg\mathcal{B}).
\end{aligned}
\tag{1.2}
$$

1.1.7. Алгебра Буля

Представим теперь теорию, которая имеет переменные, формулы, построенные из атомарных формул (переменных) с помощью трех операций $\sqcap, \sqcup, \sim$ – аналогов логических связок $\&, \vee, \neg$ и аналогов $1, 0$ двух истинностных значений $\top, \bot$. Если формулы (1.1) при замене в них $\&, \vee, \neg, \top, \bot$ на $\sqcap, \sqcup, \sim, 1, 0$ остаются справедливыми, то мы имеем новую абстрактную алгебру, называемую *алгеброй Буля*.

Таким образом, алгебра высказываний – это пример алгебры Буля, обязанной своим происхождением логике Аристотеля.

Алгебра высказываний – это логическая булева алгебра. Существуют и не логические булевы алгебры. Примером является теория множеств Кантора [4]. Переменные – это подмножества множества X. Роль операций $\sqcap, \sqcup, \sim$ играют пересечение $\cap$, объединение $\cup$ и дополнение $\setminus$ (до X), а вместо $0, 1$ надо взять пустое множество $\emptyset$ и само X.

1.1.8. Истинные и общезначимые формулы

Формула $\mathcal{A}(X_1, ..., X_n)$ называется *истинной* (выполнимой), если существует набор значений переменных $X_{10}, ..., X_{n0}$ такой, что $\mathcal{A}(X_{10}, ..., X_{n0}) = \top$.

Формула $\mathcal{A}(X_1, ..., X_n)$ называется *общезначимой*, если при любом наборе переменных она истинна. Для общезначимой формулы используется символическая запись

$$\models \mathcal{A}.$$

1.1.9. Проблема разрешимости

Пусть дана произвольная формула $\mathcal{A}(X_1, ..., X_n)$. Можно ли как-то проверить, что она является *общезначимой*? Если существует такой способ (алгоритм), позволяющий в конечное число шагов убедиться в этом, то говорят, что проблема проверки общезначимости формул алгебры высказываний *разрешима*.

Теорема 1.3. *Проблема разрешимости в алгебре высказываний имеет положительное решение.*

Доказательство.

Рассматривая набор переменных $(X_1, ..., X_n)$ на множестве $\{\bot, \top\}$, имеем 2^n возможных комбинаций. Для каждой комбинации

легко вычислить истинностное значение формулы $\mathcal{A}$. Это можно сделать, написав программу для компьютера. Найдя все значения формулы, мы узнаем всегда ли она истинна. Если «да», то формула $\mathcal{A}$ общезначима.

Теорема 1.3 доказана.

Легко понять, что на практике проверить тождественную истинность формулы бывает очень сложно, если она имеет большое число переменных. На вычисление истинностных значений формулы может потребоваться слишком много машинного времени.

1.1.10. Логическое следствие

Пусть даны формулы $\mathcal{A}_1, ..., \mathcal{A}_m, \mathcal{B}$. Формула $\mathcal{B}$ является *логическим следствием* формул $\mathcal{A}_1, ..., \mathcal{A}_m$, если, придавая значения переменным $X_1, ..., X_n$, от которых зависят все рассматриваемые формулы, всякий раз, когда истинны одновременно все формулы $\mathcal{A}_1, ..., \mathcal{A}_m$, истинна и формула $\mathcal{B}$.

Для логического следствия используется запись

$$\mathcal{A}_1, ..., \mathcal{A}_m \models \mathcal{B}.$$

Для проверки наличия логического следования достаточно построить истинностную таблицу.

Пример 1.1. Проверить, что

$$X, Y, (Z\&X \Rightarrow \neg Y) \models \neg Z. \tag{1.3}$$

Имеем истинностную таблицу:

X	Y	Z	$Z\&X \Rightarrow \neg Y$	$\neg Z$
$\top$	$\top$	$\top$	$\bot$	$\bot$
$\top$	$\top$	$\bot$	$\top$	$\top$

Из второй строки видно, что (1.3) есть логическое следствие.

Теорема 1.4 (о дедукции. Эрбран (1930)). *Если* $\mathcal{A} \models \mathcal{B}$, *то* $\models (\mathcal{A} \Rightarrow \mathcal{B})$.

Доказательство.

Используем таблицу истинности для связки $\Rightarrow$. Из условия $\mathcal{A} \models \mathcal{B}$ вытекает, что если $\mathcal{A} = \top$, то $\mathcal{B} = \top$. Но тогда $\models (\mathcal{A} \Rightarrow \mathcal{B})$, ибо случай, когда $\mathcal{A} = \top$, $\mathcal{B} = \bot$, исключен.

Теорема 1.4 доказана.

Следствие 1.1. $\mathcal{A}_1, ..., \mathcal{A}_m \models \mathcal{B}$ *тогда и только тогда, когда* $\models (\mathcal{A}_1\&...\&\mathcal{A}_m \Rightarrow \mathcal{B})$.

1.1.11. Силлогизмы

Силлогизм – это правило, позволяющее из (истинных) высказываний получать новые (истинные) высказывания.

Приведем неполный список силлогизмов:

$$\frac{\mathcal{A},\ \mathcal{A}\Rightarrow\mathcal{B}}{\mathcal{B}}\quad(\textbf{Modus ponens})$$

$$\frac{\mathcal{A}\Rightarrow\mathcal{B},\ \neg\mathcal{B}}{\neg\mathcal{A}}\quad(\textbf{Modus tollens})$$

$$\frac{\mathcal{A}\Rightarrow\mathcal{B},\ \mathcal{B}\Rightarrow\mathcal{C}}{\mathcal{A}\Rightarrow\mathcal{C}}$$

$$\frac{\mathcal{A},\ \mathcal{B}}{\mathcal{A}\&\mathcal{B}}$$

$$\frac{\mathcal{A}\&\mathcal{B}}{\mathcal{A},\ \mathcal{B}}$$

$$\frac{\mathcal{A}\vee\mathcal{B},\ \neg\mathcal{B}}{\mathcal{A}}$$

$$\frac{\neg(\mathcal{A}\&\mathcal{B}),\ \mathcal{A}}{\neg\mathcal{B}}$$

$$\frac{\mathcal{A}\Rightarrow(\mathcal{B}\Rightarrow\mathcal{C})}{\mathcal{B}\Rightarrow(\mathcal{A}\Rightarrow\mathcal{C})}$$

$$\frac{\mathcal{A}\Rightarrow(\mathcal{B}\Rightarrow\mathcal{C})}{\mathcal{A}\&\mathcal{B}\Rightarrow\mathcal{C}}$$

$$\frac{\mathcal{A}\&\mathcal{B}\Rightarrow\mathcal{C}}{\mathcal{A}\Rightarrow(\mathcal{B}\Rightarrow\mathcal{C})}$$

1.2. Логика предикатов

Логика предикатов изобретена Готлобом Фреге. «Он считал ее универсальным языком, в котором можно было бы представить систематически и математически точным образом любую возможную форму рационального мышления, которая могла бы стать частью дедуктивного мышления» [38].

1.2.1. Предикаты и формулы

Логика предикатов[6] – это расширение возможностей логики высказываний, позволяющее строить высказывания с учетом *свойств* изучаемых объектов или отношений между ними.

Одноместный предикат $P(x)$ – это утверждение об объекте x, где x рассматривается как переменная. Иначе говоря, если в $P(x)$ вместо x подставить конкретный изучаемый объект a, то получаем высказывание, принадлежащее алгебре высказываний. В таком случае $P(a) = \top$ или $P(a) = \bot$.

Многоместный предикат $P(x_1, ..., x_n)$ – это утверждение об объектах $x_1, ..., x_n$, где $x_1, ..., x_n$ рассматриваются как переменные. Следовательно, при подстановке $x_1 = a_1, ..., x_n = a_n$ получим высказывание $P(a_1, ..., a_n)$, являющееся истинным или ложным.

Расширение логики высказываний до логики предикатов получается за счет включения в формулы утверждений, являющихся предикатами. Но при этом, поскольку предикаты относятся к изучаемым объектам, мы обязаны включить в теорию и сами объекты $a_1, ...a_n, ...$ Поэтому, чтобы дать определение формул в логике предикатов, необходимо уточнить принципы описания в логике предикатов объектов. Делается это с помощью понятия *терм*.

Имеем следующее определение терма:

1) *переменные* $x_1, ..., x_n, ...$ *для объектов – это термы;*
2) *если* $f(\cdot, ..., \cdot)$ *функция* n*-переменных, ставящая в соответствие изучаемым объектам объект, и* $t_1, ..., t_n$ *термы, то* $f(t_1, ..., t_n)$ *терм.*

Теперь можно дать определение формулы:

1) *если* $P(\cdot, ..., \cdot)$ n*-местный предикат, а* $t_1, ..., t_n$ *термы, то* $P(t_1, ..., t_n)$ *(атомарная) формула;*
2) *если* $\mathcal{A}$ *и* $\mathcal{B}$ *формулы, то* $(\mathcal{A}\&\mathcal{B})$, $(\mathcal{A} \vee \mathcal{B})$, $(\mathcal{A} \Rightarrow \mathcal{B})$ *формулы;*
3) *если* $\mathcal{A}$ *формула, то* $\neg\mathcal{A}$ *формула;*
4) *если* $\mathcal{A}(x)$ *формула, содержащая переменную* x*, то*

$$\forall x \mathcal{A}(x), \quad \exists x \mathcal{A}(x) \tag{1.4}$$

формулы.

[6] *Предика́т* [лат. praedicatum] – 1) сказуемое; 2) логическое сказуемое; то, что в утверждении высказывается об объекте [40].

Обращают внимание на себя необычные значки в формуле (1.4). Это *кванторы.* Знак $\forall$ называется *квантором всеобщности*, а знак $\exists$ называется *квантором существования.* Ввел их в логику Ч.С.Пирс[7], хотя идея квантификации принадлежит Г.Фреге.

Символ $\forall x$ интерпретируется как фраза «для всех x», соответственно $\exists x$ – «существует x». Не стоит много говорить о том, что введение кванторов существенно обогащает *язык* и, следовательно, возможности логики предикатов.

Переменная x, входящая в формулу $\mathcal{A}$, называется *связанной*, если она находится под действием квантора $\forall x$ или $\exists x$. В противном случае, переменная x в формуле $\mathcal{A}$ является *свободной.* Например, в формулах

$$\exists x(x = y),$$
$$\forall x \mathcal{B}(x, y)$$

переменная x связанная, а переменная y свободная.

Ясно, что формула без свободных переменных является высказыванием.

1.2.2. Интерпретации

За каждой формулой скрывается нечто, в связи с чем она была написана и о чем она повествует, то есть скрывается ее *содержание.* Содержательная часть формул, их смысл, [8] относится к специальному разделу логики, называемому *семантикой.*[9] Выяснить содержание формулы можно обращаясь к реальному миру предметов. Делается это посредством *интерпретации* формулы.

Интерпретация состоит в указании конкретного непустого множества M, называемого областью интерпретации, и некоторого правила, по которому каждому n-местному предикату $P(\cdot, ..., \cdot)$ ставится в соответствие n-местное отношение $\overline{P} \subset M^n$ в M, а функциям (операциям) n переменных $f(\cdot, ..., \cdot)$, участвующим в определении термов, конкретные функции (операции) $\overline{f} : M^n \to M$ на M.

[7] По другим данным кванторы ввел Пеано [31, с.14].

[8] *Смысл* – это внутреннее содержание, значение формулы, постигаемое разумом.

[9] *Семантика* [гр. sëmanticos] – 1) смысловая сторона единиц языка – слов, частей слова, словосочетаний; 2) раздел логики, исследующий отношения логических знаков и составленных из них выражений (формул) к понятиям и предметам действительности [40].

При заданной интерпретации переменные, входящие в формулы, мыслятся пробегающими область M.

Формально *интерпретация* – это структура вида[10]

$$\mathfrak{M} =< M; \overline{P}_1, ..., \overline{P}_p; \overline{f}_1, ..., \overline{f}_q > .$$

Содержанием при интерпретации формулы наполняются благодаря тому, что элементы множества M уже привязаны к конкретной реальности, знакомы и понятны. Например, в случае, когда $M = \mathbb{R}$, т.е. является множеством действительных чисел, у нас не появляется чувства беспокойства, под формулами мы понимаем сведения из математического анализа, который прочно привязан к практической деятельности инженеров, физиков, химиков и т.д. Следовательно, нам становится ясным, когда формула при определенной фиксации своих переменных *истинна*, когда *ложна.*

1.2.3. Истинность и выполнимость формул. Модели, общезначимость, логическое следствие

Рассмотрим формулу $\mathcal{A}(x_1, ..., x_n)$ и некоторую ее интерпретацию $\mathfrak{M}$.

Формула $\mathcal{A}(x_1, ..., x_n)$ называется *истинной в интерпретации* $\mathfrak{M} =< M; \overline{P}_1, ..., \overline{P}_p; \overline{f}_1, ..., \overline{f}_q >$, если она истинна при любом выборе $x_1 = a_1, ..., x_n = a_n$, где $a_1, ..., a_n \in M$.

Формула называется *выполнимой в интерпретации* $\mathfrak{M}$, если она истинна при некотором выборе переменных $x_1 = a_1, ..., x_n = a_n$, где $a_1, ..., a_n \in M$.

Интерпретация $\mathfrak{M}$ называется *моделью* для совокупности формул $\mathcal{A}_1, ..., \mathcal{A}_m$, если они истинны в $\mathfrak{M}$. Символически это записывается в виде

$$\mathfrak{M} \models \mathcal{A}_1, ..., \mathcal{A}_m.$$

Формула $\mathcal{A}$ *общезначима*, если она истинна в любой интерпретации. Для общезначимой формулы пишем

$$\models \mathcal{A}.$$

[10] О понятии математической структуры см. в книге Н.Бурбаки «Теория множеств».

Если формула $\mathcal{A}$ общезначима, то формула $\neg\mathcal{A}$ называется *логически ложной*, или *противоречием.*

Пусть даны две формулы $\mathcal{A}(x_1, ..., x_n)$ и $\mathcal{B}(x_1, ..., x_n)$. Формула $\mathcal{B}$ является *логическим следствием* формулы $\mathcal{A}$, если во всякой интерпретации формула $\mathcal{B}$ выполнена на каждом наборе переменных $x_1 = a_1, ..., x_n = a_n$, на котором выполнена формула $\mathcal{A}$. Символически для логического следствия используют обозначение

$$\mathcal{A} \models \mathcal{B}.$$

Теорема 1.5. *$\mathcal{A} \models \mathcal{B}$ тогда и только тогда, когда $\models (\mathcal{A} \Rightarrow \mathcal{B})$.*

Формулы $\mathcal{A}(x_1, ..., x_n)$ и $\mathcal{B}(x_1, ..., x_n)$ называются *равносильными*, если $\mathcal{A} \models \mathcal{B}$ и $\mathcal{B} \models \mathcal{A}$. Для равносильных формул используется запись: $\mathcal{A} \equiv \mathcal{B}$.

Примеры равносильных формул:

$$\begin{gathered}
\forall x\mathcal{A}(x)\&\mathcal{B} \equiv \forall x[\mathcal{A}(x)\&\mathcal{B}] \\
\exists x\mathcal{A}(x)\&\mathcal{B} \equiv \exists x[\mathcal{A}(x)\&\mathcal{B}] \\
\forall x\mathcal{A}(x) \vee \mathcal{B} \equiv \forall x[\mathcal{A}(x) \vee \mathcal{B}] \\
\exists x\mathcal{A}(x) \vee \mathcal{B} \equiv \exists x[\mathcal{A}(x) \vee \mathcal{B}] \\
(\forall x\mathcal{A}(x) \Rightarrow \mathcal{B}) \equiv \exists x[\mathcal{A}(x) \Rightarrow \mathcal{B}] \\
(\exists x\mathcal{A}(x) \Rightarrow \mathcal{B}) \equiv \forall x[\mathcal{A}(x) \Rightarrow \mathcal{B}] \\
(\mathcal{B} \Rightarrow \forall x\mathcal{A}(x)) \equiv \forall x[\mathcal{B} \Rightarrow \mathcal{A}(x)] \\
(\mathcal{B} \Rightarrow \exists x\mathcal{A}(x)) \equiv \exists x[\mathcal{B} \Rightarrow \mathcal{A}(x)] \\
\neg\forall x\mathcal{A}(x) \equiv \exists x\neg\mathcal{A}(x) \\
\neg\exists x\mathcal{A}(x) \equiv \forall x\neg\mathcal{A}(x).
\end{gathered} \tag{1.5}$$

1.2.4. Готлоб Фреге

Готлоб Фреге (Frege) (1848 - 1925) – немецкий логик, математик и философ, основоположник логицизма. Дал первую аксиоматику логики высказываний и предикатов, построил первую систему формализованной арифметики. Один из основоположников логической семантики.

«В неопубликованной при жизни работе «Новая попытка обоснования арифметики» Фреге пришел к выводу, что «арифметика и

геометрия и, следовательно, вся математика, проистекают из одного и того же, а именно геометрического источника познания, который тем самым обретает степень подлинного математического источника познания, при этом, естественно, логический источник познания постоянно выступает вместе с ним» (см.: Frege G. Schriften zur Logik. Aus dem Nachlass. Berlin, 1973. S. 243).

Заслуживает внимания видение рассматриваемой проблемы известным геометром А.Д.Александровым. Он считал, что геометрия в своем существе есть не что иное, как органическое соединение строгой логики с наглядным представлением. Наглядное представление в геометрии пронизывается строгой логикой, а логика оживляется наглядным представлением» [30].

Рис. 1.3: Г.Фреге (1848-1925).

1.2.5. Сколемовские функции и сколемизация формул

Формулу $\mathcal{A}$ с помощью тождеств (1.5) можно привести к равносильной формуле в *предваренной форме*, в которой кванторы $\exists, \forall$ не перемешиваются, а встречаются группами и все «вынесены влево», т.е. либо к виду

$$\exists x_1 ... \exists x_{n_1} \forall y_1 ... \forall y_{n_2} \exists z_1 ... \exists z_{n_3} ... \mathcal{B}(x_1, ..., x_{n_1}, y_1, ..., y_{n_2}, z_1, ..., z_{n_3}, ...),$$

либо к виду

$$\forall u_1 ... \forall u_{n_1} \exists v_1 ... \exists v_{n_2} \forall w_1 ... \forall w_{n_3} ... \mathcal{B}(u_1, ..., u_{n_1}, v_1, ..., v_{n_2}, w_1, ..., w_{n_3}, ...),$$

где в формуле $\mathcal{B}$ нет кванторов. Легко добиться, чтобы последними стояли кванторы существования. Для этого используется тождество:

$$\mathcal{B}(x_1, ..., z_1, ...) \equiv \exists u[\mathcal{B}(x_1, ..., z_1, ...) \& \mathcal{I}(u)],$$

где $\mathcal{I}(u)$ – произвольная общезначимая формула.

Будем теперь «снимать» в формуле $\mathcal{A}$ последовательно группы кванторов слева направо, заменяя их на функции. Путеводной здесь является идея, что пара кванторов $\forall u \exists v$ – это функция

$v = f(u)$. Следовательно, набору кванторов $\forall y_1 ... \forall y_{n_2} \exists z_i$ отвечает функция $z_i = g_i(y_1, ..., y_{n_2})$. Если самой левой группой кванторов являются кванторы существования $\exists x_1 ... \exists x_{n_1}$, то им соответствуют постоянные функции, которые сводятся к заданию их значений $a_1, ..., a_{n_1}$. *Обратим внимание на то, что функция, отвечающая квантору существования, должна зависить от всех предыдущих переменных, связанных кванторами всеобщности.*

Сняв группу кванторов в формуле $\mathcal{A}$, мы в $\mathcal{B}$ оставляем переменные $y_1, ..., y_{n_2}$, по которым стояли кванторы всеобщности, а вместо следующих за ними переменных $z_1, ...$, связанных кванторами существования, подставляем полученные функции $g_i(y_1, ..., y_{n_2})$.

В результате имеем набор функций

$$\begin{aligned} a_1, ..., a_{n_1}, g_1(y_1, ..., y_{n_2}), ..., g_{n_3}(y_1, ..., y_{n_2}), ... \\ ..., h_1(y_1, ..., w_{n_{k-1}}), ...h_{n_k}(y_1, ..., w_{n_{k-1}}) \end{aligned} \tag{1.6}$$

и формулу

$$\begin{aligned} \mathcal{B}(a_1, ..., a_{n_1}, y_1, ..., y_{n_2}, g_1(y_1, ..., y_{n_2}), ..., g_{n_3}(y_1, ..., y_{n_2}), ... \\ ..., h_1(y_1, ..., w_{n_{k-1}}), ..., h_{n_k}(y_1, ..., w_{n_{k-1}})). \end{aligned} \tag{1.7}$$

Функции (1.6) называются *сколемовскими* (разрешающими) в *интерпретации* $\mathfrak{M}$, если формула (1.7) истинна в $\mathfrak{M}$. Очевидно, что формула (1.7) истинна в $\mathfrak{M}$ тогда и только тогда, когда

$$\mathfrak{M} \models \mathcal{A}_s,$$

где

$$\mathcal{A}_s = \forall y_1 ... \forall y_{n_2} ... \forall w_1 ... \forall w_{n_{k-1}} \mathcal{B}(a_1, ..., a_{n_1}, y_1, ..., g_1(y_1, ..., y_{n_2}), ...$$

$$..., w_1, ..., h_1(y_1, ..., w_{n_{k-1}}), ..., h_{n_k}(y_1, ..., w_{n_{k-1}})).$$

Формула $\mathcal{A}_s$ не содержит кванторов существования и является $\forall-$формулой, полученной в результате *сколемизации* исходной формулы $\mathcal{A}$.

Теорема 1.6 (Эрбран). *Формула $\mathcal{A}$ имеет модель $\mathfrak{M}$ тогда и только тогда, когда для нее существуют сколемовские функции.*

Доказательство.

Рассмотрим только случай формулы, начинающейся с группы кванторов существования [11]. Применим метод математической индукции по числу групп кванторов.

[11] Полное доказательство см. в [32].

Если

$$\mathcal{A} = \exists x_1 ... \exists x_n \mathcal{B}(x_1, ..., x_n),$$

то, как следует из определения модели, $\mathfrak{M} \models \mathcal{A}$ тогда и только тогда, когда существует набор элементов $a_1, ..., a_n \in M$, для которого $\mathcal{A}(a_1, ..., a_n) = \top$. Но набор $a_1, ..., a_n$ – это как раз сколемовские функции для $\mathcal{A}$.

Пусть теорема верна для формул с p группами кванторов и дана формула

$$\mathcal{A} = \exists x_1 ... \exists x_{n_1} \forall y_1 ... \forall y_{n_2} ... \mathcal{B}(x_1, ..., y_1, ...), \tag{1.8}$$

содержащая $(p+1)$ группу кванторов.

Для того чтобы формула $\mathcal{A}$ была истинна в интерпретации $\mathfrak{M}$, необходимо и достаточно, чтобы существовали $a_1, ..., a_n \in M$ такие, что формула

$$\mathcal{C} = \forall y_1 ... \forall y_{n_2} ... \mathcal{B}(a_1, ..., a_{n_1}, y_1, ...)$$

была истинна в $\mathfrak{M}$. Но формула $\mathcal{C}$ содержит уже p групп кванторов. Поэтому по индукционному предположению $\mathfrak{M} \models \mathcal{C}$ тогда и только тогда, когда существуют сколемовские функции

$$g_1(y_1, ..., y_{n_2}), ..., g_{n_3}(y_1, ..., y_{n_2}), ...$$

для формулы $\mathcal{C}$. Легко видеть, что функции

$$a_1, ..., a_{n_1}, g_1(y_1, ..., y_{n_2}), ..., g_{n_3}(y_1, ..., y_{n_2}), ...$$

являются сколемовскими для $\mathcal{A}$. Для них формула

$$\mathcal{B}(a_1, ..., a_{n_1}, y_1, ..., y_{n_2}, g_1(y_1, ..., y_{n_2}), ...)$$

истинна в $\mathfrak{M}$. Но это означает истинность формулы $\mathcal{A}$ в $\mathfrak{M}$.

Теорема 1.6 доказана.

