ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Механико-математический факультет

Н. Т. Когабаев

ЛЕКЦИИ ПО ТЕОРИИ АЛГОРИТМОВ

Учебное пособие

Новосибирск

2009

УДК 510.5(075)
ББК В12я73-1
К 570

Когабаев Н. Т. Лекции по теории алгоритмов: Учеб. пособие / Новосиб. гос. ун-т. Новосибирск, 2009. 107 с.

ISBN 978-5-94356-799-5

В настоящем учебном пособии изложены математические основы теории алгоритмов. Пособие отражает содержание лекций основного курса «Теория алгоритмов», прочитанных автором для студентов 1-го курса механико-математического факультета НГУ и охватывает материал из нескольких областей математики, так или иначе связанных с понятием алгоритма: теория автоматов и регулярных языков, машины Тьюринга и Шёнфилда, нормальные алгорифмы Маркова, классическая теория вычислимости, теория нумераций, теория сложности вычислений.

Предназначено для студентов 1-го курса механико-математического факультета НГУ, изучающих курс «Теория алгоритмов», а также для всех желающих познакомиться с основами упомянутых в пособии математических теорий.

Рецензент
канд. физ.-мат. наук В. Н. Власов

Издание подготовлено в рамках выполнения инновационно-образовательной программы *«Инновационные образовательные программы и технологии, реализуемые на принципах партнерства классического университета, науки, бизнеса и государства»* национального проекта «Образование».

ISBN 978-5-94356-799-5

Оглавление

Глава I

Предварительные сведения

§ 1. Некоторые аксиомы теории множеств

Все объекты, изучаемые в данном курсе, являются множествами. Множествами являются символы, алфавиты и языки. Множествами являются числа, кортежи и последовательности. Множествами являются предикаты, функции и операторы. Даже автоматы, машины и алгоритмы, изучению которых посвящен настоящий курс, являются множествами.

Для работы с множествами и формализации определенных понятий нам потребуются некоторые аксиомы *теории множеств Цермело-Френкеля* ZF. Теория ZF является формальной (синтаксической) теорией в языке с одним символом двухместного предиката $\in$ и символом равенства $\approx$. Однако мы будем формулировать понятия и аксиомы данной теории на естественном (общематематическом) языке. Подобная «нестрогость» не должна пугать читателя, поскольку при желании все формулировки можно «перевести» на формальный язык ZF, но в рамках данного курса в этом нет необходимости. Для более глубокого и подробного ознакомления с системой ZF можно порекомендовать книги [2], [3].

Понятия *множества* и *отношения принадлежности* $\in$ являются неопределяемыми через другие математические объекты. Неформально множество — это некоторая совокупность объектов A, отношение $x \in A$ означает, что объект x является элементом совокупности A. Мы также будем использовать термины *семейство* и *совокупность* для описания некоторых множеств.

Определение. Говорят, что множество A является *подмножеством* множества B, и пишут $A \subseteq B$, если $\forall x(x \in A \to x \in B)$. Другими словами, $A \subseteq B$, если каждый элемент множества A является элементом множества B.

Равенство двух множеств A и B определяется следующей аксиомой.

Аксиома экстенсиональности: *$A = B$ тогда и только тогда, когда $\forall x(x \in A \leftrightarrow x \in B)$. Таким образом, множества A и B равны, если $A \subseteq B$ и $B \subseteq A$.*

Следующая естественная аксиома постулирует существование наименьшего по включению множества.

Аксиома пустого множества: *существует пустое множество $\varnothing$, т. е. множество, не содержащее ни одного элемента.*

Следующие четыре аксиомы позволяют из одних множеств строить другие, более сложные по своей структуре.

Аксиома пары: *если A и B — множества, то существует неупорядоченная пара $\{A, B\}$, составленная из этих множеств.*

Аксиома суммы: *если A — множество, то существует множество $\cup A = \{x \mid x \in y$ для некоторого $y \in A\}$, которое называется объединением множества A.*

Аксиома степени: *если A — множество, то существует множество $P(A) = \{B \mid B \subseteq A\}$ всех подмножеств множества A.*

Аксиома подстановки: *если A — множество, а $\Phi(x, y)$ — некоторое условие на множества x, y такое, что для любого x существует не более одного y, удовлетворяющего условию $\Phi(x, y)$, то существует множество $\{y \mid \Phi(x, y)$ для некоторого $x \in A\}$.*

Например, из аксиомы пустого множества, аксиомы пары и аксиомы суммы следует, что существуют множества $\varnothing$, $\{\varnothing\}$, $\{\varnothing, \{\varnothing\}\}$, $\{\varnothing, \{\varnothing\}, \{\varnothing, \{\varnothing\}\}\}$ и т. д. (каждое следующее состоит из всех предыдущих), эти множества мы будем называть *натуральными числами* и обозначать их соответственно через $0, 1, 2, 3$ и т. д.

Заметим, что используя только те пять аксиом «существования», которые сформулированы выше, можно получить лишь конечные множества. В частности, из этих пяти аксиом невозможно вывести, что «совокупность» всех натуральных чисел образует множество. Для разрешения этого вопроса вводится следующая

Аксиома бесконечности: *существует множество $\omega = \{0, 1, 2, 3, \ldots\}$ всех натуральных чисел.*

Теперь, располагая каноническим бесконечным множеством ω, можно строить другие бесконечные множества. В следующем определении вводятся стандартные теоретико-множественные операции объединения, пересечения, разности и дополнения (до некоторого множества).

Определение. Если A и B — множества, то их *объединением* называется множество $A \cup B = \{x \mid x \in A$ или $x \in B\}$, *пересечением* — множество $A \cap B = \{x \mid x \in A$ и $x \in B\}$, *разностью* — множество $A \setminus B = \{x \mid x \in A$ и $x \notin B\}$.

Если рассматриваемые множества являются подмножествами некоторого фиксированного множества U, то можно говорить о *дополнении* $\overline{A} = U \setminus A$ множества A (до множества U).

Объединением семейства множеств A называется множество $\cup A = \{x \mid \exists B \in A(x \in B)\}$. *Пересечением* непустого семейства множеств A называется множество $\cap A = \{x \mid \forall B \in A(x \in B)\}$.

Из перечисленных выше аксиом следует, что применяя эти операции к множествам, мы снова получаем множества. Например, если A, B — множества, то в силу аксиомы пары существует неупорядоченная пара $\{A, B\}$, а в силу аксиомы суммы существует объединение $A \cup B = \cup\{A, B\}$. Затем, используя аксиому подстановки, заключаем, что существует пересечение $A \cap B = \{y \mid (y \in A$ и $y \in B$ и $y = x)$ для некоторого $x \in A \cup B\}$.

Аксиома пары постулирует существование множества $\{a, b\}$. Однако порядок расположения элементов в паре формально никак не задается, поскольку $\{a, b\} = \{b, a\}$. Более того, если $a = b$, то пара $\{a, b\}$ превращается в одноэлементное множество $\{a\}$. Чтобы все-таки упорядочить элементы пары, вводится следующее

Определение. *Упорядоченной парой* элементов a и b называется множество $\langle a, b\rangle = \{\{a\}, \{a, b\}\}$. В упорядоченной паре мы задаем строгий порядок расположения элементов: a — первый, b — второй. Следует различать $\langle a, b\rangle \neq \{a, b\}$!

Предложение 1. *Для любых элементов a, b, c, d имеет место: $\langle a, b\rangle = \langle c, d\rangle$ тогда и только тогда, когда $a = c$ и $b = d$.*

Доказательство. Предлагается читателю в качестве упражнения. □

Определение. Пусть $n \in \omega, n \geqslant 1$. *Упорядоченная n-ка (кортеж длины n)* определяется по индукции: $\langle a_1\rangle = a_1$, $\langle a_1, \ldots, a_{n-1}, a_n\rangle = \langle\langle a_1, \ldots, a_{n-1}\rangle, a_n\rangle$.

Пустое множество $\varnothing$ по определению называем *кортежем длины* 0.

Следствие 2. *$\langle a_1, \ldots, a_n\rangle = \langle b_1, \ldots, b_n\rangle$ тогда и только тогда, когда имеет место $a_1 = b_1, \ldots, a_n = b_n$.*

Доказательство. Следует из предыдущего предложения по индукции. □

Определение. *Декартовым произведением* множеств $A_1, \ldots, A_n$ называется множество

$$A_1 \times \ldots \times A_n = \{\langle a_1, \ldots, a_n\rangle \mid a_1 \in A_1, \ldots, a_n \in A_n\}.$$

n-ой декартовой степенью множества A называется множество $A^n = \underbrace{A \times \ldots \times A}_{n}$.

При $n = 0$ по определению полагаем $A^0 = \{\varnothing\}$.

Определение. Любое подмножество $R \subseteq A_1 \times \ldots \times A_n$ называется *отношением (предикатом)* на множествах $A_1, \ldots, A_n$. Если $\langle x_1, \ldots, x_n\rangle \in R$, то говорят, что *предикат R истиннен на элементах $x_1, \ldots, x_n$*, и пишут $R(x_1, \ldots, x_n)$, иначе говорят, что *предикат R ложен на элементах $x_1, \ldots, x_n$*, и пишут $\neg R(x_1, \ldots, x_n)$.

Любое подмножество $R \subseteq A^n$ называется *n-местным отношением (предикатом)* на множестве A.

Определение. *Композицией* отношений $R_1 \subseteq A \times B$ и $R_2 \subseteq B \times C$ называется отношение $R_1 \circ R_2 = \{\langle a, c\rangle \mid \exists b(\langle a, b\rangle \in R_1 \text{ и } \langle b, c\rangle \in R_2)\}$.