Пример 1.2. Провести сколемизацию формулы

$$\mathcal{A} = \exists x \exists y \forall z \forall u \exists v \mathcal{B}(x, y, z, u, v).$$

Переменные x, y заменяем на константы a, b соответственно. Переменные z, u оставляем, а вместо v вводим сколемовскую функцию $f(z, u)$. Получаем формулы

$$\mathcal{B}(a, b, z, u, f(z, u))$$

и

$$\mathcal{A}_s(a, b) = \forall z \forall u \mathcal{B}(a, b, z, u, f(z, u)).$$

1.3. Метод резолюций

1.3.1. Метод резолюций в логике высказываний

Предположим, что дана формула логики высказываний $\mathcal{A}$. Как проверить общезначимость формулы $\mathcal{A}$? Это задача решается с помощью *метода резолюций* Дж. Робинсона, к изложению которого мы приступаем[12].

Очевидно, что $\models \mathcal{A}$ тогда и только тогда, когда является противоречием формула $\neg\mathcal{A}$.

Формулу $\neg\mathcal{A}$ приводим к *конъюнктивной нормальной форме* (КНФ). КНФ – это формула, равносильная данной формуле и записанная в виде конъюнкции элементарных дизъюнкций, построенных на пропозициональных переменных, т.е. в данном случае

$$\neg\mathcal{A} = \mathcal{D}_1 \& ... \& \mathcal{D}_p,$$

где $\mathcal{D}_i$ есть дизъюнкция конечного числа пропозициональных переменных или их отрицаний. Тем самым мы формируем множество *дизъюнктов* $K = \{\mathcal{D}_1, ..., \mathcal{D}_p\}$.

Два дизъюнкта этого множества $\mathcal{D}_i$ и $\mathcal{D}_j$, содержащие пропозициональные переменные с противоположными знаками, – *контрарные литералы*, т.е., к примеру, Y и $\neg Y$, и, следовательно, $\mathcal{D}_i = \mathcal{D}'_i \vee Y$, $\mathcal{D}_j = \mathcal{D}'_j \vee \neg Y$, формируют третий дизъюнкт – *резольвенту* $\mathcal{D}'_i \vee \mathcal{D}'_j$, в которой исключены контрарные литералы:

$$R\frac{\mathcal{D}_i,\ \mathcal{D}_j}{\mathcal{D}'_i \vee \mathcal{D}'_j}.$$

В частности, если $\mathcal{D}_i = Y$, $\mathcal{D}_j = \neg Y$, то резольвента для них – это дизъюнкция, ничего не содержащая (отсутствуют $\mathcal{D}'_i$ и $\mathcal{D}'_j$). Ее называют *пустой резольвентой* и обозначают знаком $\Box$.

Теорема 1.7. *Резольвента $\mathcal{D}'_i \vee \mathcal{D}'_j$, – это логическое следствие дизъюнктов $\mathcal{D}_i$ и $\mathcal{D}_j$, т.е.*

$$\mathcal{D}_i, \mathcal{D}_j \models (\mathcal{D}'_i \vee\ \mathcal{D}'_j).$$

[12]Одним из первых реализовал метод Эрбрана на вычислительной машине Гилмор. Однако его программа справилась с доказательством нескольких простых формул, но столкнулась с большими трудностями в доказательстве других формул логики первого порядка.

Доказательство.
Достаточно доказать, что

$$\models \mathcal{D}_i \& \mathcal{D}_j \Rightarrow (\mathcal{D}'_i \vee \mathcal{D}'_j) \tag{1.9}$$

или

$$\models (\mathcal{D}'_i \vee Y) \& (\mathcal{D}'_j \vee \neg Y) \Rightarrow (\mathcal{D}'_i \vee \mathcal{D}'_j).$$

Но

$$[(\mathcal{D}'_i \vee Y) \& (\mathcal{D}'_j \vee \neg Y]) \equiv [(\mathcal{D}'_i \vee Y) \& (\neg Y \vee \mathcal{D}'_j)] \equiv$$
$$\equiv [(\mathcal{D}'_i \vee Y) \& (Y \Rightarrow \mathcal{D}'_j)].$$

Поэтому достаточно убедиться, что

$$\models [(\mathcal{D}'_i \vee Y) \& (Y \Rightarrow \mathcal{D}'_j) \Rightarrow (\mathcal{D}'_i \vee \mathcal{D}'_j)]. \tag{1.10}$$

Формула (1.10) получается из общезначимой[13] формулы

$$\models [(\mathcal{A} \vee \mathcal{B}) \& (\mathcal{B} \Rightarrow \mathcal{C}) \Rightarrow (\mathcal{A} \vee \mathcal{C})] \tag{1.11}$$

подстановками:

$$\mathcal{A} \to \mathcal{D}'_i, \quad \mathcal{B} \to Y, \quad \mathcal{C} \to \mathcal{D}'_j.$$

Следовательно, формула (1.10) общезначима. Поэтому общезначима и формула (1.9).

Теорема 1.7 доказана.

Неоднократно применяя правило получения резольвент к множеству дизъюнкт, стремятся получить пустой дизъюнкт $\square$.

Наличие пустого дизъюнкта $\square$ свидетельствует о получении противоречия, поскольку пустая резольвента получается из двух противоречащих друг другу дизъюнктов Y и $\neg Y$, каждый из которых логическое следствие формулы $\neg\mathcal{A}$ в соответствии с правилом

$$\frac{\mathcal{A} \& \mathcal{B}}{\mathcal{A},\ \mathcal{B}}.$$

Как следует из следующей теоремы, получение противоречия с помощью формулы $\neg\mathcal{A}$, означает общезначимость формулы $\mathcal{A}$.

Теорема 1.8 (**доказательство от противного**). *Если* $\Gamma, \neg\mathcal{A} \models \bot$, *где* Γ *множество формул, то* $\Gamma \models \mathcal{A}$.

[13] В чем легко убедиться.

Доказательство.
Из теоремы дедукции

$$(\Gamma, \neg\mathcal{A} \models \bot) \Leftrightarrow \models (\Gamma\&\neg\mathcal{A} \Rightarrow \bot).$$

Однако

$$(\Gamma\&\neg\mathcal{A} \Rightarrow \bot) \equiv (\neg(\Gamma\&\neg\mathcal{A}) \vee \bot) \equiv \neg(\Gamma\&\neg\mathcal{A}) \equiv (\Gamma \Rightarrow \mathcal{A}).$$

Поэтому $\models (\Gamma \Rightarrow \mathcal{A})$.
Теорема 1.8 доказана.

Изложим по шагам

Алгоритм метода резолюций.

Шаг 1. Принять отрицание формулы $\mathcal{A}$, т.е. $\neg\mathcal{A}$.

Шаг 2. Привести формулу $\neg\mathcal{A}$ к конъюнктивной нормальной форме.

Шаг 3. Выписать множество ее дизъюнктов:

$$K = \{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_p\}.$$

Шаг 4. Выполнить анализ пар множества K по правилу: если существуют дизъюнкты $\mathcal{D}_i$ и $\mathcal{D}_j$, один из которых ($\mathcal{D}_i$) содержит литерал X, а другой ($\mathcal{D}_j$) – контрарный литерал $\neg X$, то нужно соединить эту пару логической связкой дизъюнкции ($\mathcal{D}_i \vee \mathcal{D}_j$) и сформировать новый дизъюнкт – *резольвенту*, исключив контрарные литералы X и $\neg X$.

Шаг 5. Если в результате соединения дизъюнктов, содержащих контрарные литералы, будет получена пустая резольвента – $\square$, то результат достигнут (доказательство подтвердило противоречие), в противном случае включить резольвенту в множество дизъюнктов K и перейти к шагу 4.

При реализации указанного алгоритма возможны три случая:

- Среди текущего множества дизъюнктов нет резольвируемых. Это означает, что формула $\mathcal{A}$ не является общезначимой.
- На каком-то шаге получается пустая резольвента. Формула $\mathcal{A}$ общезначима.
- Процесс не останавливается, т.е. множество дизъюнкт пополняется все новыми резольвентами, среди которых нет пустых. В таком случае нельзя ничего сказать об общезначимости формулы $\mathcal{A}$.

Метод резолюций пригоден и для доказательства того, что формула $\mathcal{B}$ является логическим следствием формул $\mathcal{F}_1, ..., \mathcal{F}_n$, поскольку, как вытекает, из следствия 1.1,

$$\mathcal{F}_1, ..., \mathcal{F}_n \models \mathcal{B} \Leftrightarrow \models (\mathcal{F}_1 \& ... \& \mathcal{F}_n \Rightarrow \mathcal{B}).$$

Для того чтобы «запустить» алгоритм метода резолюций, нужно воспользоваться тождеством

$$(\mathcal{F}_1 \& ... \& \mathcal{F}_n \Rightarrow \mathcal{B}) \equiv \neg(\mathcal{F}_1 \& ... \& \mathcal{F}_n \& \neg\mathcal{B}).$$

Следовательно, формула $\mathcal{A} = \neg(\mathcal{F}_1 \& ... \& \mathcal{F}_n \Rightarrow \mathcal{B})$ общезначима, если формула $\neg\mathcal{A} = (\mathcal{F}_1 \& ... \& \mathcal{F}_n \& \neg\mathcal{B})$ является противоречием. Далее применяем описанный по шагам метод резолюций к формуле $\mathcal{A}$.

Пример 1.3. Доказать, что

$$X, B, (C \& X \Rightarrow \neg B) \models \neg C. \tag{1.12}$$

Имеем

$$\mathcal{A} = X \& B \& (C \& X \Rightarrow \neg B) \Rightarrow \neg C,$$
$$\mathcal{A} = \neg[X \& B \& (C \& X \Rightarrow \neg B) \& \neg(\neg C)]$$

и

$$\neg\mathcal{A} = X \& B \& (C \& X \Rightarrow \neg B) \& \neg(\neg C).$$

Так как

$$(C \& X \Rightarrow \neg B) \equiv (\neg C \vee \neg X \vee \neg B), \quad \neg(\neg C) \equiv C,$$

то

$$\neg\mathcal{A} = X \& B \& (\neg C \vee \neg X \vee \neg B) \& C.$$

Следовательно,

$$K = \{X, B, (\neg C \vee \neg X \vee \neg B), C\}.$$

Резольвента для $(\neg C \vee \neg X \vee \neg B)$ и C – дизъюнкт $(\neg X \vee \neg B)$. Поэтому

$$K = \{X, B, (\neg C \vee \neg X \vee \neg B), C, (\neg X \vee \neg B)\}.$$

Резольвента для X и $(\neg X \vee \neg B)$ – дизъюнкт $\neg B$, и

$$K = \{X, B, (\neg C \vee \neg X \vee \neg B), C, (\neg X \vee \neg B), \neg B\}.$$

Получаем пустую резольвенту $\square$ для дизъюнктов B и $\neg B$. Следовательно, логическое следствие (1.12) установлено.

Пример 1.4. Доказать, что

$$(C \& X \Rightarrow \neg B) \models \neg C.$$

Имеем

$$\mathcal{A} = (C \& X \Rightarrow \neg B) \Rightarrow \neg C,$$

$$\mathcal{A} = \neg[(C\&X \Rightarrow \neg B)\&\neg\neg C].$$

Следовательно,

$$\neg\mathcal{A} = (C\&X \Rightarrow \neg B)\&\neg(\neg C).$$

Ищем дизъюнкты:

$$(C\&X \Rightarrow \neg B) \equiv (\neg C \vee \neg X \vee \neg B), \quad \neg(\neg C) \equiv C.$$

Таким образом,

$$K = \{(\neg C \vee \neg X \vee \neg B), C\}.$$

Резольвента для $(\neg C \vee \neg X \vee \neg B)$ и C – это $(\neg X \vee \neg B)$. Поэтому

$$K = \{(\neg C \vee \neg X \vee \neg B), C, (\neg X \vee \neg B)\}.$$

Процесс останавливается. Пустой резольвенты нет. Формула $\neg C$ не является логическим следствием формулы $(C\&X \Rightarrow \neg B)$.

1.3.2. Метод резолюций в логике предикатов

Предположим, что дана формула $\mathcal{A}$. Как проверить общезначимость этой формулы? Применим ли метод резолюций? Применим, но с некоторыми дополнениями.

Если формула $\mathcal{A}$ не содержит предикатов и знаков операций (функций), то фактически имеем формулу исчисления высказываний, поскольку предикаты можно рассматривать как пропозициональные переменные. Следовательно, метод резолюций не требует существенных изменений. В случае, если формула $\mathcal{A}$ содержит кванторы, то ее надо привести к предваренной форме (§1.2.5), а затем устранить кванторы, вводя сколемовские функции. Появятся, естественно, при этом знаки операций (функций). Затем полученную формулу приводят к конъюнктивной нормальной форме и ищут резольвенты для дизъюнктов.

Что делать, если два контрарных литерала, в данном случае они имеют вид $P(..)$ и $\neg P(..)$, содержат два разных терма? Например, $\mathcal{D}_i = \mathcal{D}'_i \vee P(a,x)$ и $\mathcal{D}_j = \mathcal{D}'_j \vee \neg P(a, f(b))$. В этом случае проводят их *унификацию*, т.е. замену термов. В нашем примере она имеет вид:[14]:

$$\mathcal{D}_i < x|f(b) >= [\mathcal{D}'_i \vee P(a,x)] < x|f(b) >= \mathcal{D}'_i < x|f(b) > \vee P(a, f(b)),$$

[14] Символическая запись

$$\mathcal{A}(t_1) < t_1|t_2 >$$

означает замену в формуле $\mathcal{A}$ терма t_1 на терм t_2.

$$\mathcal{D}_j < x|f(b) >= [\mathcal{D}'_j \vee \neg P(a, f(b))] < x|f(b) >= \\ = \mathcal{D}'_j < x|f(b) > \vee \neg P(a, f(b)).$$

Далее ищется резольвента:

$$R\frac{\mathcal{D}_i,\ \mathcal{D}_j}{\mathcal{D}'_i < x|f(b) > \vee\ \mathcal{D}'_j < x|f(b) >}.$$

Сформулируем более строго правила замены термов и понятие унификации.

Замена – это пара вида $< x|t >$, где x – переменная, а t – терм. Применение замены $< x|t >$ к терму t_0, входящему в некоторую формулу, определяется индуктивно:

- $x < x|t >= t$, если x – переменная;
- $a < x|t >= a$, если a – константа (фиксированный объект из M при интерпретации $\mathfrak{M}$);
- $f(t_1, ..., t_n) < x|t >= f(t_1 < x|t >, ..., t_n < x|t >)$.

Таким образом, *унификация двух последовательностей термов* $t_1, ..., t_n$ и $\tau_1, ..., \tau_n$ – это замена $< x|t >$ такая, что

$$t_i < x|t >= \tau_i < x|t >\ \ (i = 1, ...n).$$

Резольвента для дизъюнктов $\mathcal{D}_i = \mathcal{D}'_i \vee P(t_1)$, $\mathcal{D}_i = \mathcal{D}'_i \vee \neg P(t_2)$ ищется с помощью унификации термов t_1, t_2:

$$R\frac{\mathcal{D}_i,\ \mathcal{D}_j}{\mathcal{D}'_i < x|t > \vee\ \mathcal{D}'_j < x|t >}.$$

Пример 1.5. Имеем два дизъюнкта

$$\mathcal{D}_1 = P(z) \vee \neg R(x), \quad \mathcal{D}_2 = \neg P(f(x)) \vee Q(y).$$

Очевидна замена

$$\mathcal{D}_1 < z|f(x) >= [P(z) \vee \neg R(x)] < z|f(x) >= P(f(x)) \vee \neg R(x),$$

$$\mathcal{D}_2 < z|f(x) >= [\neg P(f(x)) \vee Q(y)] < z|f(x) >= \neg P(f(x)) \vee Q(y).$$

Тогда резольвента для $\mathcal{D}_1$ и $\mathcal{D}_2$ есть дизъюнкт $\neg R(x) \vee Q(y)$.

Пример 1.6. Для дизъюнктов

$$\mathcal{D}_1 = P(f(x)) \vee \neg R(x),$$

$$\mathcal{D}_2 = \neg P(g(x)) \vee Q(y)$$

нет унификации.

Глава 2

Формальные теории (исчисления)

Формальная теория – это способ изложения логики без приписывания пропозициональным переменным, предикатам и формулам какого-либо значения. По замыслу создателей формальных теорий (Гильберт и др.) таким образом можно избежать многих неприятностей, возникающих при использовании в логике человеческого языка, допускающего двусмысленность, недосказанность, переиначивание исходно заложенного значения, смысла и т.д.

Формальная теория – это игра со знаками, игра со словами и предложениями, составленными из знаков. При этом имеется в виду, что правила составления слов из знаков и правила игры со словами и предложениями заранее оговорены, точно и строго прописаны. В основе формальной теории – *язык*, на котором «разговаривает теория». На первое место выходит *синтаксис*[1] этого языка, т.е. способ построения формул, в противопоставление *семантике*. [2]

[1] *Синтаксис* [< гр. syntaxis составление] – 1) характерные для конкретных языков средства и правила создания речевых единиц; 2) раздел грамматики, изучающий способы соединения слов в словосочетания и предложения, соединение предложений в сложные предложения [40].

[2] *Семантика* [< гр. semanticos обозначающий] – 1) смысловая сторона единиц языка, словосочетаний; 2) раздел логики, исследующий отношения логических знаков к понятиям и предметам действительности [40].

2.1. Определение формальной теории, или исчисления

Формальная теория $\mathcal{T}$ состоит из следующих компонент:

1. Множества знаков, образующих *алфавит* языка теории.

2. Множества слов, составленных из знаков алфавита, называемых *формулами*.

3. Подмножества формул всего множества формул, называемых *аксиомами*.

4. Множества *правил вывода*, с помощью которых из формул получают формулы.

В *язык* теории $\mathcal{T}$ входит алфавит и формулы. Количество аксиом может быть конечным или бесконечным. В последнем случае для наглядного представления они задаются с помощью *схем*. По схемам легко выписываются сами аксиомы. Под логическими аксиомами, как правило, понимают аксиомы *базовой логики*, над которой надстраиваются конкретные теории за счет добавления новых аксиом, отражающих специфику конкретной теории. Такие аксиомы называют *нелогическими*.

Обычно формулы состоят из конечного числа знаков. Но бывают теории с бесконечно длинными формулами, т.е. с формулами, содержащими бесконечное число знаков.

2.1.1. Доказательство. Непротиворечивость теории. Полнота теории

Формальное доказательство формулы $\mathcal{A}$ в теории $\mathcal{T}$ – это конечная последовательность формул

$$\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_n, \tag{2.1}$$

где каждая формула $\mathcal{A}_i$ является либо аксиомой, либо получена из предыдущих с помощью одного из правил вывода, а $\mathcal{A}_n$ – это формула $\mathcal{A}$.

Последняя формула в доказательстве (2.1) называется *теоремой*. Используется символическая запись для теорем [3]:

$$\vdash \mathcal{A}.$$

Формальная теория называется *полной*, если для всякого высказывания[4] $\mathcal{A}$ имеем:

$$\vdash \mathcal{A} \text{ или } \vdash \neg\mathcal{A}.$$

По замыслу создателя исчисления предикатов Фреге, любая правильно построенная формула, точнее высказывание, должна быть теоремой, т.е. должна быть доказываемым утверждением. Иначе говоря, исчисление высказываний и исчисление предикатов должны быть полными теориями. Ожидания Фреге оправдались (см. ниже теоремы 2.3 и 2.6). Но, как оказалось, более сильные теории, включающие арифметику, неполны. Это было доказано Гёделем (см. ниже теорема 2.9).

Формальная теория называется *непротиворечивой*, если в ней не является доказуемой формула

$$\mathcal{A}\&\neg\mathcal{A},$$

где $\mathcal{A}$ произвольное высказывание теории.

Смысл условия непротиворечивости в том, что оно может быть доказано средствами и в рамках самой формальной теории. Увы, таким способом, как выяснилось, невозможно установить даже непротиворечивость (формальной) арифметики.

2.2. Исчисление высказываний

2.2.1. Язык и правила вывода исчисления высказываний

Исчисление высказываний – это следующая формальная теория:

[3] Знак $\vdash$ читается как «доказуемо $\mathcal{A}$» или «$\mathcal{A}$ теорема». Знак $\vdash$ введен в 1879 г. немецким логиком Фреге.

[4] Высказывание в логике предикатов – это закрытая формула, т.е. формула, все переменные которой связаны кванторами.

1. **Алфавит**.

- Знаки пропозициональных переменных

$$X_1, X_2, ..., X_n, ...$$

- Логические связки

$$\vee,\ \&,\ \neg,\ \Rightarrow .$$

- Вспомогательные знаки

$$(\ ,\ ).$$

2. **Формулы.**

- Пропозициональная переменная есть (атомарная) формула.
- Если $\mathcal{A}$ и $\mathcal{B}$ формулы, то $(\mathcal{A}\&\mathcal{B})$, $(\mathcal{A}\vee\mathcal{B})$, $(\mathcal{A}\Rightarrow\mathcal{B})$ формулы.
- Если $\mathcal{A}$ формула, то $\neg\mathcal{A}$ формула.

3. **Аксиомы** (схемы Клини).

I

$\mathcal{A}\Rightarrow(\mathcal{B}\Rightarrow\mathcal{A})$

$(\mathcal{A}\Rightarrow(\mathcal{B}\Rightarrow\mathcal{C}))\Rightarrow((\mathcal{A}\Rightarrow\mathcal{B})\Rightarrow(\mathcal{A}\Rightarrow\mathcal{C}))$

II

$(\mathcal{A}\&\mathcal{B})\Rightarrow\mathcal{A}$

$(\mathcal{A}\&\mathcal{B})\Rightarrow\mathcal{B}$

$\mathcal{A}\Rightarrow(\mathcal{B}\Rightarrow(\mathcal{A}\&\mathcal{B}))$

III

$\mathcal{A}\Rightarrow(\mathcal{A}\vee\mathcal{B})$

$B\Rightarrow(\mathcal{A}\vee\mathcal{B})$

$(\mathcal{A}\Rightarrow\mathcal{C})\Rightarrow((\mathcal{B}\Rightarrow\mathcal{C})\Rightarrow((\mathcal{A}\vee\mathcal{B})\Rightarrow\mathcal{C}))$

IV

$(\mathcal{A}\Rightarrow\mathcal{B})\Rightarrow((\mathcal{A}\Rightarrow\neg\mathcal{B})\Rightarrow\neg\mathcal{A})$

$\neg\neg\mathcal{A}\Rightarrow\mathcal{A}.$

4. **Правила вывода.**

$$\frac{\mathcal{A},\ \mathcal{A}\Rightarrow\mathcal{B}}{\mathcal{B}}\ (modus\ ponens).$$

5. **Определение доказательства.**

Пусть Γ – множество формул. Формула $\mathcal{A}$ *выводима* из множества *гипотез* Γ, если существует конечная последовательность формул

$$\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_n, \tag{2.2}$$

где каждая формула $\mathcal{A}_i$ является либо аксиомой, либо гипотезой, либо получена из предыдущих с помощью правила вывода и $\mathcal{A}_n$ – это формула $\mathcal{A}$.

Последовательность (2.2) – это *вывод* или *доказательство* формулы $\mathcal{A}$ из множества гипотез Γ. Используется обозначение для вывода:

$$\Gamma \vdash \mathcal{A}.$$

Если $\Gamma = \emptyset$, то вывод называется доказательством формулы $\mathcal{A}$, а сама формула $\mathcal{A}$ в (2.2) называется *теоремой*. Используется символическая запись для теорем:

$$\vdash \mathcal{A}.$$

Имеет место

Теорема 2.1 (теорема дедукции). *Если*

$$\Gamma, \mathcal{A} \vdash \mathcal{B}, \text{ то } \Gamma \vdash (\mathcal{A} \Rightarrow \mathcal{B}).$$

Доказательство. См. [33, с.112].

2.2.2. Пример доказательства теоремы

Приведем пример (формального) доказательства теоремы.

Итак,

$$\vdash \mathcal{A} \Rightarrow \mathcal{A}.$$

Прежде чем изложить доказательство, надо сделать некоторые разъяснения. Формальное доказательство – это просто цепочка «голых» формул. Чтобы воспринять ее, нужно прокомментировать то, как эта цепочка возникает. Иначе говоря, приходится прибегать к русскому языку, помимо языка формального. В противном случае будем иметь подобно тому, что приписывают мифическим античным грекам, чертеж на песке – рисунок к доказательству теоремы из геометрии и слово «Смотри!»