Определение. *Обратным отношением* к отношению $R \subseteq A \times B$ называется отношение $R^{-1} = \{\langle b, a\rangle \mid \langle a, b\rangle \in R\}$.

Определение. Отношение $f \subseteq A \times B$ называется *функцией (отображением)*, если для любого $a \in A$ существует не более одного $b \in B$ такого, что $\langle a, b\rangle \in f$.

Для данного $a \in A$, если существует $b \in B$ со свойством $\langle a, b\rangle \in f$, то говорят, что *значение $f(a)$ определено и равно b*, и пишут $f(a)\downarrow$, в противном случае говорят, что *значение $f(a)$ не определено*, и пишут $f(a)\uparrow$.

Областью определения f называется множество $\mathrm{dom}(f) = \{a \in A \mid f(a)\downarrow\}$.

Областью значений f называется множество $\mathrm{range}(f) = \{f(a) \mid a \in A, f(a)\downarrow\}$.

Определение. Запись $f : A \to B$ означает, что для некоторых множеств X, Y отношение $f \subseteq X \times Y$ является функцией такой, что $\mathrm{dom}(f) = A$ и $\mathrm{range}(f) \subseteq B$. При этом говорят, что f является *функцией из A в B*.

Определение. Говорят, что функция $f : A \to B$ является *инъективной (разнозначной)*, и пишут $f : A \xrightarrow{1-1} B$, если для любого $b \in B$ существует не более одного $a \in A$ такого, что $f(a) = b$.

Определение. Говорят, что функция $f : A \to B$ является *сюръективной (отображением на)*, и пишут $f : A \xrightarrow{\text{на}} B$, если $\mathrm{range}(f) = B$.

Определение. Говорят, что функция $f : A \to B$ является *биективной (взаимно-однозначной)*, и пишут $f : A \xrightarrow[\text{на}]{1-1} B$, если f одновременно инъективна и сюръективна.

Определение. Если $f \subseteq A \times B$, $g \subseteq B \times C$ — функции, то их *композиция* $f \circ g \subseteq A \times C$ определяется как композиция отношений. Легко видеть, что $f \circ g$ тоже является функцией.

Определение. Если $f : A \xrightarrow[\text{на}]{1-1} B$ — биективная функция, то *обратная функция* f^{-1} определяется как обратное отношение. Легко видеть, что f^{-1} является биекцией вида $f^{-1} : B \xrightarrow[\text{на}]{1-1} A$.

Определение. Функция вида $f : X \to A$, где $X \subseteq A^n$ называется *n-местной частичной функцией на A*.

Замечание. Заметим, что существуют только два вида 0-местных частичных функций — это либо нигде не определенная функция $f = \varnothing$, либо функция вида $f = \{\langle \varnothing, a \rangle\}$ для некоторого $a \in A$. Во втором случае мы будем называть функцию константой и отождествлять ее с элементом a, т. е. $f = a$.

§ 2. Алфавиты и формальные языки

В большинстве случаев данные, с которыми работают алгоритмы, представляются в виде слов некоторого языка. Алгоритмы подобного вида можно назвать *словарными*. Например, все алгоритмы из следующей главы безусловно являются словарными. Более того, описание самих алгоритмов, как правило, осуществляется с помощью специального текста, который обычно называют *программой*.

Понятия слова, языка и текста имеют очень широкий спектр значений, выходящий за рамки математической науки. Мы будем работать только с *формальными языками*, т. е. с языками, которые поддаются формальному описанию в рамках теории множеств и которые можно изучать, используя строгие математические методы.

Определение. *Алфавитом* будем называть любое непустое конечное множество $\mathcal{A} = \{a_0, \ldots, a_n\}$. Элементы алфавита принято называть *буквами*, или *символами*.

Определение. *Словом длины n* в алфавите $\mathcal{A}$ мы будем называть любой кортеж $\langle a_1, a_2, \ldots, a_n \rangle$ длины n, где $a_1, \ldots, a_n$ — буквы алфавита $\mathcal{A}$. Слова обычно записывают в виде $a_1 a_2 \ldots a_n$. *Пустое слово* $\varnothing$ не содержит ни одной буквы, имеет длину 0 и обозначается через Λ.

Замечание. В дальнейшем запись слова в виде $a_1 a_2 \ldots a_n$ подразумевает, в частности, что при $n = 0$ это слово пусто. Это же соглашение распространяется на конечные кортежи и конечные множества, т. е. $\langle a_1, \ldots, a_n \rangle = \{a_1, \ldots, a_n\} = \varnothing$ при $n = 0$.

Определение. Множество всех слов в алфавите $\mathcal{A}$, включая пустое слово, обозначается через $\mathcal{A}^*$. Множество всех непустых слов в алфавите $\mathcal{A}$ обозначается через $\mathcal{A}^+$, т. е. $\mathcal{A}^+ = \mathcal{A}^* \setminus \{\Lambda\}$.

Определение. Длина слова $w \in \mathcal{A}^*$ обозначается через $|w|$.

Определение. Слово u называется *подсловом* слова v, если существуют слова w_1 и w_2 такие, что $v = w_1uw_2$, при этом *вхождением* подслова u в слово v назовем упорядоченную тройку $\langle w_1, u, w_2\rangle$. Отметим, что одно и то же слово u может иметь несколько различных вхождений в слово v.

Определение. Слово u называется *префиксом* слова v, если $v = uw$ для некоторого слова w. Слово u называется *суффиксом* слова v, если $v = wu$ для некоторого w.

Определение. Введем следующие операции над словами в алфавите $\mathcal{A}$:
(а) *Конкатенацией слов* u и v называется слово uv, полученное приписыванием к слову u слова v справа.
(б) *Степень слова* u определяется индукцией по $n \in \omega$ следующим образом: $u^0 = \Lambda$, $u^{n+1} = u^n u$.
(в) *Обращением слова* $u = a_1a_2\ldots a_n$, где $a_1, a_2, \ldots, a_n$ — буквы алфавита $\mathcal{A}$, называется слово $u^R = a_n\ldots a_2a_1$.

Пример. Конкатенацией слов `kein` и `mehrheit` является слово `keinmehrheit`. Если теперь взять конкатенацию тех же слов, но в другом порядке, то получится слово `mehrheitkein`. Слово `ei` является подсловом слова `keinmehrheit`, причем оно имеет два вхождения: первое вхождение начинается со 2-го символа слова, а второе — с 10-го символа. Обращением слова `mehrheit` будет слово `tiehrhem`. Степенями слова `kein` являются слова Λ, `kein`, `keinkein`, `keinkeinkein` и т. д. При этом каждое слово из данной последовательности является префиксом всех последующих слов.

Определение. Любое подмножество $L \subseteq \mathcal{A}^*$ называется *формальным языком над алфавитом* $\mathcal{A}$.

Определение. Введем следующие операции над языками в алфавите $\mathcal{A}$:
(а) *Объединение* $L_1 \cup L_2$ и *пересечение* $L_1 \cap L_2$ языков L_1 и L_2 определяются как обычные теоретико-множественные объединение и пересечение.
(б) *Дополнением* языка L называется язык $\overline{L} = \mathcal{A}^* \setminus L$.
(в) *Конкатенацией языков* L_1 и L_2 называется язык $L_1L_2 = \{uv \mid u \in L_1, v \in L_2\}$.
(г) *Степень языка* L определяется индукцией по $n \in \omega$ следующим образом: $L^0 = \{\Lambda\}$, $L^{n+1} = L^nL$.
(д) *Звездочкой Клини* от языка L называется язык $L^* = \bigcup\limits_{n\in\omega} L^n$. Другими словами,

$$L^* = \{w \mid w = w_1w_2\ldots w_n \text{ для некоторых } n \in \omega \text{ и } w_1, w_2, \ldots, w_n \in L\}.$$

В частности, всегда имеет место $\Lambda \in L^*$ (при $n = 0$).

Замечание. Заметим, что определение звездочки Клини согласуется с введеным выше обозначением $\mathcal{A}^*$. Звездочка Клини от алфавита — это и есть множество всех слов в данном алфавите.

Пример. С помощью введенных операций можно определять формально те языки, которые изначально имеют неформальное описание.

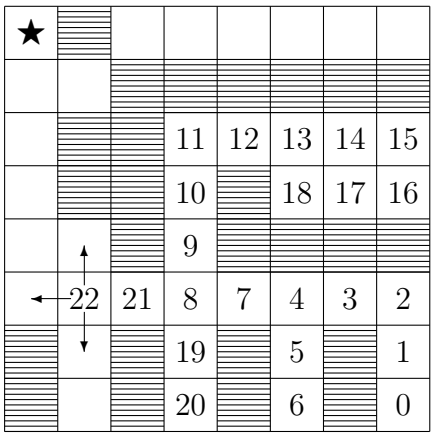
Например, язык всех слов в алфавите $\mathcal{A} = \{a, b\}$, имеющих хотя бы одно вхождение буквы a, совпадает с языком $\{b\}^*\{a\}\{a, b\}^*$. Действительно, в любом таком слове можно выделить первое вхождение буквы a, т. е. представить в виде b^naw, где $n \in \omega$, w — некоторое слово в алфавите $\mathcal{A}$. С другой стороны, любое слово вида b^naw лежит в языке $\{b\}^*\{a\}\{a, b\}^*$.

Другой пример — это язык всех слов четной длины в том же алфавите, который можно представить как $\{aa, ab, ba, bb\}^*$. Действительно, слово w имеет четную длину тогда и только тогда, когда его можно представить в виде $w = w_1 \dots w_n$ для некоторого $n \in \omega$ и некоторых слов w_i, каждое из которых имеет длину 2, т. е. каждое $w_i \in \{aa, ab, ba, bb\}$.

Отсюда следует, что язык всех слов нечетной длины можно получить как дополнение предыдущего языка, т. е. $\{a, b\}^* \setminus \{aa, ab, ba, bb\}^*$, а язык всех слов четной длины, содержащих букву a, можно получить, используя операцию пересечения $(\{b\}^*\{a\}\{a, b\}^*) \cap \{aa, ab, ba, bb\}^*$.

Язык всех слов четной длины, содержащих букву a, можно получить и другим способом, а именно как язык $\{bb\}^*\{aa, ab, ba\}\{aa, ab, ba, bb\}^*$.

§ 3. Интуитивные свойства алгоритмов



Понятие алгоритма известно в математике с древних времен. Однако до определенного времени алгоритмы возникали лишь при описании алгоритмических решений вполне конкретных математичеких задач. При этом автор решения в явном виде предъявлял описание своего алгоритма и доказывал, что его алгоритм приводит к верному результату. Этого было достаточно, чтобы по достоинству оценить новый математический результат. Но позднее, ближе к эпохе современности, в различных областях математики (чаще всего в алгебре) стали возникать задачи, относительно которых предполагалось, что они не имеют алгоритмического решения. Чтобы формально доказывать такие гипотезы, естественно, потребовалось более общее, формальное описание понятия алгоритма.

Прежде чем вводить формальные определения различных алгоритмических систем, мы попытаемся выявить с интуитивной точки зрения те характерные черты, которыми обладают алгоритмы. В дальнейшем мы убедимся в том, что все существующие подходы к формализации понятия алгоритма удовлетворяют этим свойствам. Это наблюдение будет служить подтверждением того, что данные подходы являются естественными и адекватными.

Чтобы наши интуитивные рассуждения были более наглядными, рассмотрим следующую несложную задачу, имеющую алгоритмическое решение.

Пример. На плоскости задан прямоугольник, разбитый на одинаковые клетки со стороной единичной длины. Некоторые клетки заштрихованы — это *стены*, остальные клетки — это *коридоры*. Таким образом, у нас задан *лабиринт*, вход которого находится в правой нижней клетке, а выход — в левой верхней (выход помечен символом ★, см. рисунок). В начальный момент времени на входной клетке находится *робот Бендер*, который умеет отличать стены от коридоров, умеет делать ходы на одну соседнюю незаштрихованную клетку сверху, справа, снизу или слева, умеет принимать простейшие логические решения, и способен распознать символ ★. Заранее известно, что в таком лабиринте существует как минимум один путь, ведущий

по коридорам из входа в выход. Необходимо *запрограммировать* Бендера так, чтобы он, руководствуясь данной программой несложных операций, смог найти выход из лабиринта.

Для этого мы зарезервируем достаточно большое количество *меток*, занумерованных натуральными числами. Этими метками Бендер будет помечать свои ходы. Искомая программа состоит из следующих шести пунктов:

0) Помечаем правую нижнюю клетку меткой 0. Переходим к пункту (1).

1) Проверяем, верно ли, что клетка, на которой стоит Бендер, является выходом. Если это верно, то переходим к пункту (5). Если нет, переходим к пункту (2).

2) Проверяем, существует ли хотя бы одна непомеченная соседняя (сверху, справа, снизу или слева) клетка. Если существует, то переходим к пункту (3). Если нет, то переходим к пункту (4).

3) Выбираем каким-нибудь эффективным способом одну соседнюю непомеченную клетку. Например, можно выбирать первую непомеченную клетку при обходе всех четырех соседних клеток по часовой стрелке, начиная с верхней. Помечаем выбранную клетку наименьшей неиспользованной меткой. Затем передвигаем Бендера на эту клетку и переходим к пункту (1).

4) Возвращаемся на один ход назад, т. е. передвигаем Бендера обратно на ту клетку, с которой он в последний раз перешел на текущую клетку. Переходим к пункту (2).

5) Работа алгоритма останавливается. Выход из лабиринта найден. Таким образом, задача решена.

Данный алгоритм всегда приводит к нахождению выхода из лабиринта. Действительно, если бы алгоритм не приводил к успеху, то это означало бы, что Бендер, проделав определенное число ходов и расставив метки от 0 до некоторого n, вернулся бы на исходную позицию, т. е. в правый нижний угол. Но тогда искомый путь, который тоже начинается в правом нижнем углу, полностью содержался бы в множестве помеченных клеток. В частности, левая верхняя клетка тоже была бы помечена Бендером, а значит, он побывал бы на ней в процессе своего блуждания по лабиринту, но вернулся назад. Последнее невозможно в силу пункта (5) из описания алгоритма.

Анализируя свойства алгоритма из данного примера, выделим теперь следующие

Основные черты алгоритмичеких процессов:

а) *Конечность описания* — любой алгоритм задается как набор инструкций конечных размеров, т. е. программа имеет конечную длину. В нашем примере описание алгоритма состоит из 6 пунктов, каждый из которых записан с помощью конечного числа слов.

б) *Дискретность* — алгоритм исполняется по шагам, происходящим в дискретном времени. Шаги четко отделены друг от друга. В алгоритмах нельзя использовать аналоговые устройства и непрерывные методы. В нашем примере один шаг работы алгоритма — это выполнение одного из пунктов (0)–(5), эти шаги можно занумеровать натуральными числами. В итоге Бендер проделывает

всегда лишь конечное число шагов для того, чтобы найти выход (хотя в общем случае допускается бесконечное количество шагов, т. е. алгоритмы могут не останавливаться).

в) *Направленность* — у алгоритма есть входные и выходные данные. В алгоритме четко указывается, когда он останавливается, и что выдается на выходе после остановки. В нашем примере входные данные — это схема лабиринта, выходные данные — путь по коридорам, ведущий к выходу.

г) *Массовость* — алгоритм применим к некоторому достаточно большому классу однотипных задач, т. е. входные данные выбираются из некоторого, как правило, бесконечного множества. В нашем примере один и тот же алгоритм годится для поиска выхода из любого лабиринта прямоугольной формы, а не только для того лабиринта 8×8, который изображен на рисунке.

д) *Детерминированность (или конечная недетерминированность)* — вычисления продвигаются вперед детерминированно, т. е. вычислитель однозначно представляет, какие инструкции необходимо выполнить в текущий момент. Нельзя использовать случайные числа или методы. Конечная недетерминированность означает, что иногда в процессе работы алгоритма возникает несколько вариантов для дальнейшего хода вычислений, но таких вариантов лишь конечное число. В нашем примере действия Бендера детерминированны, но этот алгоритм можно переделать и в недетерминированный, если предоставить Бендеру свободу самостоятельного выбора клетки в пункте (3). При этом, поскольку этот выбор конечен (соседних клеток всего четыре), можно представить дальнейшие действия Бендера в виде распределенных (параллельных) вычислений: в момент выбора Бендер создает несколько своих клонов, и каждый клон идет по своему пути, руководствуясь теми же пунктами (0)–(5). Если хотя бы один из клонов находит выход, то задача считается решенной.

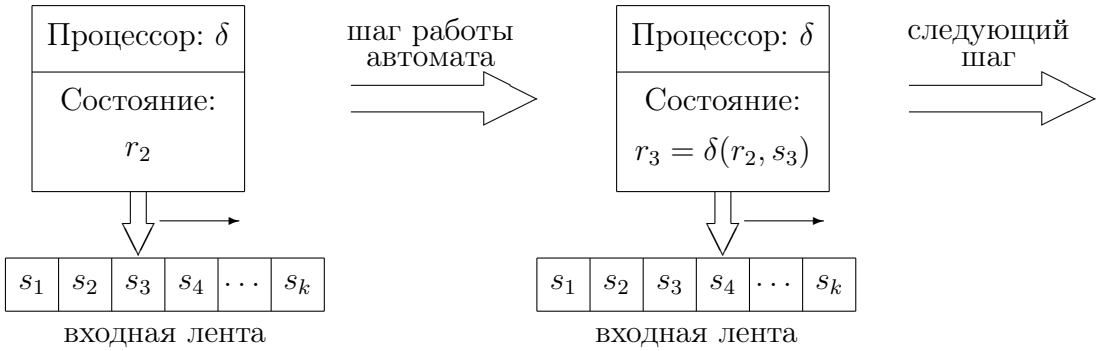
Глава II

Конечные автоматы и формальные грамматики

§ 4. Детерминированные конечные автоматы

Конечные автоматы относятся к классу словарных алгоритмов. С помощью конечных автоматов можно решать задачи из достаточно широкого семейства алгоритмических проблем. Например, технология проектирования микросхем основана на результатах теории автоматов. Другой пример — это компиляторы, перерабатывающие текст программы, написанной на языке программирования высокого уровня, в программу на машинном языке. Работа любого такого компилятора состоит из двух стадий. Первая стадия — лексический анализ текста, который реализуется с помощью определенного конечного автомата. Вторая стадия — синтаксический анализ, который осуществляется с помощью автомата с магазинной памятью. Однако, как будет видно позднее, не любая алгоритмическая задача поддается решению с помощью конечного автомата.

Дадим сначала неформальное определение конечных автоматов и описание их работы.



Входными данными любого конечного автомата являются слова в некотором заранее фиксированном алфавите, выходными данными являются две логические константы `true` и `false`. В каждый момент времени автомат находится в определенном *внутреннем состоянии*. Число состояний автомата конечно. В начальный момент времени автомат находится в *начальном состоянии*. Кроме этого, некоторые состояния автомата считаются *выделенными*. Входное слово записывается на *входной ленте*, разбитой на ячейки: в каждой ячейке содержится одна буква слова. Автомат связан с лентой посредством специальной *управляющей головки*, которая последовательно считывает символы из ячеек, двигаясь слева направо. Очередной считанный

символ отправляется в *процессор*, который по текущему состоянию и по данному считанному символу определяет новое состояние, в которое переводится автомат. Функция, вычисляющая это новое состояние называется *функцией перехода*. Функция перехода расположена внутри процессора и не изменяется в ходе вычислений. Считав все символы слова, автомат останавливается в определенном состоянии: если это состояние оказывается выделенным, то на выход подается константа `true` — автомат распознал слово, в противном случае подается константа `false` — автомат не распознал слово.