Поэтому слева будем выписывать в столбик формальное доказательство, а справа давать комментарии к происходящему на *метаязыке*, т.е. в нашем случае – на русском языке.

Доказательство.

	Язык	**Метаязык**
1	$(\mathcal{A} \Rightarrow ((\mathcal{A} \Rightarrow \mathcal{A}) \Rightarrow \mathcal{A}))$	1-я аксиома, подстановка $\mathcal{B} \to (\mathcal{A} \Rightarrow \mathcal{A})$
2	$((\mathcal{A} \Rightarrow ((\mathcal{A} \Rightarrow \mathcal{A}) \Rightarrow \mathcal{A})) \Rightarrow$ $\Rightarrow ((\mathcal{A} \Rightarrow (\mathcal{A} \Rightarrow \mathcal{A})) \Rightarrow (\mathcal{A} \Rightarrow \mathcal{A})))$	2-я аксиома, подстановки: $\mathcal{B} \to (\mathcal{A} \Rightarrow \mathcal{A})$ $\mathcal{C} \to \mathcal{A}$
3	$((\mathcal{A} \Rightarrow (\mathcal{A} \Rightarrow \mathcal{A})) \Rightarrow (\mathcal{A} \Rightarrow \mathcal{A}))$	modus ponens к пунктам 1 и 2
4	$\mathcal{A} \Rightarrow (\mathcal{A} \Rightarrow \mathcal{A})$	1-я аксиома, подстановка $\mathcal{B} \to \mathcal{A}$
5	$\mathcal{A} \Rightarrow \mathcal{A}$	modus ponens к пунктам 3 и 4
		ч.т.д.

2.2.3. Полнота и непротиворечивость исчисления высказываний

Теорема 2.2. *Для исчисления высказываний справедливо утверждение:*

$$\models \mathcal{A} \text{ тогда и только тогда, когда } \vdash \mathcal{A}.$$

Доказательство. См. [33, с.117].

Теорема 2.3. *Исчисление высказываний – полная теория.*

Доказательство. Следует из утверждения «$\models \mathcal{A}$ влечет $\vdash \mathcal{A}$» теоремы 2.2.

Теорема 2.4. *Исчисление высказываний – непротиворечивая теория.*

Доказательство. Следует из утверждения «$\vdash \mathcal{A}$ влечет $\models \mathcal{A}$» теоремы 2.2. Действительно, если бы была доказуема формула $\mathcal{A} = \mathcal{B}\&\neg\mathcal{B}$, то было бы истинно утверждение $\mathcal{B}\&\neg\mathcal{B}$. Но последняя формула ложна.

2.3. Исчисление предикатов

2.3.1. Язык и правила вывода исчисления предикатов

Исчисление предикатов – это формальная теория, получаемая за счет добавления к исчислению высказываний новых знаков, понятия терма, новых типов формул и новых правил вывода.

1. **Дополнения в алфавит**.

- Знаки индивидуальных переменных

$$x_1, x_2 ..., x_n, ...$$

- Знаки индивидуальных констант

$$c_1, c_2 ..., c_n, ...$$

- Знаки предикатов

$$P_1^{(n_1)}\underbrace{(\cdot, ..., \cdot)}_{n_1 \text{ мест}},\ P_2^{(n_2)}\underbrace{(\cdot, ..., \cdot)}_{n_2 \text{ мест}}, ...$$

- Знаки операций

$$f_1^{(n_1)}\underbrace{(\cdot, ..., \cdot)}_{n_1 \text{ мест}},\ f_2^{(n_2)}\underbrace{(\cdot, ..., \cdot)}_{n_2 \text{ мест}}, ...$$

- Логические знаки (кванторы) $\forall$, $\exists$.

2. **Термы.**

1) переменные $x_1, ..., x_n, ...$ – это термы;

1) константы $c_1, ..., c_n, ...$ – это термы;

2) если $f_m^{(n)}\underbrace{(\cdot, ..., \cdot)}_{n \text{ мест}}$ – n-местный знак операции и $t_1, ..., t_n$ термы, то $f_m^{(n)}(t_1, ..., t_n)$ терм.

3. **Формулы.**

1) если $P_m^{(n)}\underbrace{(\cdot, ..., \cdot)}_{n \text{ мест}}$ – n-местный знак предиката и $t_1, ..., t_n$ – термы, то $P_m^{(n)}(t_1, ..., t_n)$ – (атомарная) формула;

2) если $\mathcal{A}$ и $\mathcal{B}$ формулы, то $(\mathcal{A}\&\mathcal{B})$, $(\mathcal{A}\vee\mathcal{B})$, $(\mathcal{A}\Rightarrow\mathcal{B})$ формулы;

3) если $\mathcal{A}$ формула, то $\neg\mathcal{A}$ формула;

4) если $\mathcal{A}(x)$ формула, содержащая переменную x, то

$$\forall x\mathcal{A}(x),\quad \exists x\mathcal{A}(x) \tag{2.3}$$

формулы.

4. **Дополнительные аксиомы.**

$$\forall x\mathcal{A}(x)\Rightarrow\mathcal{A}(t),$$
$$\mathcal{A}(t)\Rightarrow\exists x\mathcal{A}(x),$$

где $t=t(x_1,...,x_n)$ – терм и в формуле $\mathcal{A}$ нет кванторов $\forall x_i$, $\exists x_i$ $(i=1,...,n)$.

4. **Дополнительные правила вывода.**

$$\frac{\mathcal{B}\Rightarrow\mathcal{A}(x)}{\mathcal{B}\Rightarrow\forall x\mathcal{A}(x)}$$

$$\frac{\mathcal{A}(x)\Rightarrow\mathcal{B}}{\exists x\mathcal{A}(x)\Rightarrow\mathcal{B}}.$$

Приведенный язык исчисления предикатов образует теорию, которую называют *узким исчислением предикатов*, или *чистым исчислением предикатов.*

Появление в теории аксиом, разрешающих «навешивание кванторов» по знакам операций или предикатов, приводит к исчислениям *высших порядков.* Исчисление предикатов – это теория *первого порядка.*

Расширение узкого исчисления предикатов за счет добавления новых знаков индивидуальных констант, операций (знаков функций, знаков операций), знаков предикатов, новых аксиом, связывающих новые знаки, и новых правил вывода называется часто *прикладным исчислением предикатов.*

2.3.2. Полнота и непротиворечивость исчисления предикатов

Теорема 2.5 (Гёдель, 1930). *Для исчисления предикатов справедливо утверждение:*

$$\models \mathcal{A} \text{ тогда и только тогда, когда } \vdash \mathcal{A}.$$

Доказательство. См. [46, с.71,78].

Теорема 2.6. *Исчисление предикатов – полная теория.*

Доказательство. Следует из утверждения «$\models \mathcal{A}$ влечет $\vdash \mathcal{A}$» теоремы 2.5.

Теорема 2.7. *Исчисление предикатов – непротиворечивая теория.*

Доказательство. Следует из утверждения «$\vdash \mathcal{A}$ влечет $\models \mathcal{A}$» теоремы 2.5. Действительно, если бы была доказуема формула $\mathcal{A} = \mathcal{B}\&\neg\mathcal{B}$, то была бы общезначимой формула $\mathcal{B}\&\neg\mathcal{B}$. Но последняя формула ложна.

2.4. Формальная арифметика

2.4.1. Эгалитарные теории

Теория, содержащая исчисление предикатов, называется *эгалитарной*, если она имеет дополнительный двуместный предикат $=(\cdot,\cdot)$, для которого выполняются две нелогические аксиомы:

1. $\forall x(=(x,x))$;
2. $=(x,y) \Rightarrow (\mathcal{A}(...,x,...,y,...) \Rightarrow \mathcal{A}(...,y,...,y,...))$.

Здесь $\mathcal{A}$ произвольная формула.

Вместо $=(x,y)$ пишут $x=y$. Таким образом, эгалитарная теория – это просто теория с *равенством.*

2.4.2. Язык и правила вывода формальной арифметики

Формальная арифметика – это эгалитарное прикладное исчисление, в котором дополнительно имеются:

1. Предметная константа 0.
2. Двуместные операции $+$ и $\cdot$ и одноместная операция $'$.

3. Знак равенства $\cdot = \cdot$.

4. Нелогические аксиомы равенства (§2.4.1) и следующие нелогические аксиомы арифметики:

$$(\mathcal{A}(0)\&\forall x(\mathcal{A}(x) \Rightarrow \mathcal{A}(x'))) \Rightarrow \forall x\mathcal{A}(x)$$
$$(t_1' = t_2') \Rightarrow (t_1 = t_2)$$
$$\neg(t' = 0)$$
$$(t_1 = t_2) \Rightarrow ((t_2 = t_3) \Rightarrow (t_1 = t_3))$$
$$(t_1 = t_2) \Rightarrow (t_1' = t_2')$$
$$t + 0 = t$$
$$(t_1 + t_2') = (t_1 + t_2)'$$
$$t \cdot 0 = 0$$
$$t_1 \cdot t_2' \Rightarrow t_1 \cdot t_2 + t_1,$$

где $\mathcal{A}$ – любая формула, а t, t_1, t_2 – любые термы.

Первая аксиома – это известный способ доказательства посредством математической индукции. Если вместо t' писать $t + 1$, то ясно, что t' – это следующее натуральное число, идущее за t. Другими словами, аксиомы арифметики определяют натуральные числа и правила оперирования с ними с помощью операций сложения и умножения.

2.4.3. Непротиворечивость формальной арифметики. Теорема Генцена

Метод математической индукции, который входит в качестве аксиомы в формальную арифметику, может быть усилен за счет расширения области его применения до так называемых *трансфинитных чисел* [4], которые, как видно из их названия[5], «идут следом» за финитными, т.е. натуральными числами. Получается более мощный способ доказательства теорем, названный *методом трансфинитной индукции* [4].

Теорема 2.8 (Генцен, 1936). *Непротиворечивость формальной арифметики доказывается в более широкой формальной теории, содержащей арифметику и принцип трансфинитной индукции.*

Доказательство. См. [46, с.283-295].

[5] *«Транс» – за, «финитный» – конечный.*

2.4.4. Теорема Гёделя о неполноте

В определенной мере сущность теоремы Гёделя о неполноте дедуктивных систем содержится в следующей фразе Св. Иоанна Дамаскина[6]:

> «Бог есть, это очевидно. Но что есть Он по существу и естеству, – это совершенно непостижимо и неведомо» [5, с.6].

Ему предшествовал Григорий Богослов:

> «...если познание имеет предметом своим вещи существующие, то уже то, что выше познания, конечно, выше и бытия, и снова: то, что превышает бытие, то выше и познания» [5, с.7].

То же, но уже в формулировке Гёделя:

Теорема 2.9. *Во всякой непротиворечивой теории первого порядка, включающей формальную арифметику,*

1) *существует такая (истинная) формула $\mathcal{A}$, что ни $\mathcal{A}$, ни $\neg\mathcal{A}$ не являются доказуемыми;*

2) *утверждение о непротиворечивости этой теории – это формула данной теории, которая не является доказуемой.*

Доказательство. См. [8, с.298-310] или [21, с.116].

Теорема Гёделя говорит о том, что в любой достаточно богатой теории существуют высказывания, которые воспринимаются как истинные, разумные, но тем не менее они не могут быть обоснованы, доказаны теми средствами, которые теория предоставляет исследователю[7]. По существу, теорема утверждает, что в любом мире есть вещи, познание которых требует выхода в более обширный, высший мир. Над каждой теорией нужно надстраивать более изощренную метатеорию, над метатеорией – метаметатеорию и т.д.

---

[6] Традиционная история приписывает этому греческому богослову и философу следующие годы жизни – 675-753.

[7] А.Н.Колмогоров: «Математикам хорошо известно, что в пределах каждой формальной системы, достаточно богатой математически, можно сформулировать вопросы, которые кажутся содержательными, осмысленными и должны предполагать наличие определенного ответа, хотя в пределах данной системы такого ответа найти нельзя» [15].

Собственно говоря, так вышло с доказательством непротиворечивости формальной арифметики (§ 2.4.3).

Кроме того, теорема Гёделя говорит о несостоятельности идеи полной формализации процесса логического вывода. Иначе говоря, не все может быть формализовано, как бы этого ни хотелось математикам.

2.4.5. Курт Гёдель

Рис. 2.1: Курт Гёдель (1906-1978).

«Австрийский математик Курт Гёдель родился в Брно (Словакия) на двадцать семь лет позже Эйнштейна и получил физическое и математическое образование в Венском университете. Его научные интересы частично пересекались с интересами Эйнштейна. Скромный математик – одиночка Гёдель, в зрелом возрасте также приехавший в Принстонский институт перспективных исследований, внес важнейший вклад в основы математики, настолько революционный, что раздвинул границы этой дисциплины и оказал существенное влияние на общее мировоззрение и культуру 20 века» [37].

«Оказавшись в Принстоне, Гёдель предложил оригинальное решение выведенных Эйнштейном уравнений общей теории относительности. Из этого решения, между прочим, следует принципиальная возможность машины времени. Вообще же с математики он переключился на философию, увлекся трудами Лейбница – и пришел к выводу, что тот открыл – ни много ни мало – Тайну Жизни. Впрочем, по мнению Гёделя, до нас это открытие не дошло, ибо современные Лейбницу мракобесы подвергли его сочинения жесточайшей цензуре. В последние двадцать лет жизни Гёдель не опубликовал ни одной работы. Умер он в возрасте 71 года, при явных признаках психического расстройства. Уверившись, что врачи пытаются его отравить, он отказался принимать пищу, – и голодное истощение, наряду с распадом личности, фигурирует в медицинском свидетельстве о его смерти» [12].

2.5. Автоматический вывод теорем

Развитие вычислительной техники стало мощным стимулом для поиска алгоритмов и соответствующих им машинных программ, позволяющих автоматически доказывать теоремы формального исчисления посредством нахождения соответствующего (логического) вывода.

Доказанная теорема – это новая открытая истина. Автоматический поиск выводов, таким образом, является открытием истин, которое доверено ЭВМ. Машина, «доказывая теорему» $\mathcal{A}$, ищет конечную последовательность формул, последняя из которых – это формула $\mathcal{A}$, удовлетворяющую определению (формального) доказательства, данного в § 2.1.1.

Одной из первых работ (1960) по созданию алгоритма автоматического вывода была статья Ван Хао [2]. В СССР автоматический вывод в рамках исчисления высказываний в 1960-е годы осуществлялся на машине «Урал-4» группой Н.А.Шанина [47].

Наиболее крупными достижениями по созданию методов автоматического доказательства теорем являются «обратный метод» С.Ю.Маслова [23, 24] и «метод резолюций» Дж.Робинсона (см. §1.3).

2.5.1. С.Ю. Маслов

Сергей Юрьевич Маслов родился 10 июня 1939 года. Он один из ведущих математических логиков своего поколения. Его первые исследования, подытоженные в кандидатской диссертации 1964 года, были посвящены каноническим исчислениям Поста – универсальному аппарату математической логики и теории алгоритмов.

Работы С.Ю.Маслова в области теории поиска логического вывода[8] и прежде всего созданный им (и опубликованный на год раньше метода резолюций) обратный метод поиска вывода известны не только логикам, но и специалистам по вычислительной науке и искусственному интеллекту. Обратный метод, поставивший его автора на одно из первых мест в мире в области поиска логического вывода, позволил получить глубокие результаты в проблеме разрешения для исчисления предикатов. С.Ю.Маслов был первым не

[8] Использована статья о С.Ю.Маслове, находящаяся на сайте: http://www.philosophy.ru/library/logic/maslov/01.html

только в теоретических начинаниях, но и в их практическом воплощении. Он стал ведущим участником создания двух наиболее значительных в СССР программ поиска вывода: программы АЛПЕВ для поиска натурального вывода в исчислении высказываний (под руководством Н.А.Шанина в 1961-1963 гг.) и программы для исчисления предикатов на основе обратного метода в 1966-1967 гг. Последние 10 лет своей жизни[9] Сергей Юрьевич посвятил разработке своих идей о применениях дедуктивных систем к другим естественным и гуманитарным наукам, не оставляя исследований по поиску автоматического вывода.

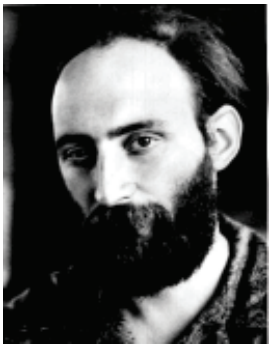

Рис. 2.2: С.Ю. Маслов (1939-1982).

Из смежных с математической логикой дисциплин и наук, в которых оказывалось целесообразным применение методов дискретной математики, С.Ю.Маслов на протяжении последних 10 лет особенно много занимался вопросами биологической теории эволюции, нейропсихологии, искусственного интеллекта, семиотики культуры (в частности архитектуры). Развивавшийся им подход к теории биологической эволюции (связанный и с его длительной и обстоятельной перепиской с выдающимся биологом-теоретиком А.А.Любищевым) представлял собой модель достаточно общего характера, которая может быть интересна и для других сложных систем передачи информации (например фольклорных текстов). Достигнутый в ряде последних его статей и докладов синтез новейших результатов нейропсихологических и нейролингвистических исследований по асимметрии полушарий головного мозга и по искусственному интеллекту привел к четкому выделению круга преимущественно правополушарных задач искусственного интеллекта, решение которых находится за очень короткие временные интервалы. В этих работах был намечен путь моделирования решения переборных задач, опиравшийся на достижения теории логического вывода.

Особый цикл последних работ С.Ю. Маслова был посвящен соединению изученного круга нейропсихологических представлений с исследованиями, ведущимися в семиотике культуры. Изучались возможности выявления преимущественно левополушарных и

[9] С.Ю.Маслов погиб в автомобильной катастрофе.

правополушарных составляющих в сменяющих друг друга стилистических тенденциях, в свою очередь сопоставимых и с другими социально-историческими факторами. Соответствующие гипотезы были проиллюстрированы и проверены на обширном материале, почерпнутом из истории архитектуры, и вызвали оживленные дискуссии среди специалистов по культурологии и семиотике.

Из воспоминаний В.Я Крейновича[10] о С.Ю.Маслове:

«На меня глубокое впечатление произвел эпизод, когда я пришел к нему, взбешенный на очередной бред коммунистического начальства, и он неожиданно сказал мне, что главная проблема России не в том, что начальство плохое, а в том, что слишком много ненависти. Я это потом вспоминал не раз, удивляясь, как я мог этого не понять» [11].

2.6. Логическое программирование

Как решается задача с помощью ЭВМ? Математик разрабатывает алгоритм решения задачи, а программист, используя один из языков программирования, например *PASCAL*, реализует предложенный алгоритм в виде программы.

Возможен иной подход. Он основывается на следующих трех моментах:

- На способности ЭВМ доказывать теоремы. Эта способность, как мы знаем, называется автоматическим доказательством теорем (см. § 2.5).
- На понимании, что вычисление – это частный случай логического вывода.
- На осознании того, что алгоритм – формальное задание функции (см. § 4.1).

Иначе говоря, задачу, решение которой ищется, надо представить в виде высказывания, доказательство которого следует предоставить ЭВМ.

[10] В.Я Крейнович в настоящее время профессор компьютерных наук в университете Техаса в El Paso (США). – http://www.cs.utep.edu/vladik

[11] См.: http://www.mathsoc.spb.ru/pers/maslov/davydova.html

Таким образом, компьютеру предлагается не алгоритм, а описание предметной области задачи и сама задача в виде формул формального исчисления. Решение задачи является в таком случае выводом в данном исчислении. Какова при этом роль программиста? От программиста при таком подходе требуется описать с достаточной степенью полноты предметную область и формулировку задачи на языке этого исчисления, а поиск вывода, приводящего к решению задачи, поручается компьютеру.

2.6.1. Логическая программа

Логическая программа[12] представляет собой конечный набор формул логики предикатов одного из следующих видов:

$$P(t_1, ..., t_n)., \tag{2.4}$$

$$Q(s_1, ..., s_k) : -Q_1(s_1, ..., s_k), ..., Q_m(s_1, ..., s_k)., \tag{2.5}$$

где $P, Q, Q_1, ..., Q_m$ – предикаты, а $t_1, ...t_n, s_1, ..., s_k$ – термы. Формулы первого вида называются *фактами*, а второго – *правилами*. (В конце каждого выражения ставится точка). Факт – одиединственный экземпляр, свойство или отношение между объектами.

Правило (2.5) читается как «$Q(s_1, ..., s_l)$ истинно, если истинны $Q_1(s_1, ..., s_l), ..., Q_m(s_1, ..., s_l)$». Формула $Q(s_1, ..., s_l)$ называется *заголовком* правила (2.5). Правило позволяет выводить новые факты из уже имеющихся.

Таким образом, логическая программа состоит из конечного числа фактов и правил:

$$\begin{array}{l} \mathcal{A}. \\ ... \\ \mathcal{M}. \\ \mathcal{N} : - \mathcal{N}_1, ..., \mathcal{N}_m. \\ ... \\ \mathcal{W} : - \mathcal{W}_1, ..., \mathcal{W}_d. \end{array} \tag{2.6}$$

В фактах и правилах описывается логическая модель предметной области, отношения предметов и правила получения новых свойств.

[12] При написании этого параграфа использованы материалы сайта http://pgap.chat.ru/zap/zap149.htm

Логическая программа задает множество следствий, которые являются результатом программы. Выполнение логической программы – это *вывод следствия* из нее.

Для *выполнения программы* требуется обратиться с *целевым запросом* (целью), который представляет собой последовательность формул вида

$$R_1(u_1, ..., u_m), ..., R_p(u_1, ..., u_m)., \tag{2.7}$$

где $R_j(u_1, ..., u_m)$ – атомарные формулы логики первого порядка, буквы u_i – термы.

Выполнение программы состоит в попытке решить задачу, т.е. доказать целевое утверждение (2.7), используя факты и предположения $\mathcal{N}_1, ..., \mathcal{N}_m, ..., \mathcal{W}_1, ..., \mathcal{W}_d$, заданные в логической программе.

Описанные конструкции логического программирования семантически интерпретируются в логике предикатов. Процедура интерпретации состоит в сопоставлении формулам логической программы формул логики предикатов.

Для этого каждому факту (2.4) ставится в соответствие формула

$$\mathcal{F} = \forall x_1 ... \forall x_s P(t_1, ..., t_n), \tag{2.8}$$

где кванторы общности навешены на все переменные $x_1, ..., x_s$ атомарной формулы (2.4), входящие в термы t_j (кроме переменных, в термах могут быть и константы).

Правилу (2.5) ставится в соответствие формула вида

$$\mathcal{G} = \forall y_1 ... \forall y_r (Q_1(s_1, ..., s_l) \& ... \& Q_m(s_1, ..., s_l) \Rightarrow Q(s_1, ..., s_l)), \tag{2.9}$$

где кванторы общности, как и выше, навешены на все переменные, входящие в термы s_i.

Запрос (2.7) получит в соответствие формулу

$$\mathcal{H}_Z = \exists z_1 ... \exists z_d (R_1(u_1, ..., u_m) \& ... \& R_p(u_1, ..., u_m)), \tag{2.10}$$

где кванторы существования связывают все переменные.

Пусть $\mathcal{F}_1, ..., \mathcal{F}_a$ – формулы, соответствующие всем фактам, $\mathcal{G}_1, ..., \mathcal{G}_b$ – всем правилам. Тогда значение пары $<$**программа, запрос**$>$ есть утверждение о том, что формула $\mathcal{H}_Z$ есть логическое следствие формул $\mathcal{F}_1, ..., \mathcal{F}_a, \mathcal{G}_1, ..., \mathcal{G}_b$. Для того чтобы выяснить, так ли это, применяется метод резолюций.

Логическое программирование предполагает наличие логической машины, называемой *интерпретатором*, который осуществляет процесс логического вывода. Механизм этого вывода использует процедуру унификации, на которой основан метод резолюций в логике предикатов (см. § 1.3.2).

Рассмотрим действия интерпретатора на примере следующей логической программы:

$$\begin{aligned} &R(a,b).\\ &Q(b,g(c)).\\ &P(x,f(y)) :- R(x,z), Q(z,f(y)).\\ &P(x,f(y)) :- R(x,z), Q(z,g(y)).\\ &R(x,z) :- Q(f(x),G(z)). \end{aligned} \tag{2.11}$$

Здесь a, b, c, d – константы, x, y, z – переменные.