Уже из неформального определения видно, что конечные автоматы обладают такими интуитивными свойствами, как конечность описания, дискретность, направленность и массовость. Детерминированность также присутствует в данном описании: по текущему состоянию и считанному символу функция перехода однозначно определяет новое состояние. Отличительной чертой конечных автоматов является отсутствие памяти вне процессора, т. е. вся память автомата занята программой. Эта особенность позволяет пролить свет на причины неспособности конечных автоматов решить любую алгоритмически разрешимую задачу (для алгоритмов определенного вида необходима дополнительная память).

Теперь перейдем к формальным определениям в терминах теории множеств.

Определение. *Детерминированным конечным автоматом* (сокращенно д.к.а.) называется упорядоченная пятерка $\mathfrak{A} = \langle Q, \mathcal{A}, \delta, q_0, F\rangle$, состоящая из следующих объектов:

а) $Q = \{q_0, \ldots, q_m\}$ — конечный алфавит *внутренних состояний* автомата;

б) $\mathcal{A} = \{a_0, \ldots, a_n\}$ — конечный *входной алфавит* автомата;

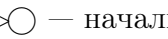
в) $\delta : Q \times \mathcal{A} \to Q$ — *функция перехода*;

г) $q_0 \in Q$ — *начальное состояние*;

д) $F \subseteq Q$ — множество *выделенных (конечных) состояний*.

Графическое изображение детерминированных автоматов

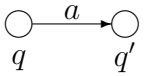
Автоматы удобно изображать графически, используя следующие геометрические фигуры:

 — начальное состояние;

 — выделенное состояние;

 — промежуточное состояние;

 — одновременно начальное и выделенное состояние;



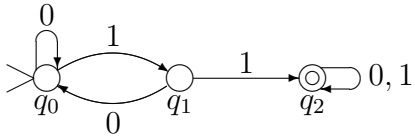
— такая дуга присутствует в автомате, если значение функции перехода $\delta(q, a) = q'$;

— такая петля присутствует в автомате, если значение функции перехода $\delta(q, a) = q$.

Пример. Например, автомат $\mathfrak{A} = \langle Q, \mathcal{A}, \delta, q_0, F\rangle$, где $\mathcal{A} = \{0, 1\}$, $Q = \{q_0, q_1, q_2\}$, $F = \{q_2\}$, а функция перехода задается соотношениями

$$\begin{array}{lll} \delta(q_0, 0) = q_0, & \delta(q_1, 0) = q_0, & \delta(q_2, 0) = q_2, \\ \delta(q_0, 1) = q_1, & \delta(q_1, 1) = q_2, & \delta(q_2, 1) = q_2, \end{array}$$

имеет следующее графическое изображение:



Определение. *Путем* в детерминированном конечном автомате $\mathfrak{A} = \langle Q, \mathcal{A}, \delta, q_0, F\rangle$ назовем любую конечную последовательность $\langle r_0, s_1, r_1, \ldots, s_k, r_k\rangle$, где $r_0, \ldots, r_k \in Q$, $s_1, \ldots, s_k \in \mathcal{A}$ и $\delta(r_i, s_{i+1}) = r_{i+1}$ для всех $i < k$. Если $\langle r_0, s_1, r_1, \ldots, s_k, r_k\rangle$ — путь в автомате, то будем обозначать его через

$$r_0 \xrightarrow{s_1} r_1 \xrightarrow{s_2} \ldots \xrightarrow{s_k} r_k$$

и говорить, что слово $w = s_1 s_2 \ldots s_k$ *читается вдоль дуг данного пути*. В частности, пустое слово Λ читается вдоль пути $\langle r_0\rangle$, состоящего из одного состояния и не содержащего ни одной дуги.

Определение. Пусть $\mathfrak{A} = \langle Q, \mathcal{A}, \delta, q_0, F\rangle$ — д.к.а. Расширим функцию δ до функции $\delta^* : Q \times \mathcal{A}^* \to Q$ следующим образом индукцией по длине слова:

$$\delta^*(q, \Lambda) = q, \quad \delta^*(q, wa) = \delta(\delta^*(q, w), a),$$

где $q \in Q$, $w \in \mathcal{A}^*$, $a \in \mathcal{A}$.

Определение. Говорят, что д.к.а. $\mathfrak{A} = \langle Q, \mathcal{A}, \delta, q_0, F\rangle$ *распознает* слово $w \in \mathcal{A}^*$, если $\delta^*(q_0, w) \in F$.

Другими словами, слово $w = s_1 s_2 \ldots s_k$ распознается автоматом, если в нем существует путь

$$q_0 = r_0 \xrightarrow{s_1} r_1 \xrightarrow{s_2} \ldots \xrightarrow{s_k} r_k \in F$$

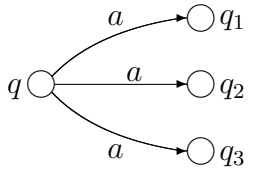
такой, что он начинается в начальном состоянии q_0, вдоль его дуг читается слово $s_1 s_2 \ldots s_k$ (в детерминированном автомате такой путь определяется однозначно по слову w) и заканчивается в некотором выделенном состоянии.

Определение. Через $T(\mathfrak{A}) = \{w \in \mathcal{A}^* \mid \delta^*(q_0, w) \in F\}$ будем обозначать *язык всех слов, распознаваемых конечным автоматом* $\mathfrak{A}$.

Определение. Язык $L \subseteq \mathcal{A}^*$ называется *автоматным*, если существует конечный автомат $\mathfrak{A}$ такой, что $L = T(\mathfrak{A})$.

Пример (продолжение)**.** Автомат из предыдущего примера распознает в точности все слова в алфавите $\{0, 1\}$, которые содержат подслово 11, т. е. $T(\mathfrak{A}) = \{w \in \{0, 1\}^* \mid w$ содержит подслово $11\}$. Действительно, начав чтение произвольного слова w в начальном состоянии данного автомата, попасть в выделенное состояние можно, только пройдя по дуге, ведущей из q_0 в q_1 (помеченной 1), а затем сразу же по дуге, ведущей из q_1 в q_2 (тоже помеченной 1), — такое возможно лишь в случае, когда w содержит две единицы подряд.

§ 5. Недетерминированные конечные автоматы



Недетерминированные конечные автоматы отличаются от детерминированных тем, что из любого состояния q после считывания символа a возможны сразу несколько (или ни одного) переходов в состояния, принадлежащие *множеству возможных состояний* $\Delta(q, a)$. Это означает, что автомат может продолжить дальнейший процесс чтения символов, перейдя в любое из возможных состояний.

Можно считать, что в таких ситуациях работа автомата разветвляется на несколько параллельных ветвей вычислений, каждая из которых начинается с определенного состояния $q_i \in \Delta(q, a)$. Если $\Delta(q, a)$ пусто, то работа автомата вдоль данной ветви вычислений заканчивается в состоянии q, даже если на входной ленте остались несчитанные символы. Таким образом, вдоль одних ветвей вычислений автомат может прочитать все входные символы и остановится в выделенном состоянии, вдоль других может прочитать все символы, но остановится в невыделенном состоянии, вдоль третьих ветвей работа автомата «обрывается на полуслове».

Перейдем к формальным определениям, связанным с недетерминированными автоматами.

Определение. *Недетерминированным конечным автоматом* (сокращенно н.к.а.) называется упорядоченная пятерка $\mathfrak{A} = \langle Q, \mathcal{A}, \Delta, q_0, F\rangle$, в которой $Q, \mathcal{A}, q_0, F$ определяются и называются так же, как в детерминированном случае, а *функция переходов* Δ является функцией вида $\Delta : Q \times \mathcal{A} \to P(Q)$, где $P(Q)$ — множество всех подмножеств Q (см. аксиому степени из § 1).

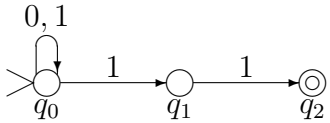
Графическое изображение недетерминированных автоматов

При графическом изображении недетерминированных автоматов используются те же обозначения, что и в детерминированном случае. При этом из одного состояния автомата могут выходить сразу несколько (или нисколько) стрелок, помеченных одной и той же буквой алфавита. Дуга, выходящая из состояния q, входящая в состояние q', помеченная символом a, присутствует в схеме автомата тогда и только тогда, когда $q' \in \Delta(q, a)$.

Пример. Например, автомат $\mathfrak{A} = \langle Q, \mathcal{A}, \Delta, q_0, F\rangle$, где $\mathcal{A} = \{0, 1\}$, $Q = \{q_0, q_1, q_2\}$, $F = \{q_2\}$, а функция переходов задается соотношениями

$$\begin{array}{lll} \Delta(q_0, 0) = \{q_0\}, & \Delta(q_1, 0) = \varnothing, & \Delta(q_2, 0) = \varnothing, \\ \Delta(q_0, 1) = \{q_0, q_1\}, & \Delta(q_1, 1) = \{q_2\}, & \Delta(q_2, 1) = \varnothing, \end{array}$$

имеет следующее графическое изображение:



Определение. *Путь* в недетерминированном конечном автомате $\mathfrak{A}=\langle Q, \mathcal{A}, \Delta, q_0, F\rangle$ определяется и обозначается так же, как в детерминированном случае, нужно лишь условие $\delta(r_i, s_{i+1}) = r_{i+1}$ заменить на условие $r_{i+1} \in \Delta(r_i, s_{i+1})$.

Определение. Говорят, что н.к.а. $\mathfrak{A}=\langle Q, \mathcal{A}, \Delta, q_0, F\rangle$ *распознает* слово $s_1 s_2 \dots s_k \in \mathcal{A}^*$, если существует последовательность состояний $q_0 = r_0, r_1, \dots, r_k$ такая, что

$$\begin{aligned} &r_1 \in \Delta(r_0, s_1),\\ &r_2 \in \Delta(r_1, s_2),\\ &\vdots \qquad\qquad \vdots\\ &r_k \in \Delta(r_{k-1}, s_k), \end{aligned}$$

и при этом $r_k \in F$.

Другими словами, слово $w = s_1 s_2 \dots s_k$ распознается автоматом, если в нем существует хотя бы один путь

$$q_0 = r_0 \xrightarrow{s_1} r_1 \xrightarrow{s_2} \dots \xrightarrow{s_k} r_k \in F$$

такой, что он начинается в начальном состоянии q_0, вдоль его дуг читается слово $s_1 s_2 \dots s_k$ (в недетерминированном автомате такой путь определяется неоднозначно по слову w) и заканчивается в некотором выделенном состоянии.

Определение. Как и раньше, через $T(\mathfrak{A})$ обозначается *язык всех слов, распознаваемых автоматом* $\mathfrak{A}$.

Пример (продолжение)**.** Автомат из предыдущего примера распознает в точности все слова в алфавите $\{0,1\}$, которые имеют суффикс 11, т. е. $T(\mathfrak{A}) = \{w \in \{0,1\}^* \mid \exists v \in \{0,1\}^* (w = v11)\}$. Действительно, любой путь, заканчивающийся в выделенном состоянии данного автомата, обязан содержать участок $q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2$. Отсюда следует, что слова, распознаваемые $\mathfrak{A}$, должны содержать хотя бы одно вхождение подслова 11. Поскольку из выделенного состояния нет выходящих стрелок, то любой путь, содержащий описанный выше участок, обрывается как раз после прохождения данного участка. Отсюда следует, что в словах, распознаваемых автоматом, после крайнего справа вхождения подслова 11 нет букв.

Теорема 3 (о детерминизации)**.** *Для любого недетерминированного конечного автомата* $\mathfrak{A}$ *существует детерминированный конечный автомат* $\overline{\mathfrak{A}}$ *такой, что* $T(\mathfrak{A}) = T(\overline{\mathfrak{A}})$.

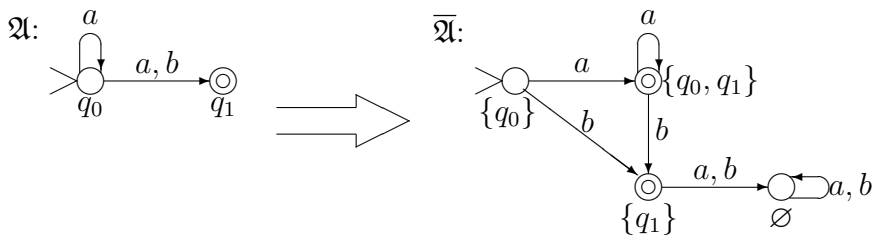
Доказательство. Пусть $\mathfrak{A} = \langle Q, \mathcal{A}, \Delta, q_0, F\rangle$ — исходный н.к.а. Определим следующий д.к.а. $\overline{\mathfrak{A}} = \langle \overline{Q}, \mathcal{A}, \delta, \overline{q}_0, \overline{F}\rangle$ (см. пример на рисунке):

а) $\overline{Q} = P(Q)$;

б) $\delta(\overline{q}, a) = \bigcup\limits_{r \in \overline{q}} \Delta(r, a)$, где $\overline{q} \in \overline{Q}$ и $a \in \mathcal{A}$;

в) $\overline{q}_0 = \{q_0\}$;

г) $\overline{F} = \{\overline{q} \in \overline{Q} \mid \overline{q} \cap F \neq \varnothing\}$.

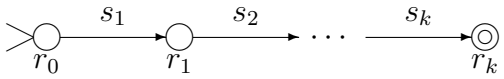


Докажем, что $T(\mathfrak{A}) = T(\overline{\mathfrak{A}})$, показав два включения: $T(\mathfrak{A}) \subseteq T(\overline{\mathfrak{A}})$ и $T(\overline{\mathfrak{A}}) \subseteq T(\mathfrak{A})$.

Пусть слово $w = s_1 \dots s_k \in T(\mathfrak{A})$. Следовательно, существуют такие состояния $r_0, r_1, \dots, r_k \in Q$, что

$$\begin{aligned} &q_0 = r_0, \\ &r_1 \in \Delta(r_0, s_1), \\ &r_2 \in \Delta(r_1, s_2), \\ &\vdots \qquad\qquad \vdots \\ &r_k \in \Delta(r_{k-1}, s_k), \\ &r_k \in F. \end{aligned}$$

Иначе говоря, слово w читается вдоль дуг следующего пути в автомате $\mathfrak{A}$.



Подадим слово w на вход автомата $\overline{\mathfrak{A}}$. Тогда он будет находиться в следующих состояниях:

$$\overline{r}_0 = \overline{q}_0, \quad \overline{r}_1 = \delta(\overline{r}_0, s_1), \quad \dots, \quad \overline{r}_k = \delta(\overline{r}_{k-1}, s_k).$$

Докажем индукцией по $i \leqslant k$, что $r_i \in \overline{r}_i$.

1^0. Базис индукции очевиден: $r_0 = q_0 \in \{q_0\} = \overline{q}_0 = \overline{r}_0$.

2^0. Пусть $r_i \in \overline{r}_i$. Отсюда следует, что $\Delta(r_i, s_{i+1}) \subseteq \bigcup\limits_{r \in \overline{r}_i} \Delta(r, s_{i+1})$. Тогда получаем $r_{i+1} \in \Delta(r_i, s_{i+1}) \subseteq \bigcup\limits_{r \in \overline{r}_i} \Delta(r, s_{i+1}) = \delta(\overline{r}_i, s_{i+1}) = \overline{r}_{i+1}$. Следовательно, $r_{i+1} \in \overline{r}_{i+1}$. Что и требовалось доказать.

Итак, для любого $i \leqslant k$ выполняется $r_i \in \overline{r}_i$. В частности, $r_k \in \overline{r}_k$. Поскольку $r_k \in F$, то $\overline{r}_k \cap F \neq \varnothing$. Следовательно, $\overline{r}_k \in \overline{F}$. Отсюда заключаем, что $w \in T(\overline{\mathfrak{A}})$, и значит включение $T(\mathfrak{A}) \subseteq T(\overline{\mathfrak{A}})$ доказано.

Пусть теперь слово $w = s_1 \dots s_k \in T(\overline{\mathfrak{A}})$, и пусть при чтении данного слова автомат $\overline{\mathfrak{A}}$ находился в состояниях

$$\overline{r}_0 = \overline{q}_0, \quad \overline{r}_1 = \delta(\overline{r}_0, s_1), \quad \dots, \quad \overline{r}_k = \delta(\overline{r}_{k-1}, s_k),$$

причем последнее состояние $\overline{r}_k \in \overline{F}$.

Так как $\overline{r}_k \in \overline{F}$, то $\overline{r}_k \cap F \neq \varnothing$, и значит найдется $r_k \in \overline{r}_k \cap F$.

Так как $r_k \in \overline{r}_k = \delta(\overline{r}_{k-1}, s_k) = \bigcup\limits_{r \in \overline{r}_{k-1}} \Delta(r, s_k)$, то найдется $r_{k-1} \in \overline{r}_{k-1}$ такое, что $r_k \in \Delta(r_{k-1}, s_k)$.

Так как $r_{k-1} \in \overline{r}_{k-1}$, то аналогично находим $r_{k-2} \in \overline{r}_{k-2}$ такое, что выполняется $r_{k-1} \in \Delta(r_{k-2}, s_{k-1})$.

И так далее.

В итоге получаем последовательность $r_0, r_1, \dots, r_k \in Q$ такую, что для каждого $i \in \{1, \dots, k\}$ выполняется $r_{i-1} \in \overline{r}_{i-1}$ и $r_i \in \Delta(r_{i-1}, s_i)$. Поскольку $r_0 \in \overline{r}_0 = \overline{q}_0 = \{q_0\}$, заключаем $r_0 = q_0$. Кроме этого, по построению $r_k \in F$. Отсюда по определению следует, что $w \in T(\mathfrak{A})$. Таким образом, включение $T(\overline{\mathfrak{A}}) \subseteq T(\mathfrak{A})$ тоже доказано. $\square$

Замечание. В заключение параграфа заметим, что в некоторых источниках (см., например, [4],[12]) используется другое определение недетерминированного конечного автомата, которое отличается от введенного выше тем, что функция переходов Δ имеет вид $\Delta : Q \times (\mathcal{A} \cup \{\Lambda\}) \to P(Q)$. Автоматы данного типа называют *автоматами с Λ-переходами*, поскольку в них допускаются дуги, помеченные пустым словом Λ. Наличие дуги, помеченной Λ, выходящей из состояние q и входящей в состояние q', означает, что автомат, находясь в состоянии q, может перейти в состояние q', не считывая символа с ленты. Пользуясь терминами из программирования, можно сказать, что в такой ситуации автомат осуществляет *безусловный переход* из q в q'.