Предположим, что целевой запрос есть формула

$$P(u, f(v)). \tag{2.12}$$

При вычислении ответа на этот запрос интерпретатор формулирует цель $P(u, f(v))$ и пытается достичь ее, унифицируя цель с фактами. В нашем случае цель $P(u, f(v))$ не унифицируется ни с одним из фактов. Тогда интерпретатор пытается ее унифицировать с заголовком одного из правил.

Это возможно с заголовком первого правила при подстановке $< x|u >, < y|v >$. Но надо предварительно проверить истинность предположений первого правила. Для этого интерпретатор формирует запрос

$$R(x,z), Q(z,f(y)).$$

Цель $R(x,z)$ достигается за счет унификации с первым фактом с помощью подстановки $< x|a >, < z|b >$. В таком случае, следующим запросом является $Q(b,f(y))$. Но эта цель не достижима, поскольку не унифицируется ни с одним из фактов.

Интерпретатор возвращается к цели $R(x,z)$ и обращается к второму правилу.

Теперь интерпретатор делает запрос:

$$R(x,z), Q(z,g(y)).$$

Цель $R(x,z)$ достигается, как мы знаем, унификацией с первым фактом при подстановке $< x|a >, < z|b >$, а цель $Q(b,g(y))$ достигается в силу того, что унифицируется со вторым фактом с помощью подстановки $< y|c >$. Следовательно, цель (2.12) достигается подстановкой $< u|a >, < v|c >$. Поэтому интерпретатор на этом может закончить работу и выдать найденную подстановку. Выполнение программы считается успешно завершенным; при этом виден пример доказательства формулы (2.12).

2.6.2. Языки логического программирования

Для реализации идей логического программирования разработаны различные языки логического программирования. Первым и наиболее известным является язык PROLOG.

В TURBO PROLOG структура логической программы имеет следующий вид:

domains <структуры и типы данных>
global domains <внешние структуры и типы данных>
data base <глобальные предикаты динамической базы данных>
predicates <определение предикатов>
global predicates <внешние предикаты>
goal <цели>
clauses <факты и правила>

Пример 2.1. Логическая программа для выяснения родственных связей:

```
domains
name=symbol

predicates
parent(name,name)
father(name,name)
mother(name,name)
grandfather(name,name)
grandmather(name,name)

clauses
mother(оля,петя).
father(тимур,петя).
father(тимур,лена).
mother(лика,кирилл).
mother(лика,вика).
father(сергей,кирилл).
mother(женя,ира).
father(толя,федя).
```

```
father(толя,ира).
parent(X,Y):- mother(X,Y),father(X,Y).
grandfather(X,Y):- father(X,Z),parent(Z,Y).
grandmather(X,Y):- mother(X,Z),parent(Z,Y).
```

Глава 3

Неклассические логики

3.1. Интуиционистская логика

Интуиционистская логика – это логика, для которой не выполняется один из основных законов классической логики, – закон исключенного третьего, т.е.

$$\models (\mathcal{A} \vee \neg\mathcal{A}).$$

Отказ от этого закона приводит к признанию существования «третьей возможности». Но в таком случае, естественно, приходим к запрету доказывать теоремы методом от противного. Последнее приводит к отказу от аксиомы

$$\models (\neg\neg\mathcal{A} \Rightarrow \mathcal{A}). \tag{3.1}$$

Действительно, при доказательстве от противного предполагают, что верно на самом деле $\neg\mathcal{A}$, и затем приходят к некоторому противоречию. Это показывает, что «не верно» $\neg\mathcal{A}$, т.е «верно» $\neg\neg\mathcal{A}$. Последнее позволяет утверждать, что «верно» $\mathcal{A}$. Иначе говоря, имеем утверждение (3.1).

Интуиционистская логика в философской форме была заявлена Брауэром (1908, 1918). Первое систематическое изучение интуиционистской логики[1] была начато 22-летним великим советским математиком А.Н.Колмогоровым [16]. В виде формального исчисления ее впервые изложил Гейтинг (1930).

[1] Так сказано в хрестоматии ван Хейеноорта «От Фреге до Гёделя» [52].

3.2. Нечеткая логика

Нечеткая логика отличается от двузначной классической логики тем, что допускает континуальное число *истинностных значений* для высказываний. В простейшем случае эти значения принадлежат отрезку $[0,1]$ действительных чисел. Иначе говоря, между значением 0, соответствующим классическому $\perp$ (ложь) и 1, или $\top$ (истина), имеется несчетное число промежуточных истинностных значений $\underset{\sim}{a} \in (0,1)$.

Нечеткая логика широко используется в современной прикладной математике и технических науках.

3.2.1. Нечеткие подмножества

Пусть E некоторое фиксированное множество и $M = [0,1]$ – отрезок действительных чисел.

Нечеткое подмножество $\underset{\sim}{A}$ множества E – это множество пар вида

$$\{(x, \mu_{\underset{\sim}{A}}(x)) : x \in E\}, \tag{3.2}$$

где $\mu_{\underset{\sim}{A}} : E \to M$ – функция.

Если $M = \{0,1\}$, то $\mu_{\underset{\sim}{A}} : E \to M$ – обычная характеристическая функция. Поэтому в данном случае множество $\underset{\sim}{A}$ отождествляется с классическим канторовским подмножеством $A = \{x : \mu_A(x) = 1\}$.

Пара $(x, \mu_{\underset{\sim}{A}}(x))$ интерпретируется как элемент $x \in E$, который *принадлежит подмножеству* $\underset{\sim}{A}$ *со степенью* $\mu_{\underset{\sim}{A}}(x)$. В классической теории множеств элемент $x \in E$ либо принадлежит, т.е. $\mu_A(x) = 1$, либо не принадлежит ($\mu_A(x) = 0$) подмножеству A. Третьего не дано! Для нечеткого множества есть и третье, и четвертое, и т.д. Налицо размытость, нечеткость подмножества $\underset{\sim}{A}$.

Множество M называется *множеством принадлежности*, а функция $\mu_{\underset{\sim}{A}}$ – *функцией принадлежности*.

Обычные множества будем обозначать как $A, B, C, ...$, а нечеткие множества – $\underset{\sim}{A}, \underset{\sim}{B}, \underset{\sim}{C}, ...$

Имеем

$$E = \underset{\sim}{E} = \{(x,1) : x \in E\},$$

иначе говоря, $\mu_{\underset{\sim}{E}} \equiv 1$, и

$$\emptyset = \underset{\sim}{\emptyset} = \{(x, 0) : x \in E\},$$

или $\mu_{\emptyset} \equiv 0$.

3.2.2. Операции над нечеткими подмножествами

Над нечеткими множествами можно исполнить такие же операции, как и над обычными.

Объединение нечетких множеств $\underset{\sim}{A}$ и $\underset{\sim}{B}$ – это нечеткое множество

$$\underset{\sim}{C} = \underset{\sim}{A} \cup \underset{\sim}{B},$$

для которого

$$\mu_{\underset{\sim}{C}} = \max\{\mu_{\underset{\sim}{A}}, \mu_{\underset{\sim}{B}}\}.$$

Аналогично имеем *пересечение* $\underset{\sim}{C}$ нечетких множеств $\underset{\sim}{A}$ и $\underset{\sim}{B}$

$$\underset{\sim}{C} = \underset{\sim}{A} \cap \underset{\sim}{B},$$

если по определению

$$\mu_{\underset{\sim}{C}} = \min\{\mu_{\underset{\sim}{A}}, \mu_{\underset{\sim}{B}}\}.$$

Нечеткое множество $\underset{\sim}{B}$ есть *дополнение* для $\underset{\sim}{A}$, т.е.

$$\underset{\sim}{B} = \overline{\underset{\sim}{A}},$$

если

$$\mu_{\underset{\sim}{B}} = 1 - \mu_{\underset{\sim}{A}}.$$

Ясно: $\overline{\overline{\underset{\sim}{A}}} = \underset{\sim}{A}$.

Если даны нечеткие множества $\underset{\sim}{A}$ и $\underset{\sim}{B}$, то пишем $\underset{\sim}{A} \subset \underset{\sim}{B}$ тогда и только тогда, когда

$$\forall x \in E(\mu_{\underset{\sim}{A}}(x) \le \mu_{\underset{\sim}{B}}(x)).$$

3.2.3. Свойства множества нечетких подмножеств

Справедливы следующие равенства:

$$
\begin{aligned}
\underset{\sim}{A} \cap \underset{\sim}{B} &= \underset{\sim}{B} \cap \underset{\sim}{A} \\
\underset{\sim}{A} \cup \underset{\sim}{B} &= \underset{\sim}{B} \cup \underset{\sim}{A} \\
(\underset{\sim}{A} \cap \underset{\sim}{B}) \cap \underset{\sim}{C} &= \underset{\sim}{A} \cap (\underset{\sim}{B} \cap \underset{\sim}{C}) \\
(\underset{\sim}{A} \cup \underset{\sim}{B}) \cup \underset{\sim}{C} &= \underset{\sim}{A} \cup (\underset{\sim}{B} \cup \underset{\sim}{C}) \\
\underset{\sim}{A} \cap (\underset{\sim}{B} \cup \underset{\sim}{C}) &= (\underset{\sim}{A} \cap \underset{\sim}{B}) \cup (\underset{\sim}{A} \cap \underset{\sim}{C}) \\
\underset{\sim}{A} \cup (\underset{\sim}{B} \cap \underset{\sim}{C}) &= (\underset{\sim}{A} \cup \underset{\sim}{B}) \cap (\underset{\sim}{A} \cup \underset{\sim}{C}) \\
\underset{\sim}{A} \cup \underset{\sim}{A} &= \underset{\sim}{A} \\
\underset{\sim}{A} \cap \underset{\sim}{A} &= \underset{\sim}{A} \\
\underset{\sim}{A} \cap E &= \underset{\sim}{A} \\
\underset{\sim}{A} \cup E &= E \\
\underset{\sim}{A} \cup \emptyset &= \underset{\sim}{A} \\
\overline{\overline{\underset{\sim}{A}}} &= \underset{\sim}{A} \\
\overline{\underset{\sim}{A} \cap \underset{\sim}{B}} &= \overline{\underset{\sim}{A}} \cup \overline{\underset{\sim}{B}} \\
\overline{\underset{\sim}{A} \cup \underset{\sim}{B}} &= \overline{\underset{\sim}{A}} \cap \overline{\underset{\sim}{B}}.
\end{aligned}
\tag{3.3}
$$

Однако

$$
\begin{aligned}
&\underset{\sim}{A} \cap \overline{\underset{\sim}{A}} \neq \emptyset \ \ (\text{кроме } \underset{\sim}{A} = \emptyset \text{ или } \underset{\sim}{A} = E) \\
&\underset{\sim}{A} \cup \overline{\underset{\sim}{A}} \neq E \ \ (\text{кроме } \underset{\sim}{A} = \emptyset \text{ или } \underset{\sim}{A} = E),
\end{aligned}
\tag{3.4}
$$

которые для обычных множеств имеют вид

$$
\begin{aligned}
A \cap \overline{A} &= \emptyset \\
A \cup \overline{A} &= E
\end{aligned}
\tag{3.5}
$$

и справедливы.

Таким образом, множество нечетких множеств не является булевой алгеброй (см. § 1.1.7). Следовательно, можно ожидать, что логика, построенная на нечетких множествах, будет неклассической.

Для того чтобы убедиться в неравенствах (3.4), достаточно взять $\mu_{\underset{\sim}{A}} = 1/2$. Тогда

$$\mu_{\underset{\sim}{A}\cap\overline{\underset{\sim}{A}}} = \min\{\mu_{\underset{\sim}{A}}, \mu_{\overline{\underset{\sim}{A}}}\} =$$

$$= \min\{\mu_{\underset{\sim}{A}}, 1 - \mu_{\underset{\sim}{A}}\} = \min\{1/2, 1/2\} = 1/2 \neq 0 = \mu_{\emptyset},$$

и для второго неравенства

$$\mu_{\underset{\sim}{A}\cup\overline{\underset{\sim}{A}}} = \max\{\mu_{\underset{\sim}{A}}, \mu_{\overline{\underset{\sim}{A}}}\} =$$

$$= \max\{\mu_{\underset{\sim}{A}}, 1 - \mu_{\underset{\sim}{A}}\} = \max\{1/2, 1/2\} = 1/2 \neq 1 = \mu_E.$$

3.2.4. Нечеткая логика высказываний

Для того чтобы перейти к нечеткой логике, нужно ввести нечеткие пропозициональные переменные.

Нечеткие пропозициональные переменные $\underset{\sim}{a}, \underset{\sim}{b}, \underset{\sim}{c}, \ldots$ – это

$$\underset{\sim}{a} = \mu_{\underset{\sim}{A}}(x), \ \underset{\sim}{b} = \mu_{\underset{\sim}{B}}(x), \ \underset{\sim}{c} = \mu_{\underset{\sim}{C}}(x), \ \ldots \tag{3.6}$$

При таком определении переменные $\underset{\sim}{a}, \underset{\sim}{b}, \underset{\sim}{c}, \ldots$ начинают принимать не только классические значения 0,1, но и любые другие из интервала $(0, 1)$.

Полагаем, что

$$0 = \mu_{\emptyset}(x) \equiv 0 \ \text{ и } \ 1 = \mu_E(x) \equiv 1.$$

Определяем нечеткие логические операции:

$$\underset{\sim}{a} \ \& \ \underset{\sim}{b} = \min(\underset{\sim}{a}, \underset{\sim}{b}),$$

$$\underset{\sim}{a} \vee \underset{\sim}{b} = \max(\underset{\sim}{a}, \underset{\sim}{b}),$$

$$\neg \underset{\sim}{a} = 1 - \underset{\sim}{a} \ .$$

Мы можем теперь ввести понятие *нечеткой формулы*:

1) *нечеткая пропозициональная переменная есть (атомарная) нечеткая формула;*

2) *если* $\mathcal{A}$ *и* $\mathcal{B}$ *нечеткие формулы, то* $(\mathcal{A}\&\mathcal{B})$, $(\mathcal{A}\vee\mathcal{B})$, $(\mathcal{A}\Rightarrow\mathcal{B})$ *нечеткие формулы;*

3) *если* $\mathcal{A}$ *нечеткая формула, то* $\neg\mathcal{A}$ *нечеткая формула.*

Справедливы тождества:

$$
\begin{aligned}
\underset{\sim}{a}\ \&\ \underset{\sim}{b} &\equiv \underset{\sim}{b}\ \&\ \underset{\sim}{a}\\
\underset{\sim}{a}\ \vee\ \underset{\sim}{b} &\equiv \underset{\sim}{b}\ \vee\ \underset{\sim}{a}\\
(\underset{\sim}{a}\ \&\ \underset{\sim}{b})\&\ \underset{\sim}{c} &\equiv \underset{\sim}{a}\ \&(\underset{\sim}{b}\ \&\ \underset{\sim}{c})\\
(\underset{\sim}{a}\ \vee\ \underset{\sim}{b})\vee\ \underset{\sim}{c} &\equiv \underset{\sim}{a}\ \vee(\underset{\sim}{b}\ \vee\ \underset{\sim}{c})\\
\underset{\sim}{a}\ \&(\underset{\sim}{b}\ \vee\ \underset{\sim}{c}) &\equiv (\underset{\sim}{a}\ \&\ \underset{\sim}{b})\vee(\underset{\sim}{a}\ \&\ \underset{\sim}{c})\\
\underset{\sim}{a}\ \vee(\underset{\sim}{b}\ \&\ \underset{\sim}{c}) &\equiv (\underset{\sim}{a}\ \vee\ \underset{\sim}{b})\&(\underset{\sim}{a}\ \vee\ \underset{\sim}{c})\\
\underset{\sim}{a}\ \vee\ \underset{\sim}{a} &\equiv \underset{\sim}{a}\\
\underset{\sim}{a}\ \&\ \underset{\sim}{a} &\equiv \underset{\sim}{a}\\
\underset{\sim}{a}\ \&1 &\equiv \underset{\sim}{a}\\
\underset{\sim}{a}\ \vee 1 &\equiv 1\\
\underset{\sim}{a}\ \vee 0 &\equiv \underset{\sim}{a}\\
\neg\neg\ \underset{\sim}{a} &\equiv \underset{\sim}{a}\\
\neg(\underset{\sim}{a}\ \&\ \underset{\sim}{b}) &\equiv \neg\ \underset{\sim}{a}\ \vee\neg\ \underset{\sim}{b}\\
\neg(\underset{\sim}{a}\ \vee\ \underset{\sim}{b}) &\equiv \neg\ \underset{\sim}{a}\ \&\neg\ \underset{\sim}{b}\ .
\end{aligned}
\tag{3.7}
$$

Однако

$$
\begin{aligned}
&\underset{\sim}{a}\ \&\neg\ \underset{\sim}{a} \neq 0 \quad (\text{кроме } \underset{\sim}{a}=0 \text{ или } \underset{\sim}{a}=1)\\
&\underset{\sim}{a}\ \vee\neg\ \underset{\sim}{a} \neq 1 \quad (\text{кроме } \underset{\sim}{a}=0 \text{ или } \underset{\sim}{a}=1).
\end{aligned}
\tag{3.8}
$$

Таким образом, нечеткая логика высказываний не является классической.

3.2.5. Нечеткие релейно-контактные схемы

Поставим в соответствие формуле $\underset{\sim}{a}$ & $\underset{\sim}{b}$ последовательную (рис.3.1, a)), а $\underset{\sim}{a} \vee \underset{\sim}{b}$ – параллельную (рис.3.1, b) (электрические) релейно-контактные схемы:

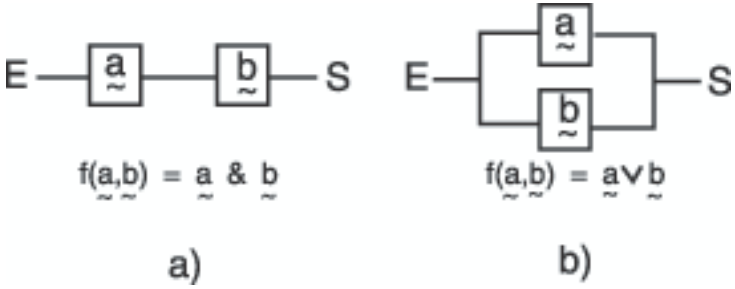

Рис. 3.1: Простейшие нечеткие релейно-контактные схемы.

В этих схемах указывается вход E и выход S. Результат выполнения нечетких логических операций называется *током схемы*.

Если взять $\underset{\sim}{a} = 0,7$, $\underset{\sim}{b} = 0,4$, то для последовательной схемы ток S равен 0,4; для параллельной – 0,7. В случае классической логики ток мог принимать только значения 0 или 1 (см. §1.1.5).

3.3. Модальные логики

Модальная логика строится на основе логики высказываний за счет добавления новых знаков, позволяющих выражать отношение тех или иных высказываний к окружающей действительности. Как правило, это суждения о *возможности* или *необходимости* чего-либо.

Классическая логика имеет дела с *ассерторическими*[2] высказываниями, которые утверждают наличие или отсутствие той или иной ситуации. Однако в жизни приходится иметь дело с высказываниями, содержащими указание на необходимость или возможность чего-либо. Это связано с элементами случайного в природе

[2] *Assertion* (англ.) – утверждение, отстаивание.

либо с констатацией того, что что-то может произойти в будущем или имело место в прошлом, но чего нет в данный момент. Высказывания этого рода называют *модальными* [3].

«Для классической логики вещь существует или не существует, и нет никаких других вариантов. Но как в обычной жизни, так и в науке постоянно приходится говорить не только о том, что есть в действительности и чего нет, но и о том, что должно быть или не должно быть и т.д. Действительный ход событий можно рассматривать как реализацию одной из многих мыслимых возможностей, а действительный мир, в котором мы находимся, – как один из бесчисленного множества возможных миров» [10].

3.3.1. Типы модальности

Различают три типа модальностей, каждый из которых подразделяется на виды [3, с.313]:

- **Алетические модальности**. Это высказывания, содержащие такие *виды* модальности, как «необходимо», «возможно», «невозможно», «случайно».
- **Деонтические модальности.** Это модальности, связанные с характеристиками действий и поступками людей в обществе. Например, «обязательно», «разрешено», «запрещено», «безразлично».
- **Эпистемические модальности.** Характеристики наших знаний. Можно назвать такие виды модальности этого типа, как «доказано», «опровергнуто», «не доказано», «не опровергнуто», «знает», «верит», «убежден», «сомневается».

3.3.2. Исчисления I и T (Фейса-фон Вригта)

Модальная логика – исчисление I – строится за счет расширения языка логики высказываний.

[3] *Модальность* – [фр. modalé < лат. modus способ, наклонение] – 1) *лингв.* грамматическая категория, обозначающая отношение содержания предложения к действительности и выражающаяся формами наклонения глагола, вводными словами и т.п.; 2) *лог.* модальное суждение – характеристика суждения в зависимости от характера устанавливаемой им достоверности, от того, выражает ли оно возможность, действительность или необходимость чего-либо [40].

1. Дополнительный *модальный* знак: $\Box$.
2. В определение формул добавляется новая фраза:

   если $\mathcal{A}$ *– формула, то* $\Box\mathcal{A}$ *– формула.*

   Формула $\Box\mathcal{A}$ читается как «необходимо $\mathcal{A}$». Выражения, содержащие модальный знак, называются *модальностями.*
3. Дополнительная аксиома:

$$\Box(\mathcal{A} \Rightarrow \mathcal{B}) \Rightarrow (\Box\mathcal{A} \Rightarrow \Box\mathcal{B}).$$

4. Дополнительное правило вывода:

$$\frac{\mathcal{A}}{\Box\mathcal{A}} \qquad \text{(Правило Гёделя).}$$

Если добавить к исчислению I еще одну аксиому

$$\Box\mathcal{A} \Rightarrow \mathcal{A},$$

то получаем исчисление T, называемое также *исчислением Фейса - фон Вригта.*

3.3.3. Исчисления S4, S5 и исчисление Брауэра

Исчисление S4 получается за счет добавления к исчислению T аксиомы

$$\Box\mathcal{A} \Rightarrow \Box(\Box\mathcal{A}).$$

Если же к исчислению T добавить аксиому

$$\neg(\Box\mathcal{A}) \Rightarrow \Box(\neg(\Box\mathcal{A})),$$

то получают *исчисление* S5.

Наконец, *брауэрово исчисление* получается за счет добавления к исчислению T аксиомы Брауэра:

$$\mathcal{A} \Rightarrow \Box(\Diamond\mathcal{A}),$$

где введено обозначение

$$\Diamond\mathcal{A} = \neg(\Box(\neg\mathcal{A})).$$

Формула $\Diamond\mathcal{A}$ читается как «возможно $\mathcal{A}$».

Модальное исчисление, содержащее правило вывода Гёделя, называется *нормальным*. В противном случае *ненормальным*.

Исчисление S4, расширенное за счет добавления аксиомы Брауэра, эквивалентно исчислению S5 [44, с.255].

Теорема 3.1. *Исчисления* T, S4 *и* S5 *непротиворечивы.*

Доказательство дано в [17, с.46].

3.3.4. Означивание формул

Пусть дана формула $\mathcal{A}(X_1, ..., X_n)$. *Означивание* формулы $\mathcal{A}$ – это отображение

$$v_m(\mathcal{A}) : \{0, 1, ..., m\}^n \to \{0, 1, ..., m\}.$$

Другими словами, пропозициональным переменным X_i и самой формуле $\mathcal{A}$ придаются значения из множества $\{0, 1, ..., m\}$ по некоторым правилам. **Значению 0 соответствует $\top$ (истина)**.

Формула $\mathcal{A}$ называется m-общезначимой, если $\mathcal{A} = 0$ при любых значениях своих переменных. Обозначение:

$$\models_m \mathcal{A}.$$

Означивание v_m называется *характеристическим,* если оно удовлетворяет двум условиям:

a) при $m = 1$ для формул без знаков $\Box, \Diamond$ оно совпадает с классическими таблицами истинности;

b) класс теорем совпадает с классом m-общезначимых формул.