Соответствующим образом изменяется и определение распознаваемости слова. Слово $w = s_1 s_2 \dots s_k$, где s_i — буквы алфавита, распознается автоматом с Λ-переходами, если в нем существует хотя бы один путь

$$r_0 \xrightarrow{t_1} r_1 \xrightarrow{t_2} \dots \xrightarrow{t_m} r_m$$

такой, что он начинается в начальном состоянии r_0, заканчивается в некотором выделенном состоянии r_m, и вдоль его дуг читается слово $t_1 t_2 \dots t_m = s_1 s_2 \dots s_k$, причем $m \geqslant k$, т. е. некоторые из символов $t_1, \dots, t_m$ могут быть пустыми.

Такое определение удобно использовать при доказательстве некоторых свойств. Однако добавление Λ-переходов в определение недетерминированного автомата не расширяет класс автоматных языков, т. е. язык распознается некоторым недетерминированным конечным автоматом (согласно нашему определению) тогда и только тогда, когда он распознается некоторым недетерминированным конечным автоматом с Λ-переходами. Таким образом, можно окончательно утверждать, что три различных подхода — детерминированные автоматы, недетерминированные автоматы и недетерминированные автоматы с Λ-переходами, эквивалентны.

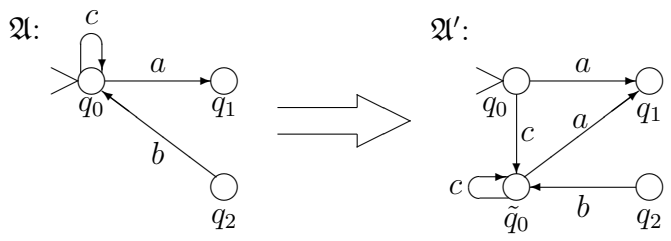
§ 6. Свойства автоматных языков

В данном параграфе мы докажем важные теоретико-множественные свойства автоматных языков, а также покажем, что существуют неавтоматные языки. Для этих целей нам понадобятся следующие определение и лемма.

Определение. Говорят, что д.к.а. $\mathfrak{A} = \langle Q, \mathcal{A}, \delta, q_0, F \rangle$ обладает *свойством вахтера*, если для любых $q \in Q$, $a \in \mathcal{A}$ имеет место $\delta(q, a) \neq q_0$, т. е. автомат находится в начальном состоянии только в самом начале своей работы.

Лемма 4 (о вахтере)**.** *Для любого д.к.а. $\mathfrak{A} = \langle Q, \mathcal{A}, \delta, q_0, F \rangle$ существует д.к.а. $\mathfrak{A}' = \langle Q', \mathcal{A}, \delta', q_0, F' \rangle$ такой, что $T(\mathfrak{A}) = T(\mathfrak{A}')$ и $\mathfrak{A}'$ обладает свойством вахтера.*

Доказательство. Мы дадим неформальное доказательство, описав процесс преобразования графической диаграммы автомата $\mathfrak{A}$ в графическую диаграмму искомого автомата $\mathfrak{A}'$ (см. схему преобразования на рисунке):



а) Добавим в автомат $\mathfrak{A}$ новое состояние $\tilde{q}_0$. Если q_0 было выделенным, то $\tilde{q}_0$ тоже сделаем выделенным.

б) Для каждой стрелки из q_0 в $q_1 \neq q_0$ добавим в автомат новую стрелку из $\tilde{q}_0$ в q_1 и пометим ее той же буквой.

в) Для каждой стрелки из q_0 в q_0 добавим в автомат новую стрелку из $\tilde{q}_0$ в $\tilde{q}_0$ и пометим ее той же буквой.

г) Перенаправим все стрелки, приходящие в q_0 из q_2 (включая стрелки из q_0 в q_0) так, чтобы они приходили из q_2 в $\tilde{q}_0$.

Построенный таким образом автомат обозначим как $\mathfrak{A}'=\langle Q', \mathcal{A}, \delta', q_0, F'\rangle$. Нетрудно видеть, что $\mathfrak{A}'$ детерминированный. В силу пункта (г) автомат $\mathfrak{A}'$ обладает свойством вахтера. Равенство $T(\mathfrak{A}) = T(\mathfrak{A}')$ следует из того, что если слово $w \in \mathcal{A}^*$ читается вдоль пути в автомате $\mathfrak{A}$, начинающегося в q_0 и заканчивающегося в некотором $r \in F$, то, преобразовав этот путь в соответствии с пунктами (а)–(г), мы получим путь в автомате $\mathfrak{A}'$, начинающийся в q_0, заканчивающийся в некотором $r' \in F'$, вдоль которого читается то же самое слово w, и наоборот, если w читается вдоль пути в автомате $\mathfrak{A}'$, начинающегося в q_0 и заканчивающегося в некотором $r' \in F'$, то, сделав обратное преобразование, мы получим путь в автомате $\mathfrak{A}$, который начинается в q_0, заканчивается в $r \in F$, и вдоль которого читается w. □

Определение. Говорят, что подмножество X множества A *замкнуто относительно операции* $f : A^n \to A$, если для любых $x_1, \ldots, x_n \in X$ имеет место $f(x_1, \ldots, x_n) \in X$.

Теорема 5. *Автоматные языки замкнуты относительно объединения, пересечения, дополнения, конкатенации и звездочки Клини.*

Доказательство. Для каждой из пяти операций мы неформально опишем, как по заданым автоматам, распознающим исходные языки, построить автомат, распознающий результат применения данной операции к исходным языкам. Заметим, что в силу теоремы 3 для каждого из случаев достаточно строить недетерминированный автомат.

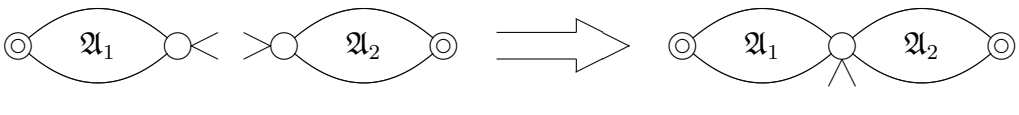
Объединение. Пусть $\mathfrak{A}_1$, $\mathfrak{A}_2$ — детерминированные конечные автоматы над алфавитом $\mathcal{A}$. Нам необходимо определить автомат $\mathfrak{A}$ такой, что $T(\mathfrak{A}) = T(\mathfrak{A}_1) \cup T(\mathfrak{A}_2)$. В силу Леммы о вахтере можно считать, что $\mathfrak{A}_1$ и $\mathfrak{A}_2$ обладают свойством вахтера. Кроме этого, можно считать, что множества состояний $\mathfrak{A}_1$ и $\mathfrak{A}_2$ не пересекаются, в противном случае следует переименовать состояния.

Автомат $\mathfrak{A}$ строится следующим образом (см. схему построения на рисунке):

а) Соединим графические диаграммы автоматов $\mathfrak{A}_1$ и $\mathfrak{A}_2$ в одну диаграмму, отождествив их начальные состояния. Объявим полученное таким отождествлением состояние начальным в $\mathfrak{A}$. Поскольку множества состояний автоматов не пересекались, никакие другие состояния (кроме начальных) не склеиваются.

б) Множеством выделенных состояний $\mathfrak{A}$ будет объединение множеств выделенных состояний автоматов $\mathfrak{A}_1$ и $\mathfrak{A}_2$.

в) Начальное состояние $\mathfrak{A}$ будет выделенным тогда и только тогда, когда оно является выделенным хотя бы в одном из исходных автоматов.



Такое описание полностью задает автомат $\mathfrak{A}$. Поскольку $\mathfrak{A}_1$ и $\mathfrak{A}_2$ обладают свойством вахтера, то любой путь по дугам автомата $\mathfrak{A}$, который начинается в начальном состоянии и заканчивается в выделенном состоянии, будет полностью расположен внутри $\mathfrak{A}_1$ или внутри $\mathfrak{A}_2$. Следовательно, любое слово, распознаваемое $\mathfrak{A}$, — это слово, распознаваемое $\mathfrak{A}_1$ или $\mathfrak{A}_2$. С другой стороны, любое слово, распознаваемое $\mathfrak{A}_1$ или $\mathfrak{A}_2$, очевидно, распознается автоматом $\mathfrak{A}$. Таким образом, автомат $\mathfrak{A}$ — искомый.

Дополнение. Пусть д.к.а. $\mathfrak{A} = \langle Q, \mathcal{A}, \delta, q_0, F\rangle$ распознает язык L, т. е. для любого слова $w \in \mathcal{A}^*$ имеет место эквивалентность $w \in L \iff \delta^*(q_0, w) \in F$. Отсюда вытекает эквивалентность $w \notin L \iff \delta^*(q_0, w) \notin F$. Другими словами, имеет место условие $w \in \mathcal{A}^* \setminus L \iff \delta^*(q_0, w) \in Q \setminus F$. Отсюда следует, что д.к.а $\mathfrak{A}' = \langle Q, \mathcal{A}, \delta, q_0, Q \setminus F\rangle$ распознает дополнение $\overline{L} = \mathcal{A}^* \setminus L$.

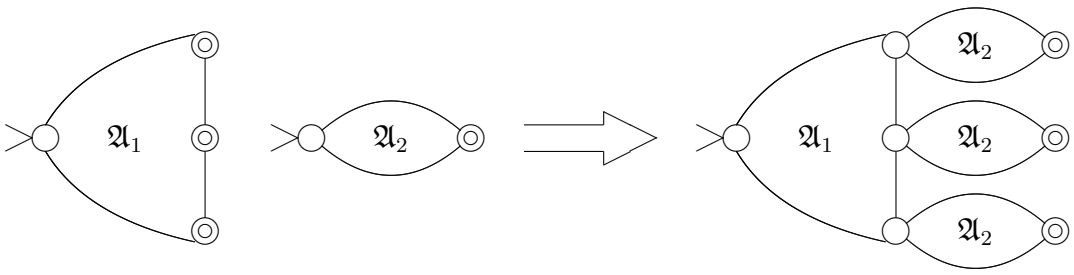
Пересечение. Замкнутость автоматных языков относительно пересечения следует из замкнутости относительно объединения и дополнения, а также из теоретико-множественного тождества $A \cap B = \overline{\overline{A} \cup \overline{B}}$.