Теорема 3.2. *Означивание*

$$v_m(\neg\mathcal{A}) = \begin{cases} 0, & v_m(\mathcal{A}) = m, \\ m, & v_m(\mathcal{A}) \neq m; \end{cases}$$

$$v_m(\mathcal{A}\ \&\mathcal{B}) = \max(v_m(\mathcal{A}), v_m(\mathcal{B}));$$

$$v_m(\mathcal{A} \vee \mathcal{B}) = \min(v_m(\mathcal{A}), v_m(\mathcal{B}));$$

$$v_m(\mathcal{A} \Rightarrow \mathcal{B}) = \begin{cases} 0, & v_m(\mathcal{A}) \geq v_m(\mathcal{B}), \\ v_m(\mathcal{B}), & v_m(\mathcal{A}) < v_m(\mathcal{B}); \end{cases}$$

$$v_m(\Box\mathcal{A}) = \begin{cases} 0, & v_m(\mathcal{A}) = 0, \\ m, & v_m(\mathcal{A}) \neq 0; \end{cases}$$

$$v_m(\Diamond\mathcal{A}) = \begin{cases} 0, & v_m(\mathcal{A}) \neq m, \\ m, & v_m(\mathcal{A}) = m. \end{cases}$$

удовлетворяет условию а) *определения характеристического означивания.*

Теорема 3.3. *В модальных исчислениях* T, S4 *и* S5 *если* $\vdash \mathcal{A}$, *то* $\models_m \mathcal{A}$ $(m \geq 1)$.

Доказательство дано в [17, с.47].

Теорема 3.4. *В исчислениях* T, S4 *и* S5 *нет характеристического* v_m*-означивания.*

Доказательство дано в [17, с.48].

Следствие 3.1. *Модальные исчисления* T, S4 *и* S5 *бесконечнозначны относительно* v_m*-означивания. Они отличны от классической логики и близки к интуиционистской.*

3.3.5. Семантика Крипке

Среди различных семантических интерпретаций модальных логик особое место занимает семантика Крипке.

Крипке строит модели вида $< W, R, G, v >$, где W – фиксированное непустое множество, R – рефлексивное бинарное отношение на W, т.е. $R \subset W \times W$, $G \in W$ – фиксированный элемент и v – T-означивание формул, описываемое ниже. Для этого рассматриваются пары $(\mathcal{A}, w)$, где $\mathcal{A} = \mathcal{A}(X_1, ..., X_n)$ – формула и $w \in W$.

Элемент w трактуется как *возможный мир*. Поэтому W – это множество возможных миров. Элемент G называем *действительным миром*. Истинность отношения $R(w, w')$ означает *достижимость* мира w' из мира w.

T-*означивание* – это отображение $v(\mathcal{A}, w) : \{\bot, \top\}^n \to \{\bot, \top\}$, задаваемое следующим образом:

$v(\neg\mathcal{A}, w) = \top \Longleftrightarrow v(\mathcal{A}, w) = \bot$;
$v(\mathcal{A}\&\mathcal{B}, w) = \top \Longleftrightarrow v(\mathcal{A}, w) = v(\mathcal{B}, w) = \top$;
$v(\mathcal{A} \vee \mathcal{B}, w) = \top \Longleftrightarrow v(\mathcal{A}, w) = \top$ или $v(\mathcal{B}, w) = \top$;
$v(\mathcal{A} \Rightarrow \mathcal{B}, w) = \top \Longleftrightarrow v(\mathcal{A}, w) = \bot$ или $v(\mathcal{B}, w) = \top$;
$v(\Box\mathcal{A}, w) = \top \Longleftrightarrow v(\mathcal{A}, w') = \top$ для любого w' с $R(w, w')$;
$v(\neg\Box\mathcal{A}, w) = \top \Longleftrightarrow v(\mathcal{A}, w') = \bot$ для любого w' с $R(w, w')$;
$v(\Diamond\mathcal{A}, w) = \top \Longleftrightarrow v(\mathcal{A}, w') = \top$ хотя бы для одного w' с $R(w, w')$;
$v(\neg \Diamond\mathcal{A}, w) = \top \Longleftrightarrow v(\mathcal{A}, w') = \bot$ хотя бы для одного w' с $R(w, w')$.

Как видим, высказывание *необходимо*, если оно «истинно во всех возможных достижимых мирах», и *возможно*, если оно истинно хотя бы в одном мире. Семантика Крипке реализует идеи Лейбница.

Если дополнительно потребовать, чтобы отношение R было симметричным, то имеем *брауэровское* означивание; в случае рефлексивного и транзитивного R – S4-означивание. Наконец, для рефлексивного, симметричного и транзитивного R получаем S5 - означивание.

Множество формул Σ T-*выполнимо* (соотв.: S4-, S5-выполнимо), если и только если существует T-означивание (соотв.: S4-, S5-означивание) такое, что для всякой $\mathcal{A} \in \Sigma$ имеем $v(\mathcal{A}, G) = \top$, и невыполнимо в противном случае.

Формула $\mathcal{A}$ T-*общезначима* тогда и только тогда, когда множество формул $\{\neg\mathcal{A}\}$ невыполнимо. Символически T-общезначимость записывается в виде:

$$\models \mathcal{A}.$$

Теорема 3.5. *В исчислениях* T, S4 *и* S5 *если* $\vdash \mathcal{A}$, *то* $\models \mathcal{A}$.

Доказательство дано в [17, с.50].

Следствие 3.2. *Модальные исчисления* T, S4 *и* S5 *непротиворечивы.*

3.3.6. Другие интерпретации модальных знаков

Знаки $\Box$ и $\Diamond$могут пониматься иначе, чем «необходимо» и «возможно».

Например, допустимы следующие интерпретации:

1) $\Box$ – «обязательность», а $\Diamond$– «позволение»;

2) $\Box$ – «доказуемость», а $\Diamond$– «непротиворечивость»;

3) $\Box$ – «везде» или «всегда», а $\Diamond$ – «кое-где» или «иногда» (пространственно-временные модальности).

3.4. Георг фон Вригт

Георг Хенрик фон Вригт (Wright) (р. 1916), финский философ и логик. Учился в Хельсинском университете и в Кембридже. Докторскую диссертацию защитил в 1941 г. в Хельсинки. В 1943 г. стал профессором Хельсинского университета. В 1945 г. был приглашен в Кембридж, где стал преемником ушедшего на пенсию Л.Витгенштейна. В 1951 г. вернулся в Финляндию. В 1968-70 годах президент Академии наук Финляндии.

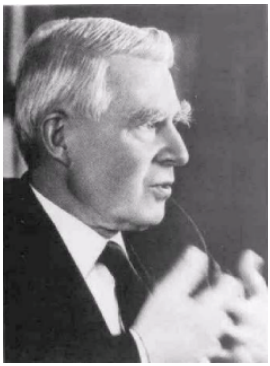

Рис. 3.2: Г. фон Вригт.

В центре исследований фон Вригта – проблемы индукции и вероятности, деонтическая и временная логики, анализ человеческого действия и его мотивов, логическая природа отношений между детерминированностью и свободой.

3.5. Временные логики

Временные (темпоральные) логики – это модальные логики. Они строятся добавлением к логике высказываний новых знаков, отражающих свойства времени.

Приведем рассуждение Георга Хенрика фон Вригта: «Возьмем, например, процесс выпадения дождя. Этот процесс продолжается

некоторое время, а затем прекращается. Но предположим, что это происходит не внезапно, а постепенно. Пусть

$$\mathcal{A}.....\neg\mathcal{A}$$

иллюстрирует, что на определенном отрезке времени вначале определенно идет дождь ($\mathcal{A}$), потом определенно не идет дождь ($\neg\mathcal{A}$), а между этими временными точками находится переходная область, когда может капать небольшое количество капель – слишком мало для того, чтобы заставить нас сказать, что идет дождь, но слишком много для того, чтобы мы могли воздержаться от утверждения, что дождь определенно закончился. В этой области высказывание $\mathcal{A}$ ни истинно, ни ложно». Таким образом, появляется еще третье значение высказывания: «ни истинно, ни ложно»; или «и истинно, и ложно»; или «неопределенно». Рассмотрение реального процесса (во времени) заставляет отступиться от классической двузначной логики. Логика примера фон Вригта многозначная, значение высказывания изменяется во времени.

3.5.1. Временная логика Прайора

Логика времени Прайора [51] – это логика будущего[4]. Она содержит новый знак F. Формула $F\mathcal{A}$ интерпретируется как «будет $\mathcal{A}$».

Можно ввести знак G:

$$G\mathcal{A} = \neg F\neg\mathcal{A},$$

который читается как «всегда будет $\mathcal{A}$».

К логике высказываний добавляются следующие аксиомы:

$$\begin{aligned} &F(\mathcal{A}\vee\mathcal{B}) \equiv F\mathcal{A}\vee F\mathcal{B} \\ &FF\mathcal{A} \Rightarrow F\mathcal{A} \end{aligned} \tag{3.9}$$

и правила вывода:

$$\frac{\mathcal{A}}{G\mathcal{A}}, \qquad \frac{\mathcal{A}\equiv\mathcal{B}}{F\mathcal{A}\equiv F\mathcal{B}}.$$

Данная логика была создана с целью формализации представлений философа Диодора из Мегар о возможности и необходимости.

[4] Будущее по-английски – Future.

Возможное и необходимое определяются через знаки F и G как

$$<>\mathcal{A} = \mathcal{A} \vee F\mathcal{A}, \qquad \Box\mathcal{A} = \mathcal{A}\&G\mathcal{A}.$$

Прайор показал, что алетическая система, содержащаяся в предложенной временной логике, сильнее исчисления S4, но слабее S5.

3.5.2. Временная логика Леммона

Леммон предложил [9] *минимальную* временную логику, основанную на модальностях F – «будет» и P – «было».

Предложенное им исчисление K_t добавляет к логике высказываний следующие аксиомы:

$$\begin{aligned} &\neg F\neg(\mathcal{A} \Rightarrow \mathcal{B}) \Rightarrow (F\mathcal{A} \Rightarrow F\mathcal{B}) \\ &\neg P\neg(\mathcal{A} \Rightarrow \mathcal{B}) \Rightarrow (P\mathcal{A} \Rightarrow P\mathcal{B}) \\ &F\neg P\neg\mathcal{A} \Rightarrow \mathcal{A} \\ &P\neg F\neg\mathcal{A} \Rightarrow \mathcal{A}. \end{aligned} \tag{3.10}$$

Добавочные правила вывода:

$$\frac{\mathcal{A}}{\neg F\neg\mathcal{A}}, \qquad \frac{\mathcal{A}}{\neg P\neg\mathcal{A}}.$$

Данная логика не делает никаких предположений о природе времени, как-то: бесконечность времени в прошлом или будущем, непрерывность или неразветвленность.

3.5.3. Временная логика фон Вригта

К логике высказываний добавляется новый символ T, представляющий бинарную связку. Формула $\mathcal{A}T\mathcal{B}$ читается как «сейчас происходит событие $\mathcal{A}$, а затем, т.е. *в следующий момент времени*, происходит событие $\mathcal{B}$».

Можно построить формулу $-T(-T(-T...))...$, являющуюся цепочкой формул, описывающих состояния, которые последовательно, т.е. в различные моменты времени некоторого отрезка времени, проходит мир. Особый интерес представляет случай, когда выражения, обозначенные как «—», являются описаниями состояния.

Такую цепочку назовем фрагментом *истории* мира. Термин «история» имеет двойственное значение: он может означать последовательность как самих полных состояний мира, так и их описаний.

Дополнительные аксиомы:

$$\begin{array}{l}(\mathcal{A} \vee \mathcal{B}T\mathcal{C} \vee \mathcal{D}) \Leftrightarrow (\mathcal{A}T\mathcal{C}) \vee (\mathcal{A}T\mathcal{D}) \vee (\mathcal{B}T\mathcal{C}) \vee (\mathcal{B}T\mathcal{D}) \\ (\mathcal{A}T\mathcal{B})\&(\mathcal{A}T\mathcal{C}) \Rightarrow (\mathcal{A}T\mathcal{B}\ \&\ \mathcal{C}) \\ \mathcal{A} \Leftrightarrow (\mathcal{A}T\mathcal{B} \vee \neg\mathcal{B}) \\ \neg(\mathcal{A}T\mathcal{B}\ \&\ \neg\mathcal{B}).\end{array} \tag{3.11}$$

К правилам вывода добавляется правило экстенциональности: если эквивалентность некоторых формул доказана, то они взаимозаменяемы.

Время в этой темпоральной логике дискретное; это линейное течение исчислимых последовательных случаев. Если число полных состояний мира равно 2^n, то число возможных историй в m последовательных моментах равно 2^{mn} [45, с.80-81].

3.5.4. Приложение временных логик к программированию

Пусть α – программа, $\mathfrak{P}$ – высказывание, относящееся к входным данным, которое должно быть истинно *перед* выполнением программы α, и $\mathfrak{R}$ – высказывание, которое должно быть истинно после выполнения программы α. Высказывание $\mathfrak{P}$ называется *предусловием*, а $\mathfrak{R}$ – *постусловием* программы α [5].

Программа α называется *частично правильной* по отношению к $\mathfrak{P}$ и $\mathfrak{R}$, если всякий раз, когда предусловие $\mathfrak{P}$ истинно перед выполнением α и α заканчивает работу, постусловие $\mathfrak{R}$ будет также истинно. В этом случае используется запись

$$\{\mathfrak{P}\}\alpha\{\mathfrak{R}\}.$$

Программа α называется *тотально правильной* по отношению к $\mathfrak{P}$ и $\mathfrak{R}$, если она частично правильна по отношению к $\mathfrak{P}$ и $\mathfrak{R}$, и

[5] Проведем аналогию с модальными (временными) логиками: каждое состояние памяти, возникающее при выполнении программы, аналогично «возможному миру» в семантике Крипке; пути выполнения программы определяют переходы между этими мирами [22, с.75].

обязательно завершает работу, если $\mathfrak{P}$ истинно[6]. В этом случае используется запись

$$\{\mathfrak{P}\}\alpha\downarrow\{\mathfrak{R}\}.$$

Предусловие $\mathfrak{P}$ и постусловие $\mathfrak{R}$ связаны с конкретной задачей, которую требуется решить и для решения которой и написана программа α. Нам важно *доказать,* что она правильна!

Проверка того, что программа действительно выполняет задачи, для решения которых она создавалась, – это задачи *тестирования* и *верификации* программы.

Для решения задачи верификации программы используются временные логики.

Основные идеи приложения временных логик к программированию сводятся к следующим двум положениям [22, с.146]:

1. **Описание программ.** С помощью языка временной логики, специализированного особым образом, выразить свойства программ, характеризующих их правильное вычислительное поведение.
2. **Верификация программ.** Использовать аппарат формальной теории временной логики для доказательства того, что данная программа обладает интересующим нас свойством.

Одно из первых упоминаний идеи доказательства правильности программ содержится в статье Дж. Маккарти (1961): «Вместо того, чтобы испытывать программы до тех пор, пока они не окажутся отлаженными, нужно доказывать, что они имеют желаемые свойства». К этому же времени относится выступление Э. Дейкстры с докладом «Некоторые соображения о более развитом программировании».

Впервые (1974) мысль об использовании логических временных знаков «всегда» и «иногда» для верификации программ проявилась в статье Р.Бурсталла [49]. Однако речь шла о последовательных программах, представляющих собой один вычислительный процесс. Временные верификации программ в полной мере развились по отношению к параллельным программам, т.е. программам, состоящим из нескольких процессов, работающих одновременно и *взаимодействующих* между собой.

[6] Исполнение α обязательно завершается, и $\mathfrak{R}$ должно быть истинно в любом состоянии памяти ЭВМ, которое может получиться при выполнении α. Это соответствует $\Box\,\mathfrak{R}$ (необходимо $\mathfrak{R}$) в модальной логике [22, с.75].

3.5.5. Временная логика Пнуели

Темпоральная логика Пнуели создана с целью верификации компьютерных программ. Другими словами, логика Пнуели – это формальная теория, с помощью которой можно доказывать «наличие у данной программы свойств, характеризующих ее правильное вычислительное поведение» [22, с.151].

Приведем рассуждения Пнуели по этому поводу: «...Нужно ли понятие внешнего времени, или темпоральности, для рассуждений о программах?... для последовательных программ темпоральность не является существенной. Это так потому, что эти программы имеют «внутренние часы», а именно само выполнение. Зная метку в программе и значения программных переменных, мы можем точно определить, в каком месте выполнения мы находимся. Поэтому для таких программ простые временные понятия «до» и «после» выполнения программного сегмента ... адекватны. Но при обращении к программам индетерминистским, параллельным, в которых выполнение состоит из перемешанных между собой операций из различных процессов, мы должны различать «где» и «когда» и сохранять внешнюю временную шкалу, независимую от выполнения» ([22, с.156], [50]).

Таким образом, логика Пнуели предназначена для верификации программ, связанных с параллельными вычислениями.

Временная логика Пнуели строится как логика первого порядка с прибавлением множества новых пропозициональных переменных вида atl, которые пробегают по выражениям вида «процесс P_i находится в метке l». Это дает средства для описания контрольного компонента программных состояний. Кроме того, переменные языка разбиваются на два типа: глобальные переменные и локальные переменные.

Глобальные переменные не меняют своих значений от состояния к состоянию, а значения локальных переменных меняются.

Логика Пнуели содержит дополнительные знаки F (когда-нибудь будет), G (всегда будет) и $\aleph$ (в следующий момент будет) и аксиомы:

$$G(\mathcal{A} \Rightarrow \mathcal{B}) \Rightarrow (G\mathcal{A} \Rightarrow G\mathcal{B})$$
$$\aleph(\mathcal{A} \Rightarrow \mathcal{B}) \Rightarrow (\aleph\mathcal{A} \Rightarrow \aleph\mathcal{B})$$
$$G\mathcal{A} \Rightarrow \mathcal{A}$$
$$G\mathcal{A} \Rightarrow \aleph G\mathcal{A}$$
$$G\mathcal{A} \Rightarrow \aleph\mathcal{A}$$
$$\aleph\neg\mathcal{A} \equiv \neg\aleph\mathcal{A}$$
$$(\mathcal{A}\&G(\mathcal{A} \Rightarrow \aleph\mathcal{A})) \Rightarrow G\mathcal{A}.$$

Дополнительное правило вывода:

$$\frac{\mathcal{A}}{G\mathcal{A}}.$$

В данной логике устанавливается *инвариантное правило*, гласящее, что

> *для того, чтобы установить инвариантность (неизменность) некоторого свойства $\mathcal{A}$ в данной программе, надо установить: а) оно имеет место в начале программы и б) сохраняется после выполнения каждой команды этой программы.*

Символически это правило записывается как

$$\frac{\mathcal{B} \Rightarrow \mathcal{A},\ atl\&\mathcal{B} \Rightarrow \aleph\mathcal{A}\ (\text{для всех меток } l)}{\mathcal{B} \Rightarrow G\mathcal{A}}.$$

С помощью этого правила доказывается наличие у программы различных инвариантных свойств. Например, доказывается *свойство исключения критических секций.*

О чем идет речь? В параллельных вычислениях параллельные процессы могут включать блоки (секции), содержащие команды, критические для кооперирования этих процессов. Иначе говоря, пока один процесс находится в своей критической секции и выполняет команды этой секции, второй процесс не должен войти в свою критическую секцию, поскольку изменения в ячейках памяти, происходящие при выполнении команд идущего первого процесса, исказят вычисления, на которые сориентирован второй процесс. Второй процесс должен ждать,[7] пока содержимое используемых им ячеек

[7] Для этого требуется понятие «внешнего времени».

не окажется требуемым. Ожидание происходит в силу выполнения особых команд программы, называемых *семафорными инструкциями*.

Рассмотрим пример семафорной инструкции [22, с.148-149].

Пример 3.1. В параллельной программе имеются две критические секции (КС). Семафорная инструкция – оператор «**wait ... then...**» с семафорной переменной x_4:

$$x_4 := 1;$$

	процесс 1		процесс 2
	l_0: **wait** $x_4 > 0$ **then** $x_4 := x_4 - 1;$		m_0 : **wait** $x_4 > 0$ **then** $x_4 := x_4 - 1;$
КС1	$l_1 : t_1 := x_3 * x_1;$ $l_2 : x_3 := t_1;$ $l_3 : x_4 := x_4 + 1;$	КС2	$m_1 : t_2 := x_3/x_2;$ $m_2 : x_2 := t_2;$ $m_3 : x_4 := x_4 + 1;$
	l_4 : **end**		m_4 : **end**

Предположим, что при выполнении процесса 1 значение x_4 уменьшилось до 0. Тогда компьютер вынужден остановить процесс 2, т.к. преградой к его выполнению является семафорная инструкция под меткой m_0, для выполнения которой необходимо, чтобы x_4 было больше 0. Следовательно, компьютер начнет выполнять команды с метками $l_1 - l_3$ и только после этого перейдет к продолжению выполнения процесса 2. Другими словами, пока процесс 1 находится в своей критической секции КС1, процесс 2 не достигнет своей критической секции КС2. То же самое будет происходить в случае, когда процесс 2 начнет выполняться первым.

Свойство, утверждающее, что процессы никогда не выполняют одновременно своих критических секций, называют *взаимным исключением критических секций* и записывают в виде следующей теоремы временной логики:

$$\vdash \mathcal{A} \Rightarrow G\neg(atl_1 \& ... \& atl_n), \tag{3.12}$$

где l_i – метка критической секции i-го процесса, $\mathcal{A}$ – формула, выражающая исходные условия[8].

Формальное доказательство теоремы (3.12) дано в [22, с.152-155].

[8] Исходные условия включают в себя значения программных переменных, предусловие (если оно имеется) и исходные метки всех параллельных процессов программы.

3.6. Алгоритмические логики

В начале 70-х годов XX века возникли *алгоритмические логики*. Они создавались с целью описания семантики языков программирования.

Алгоритмические логики включают высказывания вида

$$\{\mathfrak{P}\}S\{\mathfrak{R}\},$$

читающиеся как «если до выполнения оператора S было выполнено $\mathfrak{P}$, то после него будет $\mathfrak{R}$».

Эти логические языки практически одновременно изобретены Р.У.Флойдом (1967), К.Хоаром (1969) и представителями польской логической школы (А.Сальвиницкий и др. (1970)).

«В 1969 году Хоар определил простой язык программирования через логическую систему аксиом и правил вывода для доказательства частичной правильности программ. В его работе показано, что определение семантики языка не в терминах выполнения программы, а в терминах доказательства ее правильности упрощает процесс построения программы.

В 70-х годах на базе работы Хоара начинаются исследования в области аксиоматических определений языков программирования. Появляется много работ по аксиоматизации различных конструкций: от оператора присваивания до различных форм циклов, от вызовов процедур до сопрограмм. Главным упущением исследователей в эти годы является то, что они уделяли недостаточно внимания формальной логике.

В 1972 году вышла очередная статья Хоара о доказательстве правильности представления данных, что ускорило исследования по абстрактным типам данных. В СССР работы в этой области велись в Новосибирске (А.П.Ершов, В.Н.Агафонов, А.В.Замулин, И.Н.Скопина).

В 1973 году были сформулированы правила доказательства правильности для большинства конструкций языка Паскаль. В 1975 была построена автоматическая система верификации для подмножества языка Паскаль, основанная на аксиомах и правилах вывода. В 1979 году был определен язык программирования Евклид, в проект которого с самого начала была заложена идея аксиоматизации.

В 1976 году (в русском переводе в 1978 году) вышла книга Э.Дейкстры «Дисциплина программирования», в которой предлагается метод доказательства правильности программ. Суть мето-

да заключается в том, чтобы строить программу вместе с доказательством, причем доказательство должно опережать построение программы. Э.Дейкстра определил для простого языка программирования слабейшие предусловия и показал, как их можно использовать в качестве исчисления для вывода программ. Стало ясно, что использование формализма может привести к построению программ более надежным способом» [9].

3.6.1. Принципы построения алгоритмической логики

Опишем принципы построения алгоритмической логики $\mathbf{L_0}$.

Память в $\mathbf{L_0}$ разделена на *ячейки*. Каждая ячейка имеет *идентификатор*, представляющий собой слово из латинских букв и цифр и начинающийся с буквы. Ячейки содержат натуральные числа.

Программа в $\mathbf{L_0}$ состоит из *операторов*. Исходный оператор – *присваивание*:

$$x := e,$$

где x – идентификатор, а e – выражение, составленное из идентификаторов и натуральных чисел.