Конкатенация. Пусть $\mathfrak{A}_1$, $\mathfrak{A}_2$ — детерминированные конечные автоматы над алфавитом $\mathcal{A}$. Можно считать, что $\mathfrak{A}_2$ обладает свойством вахтера. Построим автомат $\mathfrak{A}$, распознающий язык $T(\mathfrak{A}_1)T(\mathfrak{A}_2)$, следующим образом (см. схему построения на рисунке):

а) К каждому выделенному состоянию автомата $\mathfrak{A}_1$ подвесим копию автомата $\mathfrak{A}_2$, отождествив данное выделенное состояние с начальным состоянием копии. При этом считаем, что остальные состояния автоматов не склеиваются (этого можно добиться путем переименования состояний).

б) Начальным состоянием полученного автомата объявляется начальное состояние $\mathfrak{A}_1$.

в) Выделенными состояниями полученного автомата объявляются все выделенные состояния всех копий автомата $\mathfrak{A}_2$. Других выделенных состояний нет.



В силу Леммы о вахтере любой путь вдоль дуг нового автомата, начинающийся в начальном состоянии и заканчивающийся в выделенном состоянии, разбивается на два участка. Первый участок состоит только из дуг автомата $\mathfrak{A}_1$ и заканчивается в одном из (бывших) выделенных состояний $\mathfrak{A}_1$. Второй участок состоит только из дуг соответствующей копии автомата $\mathfrak{A}_2$ и заканчивается в одном из выделенных состояний нового автомата. Отсюда следует, что $T(\mathfrak{A}) = T(\mathfrak{A}_1)T(\mathfrak{A}_2)$.

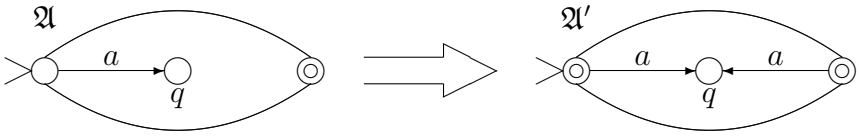
Звездочка Клини. Пусть $\mathfrak{A}$ — д.к.а., обладающий свойством вахтера. Построим автомат $\mathfrak{A}'$, распознающий язык $(T(\mathfrak{A}))^*$, следующим образом (см. схему построения на рисунке):

а) Для каждой дуги автомата $\mathfrak{A}$, выходящей из начального состояния, входящей в состояние q и помеченной символом a, проделаем следующее: для каждого выделенного состояния добавим дугу, выходящую из этого выделенного состояния, входящую

в q и помеченную буквой a (если такая дуга уже есть в автомате $\mathfrak{A}$, то не добавляем ее).

б) Начальным состоянием автомата $\mathfrak{A}'$ объявляется начальное состояние $\mathfrak{A}$.

в) Выделенными состояниями автомата $\mathfrak{A}'$ объявляются все выделенные состояния $\mathfrak{A}$. Кроме этого, сделаем начальное состояние выделенным (если оно уже не было таковым).



Докажем, что $(T(\mathfrak{A}))^* = T(\mathfrak{A}')$. Для этого сначала установим справедливость включения $(T(\mathfrak{A}))^* \subseteq T(\mathfrak{A}')$. Пусть слово $w \in (T(\mathfrak{A}))^*$. Если $w = \Lambda$, то $w \in T(\mathfrak{A}')$ в силу пункта (в). Если же $w \neq \Lambda$, то слово w представимо в виде $w = w_1 \dots w_n$, где $w_i \in T(\mathfrak{A})$ для каждого $1 \leqslant i \leqslant n$. Следовательно, для каждого $1 \leqslant i \leqslant n$ слово w_i читается вдоль следующего пути в автомате $\mathfrak{A}$:

$$q_0 = r_0^i \xrightarrow{s_1^i} r_1^i \xrightarrow{s_2^i} \dots \xrightarrow{s_{k_i}^i} r_{k_i}^i \in F,$$

в котором последнее состояние $r_{k_i}^i$ является выделенным. Поэтому в силу пункта (а) для каждого $2 \leqslant i \leqslant n$ в новом автомате $\mathfrak{A}'$ существует дуга, выходящая из $r_{k_{i-1}}^{i-1}$, входящая в r_1^i и помеченная буквой s_1^i. Таким образом, для каждого $2 \leqslant i \leqslant n$ слово w_i будет читаться вдоль следующего пути в автомате $\mathfrak{A}'$:

$$r_{k_{i-1}}^{i-1} \xrightarrow{s_1^i} r_1^i \xrightarrow{s_2^i} \dots \xrightarrow{s_{k_i}^i} r_{k_i}^i \in F.$$

Соединив последовательно все такие пути в одну цепочку, мы получим путь в автомате $\mathfrak{A}'$, который начинается в начальном состоянии, заканчивается в выделенном состоянии, и вдоль дуг которого читается w. Следовательно, $w \in T(\mathfrak{A}')$.

Теперь установим обратное включение $T(\mathfrak{A}') \subseteq (T(\mathfrak{A}))^*$. Пусть $w \in T(\mathfrak{A}')$. Можно считать, что $w \neq \Lambda$ (случай пустого слова тривиален). Следовательно, w читается вдоль пути в автомате $\mathfrak{A}'$, который начинается в начальном состоянии q_0 и заканчивается в некотором выделенном состоянии q'. Заметим, что в силу свойства вахтера $q_0 \neq q'$, поэтому состояние q' является выделенным и в исходном автомате $\mathfrak{A}$. Пусть в данном пути встречается ровно k *новых* дуг, т. е. дуг, добавленных по пункту (а). Для $1 \leqslant i \leqslant k$ введем следующие обозначения. Пусть i-ая новая дуга, встретившаяся в данном пути, имеет вид $p_i \xrightarrow{s_i} r_i$, т. е. выходит из состояния p_i, входит в состояние r_i и помечена символом s_i. В частности, p_i является выделенным. Обозначим через w_0 слово, которое читается вдоль участка нашего пути, начинающегося в состоянии q_0 и заканчивающегося в p_1. Для $1 \leqslant i \leqslant k-1$ обозначим через w_i слово, которое читается вдоль участка пути, начинающегося в p_i и заканчивающегося в p_{i+1}. Наконец через w_k обозначим слово, которое читается вдоль участка пути, начинающегося в p_k и заканчивающегося в q'. Очевидно слово $w_0 \in T(\mathfrak{A})$, поскольку оно читается вдоль участка, не содержащего новых дуг. Для каждого $1 \leqslant i \leqslant k$ заменим в i-ом участке дугу $p_i \xrightarrow{s_i} r_i$ на дугу $q_0 \xrightarrow{s_i} r_i$ из исходного автомата $\mathfrak{A}$ (такая дуга существует по

построению), в результате получим путь по дугам исходного автомата, который начинается в q_0, заканчивается в выделенном состоянии, и вдоль которого по-прежнему читается слово w_i. Следовательно, $w_i \in T(\mathfrak{A})$. Таким образом, $w = w_0 w_1 \dots w_k$, где все $w_i \in T(\mathfrak{A})$, т. е. $w \in (T(\mathfrak{A}))^*$. □

Замечание. В предыдущей теореме можно предложить *прямое* доказательство замкнутости автоматных языков относительно пересечения, а именно, если заданы два д.к.а. $\mathfrak{A}_1 = \langle Q_1, \mathcal{A}, \delta_1, q_0^1, F_1\rangle$ и $\mathfrak{A}_2 = \langle Q_2, \mathcal{A}, \delta_2, q_0^2, F_2\rangle$, то определим д.к.а. $\mathfrak{A}_1 \times \mathfrak{A}_2$, который называется *прямым произведением* исходных автоматов, следующим образом: $\mathfrak{A}_1 \times \mathfrak{A}_2 = \langle Q_1 \times Q_2, \mathcal{A}, \delta, \langle q_0^1, q_0^2\rangle, F_1 \times F_2\rangle$, где функция перехода $\delta(\langle q_1, q_2\rangle, a) = \langle \delta_1(q_1, a), \delta_2(q_2, a)\rangle$ для любых $q_1 \in Q_1$, $q_2 \in Q_2$, $a \in \mathcal{A}$. Несложно убедиться в том, что $T(\mathfrak{A}_1 \times \mathfrak{A}_2) = T(\mathfrak{A}_1) \cap T(\mathfrak{A}_2)$.

Теорема 6. *Любой конечный язык является автоматным.*

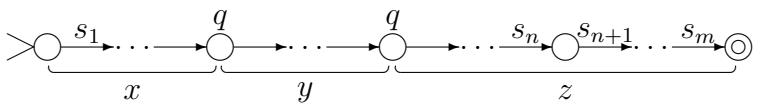
Доказательство. а) Нетрудно построить в явном виде автоматы, распознающие языки $\varnothing$, $\{\Lambda\}$, $\{a\}$, где a — буква.

б) Из (а) и теоремы 5 (конкатенация) следует, что любой язык вида $\{w\}$, где w — слово, является автоматным.