Пусть состояние памяти после присваивания удовлетворяет условию $\mathcal{A}(x)$, где $\mathcal{A}$ – формула формальной арифметики. Тогда состояние, которое было до присваивания, опишем формулой $\mathcal{A}(x|e)$.[10] Иначе говоря, имеем высказывание

$$\{\mathcal{A}(x|e)\}x := e\{\mathcal{A}(x)\}.$$

Пусть два оператора S_1, S_2 выполняются один за другим. Тогда этому соответствует логическая запись вида

$$\frac{\{\mathfrak{A}\}S_1\{\mathfrak{B}\},\ \{\mathfrak{C}\}S_2\{\mathfrak{D}\},\ \mathfrak{B} \Rightarrow \mathfrak{C}}{\{\mathfrak{A}\}S_1S_2\{\mathfrak{D}\}}.$$

Условный оператор – это конструкция

$$\mathbf{IF}\ \mathcal{A}_1 \to S_1 \triangle \mathcal{A}_2 \to S_2 \triangle \ ... \ \triangle \mathcal{A}_n \to S_n\ \mathbf{FI},$$

[9] Использованы материалы сайта http://stud.math.rsu.ru/deikstra/.

[10] К примеру, если $\mathcal{A}(x)$ – это $x < 2$ и $x := x + 1$, то $\mathcal{A}(x|x + 1)$ – это $x + 1 < 2$. Следовательно, чтобы после присваивания $x := x + 1$ стало истинным $x < 2$, требуется, чтобы до присваивания выполнялось неравенство $x + 1 < 2$, т.е. $\models \{x + 1 < 2\}x := x + 1\{x < 2\}$.

где $\mathcal{A}_1, ...\mathcal{A}_n$ – логические выражения, построенные из отношений $e_1 = e_2, e_1 > e_2$ при помощи логических связок; S_i – последовательность операторов. Проверяются формулы $\mathcal{A}_i$ при настоящем состоянии памяти. Если ни одна из $\mathcal{A}_i$ не истинна, то фиксируется ошибка. Если же некоторые $\mathcal{A}_i$ истинны, то выбирается одна из них (неважно как) и выполняется соответствующая последовательность операторов S_i.

Если каждая из команд S_i описана в логике $\mathbf{L_0}$ как $\{\mathfrak{A}_i\}S_i\{\mathfrak{B}\}$, то условный оператор описывается логической формулой

$$\{(\mathcal{A}_1 \Rightarrow \mathfrak{A}_1)\&...\&(\mathcal{A}_n \Rightarrow \mathfrak{A}_n)\}\mathbf{IF}\ \mathcal{A}_1 \to S_1 \triangle \mathcal{A}_2 \to S_2 \triangle ...$$
$$... \triangle \mathcal{A}_n \to S_n\ \mathbf{FI}\{\mathfrak{B}\}.$$

Циклам отвечает конструкция

$$\mathbf{DO}\ \mathcal{A}_1 \to S_1 \triangle ... \triangle \mathcal{A}_n \to S_n\ \mathbf{OUT}\ \mathcal{B}_1 \to T_1 \triangle ... \triangle \mathcal{B}_m \to T_m\ \mathbf{OD}.$$

Выполняется оператор цикла следующим образом. Проверяются формулы $\mathcal{A}_1, ..., \mathcal{A}_n, \mathcal{B}_1, ..., \mathcal{B}_m$ при настоящем состоянии памяти. Если ни одна из них не истинна, то фиксируется ошибка. Если же некоторые истинны, то выбирается одна из них (неважно как). Если выбрана $\mathcal{A}_i$, то выполняется соответствующая последовательность операторов S_i и выполнение цикла возобновляется. Если выбрана $\mathcal{B}_i$, то выполняется соответствующая последовательность операторов T_i и выполнение цикла завершается.

Если каждая из команд S_i описана в логике $\mathbf{L_0}$ как $\{\mathfrak{A}_i\}S_i\{\mathfrak{C}_i\}$, а T_i как $\{\mathfrak{D}_i\}T_i\{\mathfrak{B}_i\}$, то однократное выполнение цикла описывается логической формулой

$$\{(\mathcal{A}_1 \Rightarrow \mathfrak{A}_1)\&...\&(\mathcal{A}_n \Rightarrow \mathfrak{A}_n)\&(\mathcal{B}_1 \Rightarrow \mathfrak{D}_1)\&...\&(\mathcal{B}_m \Rightarrow \mathfrak{D}_m)\&$$
$$\&(\mathcal{A}_1 \vee ... \vee \mathcal{A}_n \vee \mathcal{B}_1 \vee ... \vee \mathcal{B}_m)\}\ \mathbf{IF}\mathcal{A}_1 \to S_1 \triangle ... \triangle \mathcal{A}_n \to S_n \triangle$$
$$\triangle \mathcal{B}_1 \to T_1 \triangle ... \triangle \mathcal{B}_m \to T_m\ \mathbf{FI}\{\mathfrak{B}\}.$$

Двукратное выполнение цикла дает формулу

$$\{(\mathcal{A}_1 \Rightarrow \mathfrak{A}_1)\&...\&(\mathcal{A}_n \Rightarrow \mathfrak{A}_n)\&(\mathcal{B}_1 \Rightarrow \mathfrak{D}_1)\&...\&(\mathcal{B}_m \Rightarrow \mathfrak{D}_m)\&$$
$$\&(\mathcal{A}_1 \vee ... \vee \mathcal{A}_n \vee \mathcal{B}_1 \vee ... \vee \mathcal{B}_m)\}\ \mathbf{IF}\mathcal{A}_1 \to S_1 \triangle ... \triangle \mathcal{A}_n \to S_n \triangle$$
$$\triangle \mathcal{B}_1 \to T_1 \triangle ... \triangle \mathcal{B}_m \to T_m\ \mathbf{FI}\{\mathfrak{C}_1 \vee ... \vee \mathfrak{C}_n\},$$
$$\{(\mathcal{A}_1 \Rightarrow \mathfrak{A}_1)\&...\&(\mathcal{A}_n \Rightarrow \mathfrak{A}_n)\&(\mathcal{B}_1 \Rightarrow \mathfrak{D}_1)\&...\&(\mathcal{B}_m \Rightarrow \mathfrak{D}_m)\&$$

$$\&(\mathfrak{C}_1 \vee ... \vee \mathfrak{C}_n)\&(\mathcal{A}_1 \vee ... \vee \mathcal{A}_n \vee \mathcal{B}_1 \vee ... \vee \mathcal{B}_m)\}\ \mathbf{IF}\mathcal{A}_1 \to S_1 \triangle ...$$

$$... \triangle \mathcal{A}_n \to S_n \triangle \mathcal{B}_1 \to T_1 \triangle ... \triangle \mathcal{B}_m \to T_m\ \mathbf{FI}\{\mathfrak{B}\}$$

и т.д. до бесконечности. Условие правильности k шагов цикла обозначим через $\mathbf{DO}^k\{\mathfrak{B}\}$. Цикл корректен, если

$$\vdash \mathbf{DO}^1\{\mathfrak{B}\} \vee \mathbf{DO}^2\{\mathfrak{B}\} \vee ... \vee \mathbf{DO}^k\{\mathfrak{B}\} \vee ...$$

Но это бесконечно длинная формула! Она не является формулой языка $\mathbf{L_0}$. Появление таких формул – это серьезная проблема для алгоритмических логик.

3.6.2. Чарльз Хоар

Рис. 3.3: Ч.А.Р. Хоар.

Профессор Чарльз Хоар (Charles Antony Richard Hoare) родился в 1934 г. в Англии. В 1980 г. получил престижную премию Алана Тьюринга за вклад в формальное определение языков программирования посредством аксиоматической семантики. Хоар известен своими работами в области алгебры программ. Превращение программирования в серьезную профессиональную дисциплину стало ведущим мотивом его научной деятельности [11].

С начала 60-х годов профессор Хоар является ведущим ученым в мире в области компьютерных наук[12]. В числе его заслуг – разработка логики Хоара (Hoare Logic), которая является научной основой для конструирования корректных программ и используется для определения и разработки языков программирования. Хоар написал ряд трудов по созданию спецификаций, проектированию, реализации и сопровождению программ, показывающих важность научных результатов для увеличения производительности компьютеров и повышения надежности программного обеспечения. За заслуги в области образования и

[11] См. http://www.vavilon.ru/koval/ruslaslov-01.html

[12] Использована статья Anatem с сайта: http://www.ct.kz/

компьютерных наук королева Великобритании пожаловала Хоару рыцарское звание.

3.6.3. Алгоритмическая логика Хоара

Основой для логики выводов правильных программ служат аксиомы Хоара (правила верификации). Они допускают интерпретации в терминах программных конструкций.

Сформулируем аксиомы Хоара, которые определяют предусловия как достаточные условия, гарантирующие, что исполнение соответствующего оператора при успешном завершении приведет к желаемым постусловиям[13].

$A1$. *Аксиома присваивания*:

$$\{\mathfrak{R}(x|e)\}x := e\{\mathfrak{R}\}.$$

Неформальное объяснение аксиомы: так как x после выполнения будет содержать значение e, то $\mathfrak{R}$ будет истинно после выполнения, если результат подстановки e вместо x в $\mathfrak{R}$ истинен перед выполнением.

$A2$. $\{\mathfrak{Q}\}S\{\mathfrak{P}\}$ & $(\mathfrak{P} \Rightarrow \mathfrak{R}) \Rightarrow \{\mathfrak{Q}\}S\{\mathfrak{R}\}$.
$A3$. $\{\mathfrak{Q}\}S\{\mathfrak{P}\}$ & $(\mathfrak{R} \Rightarrow \mathfrak{Q}) \Rightarrow \{\mathfrak{R}\}S\{\mathfrak{P}\}$.

Пусть S – это последовательность из двух операторов S_1, S_1 (составной оператор).

$A4$. $\{\mathfrak{Q}\}S_1\{\mathfrak{P}_1\}$ & $\{\mathfrak{P}_1\}S_2\{\mathfrak{R}\} \Rightarrow \{\mathfrak{Q}\}S\{\mathfrak{R}\}$.

Очевидно, что это правило можно сформулировать для последовательности, состоящей из n операторов.

Сформулируем правило для условного оператора (краткая форма):

$A5$. $\{\mathfrak{Q}\&\mathfrak{B}\}S_1\{\mathfrak{R}\}\&(\mathfrak{Q}\&\neg\mathfrak{B} \Rightarrow \mathfrak{R}) \Rightarrow \{\mathfrak{Q}\}$**if** $\mathfrak{B}$ **then** $S_1\{\mathfrak{R}\}$.

Аксиома $A5$ соответствует интерпретации условного оператора в языке программирования.

Сформулируем правило для альтернативного оператора (полная форма условного оператора):

$A6$. $(\{\mathfrak{Q}\&\mathfrak{B}\}S_1\{\mathfrak{R}\})\&(\{\mathfrak{Q}\&\neg\mathfrak{B}\}S_2\{\mathfrak{R}\}) \Rightarrow \{\mathfrak{Q}\}$**if** $\mathfrak{B}$ **then** S_1 **else** $S_2\{\mathfrak{R}\}$.
и для операторов цикла.

[13] Использованы материалы сайта http://stud.math.rsu.ru/deikstra/.

Предусловия и постусловия цикла **until** (до) удовлетворяют правилу:

$A7.\ \{\mathfrak{Q}\&\mathfrak{B}\}S_1\{\mathfrak{Q}\} \Rightarrow \{\mathfrak{Q}\}$**repeat** S_1 **until** $\mathfrak{B}\{\mathfrak{Q}\&\neg\mathfrak{B}\}$.

Правило вводит важное понятие инварианта цикла. Предикат $\mathfrak{Q}$, истинный перед выполнением и после выполнения каждого шага цикла, называется *инвариантным отношением* или просто *инвариантом цикла*. В математике термин «инвариантный» означает не изменяющийся под воздействием совокупности рассматриваемых математических операций. В данном случае единственная операция – это выполнение шага цикла при условии истинности $\mathfrak{Q}$ вначале.

Предусловия и постусловия цикла **while** (пока) удовлетворяют правилу:

$A8.\ \{\mathfrak{Q}\&\mathfrak{B}\}S_1\{\mathfrak{Q}\} \Rightarrow \{\mathfrak{Q}\}$**while** $\mathfrak{B}$ **do** $S_1\{\mathfrak{Q}\&\neg\mathfrak{B}\}$.

Аксиомы $A1 - A8$ можно использовать для проверки согласованности передачи данных от оператора к оператору, для анализа структурных свойств текстов программ, для установления условий окончания цикла. Кроме того, правила можно использовать для анализа результатов выполнения программы, что связано с семантикой программы.

Пример 3.2. Пусть надо определить частное q и остаток r от деления n на m.

Входные данные:

n, m – целые неотрицательные числа, причем $m > 0$.

Выходные данные:

q, r – целые неотрицательные числа.

Описание программы S:

$$\begin{array}{l}\textbf{задать}(n, m)\\ r := n;\ q := 0;\\ \textbf{while}\ m <= r\ \textbf{do}\\ \textbf{begin}\\ r := r - m;\ q := q + 1\\ \textbf{end};\\ \textbf{выдать}(q, r);\end{array} \qquad (3.13)$$

Сформулируем предусловие $\mathfrak{A}$:

$$m > 0\&(n, m) - \text{целые числа}.$$

Сформулируем постусловие $\mathfrak{B}$:

$$(r < m)\&(n = m * q + r).$$

Нужно доказать, что

$$\vdash \{\mathfrak{A}\}S\{\mathfrak{B}\}$$

или

$$\vdash \{m > 0 \& (n, m) - \text{целые}\}\ S\ \{(r < m)\&(n = m * q + r)\}.$$

Доказательство

	Язык	**Метаязык**
1	$\mathfrak{A} \Rightarrow n = n + m * 0$	вывод в формальной арифметике
2	$\{n = n + m * 0\}r := n\{n = r + m * 0\}$	аксиома $A1$
3	$\{n = r + m * 0\}q := 0\{n = r + m * q\}$	аксиома $A1$
4	$\{\mathfrak{A}\}r := n\{n = r + m * 0\}$	$A3$ к пунктам 1 и 2
5	$\{\mathfrak{A}\}r := n; q := 0\{n = r + m * q\}$	$A4$ к пунктом 3 и 4
6	$n = r + m * q\&$ $\&m <= r => n = (r - m) + m * (q + 1)$	арифметика
7	$\{n = (r - m) + m * (q + 1)\}r := r - m$ $\{n = r + m * (q + 1)\}$	аксиома $A1$
8	$\{n = r + m * (q + 1)\}q := q + 1$ $\{n = r + m * q\}$	аксиома $A1$
9	$\{n = (r - m) + m * (q + 1)\}r := r - m;$ $q := q + 1\{n = r + m * q\}$	$A4$ к пунктам 7 и 8
10	$\{n = r + m * q\&m <= r\}r := r - m;$ $q := q + 1\{n = r + m * q\}$	$A2$ к пунктам 6 и 9
11	$\{n = r + m * q\&m <= r\}$**while** $m <= r$ **do beqin** $r := r - m; q := q + 1$ **end** $\{(n = r + m * q)\&\neg(m <= r)\}$	$A8$ к пункту 10
12	$\{\mathfrak{A}\}S\{(n = r + m * q)\&\neg(m <= r)\}$	$A4$ к пунктам 5 и 11 ч.т.д.

Докажем, что выполнение программы заканчивается. Предположим, что программа не заканчивается. Тогда должна существовать бесконечная последовательность значений r и q, удовлетворяющая условиям:

1) $m <= r$;

2) r, q – неотрицательные целые числа.

Значение r на каждом шаге цикла уменьшается на положительную величину: $r := r - m(m > 0)$. Значит, последовательность значений r и q является конечной, т.е. найдется такое значение r, для которого не будет выполняться условие $m <= r$, и циклический процесс завершится.

Таким образом, установлено, что

$$\{\mathfrak{A}\}S \downarrow \{\mathfrak{B}\}.$$

Иначе говоря, наша программа S является тотально правильной.

Часть II

Алгоритмы

Глава 4

Алгоритмы

4.1. Понятие алгоритма и вычислимой функции

Алгоритм – это точное предписание, определяющее вычислительный процесс, ведущий к искомому результату (А.А.Марков, 1954, [27]).

Данное определение возникло до появления ЭВМ и науки программирования. Но как только последние стали заметным явлением в сумме технологий XX века, стало возможным следующее определение алгоритма.

«*Алгоритм* – это раз и навсегда составленная программа, в которой все заранее предусмотрено. Выражаясь популярно, алгоритм – это точное, воспроизводимое, поддающееся исполнению предписание, определяющее – шаг за шагом, – каким путем надлежит решать данную задачу. Алгоритмом является любое формализованное доказательство математической теоремы, равно как и программа цифровой машины, переводящей с одного языка на другой» (С.Лем, 1967, [20, с.137]).

«Алгоритмом принято называть систему вычислений, которая для некоторого класса математических задач из записи A «условий» задачи позволяет при помощи однозначно определенной последовательности операций, совершаемых «механически», без вмешательства творческих способностей человека, получить запись B

«решения» задачи» (А.Н.Колмогоров, В.А.Успенский, 1958, [14]).

Таким образом, *алгоритм* – это процедура $\mathfrak{A}$, которая [13]:

1) Применяется к строго определенным исходным данным A. Ее нельзя применять к другому типу данных, она либо будет не способна с ними что-то сделать, либо может выдать бессмысленный результат, т.е. результат, не поддающийся анализу с точки зрения тех ожиданий, которые связывались с алгоритмом при его создании.
2) Расчленена на отдельные простые шаги; каждый шаг состоит в непосредственной обработке возникшего к этому шагу состояния S в состояние $S^* = \Omega_{\mathfrak{A}}(S)$.
3) Непосредственная переработка S в $S^* = \Omega_{\mathfrak{A}}(S)$ производится однозначно заданным способом лишь на основании информации о виде заранее ограниченной «активной» части состояния S и затрагивает лишь эту активную часть.
4) Процесс переработки $A_0 = A$ в $A_1 = \Omega_{\mathfrak{A}}(A_0)$, A_1 в $A_2 = \Omega_{\mathfrak{A}}(A_1)$, A_2 в $A_3 = \Omega_{\mathfrak{A}}(A_2)$ и т.д. продолжается до тех пор, пока либо не произойдет безрезультатная остановка (или оператор $\Omega_{\mathfrak{A}}$ не определен для возникшего состояния), либо не появится сигнал о получении «решения». При этом не исключается возможность непрекращающегося процесса переработки.

Каждый алгоритм задает функцию, поскольку по набору исходных данных выдает результат применения алгоритма к этим данным. Естественно назвать функцию, значения которой могут находится с помощью некоторого алгоритма, – *вычислимой функцией*. Таким образом, вычислимая функция – это такая функция, для которой существует вычисляющий ее значения алгоритм [43, с.15].

4.2. Рекурсивные функции

Какие функции могут быть вычислимыми? Хотелось бы иметь их описание, не связанное напрямую с понятием алгоритма.

Функцию $f : X \to Y$ будем называть *частичной*, если она определена не для каждого значения $x \in X$. Множество тех $x \in X$, для которых однозначно указано соответствующее значение функции f, называется *областью определения* функции.

Ограничимся только функциями, заданными на множестве натуральных чисел N.

4.2.1. Примитивно рекурсивные функции

Введем следующие функции

$$s^1(x) = x + 1,$$

$$o^n(x_1, ..., x_n) = 0,$$

$$I^n_m(x_1, ..., x_n) = x_m \ (1 \le m \le n;\ n = 1, 2, ...),$$

называемые далее *простейшими.*

Пусть даны частичные функции $g : N^n \to N$ и $h : N^{n+2} \to N$. Частичная функция $f : N^{n+1} \to N$ получена из функций g, h *примитивной рекурсией,* если

$$\begin{aligned} &f(x_1, ..., x_n, 0) = g(x_1, ..., x_n), \\ &f(x_1, ..., x_n, y + 1) = h(x_1, ..., x_n, y, f(x_1, ..., x_n, y)). \end{aligned} \tag{4.1}$$

Для $n = 0$ уравнения (4.1) принимают вид ($g = const = a$)

$$\begin{aligned} &f(0) = a, \\ &f(x + 1) = h(x, f(x)). \end{aligned} \tag{4.2}$$

Как видно из этих уравнений, задав исходные данные $(x_1, ..., x_n, 0)$, можно шаг за шагом найти значения функции для $(x_1, ..., x_n, 1), (x_1, ..., x_n, 2), ..., (x_1, ..., x_n, m), ...$

Символически задание f через примитивную рекурсию можно записать как

$$f = \mathfrak{R}(g, h).$$

Частичная функция $f : N^{n+1} \to N$ называется *примитивно рекурсивной относительно множества частичных функций* $\sum$, если f получается из функций множества $\sum$ и простейших функций конечным числом операций подстановки (суперпозиции) и примитивной рекурсии.

Если $\sum = \emptyset$, то примитивно рекурсивная относительно множества частичных функций $\sum$ функция получается только из простейших функций и поэтому ее называют просто *примитивно рекурсивной.*

Нетрудно понять, что примитивно рекурсивные функции являются всюду определенными функциями.

Пример 4.1. Функция $f(x,y) = x + y$ примитивно рекурсивна.

В самом деле,

$$f(x,0) = x + 0 = x = I_1^1(x),$$
$$f(x,y+1) = x + (y+1) = (x+y) + 1 = s^1(x+y) = s^1(f(x,y)).$$

Это означает, то функция $x + y$ строится из примитивно рекурсивных функций $I_1^1, h(x,y,z) = z + 1 = s^1(z)$ с помощью примитивной рекурсии. Поэтому $x + y$ примитивно рекурсивная функция.

Пример 4.2. Функция $f(x,y) = xy$ примитивно рекурсивна.

Действительно,

$$\begin{aligned} f(x,0) &= x \cdot 0 = o^1(x), \\ f(x,y+1) &= x(y+1) = xy + x = f(x,y) + x. \end{aligned} \tag{4.3}$$

Если взять

$$g(x) = o^1(x), \quad h(x,y,z) = z + x$$

и вспомнить, что h – это примитивно рекурсивная функция (пример 4.1), то убеждаемся в примитивной рекурсивности функции xy.

4.2.2. Частично рекурсивные функции

Пусть дана частичная функция $f : N^n \to N$. Зафиксируем данные $(x_1, ..., x_n)$. Введем функцию

$$\mathbf{M}f(x_1, ..., x_n) = \min_{y \in N}\{f(x_1, ..., x_{n-1}, y) = x_n\}.$$

По существу, $\mathbf{M}$ – это операция на множестве частичных функций. Результатом является новая частичная функция с тем же числом аргументов. Назовем эту операцию *минимизацией*.

Частичная функция $f : N^{n+1} \to N$ называется *частично рекурсивной относительно множества частичных функций* $\sum$, если f получается из функций множества $\sum$ и простейших функций конечным числом операций подстановки (суперпозиции), примитивной рекурсии и минимизации.

Если $\sum = \emptyset$, то частично рекурсивная относительно множества частичных функций $\sum$ функция получается только из простейших функций, и поэтому ее называют просто *частично рекурсивной*.

Пример 4.3. Функция

$$g(x) = \begin{cases} x - 1, & x > 0, \\ \text{не определено}, & x = 0 \end{cases}$$

– частично рекурсивна.

Действительно,

$$g(x) = \mathbf{M}s^1(x) = \min_{y \in N}\{s^1(y) \equiv y + 1 = x\}.$$

Следовательно, g получена из простейшей функции s^1 с помощью операции минимизации.

Каждая примитивно рекурсивная функция является частично рекурсивной. Обратное неверно, поскольку в класс частично рекурсивных функция в соответствии с определением попадают частичная функция $\mathbf{M}s^1$ (пример 4.3) и, например, нигде не определенная функция

$$f(x) = \min_{y \in N}\{x + 1 + y = 0\}.$$

Обратим внимание на то, что минимизация может быть организована как последовательный (алгоритмический) процесс.

Действительно, последовательно находим

$$f(x_1, ..., x_{n-1}, 0), f(x_1, ..., x_{n-1}, 1), ..., f(x_1, ..., x_{n-1}, y), ...$$

Наименьшее a, для которого

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

– это значение для $\mathbf{M}f(x_1, ..., x_n)$.[1]

4.2.3. Тезис Чёрча

Теперь мы в состоянии ответить на вопрос, какие функции являются вычислимыми. Это частично рекурсивные функции. Действительно, внимательный анализ определения этих функций выявляет заложенную в это определение процедуру их *вычисления*.

Напомним, что понятие вычислимости связывалось с понятием алгоритма, поэтому общепринятой является гипотеза[2], именуемая как

[1] Возможно, что процесс нахождения не будет завершен, но это соответствует тому, что $\mathbf{M}f$ может быть истинной частичной функцией.

[2] Этот тезис не может быть доказан из-за того, что понятие вычислимой функции является интуитивным, т.е. отсутствует строгое определение.

Тезис Чёрча. *Класс алгоритмически (машинно) вычислимых частичных функций совпадает с классом всех частично рекурсивных функций.*

4.3. Машина Тьюринга-Поста

Машины Тьюринга-Поста – это пример алгоритма. Придуман этот алгоритм независимо Аланом Тьюрингом и Эмилем Постом.

Машина Тьюринга-Поста $\mathfrak{T}$ состоит из следующих «частей»:

1. Ленты, разбитой на конечное число ячеек.
2. Внешней памяти, принимающей одно из состояний, входящих в множество $A = \{a_0, a_1, ..., a_m\}$. Ячейки ленты находятся в одном и только в одном из состояний из множества A. Состояние a_0 называется *пустым*.
3. Внутренней памяти, принимающей одно из состояний, входящих в множество $Q = \{q_0, q_1, ..., q_n\}$. Состояние q_0 называется «стоп».
4. Головки, двигающейся вдоль ленты и считывающей содержимое ячейки, напротив которой она останавливается.
5. Механического устройства, передвигающего головку и меняющего состояния внешней и внутренней памяти. Если головка в состоянии q стоит напротив ячейки с номером k, то изменения состояния внутренней памяти и состояния ячейки происходят одновременно.

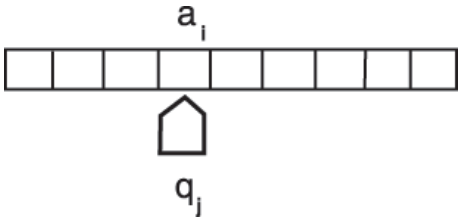

Рис. 4.1: Машина Тьюринга-Поста.

Работа машины Тьюринга-Поста $\mathfrak{T}$ осуществляется посредством команд, которые выполняет механическое устройство. *Команда* имеет один из следующих трех возможных видов:

$$q_s a_i \to q_t a_j, \qquad q_s a_i \to q_t L, \qquad q_s a_i \to q_t R,$$

где L – это движение головки влево на одну ячейку, а R – вправо. При этом всегда самое левое в записи команды $q_s \neq q_0$.

Смысл команд таков: если головка в состоянии q_s обозревает ячейку в состоянии a_i, то в первом случае она меняет свое состояние на q_t, а ячейки на a_j, во втором случае – свое состояние на q_t и сдвигается влево и, наконец, в третьем – вправо.

Конечный набор команд образует *программу*.

Состояние машины $\mathfrak{T}$ – это последовательность состояний $a_{i_1}, ..., a_{i_r}$ ячеек ленты, состояние внутренней памяти g_s головки и номер k воспринимаемой (читаемой) ячейки в состоянии a_{i_k}.

Состояние машины $\mathfrak{T}$ записываем в виде

$$a_{i_1} ... a_{i_{k-1}} q_s a_{i_k} ... a_{i_r} \tag{4.4}$$

и называем *машинным словом* $\mathfrak{m}$ в алфавите $A \cup Q$. Символ g_s может быть самым левым, но не может быть самым правым в машинном слове, так как справа от него должно быть считываемое состояние ячейки.

Под воздействием программы происходит изменение состояния машины, сопровождающееся переделкой исходного машинного слова

$$\mathfrak{m} \to \mathfrak{m}^{(1)} \to ... \to \mathfrak{m}^{(p)}.$$

4.3.1. Вычисления функций на машине Тьюринга-Поста

Пусть $\mathfrak{m}$ машинное слово в алфавите $A \cup Q$ машины $\mathfrak{T}$. Вычеркнем из слова $\mathfrak{m}$ букву a_0 и буквы из Q. Получаем редуцированное слово $[\mathfrak{m}]$.

Редуцированное слово $[\mathfrak{m}]$ перерабатывается машинной $\mathfrak{T}$ в (редуцированное) слово $\mathfrak{b}$, если для некоторой программы $\mathfrak{P}$ и некоторого p

$$q_1 a_0 [\mathfrak{m}] \to (q_1 a_0 [\mathfrak{m}])^{(1)} \to ... \to (q_1 a_0 [\mathfrak{m}])^{(p)}$$

и

$$q_0 \in (q_1 a_0 [\mathfrak{m}])^{(p)}, \quad \mathfrak{b} = [(q_1 a_0 [\mathfrak{m}])^{(p)}].$$

Символически факт переработки слова $[\mathfrak{m}]$ записываем в виде

$$\mathfrak{b} = \mathfrak{P}([\mathfrak{m}])).$$

Если ни для какого натурального p не выполняется условие

$$q_0 \in (q_1 a_0 [\mathfrak{m}])^{(p)},$$

то говорят, что машина *неприменима* к слову $[\mathfrak{m}]$. В этом случае выражение $\mathfrak{P}([\mathfrak{m}])$ является *неопределенным.*

Пусть $f(\mathfrak{r})$ функция, определенная на словах редуцированного алфавита $[A] = \{a_1, ..., a_m\}$. Если для каждого слова $\mathfrak{r}$ в этом алфавите

$$f(\mathfrak{r}) = \mathfrak{P}(\mathfrak{r}),$$

то говорим, что машина $\mathfrak{T}$ *вычисляет функцию* f.

Машинное предложение – это выражение вида

$$a_0 \mathfrak{r}_1 a_0 \mathfrak{r}_2 a_0 ... a_0 \mathfrak{r}_t,$$

где $\mathfrak{r}_i$ слова в словаре $[A]$.

Программа $\mathfrak{P}$ перерабатывает машинное предложение в редуцированное слово $\mathfrak{b}$, если

$$q_1 a_0 \mathfrak{r}_1 a_0 \mathfrak{r}_2 a_0 ... a_0 \mathfrak{r}_t \to (q_1 a_0 \mathfrak{r}_1 a_0 \mathfrak{r}_2 a_0 ... a_0 \mathfrak{r}_t)^{(1)} \to ...$$

$$... \to (q_1 a_0 \mathfrak{r}_1 a_0 \mathfrak{r}_2 a_0 ... a_0 \mathfrak{r}_t)^{(p)}$$

и

$$q_0 \in (q_1 a_0 \mathfrak{r}_1 a_0 \mathfrak{r}_2 a_0 ... a_0 \mathfrak{r}_t)^{(p)}, \ \mathfrak{b} = [(q_1 a_0 \mathfrak{r}_1 a_0 \mathfrak{r}_2 a_0 ... a_0 \mathfrak{r}_t])^{(p)}].$$

Символически

$$\mathfrak{b} = \mathfrak{P}(\mathfrak{r}_1, \mathfrak{r}_2, ..., \mathfrak{r}_t).$$

Если $f(\mathfrak{r}_1, ..., \mathfrak{r}_t)$ – *частично словарная функция*, т.е. функция, определенная на словах некоторого редуцированного алфавита $[A] = \{a_1, ..., a_m\}$, обладает свойством

$$f(\mathfrak{r}_1, ..., \mathfrak{r}_t) = \mathfrak{P}(\mathfrak{r}_1, \mathfrak{r}_2, ..., \mathfrak{r}_t),$$

то говорим, что *она вычислима на машине* $\mathfrak{T}$.

Теорема 4.1. *Все частично словарные функции, вычислимые на машине Тьюринга-Поста, являются частично рекурсивными.*

Доказательство. См. [26, с.228].

Теорема 4.2. *Для любой частично рекурсивной функции существует вычисляющая ее машина Тьюринга-Поста.*

Доказательство. См. [26, с.230].

4.3.2. Примеры вычислений

Приведем примеры вычислений на машине Тьюринга-Поста.

Полагаем

$$A = \{a_0, 1\}, \quad Q = \{q_0, ..., q_n\}.$$

Вводим следующее представление натуральных чисел в виде слов словаря A

$$0 = 1^0 = a_0, 1 = 1^1, 2 = 11 = 1^2, 3 = 111 = 1^3, ..., n = \underbrace{1...1}_{n \text{ раз}} = 1^n, ...$$

Пример 4.4. Вычислим функцию $f(n) = n + 1$.

Начальное машинное слово $\mathfrak{m} = q_1 a_0 1^n a_0$. Это число n.

Программа $\mathfrak{P}$

$$q_1 a_0 \to q_2 R,\ q_2 1 \to q_2 R,\ q_2 a_0 \to q_3 1,\ q_3 1 \to q_3 L,\ q_3 a_0 \to q_0 a_0.$$

Вычисление

$$q_1 a_0 1^n a_0 \to a_0 q_2 1^n a_0 \to ... \to a_0 1^n q_2 a_0 \to a_0 1^n q_3 1 \to a_0 1^{n-1} q_3 1^2 \to ...$$
$$... \to a_0 q_3 1^{n+1} \to q_3 a_0 1^{n+1} \to q_0 a_0 1^{n+1}\ (stop).$$

Машинное слово $q_0 a_0 1^{n+1}$ – это число $n + 1$. Следовательно,

$$n + 1 = 1^{n+1} = \mathfrak{P}(1^n)$$

или

$$f(n) = n + 1.$$

Пример 4.5. Вычислим функцию $g(n, m) = n + m$.

Начальное машинное слово $\mathfrak{m} = q_1 a_0 1^n a_0 1^m a_0$. Это пара чисел (n, m).

Программа $\mathfrak{P}$

$$q_1 a_0 \to q_2 R,\ q_2 1 \to q_2 R,\ q_2 a_0 \to q_3 1,\ q_3 1 \to q_3 R,\ q_3 a_0 \to q_4 L,$$
$$q_4 1 \to q_5 a_0,\ q_5 a_0 \to q_6 L,\ q_6 1 \to q_6 L,\ q_6 a_0 \to q_0 a_0.$$

Вычисление

$$q_1 a_0 1^n a_0 1^m a_0 \to a_0 q_2 1^n a_0 1^m a_0 \to ... \to a_0 1^n q_2 a_0 1^m a_0 \to$$
$$\to a_0 1^n q_3 1^{m+1} a_0 \to ... \to a_0 1^{n+m+1} q_3 a_0 \to q_3 1^{n+m} q_4 1 a_0 \to$$
$$\to q_3 1^{n+m} q_5 a_0 \to a_0 1^{n+m-1} q_6 1 a_0 \to ... \to a_0 q_6 1^{n+m} a_0 \to$$
$$\to q_6 a_0 1^{n+m} a_0 \to q_0 a_0 1^{n+m} a_0\ (stop).$$

Машинное слово $q_0 a_0 1^{n+m} a_0$ – это число $n + m$. Следовательно,

$$n + m = 1^{n+m} = \mathfrak{P}(1^{n+m})$$

или

$$g(n, m) = n + m.$$

4.3.3. Тезис Тьюринга

Вспоминая тезис Чёрча, естественно провести следующее рассуждение: если вычислимые функции – это частично рекурсивные, а последние вычисляются на машинах Тьюринга-Поста, то не содержится ли в этом рассуждении ответ на то, что такое вычислимая функция. Утверждение, которое хотелось бы сделать, но которое не поддерживается строгим доказательством, – это [8, с.266]

> **Тезис Тьюринга.** *Все вычислимые частичные функции вычисляются на машинах Тьюринга-Поста.*

4.3.4. Универсальная машина Тьюринга-Поста

Машина Тьюринга-Поста называется *универсальной*, если она может при определенных начальных входных данных вычислить любую функцию, которая вычислима на какой-либо машине Тьюринга-Поста.

Иначе говоря, с учетом тезиса Тьюринга можно сказать, что универсальная машина Тьюринга-Поста способна вычислить любую вычислимую функцию. Доказано, что универсальная машина Тьюринга-Поста существует.

4.4. Алан Тьюринг

Алан Матисон Тьюринг (23.6.1912-7.6.1954) – английский инженер и математик. Член Лондонского королевского общества (1951). Родился в Лондоне. Окончил Кембриджский университет (1935). Работал сначала там же. Математические работы Тьюринга в основном посвящены математической логике и вычислительным машинам. Кроме того, ему принадлежат отдельные результаты в области аппроксимации групп Ли конечными группами и в вычислении дзета-функции Римана. В 1936 году Тьюринг выяснил связь общерекурсивных функций с общими идеями «вычислимости» и «выводимости». Используя схему абстрактной машины, доказал ряд важных для кибернетики положений в области математической логики.

Во время второй мировой войны занимался электроникой, позднее возглавил работу по созданию вычислительных машин в Национальной физической лаборатории в Таддингтоне. Однако его проект «автоматического вычислительного устройства» (Automatic Computing Engine – ACE), законченный в 1946 году, был признан излишне самонадеянным и не получил одобрения. По сути, эта разработка содержала первое подробное описание компьютера в современном понимании этого слова [3].

Рис. 4.2: А.Тьюринг (1912-1954).

В Манчестере в 1951 году в результате развития проекта MADAM вводится в действие один из первых в мире полностью работоспособных компьютеров Ferranti Mark 1, и Тьюринг начинает первые вычислительные эксперименты по нелинейному моделированию формообразования у растений и раковин. Он пишет работу по вопросам морфогенеза – научному направлению, определяющему математические закономерности в развитии форм живых существ.

Конечно, расчет конфигурации цветка был не единственной и не главной задачей, выполнявшейся на «Марке 1». Спустя несколько лет он был использован при создании английской атомной бомбы. Сохранились свидетельства того, что Тьюринг проводил на нем и «несанкционированные» вычисления. Что еще считал «Марк» – известно только ему и Тьюрингу. Возможно, это были работы для правительственного Департамента кодов GCHQ (Government Code unit) – организации, продолжавшей работы по дешифровке, начатые во время войны в Блечли. С этой организацией Тьюринг не прекращал сотрудничество. На этот раз в центре внимания GCHQ были шифры советской резидентуры в Англии. Но не исключено, что вычислительные эксперименты касались ранних «детских» увлечений Тьюринга: в этот период он разрабатывает квантовую теорию спиноров – компонентов элементарных частиц – и работает над некоторыми вопросами теории относительности. Тогда же он проявляет серьезный интерес к методике психоанализа Юнга и записывает все свои сны. Эти записи не сохранились.

8 июня 1954 года Алан Матисон Тьюринг, кавалер ордена Бри-

[3] По статье Далидовича Г. Заметки об искусственном интеллекте: Яблоко Тьюринга. – http://www.russ.ru/netcult/20001220.html

танской империи, член Королевского ученого общества, математик, многие работы которого были поняты только в 70-х годах <..>, был найден мертвым в постели у себя дома в Вилмслоу. По полицейскому отчету, предположительно за день до этого он принял цианистый калий. Тьюрингу был 41 год. Рядом было найдено надкусанное яблоко, на которое был нанесен цианид. Он использовал этот реактив в опытах по электролитическому нанесению покрытия на ложки. Химические эксперименты были его любимым увлечением с десятилетнего возраста.

Одно их стихотворений Тьюринга:

Hyperboloids of wondrous Light
Rolling for aye through Space and Time
Harbour those waves which somehow Might
Play out God's holy pantomime.

Перевод 1.

Удивительный свет не прекращает вековых скитаний,
Пронзая пространства завесы.
Прими его, и он, быть может, откроет сценарий
Божественной пьесы.

Перевод 2.

Сверкающий света поток
Не остановит пространство и время.
Замысел своей пьесы откроет Бог
Тому, кто примет знания бремя (или поток сей в темя).

4.5. Эмиль Пост

Эмиль Леон Пост родился в Польше в 1897 г. В возрасте семи лет его родители эмигрировали в Нью-Йорк [4].

As a child growing up in Harlem, Post was especially interested in astronomy. Tragically, before age thirteen he lost his left arm in an accident. Post wrote to several observatories asking whether his handicap would exclude him from the profession of astronomy. While the response from Harvard College Observatory was encouraging ("there is no reason why you may not become eminent in astronomy"), the superintendent of the U.S. Naval Observatory wrote that "in my opinion the loss of your left arm would be a very serious handicap to your becoming a professional astronomer. In observational work

[4] По материалам сайта «Emil Leon Post Papers»:
http://www.amphilsoc.org/library/browser/p/post.htm

with instruments the use of both hands is necessary in all the work of this observatory."Discouraged, Post turned his intellect away from the heavens and toward mathematics.

Рис. 4.3: Э.Л. Пост (1897-1954).

After graduating from Townsend Harris High School, Post entered City College of New York. By the time he received a B.S. in mathematics in 1917, Post had already done much of the work for a paper on generalized differentiation that was eventually published in 1930. From 1917-1920 Post was a graduate student at Columbia University. His doctoral dissertation involved the mathematical study of systems of logic, specifically the application of the truth table method to the propositional calculus of Whitehead and Russell's Principia Mathematica. Post was able to show that the axioms of propositional calculus were both complete and consistent with respect to the truth table method. This dissertation was to help form the foundation of modern proof theory.

Post spent the 1920-1921 academic year at Princeton on a post-doctoral fellowship. It was during this period that he continued to analyze the Principia Mathematica and began to grapple with a revolutionary idea that would become famous in the 1930s: the fundamental incompleteness of any formal logic. Unfortunately for Post, his early formulations were fragmentary and as he struggled to work them out, Kurt Godel, who had no knowledge of Post's work, announced his landmark "incompleteness theorem"in 1931.

In 1936 Post contributed a paper to the first issue of the Journal of Symbolic Logic entitled "Finite Combinatory Processes – Formulation I."This paper had much in common with Alan Turing's work on a universal computing machine. While Turing's work described the mechanics of such a machine, Post focused on the instructions, or "software,"that would make the machine work. Post was able to prove that all computational processes could be reduced to a set of instructions that manipulated two symbols, "0"and "1."

Пост боролся с маниакальной депрессией в течение всей своей жизни. В 1954 году Пост умер от инфаркта, настигшего его после

электрошкоковой терапии в нью-йоркской больнице.

4.6. Эффективные алгоритмы

Алгоритм, трудоемкость которого (число шагов) ограничена полиномом от характерного размера задачи, называется *эффективным.*

Пример 4.6. (Жадный алгоритм). Рассмотрим конечное множество X, содержащее n элементов, некоторое семейство его подмножеств $\mathcal{X} \subset 2^X$ и (весовую) функцию $w : X \to [0, +\infty)$.

Пусть

$$w(A) = \max_{B \in A} w(B), \quad w(B) = \sum_{b \in B \subset X} w(b).$$

Алгоритм, который в указанном семействе $\mathcal{X}$ выбирает подмножество A с максимальным значением (весом) $w(A)$, называется *жадным.*

Жадный алгоритм является эффективным; количество его шагов составляет $O(n)$.

Пример 4.7. (Полный перебор). Дано конечное множество $X = \{a_1, ..., a_n\}$, содержащее n элементов, и предикат $P(x_1, ..., x_n)$. Этот предикат не является симметричным, т.е. $P(..., x, ..., y, ...) \neq P(..., y, ..., x, ...)$. Требуется найти набор элементов $(a_{i_1}, ..., a_{i_n})$ такой, что $P(a_{i_1}, ..., a_{i_n}) = \top$. Самое простое решение этой задачи состоит в том, что перебираются все возможные перестановки $(a_{i_1}, ..., a_{i_n})$ из n элементов с проверкой истинности предиката на них. Известно, что число перестановок равно $n!$. Следовательно, трудоемкость такого алгоритма полного перебора $O(n!)$. Поскольку $n!$ растет с ростом n быстрее любого полинома степени n и быстрее, чем 2^n, то данный алгоритм не является эффективным.

4.7. Алгоритмически неразрешимые проблемы

Рассмотрим *массовую задачу*, т.е. класс П однотипных, похожих конкретных задач. Если для решения массовой задачи найден алгоритм[5], решающий эту задачу, то о задаче говорят как об *алгоритмически разрешимой проблеме.*

[5]Этот алгоритм находит решение для любой задачи из П.

Всякая ли задача может решаться с помощью некоторого алгоритма? Иначе говоря, всякая ли проблема алгоритмически разрешима? Ответить на этот вопрос можно лишь в случае, когда оговаривается, что понимается под алгоритмом. Как мы знаем, алгоритм и связанное с ним понятие вычислимой функции – это интуитивные понятия. Поэтому для того, чтобы заявить, что какая-то проблема алгоритмически неразрешима, необходимо уточнить, о каком определении алгоритма при этом идет речь.

В силу тезиса Тьюринга можно считать, что под алгоритмом понимается машина Тьюринга-Поста, а на основании тезиса Чёрча следует говорить, что для алгоритмической разрешимости необходимо, чтобы решение задачи находилось с помощью частично рекурсивной функции. Отсюда вытекает, что если доказать, что решение задачи дается функцией, не являющейся частично рекурсивной, то такую задачу имеет смысл объявить алгоритмически неразрешимой.

Существуют ли алгоритмически неразрешимые проблемы (в указанном выше смысле)? Да, такие проблемы математикам известны. Таковыми является, например, проблема доказательства общезначимости формулы в исчислении предикатов (теорема Чёрча), проблема самоприменимости машин Тьюринга и проблема останова машины Тьюринга [34]. Большой список алгоритмически неразрешимых проблем дан в книге [39].

Глава 5

Сложность алгоритмов

5.1. Понятие о сложности алгоритмов

Сложность – это способ сравнения алгоритмов. Их сравнивают по количеству необходимых для выполнения алгоритма шагов (*временная сложность*) и по объему памяти, необходимой для работы алгоритма (*ёмкостная сложность*).

Для машины Тьюринга $\mathfrak{T}$, вычисляющей (словарную) функцию $f(x)$, временная сложность – это функция $t_{\mathfrak{T}}(x)$, равная числу шагов машины, совершенных при вычислении $f(x)$, если $f(x)$ определено (в противном случае $t_{\mathfrak{T}}(x)$ считается неопределенным).

Пусть $s_{\mathfrak{T}}(x)$ равно числу ячеек на ленте машины Тьюринга, задействованных при вычислении функции $f(x)$ (если $f(x)$ определено; в противном случае неопределенным считается и $s_{\mathfrak{T}}(x)$). Функция $s_{\mathfrak{T}}(x)$ называется емкостной (ленточной) сложностью.

Тогда [34, с.77]:

$$s_{\mathfrak{T}}(x) \le |x| + t_{\mathfrak{T}}(x),$$

$$t_{\mathfrak{T}}(x) \le |Q| s^2_{\mathfrak{T}}(x) |A|^{s_{\mathfrak{T}}(x)},$$

где $|x|$ –длина слова x, $|Q|, |A|$ – мощности внутренней и внешней памяти (алфавита).

Можно ввести функцию сложности (по худшему случаю)

$$t_{\mathfrak{T}}(n) = \max_{x,\ |x|=n} t_{\mathfrak{T}}(x),$$

которая характеризует временные затраты машины на вычисления слов длины n.

5.2. Классы задач P и NP

Сложность задачи оценивают, как правило, с точки зрения затраты времени, необходимого машине Тьюринга-Поста для того, чтобы вычислить функцию, посредством которой находится решение рассматриваемой задачи.

5.2.1. Класс задач P

Рассмотрим массовую задачу, под которой в действительности имеют в виду класс П однотипных задач, состоящий из бесконечного числа индивидуальных конкретных задач и описываемый совокупностью параметров. Каждой конкретной задаче $Z \in \Pi$ отвечает свой набор параметров. Относительно каждой задачи ставится вопрос, ответ на который имеет вид «ДА» или «НЕТ». Например, нас интересует, обладают ли частично рекурсивные функции свойством X.

Полагается, что решение задачи П сводится к вычислению на машине Тьюринга-Поста некоторой функции.

Предполагаем, что с классом П связана система кодирования α, которая каждой задаче Z ставит в соответствие слово $\alpha(Z)$ в некотором алфавите. Размер задачи Z – это длина слова $|\alpha(Z)|$. Пусть машина Тьюринга $\mathfrak{T}$ решает задачи класса П и

$$t_{\mathfrak{T}}(n) = \max_{Z,\ |\alpha(Z)|=n} t_{\mathfrak{T}}(\alpha(Z))$$

– соответствующая временная сложность (по худшему случаю).

Говорят, что машина Тьюринга $\mathfrak{T}$ решает задачу П за *полиномиальное время*, если

$$t_{\mathfrak{T}}(n) = O(p(n))$$

для некоторого полинома $p(n)$. В противном случае говорят, что задача решается за *экспоненциальное время.*[1]

[1] К экспоненциальным относят, например, оценки вида $O(n^{\log_2 n})$.

Класс задач П называется *полиномиально разрешимым*, если существует машина Тьюринга $\mathfrak{T}$, решающая задачи $z \in \Pi$ за полиномиальное время.

Совокупность (классов) задач, разрешимых за полиномиальное время, называем *классом задач* $\mathbf{P}$.

5.2.2. Класс задач NP

Массовая задача П принадлежит классу **NP**, если в случае ответа «ДА» для задачи $Z \in \Pi$ существует слово $c(Z)$ длины $O(p(|\alpha(Z)|))$ такое, что задача $(Z, c(Z))$ принадлежит классу $\mathbf{P}$. Слово $c(Z)$ называется *удостоверением*, или *догадкой*, задачи Z. Его наличие позволяет проверить принадлежность задачи Z классу $\mathbf{P}$.

К примеру, выполнимость формулы $\mathcal{A}(X_1, ..., X_n)$ проверяется, если кроме самой формулы дан конкретный набор переменных $X_1^0, ..., X_1^0$ – догадка, способствующая установлению выполнимости формулы $\mathcal{A}$.

5.2.3. Недетерминированная машина Тьюринга

Определим класс **NP** в терминах недетерминированной машины Тьюринга [34].

Недетерминированная машина Тьюринга $\mathfrak{nT}$ – это машина Тьюринга с *дополнительным* **угадывающим модулем** (УМ), который способен записывать на ленту слова из внешнего алфавита A, к которому добавлен знак раздела *.

Поскольку изучается массовая задача, то внутренний алфавит Q содержит буквы q_Y и q_N, соответствующие ответам «ДА» и «НЕТ».

Машина осуществляет работу следующим образом:

- **Стадия 1 (Угадывание).** На ленте записано слово $\alpha(Z)$ в алфавите A – код задачи Z, начиная с ячейки с номером 1 (и вправо). Левее, в ячейке 0, записан знак раздела *. УМ, просматривая ячейку за ячейкой, идя влево после ячейки 0, пишет произвольное слово $c(Z)$. Если УМ останавливается, то происходит переход ко второй стадии работы машины. При этом имеем состояние машины

$$c(Z) * q_1\alpha(Z). \tag{5.1}$$

- **Стадия 2 (Решение).** На этой стадии машина работает как обычная машина Тьюринга, стартуя с начального состояния (5.1). Если через конечное число шагов она придет к состоянию, содержащему q_Y, то говорят, что она *принимает* состояние (5.1). Недетерминированная машина Тьюринга *принимает слово* $\alpha(Z)$, если существует слово $c(Z)$ такое, что машиной принимается состояние (5.1).

Время работы недетерминированной машины Тьюринга $\mathfrak{n}\mathfrak{T}$– это число

$$t_{\mathfrak{n}\mathfrak{T}}(Z) = t_1(Z) + t_2(Z),$$

если принимается слово $\alpha(Z)$.

Здесь

$t_1(Z)$ – время работы на 1-й стадии, т.е. $t_1(Z) = |c(Z)|$;

$t_2(Z)$ – время работы на 2-й стадии, т.е. $t_2(Z) = t_{\mathfrak{T}}(c(Z) * \alpha(Z))$.

Пусть

$$t_{\mathfrak{n}\mathfrak{T}}(n) = \max_{Z, |\alpha(Z)|=n} t_{\mathfrak{n}\mathfrak{T}}(Z). \tag{5.2}$$

Недетерминированная машина Тьюринга *решает массовую задачу* П *за полиномиальное время*, если

$$t_{\mathfrak{n}\mathfrak{T}}(n) = O(p(n))$$

для некоторого полинома $p(n)$.

Класс **NP** – это множество задач, для которых существует недетерминированная машина Тьюринга, принимающая за полиномиальное время те и только те слова, которые соответствуют индивидуальным задачам с ответом «ДА».

Ясно, что

$$\mathbf{P} \subseteq \mathbf{NP}.$$

Теорема 5.1. *Если задача* $\Pi \in \mathbf{NP}$*, то существует полином* $p(n)$ *такой, что задача* П *может быть решена с помощью (детерминистского) алгоритма со сложностью* $O(2^{p(n)})$*,* n *– размер задачи* П.

Доказательство дано в [34].

5.3. О понятии сложности

5.3.1. Три типа сложности

Сложность задания (записи) объекта – это кратчайшая длина программы, наименьшее число команд, которое должно быть введено в идеализированную машину для того, чтобы она могла решить любую задачу некоторого класса задач, т.е. предъявить объект, считающийся ее решением [1, с.50]. Как правило, в этот класс включают однотипные задачи, поэтому такой класс задач называют *массовой задачей*.

Однако может оказаться, что достаточно **простая** программа, с помощью которой решается некоторая массовая задача, может привести к **большому** объему вычислений. Следовательно, полезно ввести понятие *сложность вычислений*. К примеру, под *сложностью алгоритмов теории чисел* понимается количество арифметических операций (сложений, вычитаний, умножений и делений без остатка), необходимых для выполнения всех действий, предписанных алгоритмом [48, с.91].

Таким образом, следует различать три типа понятия «сложность» [1, с.50]:

- *Сложность задания (описания) объекта.*
- *Сложность вычисления* (функционирования).
- *Конструктивно-энергетическая сложность.*

Последнее понятие сложности связано с намерением измерять сложность посредством энергетических затрат. Очевидно, здесь просматривается инженерно-физический подход, связанный с физической реализацией, построением, созданием объекта, сложность которого определяется.

5.3.2. Четыре категории чисел по Колмогорову

Обратим внимание на то, что в понятие сложности входит числовая характеристика (длина программы, число операций и т.д.). Изучая эту сторону понятия сложности, А.Н.Колмогоров пришел к представлению о наличии четырех категорий чисел: малых, средних, больших и сверхбольших [15]:

- Число n называется *малым*, если практически (для человека или машины) возможно перебрать все релейно-контактные схемы из n элементов, или равно, выписать для них все соответствующие им булевы функции (см. § 1.1.5)[2].
- Число n называется *средним*, если перебрать все схемы невозможно, но можно перебрать сами элементы или (что посложнее!) выработать систему обозначений (меток) для любой схемы из n элементов.
- Число n – *большое*, если невозможен перебор и такого числа элементов, но возможно установить систему обозначений для этих элементов.
- Если и этого сделать нельзя, то число n – *сверхбольшое*.

К малым числам С.Н.Колмогоров относит, например, 3, а к средним – 1000. Число видимых звезд на небе – это большое число: их нельзя перебрать, но все они перечислены в звездном каталоге с помощью выработанной системы обозначений (меток).

5.3.3. Тезис Колмогорова

Ясно, что вычислительная техника вносит коррективы в эту классификацию чисел по Колмогорову. Число 5, не малое для человека[3], является малым для ЭМВ. Но это не отменяет принципиального различия между числами в четырех категориях чисел. По мнению С.Н.Колмогорова, объем памяти живого существа и машины характеризуется средними числами, а многие проблемы, решение которых предполагает простой (полный) перебор, – большими.

> **Тезис Колмогорова.** *Проблемы, которые не могут быть решены без большого перебора, останутся за пределами возможностей машины на сколь угодно высокой ступени развития техники и культуры.*

Принципиальным является вопрос: существуют ли проблемы, которые ставятся и решаются без необходимости большого перебора? Является ли такой проблемой создание живых существ? Такие

[2] Уместно заметить, что число попарно различных булевых функций от n переменных равно 2^{2^n}.

[3] Число булевых функций от 5 переменных равно $2^{2^5} = 256^4$.

проблемы должны прежде всего интересовать кибернетиков, ибо они реально разрешимы. Из анализа А.Н.Колмогорова вытекает, что нет препятствий для создания полноценных живых существ, построенных полностью на дискретных (цифровых) механизмах переработки информации и управления. «При анализе явлений жизни существенна не диалектика бесконечного, а диалектика большого числа (чисто арифметическая комбинация большого числа элементов создаст и непрерывность и новые качества!)» [15].

«Эволюция никогда бы не произошла, если бы задача передачи определенных свойст первых, простейших свойств сред обитания не была бы легко обрабатываемой (т.е. вычислимой в течение разумного периода времени)» [7, с.199].

5.4. А.Н. Колмогоров

Рис. 5.1: А.Н.Колмогоров (1903-1987).

«Колмогоров Андрей Николаевич [р.12(25).4.1903, Тамбов], советский математик, академик АН СССР (1939), Герой Социалистического Труда (1963). Окончил Московский университет (1925), с 1931 профессор там же. Научную деятельность начал в области теории функций действительного переменного, где ему принадлежат фундаментальные работы по тригонометрическим рядам, теории меры, теории множеств, теории интеграла, теории приближения функций. В дальнейшем К. внес существенный вклад в разработку конструктивной логики, топологии (где им создана теория верхних гомологий), механики (теория турбулентности), теории дифференциальных уравнений, функционального анализа. Основополагающее значение имеют работы К. в области теории вероятностей, где он вместе с А.Я.Хинчиным начал (с 1925) применять методы теории функций действительного переменного. Это позволило ему решить ряд трудных проблем и построить широко известную систему аксиоматического обос-

нования теории вероятностей (1933), заложить основы теории марковских случайных процессов с непрерывным временем. Позднее К. развил (примыкая к исследованиям А.Я.Хинчина) теорию стационарных случайных процессов, процессов со стационарными приращениями, ветвящихся процессов. К. внес важный вклад в теорию информации. Ему принадлежат исследования по теории стрельбы, статистическим методам контроля массовой продукции, применениям математических методов в биологии, математической лингвистике <..> К. принимает деятельное участие в разработке вопросов математического образования в средней школе и в университетах. К. создал большую школу в области теории вероятностей и теории функций. Среди его учеников академики АН СССР А.И.Мальцев, М.Д.Миллионщиков, С.М.Никольский, Ю.В.Прохоров, член-корреспонденты АН СССР И.М.Гельфанд, А.С.Монин, академик АН УССР Б.В.Гнеденко, академик АН Узбекской ССР С.Х.Сираждинов, лауреаты Ленинской премии И.В.Арнольд[4], Ю.А.Розанов и др. Иностранный член Парижской АН, Лондонского королевского общества, ряда др. зарубежных академий (в Нидерландах, Польше, Румынии, США), научных учреждений и обществ. Международная премия Бальзана (1963), Государственная премия СССР (1941), Ленинская премия (1965). Награжден 6 орденами Ленина, орденом Трудового Красного Знамени и медалями»[5]

[4]В настоящее время И.В.Арнольд – академик РАН.

[5]Б.В.Гнеденко. Колмогоров Андрей Николаевич. Большая Советская Энциклопедия.

Глава 6

Алгоритмы реальности

Вычисление – это всегда вычисление функции, которое может быть выполнено на универсальной машине Тьюринга-Поста. До сих пор мы под вычислением понимали абстрактную математическую процедуру, осуществляемую посредством абстрактной машины. Можно ли спроецировать теорию, изложенную в предыдущих главах, на реальную жизнь? Точнее, нас интересует, можно ли построить такой *физический* компьютер, который способен, используя некоторый *алгоритм* и работая по соответствующей *программе*, давать в полном объеме человеку ощущение пребывания в некоторой *физической реальности*? В современной литературе в таком случае говорят о компьютерном порождении *виртуальной реальности*, констатируя тем самым ограниченные возможности (в том числе и будущих) компьютеров, а также как бы говоря, что создаваемая реальность – это реальность мнимая, ненастоящая, т.е. не физическая.

Здесь уместно отметить, что все наши человеческие ощущения связаны с виртуальной реальностью, которую создает наш мозг. А поскольку по мнению А.Н.Колмогорова (см. § 5.3.3), создание полноценных живых существ, построенных на машинных, дискретных (цифровых) механизмах переработки информации и управления возможно, то эти существа будут видеть окружающий мир таким, каким его сформирует их мозг-компьютер. Иначе говоря, эти существа имеют дело с виртуальной реальностью. Столь ли важ-

но, что мозг-компьютер формирует изображения[1] как отклик на раздражения внешней физической реальности (или «нервных» импульсов искусственного существа) или генерирует эти раздражения самостоятельно, подменяя внешнюю физическую реальность «внешней» виртуальной реальностью.

6.1. Генератор виртуальной реальности

Генератор виртуальной реальности[2] – это машина, способная дать пользователю ощущение какой-то реальной или вымышленной среды. Она имеет следующие составляющие:

- набор сенсоров, посредством которых машина узнает о действиях пользователя;
- набор генераторов изображения (например, стимуляторы нервных окончаний пользователя). Под изображениями понимается что-то, что рождает ощущения;
- управляющий компьютер.

Каждый построенный генератор виртуальной реальности способен порождать какое-то число физических или вымышленных сред.

Набор сред, ощущение нахождения пользователя в которых, может создать генератор, называется *репертуаром генератора виртуальной реальности.*

6.2. Принцип Тьюринга

Логика развития идей Тьюринга приводит к предположению, что должен существовать *универсальный генератор виртуальной реальности,* репертуар которого включает репертуары всех генераторов виртуальной реальности.

Поскольку речь идет не о математически абстрактном управляющем компьютере, производящем абстрактные математические вычисления, а о физической реальной машине, то возникает вопрос:

[1] *Изображение* – что-либо, рождающее ощущение [7, с.126].

[2] Станислав Лем называл эту машину *фантоматом* [20].

можно ли вычислить на реальном компьютере то, что придумано математиками и названо вычислением? По мнению Тьюринга, всё то, «что естественно сочли бы вычислимым», могло бы, по крайней мере в принципе, быть вычислимым и в природе [7, с.136].

Следовательно, можно сформулировать [7, с.138]

> **Принцип Тьюринга.** *Возможно построить генератор виртуальной реальности с репертуаром, включающим все среды, существование которых не противоречит законам физики.*

Виртуальная среда формируется в управляющем компьютере. Это означает: 1) наличие соответствующей программы, 2) наличие необходимых вычислительных ресурсов (быстродействие, объем памяти и др.).

То, что касается вычислительных ресурсов, то, как говорилось в § 5.3.3, порождение среды обитания человека и создание искусственной жизни не потребует *больших* по Колмогорову вычислений. Следовательно, будут задействованы ресурсы, вполне достижимые в ходе научно-технического прогресса. Определенные надежды в этом направлении связаны с теорией *квантовых вычислений* и *квантовыми компьютерами*, призванными осуществлять квантовые вычисления.

По этому поводу Дойч заявил следующее: «Эволюция никогда бы не произошла, если бы задача передачи определенных свойств первых, простейших свойств сред обитания не была бы легко обрабатываемой (т.е. вычислимой в течение разумного периода времени) при использовании в качестве компьютеров легко доступных молекул» [7, с.199].

Теперь несколько слов о программном обеспечении. Программы всегда дискретны, они содержат конечное число символов, которые при физической реализации задействуют лишь конечное число материальных объектов (импульсов, атомов и т.д.) и выполняются последовательно, шаг за шагом, т.е. также дискретно. В силу этого, если каждая программа соответствует одной физически возможной среде, входящей в репертуар генератора виртуальной реальности, то число (мощность) всевозможных программ, а значит, и сред не более, чем счетное (т.е. $\aleph_0$).

Но в мире мы наблюдаем непрерывное – непрерывные линии, фигуры и тому подобное. Можно ли непрерывное передать через

дискретное? Прежде всего, как пишет А.Н.Колмогоров [15], «скорее всего интуиция непрерывной линии в мозге осуществляется на базе дискретного механизма». И далее: «Несомненно, что переработка информации и процессы управления в живых организмах построены на сложном переплетении дискретных (цифровых) и непрерывных механизмов, с одной стороны, детерминированного и вероятностного принципов действия – с другой. Однако дискретные механизмы являются ведущими в процессах переработки информации и управления в живых организмах. Не существует состоятельных аргументов в пользу принципиальной ограниченности возможностей дискретных механизмов по сравнению с непрерывными».

Остается добавить, что, с точки зрения квантовой теории, все физические переменные квантуются. Следовательно, программы, физически реализуемые программы, дискретны в силу квантовой природы вещей.

6.3. Логически возможные среды Кантгоуту

Можно ли придумать среды, которые не противоречат законам логики, но которые не сможет передать ни один генератор виртуальной реальности? Другими словами, эти среды не может породить машина, построенная в соответствии с действующими законами физики.

Такие логически возможные среды существуют [7, с.132], их бесчисленное множество, и называются они средами Кантгоуту[3].

Для доказательства их существования используется диагональный метод Кантора (см. [4, с.17]), с помощью которого была установлена несчетность множества действительных чисел и доказана теорема Гёделя о неполноте формальных теорий.

Следовательно, логика способна породить больше миров, чем это доступно для видения человека. Миры *невиданные* наблюдает иное существо, живущее в мире с другими физическими законами, поскольку что толку в мире, у которого нет наблюдателя!

[3]От английской фразы can't go to (не могу пойти в), а также в честь Кантора (Cantor), Гёделя (Gödel) и Тьюринга (Turing).

Литература

[1] Бирюков Б.В., Гутчин И.Б. *Машина и творчество.* М.: Радио и связь, 1982.

[2] Ван Хао. *На пути к механической математике* // Кибернетический сборник. Вып.5. М.: ИЛ, 1962.

[3] Войшвилло Е.К., Дегтярев М.Г. *Логика.* М.: Владос-Пресс, 2001.

[4] Гуц А.К. *Кардинальные и трансфинитные числа.* Омск: ОмГУ, 1995.

[5] Св.Иоанн Дамаскин. *Точное изложение православной веры.* М.,1992.

[6] Джордж Ф. *Основы кибернетики.* М.: Радио и связь, 1984.

[7] Дойч Д. *Структура реальности.* Москва-Ижевск: РХД, 2001.

[8] Ершов Ю.Л., Палютин Е.А. *Математическая логика.* М.: Наука, 1979.

[9] Ивин А.А. *Логика времени* / В кн.: Неклассическая логика. М.: Наука, 1979. С.124-190.

[10] Ивин А.А. *Логика.*
– http://www.i-u.ru/biblio/arhiv/books/ivin_logik/default.asp

[11] Жоль К.К. *Логика в лицах и символах.* М.: Педагогика-Пресс, 1993.

[12] Колкер Ю. *Курт Гедель, или возможна ли диктатура в США.* – http://www.vestnik.com/issues/98/0203/win/kolker.htm

[13] Колмогоров А.Н. *О понятии алгоритма* // УМН. 1953. Т.8, вып.4. С.175-176.

[14] Колмогоров А.Н., Успенский В.А. *К определению алгоритма* // УМН. 1958. Т.13, вып.4. С.3-28.

[15] Колмогоров А.Н. *Автоматы и жизнь.* – В кн.: Кибернетика ожидаемая и кибернетика неожиданная. М.: Наука, 1968.
– http://mbur.narod.ru/misc/kolmogorov.html
– http://vivovoco.rsl.ru/VV/PAPERS/BIO/KOLMOGOR/KOL_REP.HTM♯b

[16] Колмогоров А.Н. *О принципе tertium non datur* // Математ. сб. 1925. Т.32, N.4. С.646-667.

[17] Костюк В.Н. *Элементы модальной логики.* Киев: «Наукова думка», 1978.

[18] Кофман А. *Введение в теорию нечетких множеств.* М.: Радио и связь, 1982.

[19] Лавров И.А., Максимова Л.Л. *Задачи по теории множеств, математической логике и теории алгоритмов.* М.: Наука, 1975.

[20] Лем С. *Сумма технологии.* М.: Мир, 1968.

[21] Линдон Р. *Заметки по логике.* М.: Мир, 1968.

[22] *Логика и компьютер. Моделирование рассуждений и проверка правильности программ.* М.: Наука, 1990.

[23] Маслов С.Ю. *Обратный метод установления выводимости в классическом исчислении предикатов* // ДАН СССР. Т.159, N.1.

[24] Маслов С.Ю. *Обратный метод установления выводимости для логических исчислений* //Труды математического института им. В.А.Стеклова АН СССР. 1968. Т.98. С.26-87.

[25] Маслов С.Ю. *Теория дедуктивных систем и ее применения.* М.: Советское радио, 1986.

[26] Мальцев А.И. *Алгоритмы и рекурсивные функции.* М.: Наука, 1986.

[27] Марков А.А. *Теория алгорифмов.* Тр. МИАН АН СССР. 1954. Т.42. 374 с.

[28] *Машины Тьюринга и рекурсивные функции.* М.: Мир, 1972.

[29] Мендельсон Э. *Введение в математическую логику.* М.: Наука, 1971.

[30] Михайлова Е.М. *Готлоб Фреге – философ, логик, математик.* – http://www.auditorium.ru/aud/v/index.php?a=vconf&c=getForm&r=thesisDesc&id_thesis=1217&PHPSESSID=69abfe0b0a317feb3dc30f51dadf793b

[31] Непейвода Н.Н. *Прикладная логика.* Новосибирск: НГУ, 2000.

[32] Новиков П.С. *Элементы математической логики.* М.:Наука, 1973.

[33] Новиков Ф.А. *Дискретная математика для программистов.* СПБ.: Питер, 2002.

[34] Носов В.А. *Основы теории алгоритмов и анализа их сложности.* М.: МГУ, 1992.

[35] Пономарев В.Ф. *Математическая логика. Часть 1. Логика высказываний. Логика предикатов.* Учебное пособие. Калининград: КГТУ, 2001.

[36] Пономарев В.Ф. *Математическая логика. Часть 2. Логика реляционная. Логика нечеткая.* Учебное пособие. Калининград: КГТУ, 2001.

[37] Редже Т. *Этюды о Вселенной.* М.:Мир,1985.
– http://gng.boom.ru/library/gedel.htm

[38] Робинсон Дж. *Логическое программирование – прошлое, настоящее и будущее* / В кн.: Логическое программирование.М.:Мир, 1988. С.7-26.

[39] Роджерс Х. *Теория рекурсивных функций и эффективная вычислимость.* М., 1972.

[40] *Современный словарь иностранных слов.* М.: Русский язык, 1992.

[41] Столл Р.Р. *Множества. Логика. Аксиоматические теории.* М.: Просвещение, 1968.

[42] Тьюринг А. *Могут ли машины мыслить?* М.: Физматгиз, 1960.

[43] Успенский В.А. *Лекции о вычислимых функциях.* М.: ФМ, 1960.

[44] Фейс Р. *Модальная логика.* М.: Наука, 1974.

[45] Фон Вригт Г.Н. *Логико-философские исследования. Избранные труды.* М.: Прогресс, 1986.

[46] Черч А. *Введение в математическую логику.* М.: ИЛ, 1960.

[47] Шанин Н.А., Давыдов Г.В. и др. *Алгорифм машинного поиска логического вывода в исчислении высказываний.* М.: Наука, 1970.

[48] Ященко В.В. *Введение в криптографию.* СПб.: ПИТЕР, 2001.

[49] Burstall R. *Program proving as hand simulation with little induction* // Inform. Proc. 1974. V.74. P.263-312.

[50] Pnueli A. *The temporal logic of programs* // Proc. 18th IEEE Symp. on the foundations of computer science. San-Francisco, 1977. P.46-57.

[51] Prior A.N.*Diodorian modalities* // The Philosophical Quarterly. 1955. V.5. N.20.

[52] Van Heijenoort J. *From Frege to Gëdel. A Source Book in Mathematical Logic, 1879-1931.* Cambridge, Mass.: Harvard Univ. Press, 1967.

А.К. Гуц

Математическая логика и теория алгоритмов

Учебное пособие

Редактор Е.В. Брусницына

Подготовлена к печати
ООО «Издательство Наследие. Диалог-Сибирь»
Лицензия ЛР є 071680 от 04.06.98.

Подписано в печать 12.04.03.
Формат 60×84 1/16. Печ.л. 6,9. Уч.-изд.л. 6,7.
Тираж 200 экз.

Полиграфический центр КАН
644050, Омск-50, пр. Мира, 32, к.11,
тел. (3812) 65-47-31.
Лицензия ПЛД N 58-47 от 21.04.97 г.