в) Из (б) и теоремы 5 (объединение) следует, что любой непустой конечный язык является автоматным. □

Лемма 7 (о накачивании)**.** *Пусть L — автоматный язык. Тогда существует $n \geqslant 1$ такое, что для любого слова $w \in L$, где $|w| \geqslant n$, существует представление в виде $w = xyz$, где $y \neq \Lambda$, $|xy| \leqslant n$ и $xy^i z \in L$ для всех $i \geqslant 0$.*

Доказательство. Пусть $\mathfrak{A}$ — д.к.а. такой, что $T(\mathfrak{A}) = L$, и пусть n — число состояний автомата $\mathfrak{A}$. Рассмотрим произвольное слово $w \in L$ со свойством $|w| \geqslant n$. Следовательно, можно представить $w = s_1 \dots s_n s_{n+1} \dots s_m$, где s_i — буквы. Так как w распознается автоматом $\mathfrak{A}$, то существует путь по дугам $\mathfrak{A}$, начинающийся в начальном состоянии, заканчивающийся в выделенном состоянии, и вдоль которого читается слово w.



Рассмотрим первые n переходов в этом пути, вдоль которых читаются первые n букв слова w. Так как число состояний, пройденных на этом участке пути, равно $n + 1$, то существует хотя бы одно состояние q, которое встречается не менее двух раз.

Пусть x — часть слова $s_1 \dots s_n$, которая читается от начального состояния до первого попадания в состояние q; y — часть слова $s_1 \dots s_n$, которая читается от первого попадания в состояние q до последнего попадания в состояние q; z — остальная часть w (см. рисунок). Тогда $y \neq \Lambda$ и $|xy| \leqslant n$. Кроме этого, чтение подслова y начинается и заканчивается в состоянии q. Следовательно, этот участок можно удалить из нашего пути или пройти по нему произвольное количество раз. Отсюда заключаем, что слово $xy^i z \in T(\mathfrak{A})$ для всех $i \geqslant 0$. □

Следствие 8. *Существуют неавтоматные языки.*

Доказательство. Рассмотрим язык $L = \{a^m b^m \mid m \in \omega\}$ над алфавитом $\{a, b\}$. Допустим, L — автоматный. Следовательно, существует $n \geqslant 1$ как в лемме о накачивании. Рассмотрим слово $a^n b^n \in L$. Длина $|a^n b^n| > n$. Следовательно, по лемме можно представить $a^n b^n = xyz$, где $y \neq \Lambda$, $|xy| \leqslant n$ и $xy^i z \in L$ для любого $i \geqslant 0$. Отсюда следует, что существует $k \geqslant 1$ такое, что $y = a^k$, и слово x не содержит букв b. Следовательно, слово $xz = a^{n-k} b^n \in L$, что невозможно, поскольку $n - k < n$. Таким образом, L не является автоматным. □

Замечание. Лемма о накачивании является необходимым условием автоматности языка. Это условие является достаточно сильным и демонстрирует определенную ограниченность вычислительных возможностей конечных автоматов. Например, как мы заметили, никакой конечный автомат не способен распознать язык $\{a^m b^m \mid m \in \omega\}$. Однако несложно построить формальную грамматику, которая порождает этот язык. Другими словами, не любую алгоритмически разрешимую задачу можно решить с помощью конечных автоматов. Тем не менее, язык конечных автоматов оказывается достаточным для алгоритмического описания многих важнейших классов задач в различных разделах математики и приложениях.

§ 7. Регулярные языки

В этом параграфе будет предложен другой подход для описания класса автоматных языков. Будет доказано, что автоматные языки — это в точности те языки, которые имеют «синтаксическое» описание в терминах регулярных выражений.

Определение. Пусть $\mathcal{A}$ — конечный алфавит, не содержащий символов $(,)$, $\cup$, $*$. Определим по индукции множество *регулярных выражений* над алфавитом $\mathcal{A}$:

1^0. Множества $\varnothing$, Λ, a, где $a \in \mathcal{A}$, являются *регулярными выражениями.*

2^0. Если α и β — регулярные выражения, то $(\alpha\beta)$, $(\alpha \cup \beta)$ и (α^*) тоже являются *регулярными выражениями.*

Таким образом, слово в алфавите $\mathcal{A} \cup \{(,), \cup, ^*\}$ называется регулярным выражением, если оно может быть получено конечным числом применений пунктов 1^0 и 2^0.

Определение. Схожим образом определим множество *обобщенно регулярных выражений* над алфавитом $\mathcal{A}$:

1^0. Множества $\varnothing$, Λ, a, где $a \in \mathcal{A}$, являются *обобщенно регулярными выражениями.*

2^0. Если α и β — обобщенно регулярные выражения, то $(\alpha\beta)$, $(\alpha \cup \beta)$, (α^*) и $(\overline{\alpha})$ тоже являются *обобщенно регулярными выражениями.*

Замечание. В дальнейшем мы будем опускать некоторые (в том числе внешние) скобки при записи обобщенно регулярных выражений, как это делается в обычной алгебре, считая, что операции имеют следующий приоритет: α^* — самая сильная операция, далее идет $\overline{\alpha}$, затем следует $\alpha\beta$, а операция $\alpha \cup \beta$ — самая слабая. Например, запись $\alpha^*\beta \cup \overline{\beta}\alpha$ на самом деле означает $(((\alpha^*)\beta) \cup ((\overline{\beta})\alpha))$.

Определение. Определим отображение $\mathcal{L}$ из множества всех обобщенно регулярных выражений над алфавитом $\mathcal{A}$ в множество всех языков над $\mathcal{A}$ следующим образом:

$$\begin{aligned}
&\mathcal{L}(\varnothing) = \varnothing,\\
&\mathcal{L}(\Lambda) = \{\Lambda\},\\
&\mathcal{L}(a) = \{a\}, \text{ для любого } a \in \mathcal{A},\\
&\mathcal{L}(\alpha\beta) = \mathcal{L}(\alpha)\mathcal{L}(\beta),\\
&\mathcal{L}(\alpha \cup \beta) = \mathcal{L}(\alpha) \cup \mathcal{L}(\beta),\\
&\mathcal{L}(\alpha^*) = \mathcal{L}(\alpha)^*,\\
&\mathcal{L}(\overline{\alpha}) = \mathcal{A}^* \setminus \mathcal{L}(\alpha).
\end{aligned}$$

Определение. Язык L над алфавитом $\mathcal{A}$ называется *(обобщенно) регулярным*, если существует (обобщенно) регулярное выражение α над алфавитом $\mathcal{A}$ такое, что $\mathcal{L}(\alpha) = L$. При этом будем говорить, что выражение α *задает* язык L.

Пример. Язык $L = \{w \in \{0,1\}^* \mid w \text{ содержит подслово } 11\}$ является регулярным, поскольку его можно задать регулярным выражением $(0^* \cup 10)^*11(0 \cup 1)^*$. Заметим, что для любого (обобщенно) регулярного языка существует бесконечно много (обобщенно) регулярных выражений, задающих его. Например, тот же язык L можно задать регулярным выражением $(0 \cup 1)^*11(0 \cup 1)^*$ или обобщенно регулярным выражением $\overline{\varnothing}11\overline{\varnothing}$.

Теорема 9. *Класс автоматных языков совпадает с классом регулярных языков.*

Доказательство. Сначала докажем, что любой регулярный язык автоматен. Пусть L — регулярный язык над алфавитом $\mathcal{A}$. Следовательно, найдется хотя бы одно регулярное выражение γ, которое задает L. Индукцией по длине выражения γ докажем, что L — автоматный.

1^0. Если γ является выражением вида $\varnothing$, Λ или a, где $a \in \mathcal{A}$, т. е. $L = \varnothing$, $L = \{\Lambda\}$ или $L = \{a\}$, то в силу теоремы 6 язык L является автоматным.

2^0. Если γ является выражением вида $(\alpha\beta)$, $(\alpha \cup \beta)$ или (α^*), где α, β — регулярные, то по индукционному предположению заключаем, что языки $L_1 = \mathcal{L}(\alpha)$ и $L_2 = \mathcal{L}(\beta)$ — автоматные. Тогда $L = L_1L_2$, или $L = L_1 \cup L_2$, или $L = L_1^*$ соответственно. В любом случае по теореме 5 получаем, что L автоматен.

Теперь докажем, что любой автоматный язык регулярен. Пусть L — произвольный автоматный язык. Следовательно, существует д.к.а. $\mathfrak{A} = \langle Q, \mathcal{A}, \delta, q_1, F\rangle$ с состояниями $Q = \{q_1, \ldots, q_n\}$ такой, что $T(\mathfrak{A}) = L$. Докажем, что L — регулярный. Для этого определим для всех $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant n$ и $0 \leqslant k \leqslant n$ множество слов:

$$\begin{aligned}
R(i,j,k) = \{w \in \mathcal{A}^* \mid\ & w \text{ читается вдоль пути автомата } \mathfrak{A}, \text{который}\\
& \text{начинается в } q_i, \text{заканчивается в } q_j \text{ и который в промежутке}\\
& \text{между ними не заходит в состояния } q_{k+1}, q_{k+2}, \ldots, q_n\}.
\end{aligned}$$

(Здесь термином «промежуток между q_i и q_j» мы называем множество всех состояний пути, исключая его начало q_i и его конец q_j. Напомним также, что пустое слово Λ по определению читается вдоль пути, который содержит только одно состояние и не содержит ни одной дуги.)

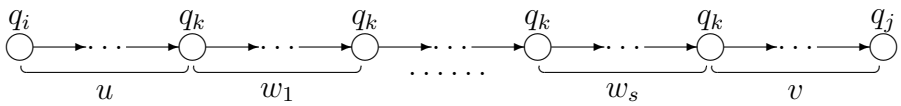
Заметим, что при $k = n$ множество $R(i, j, n)$ состоит в точности из всех слов, читаемых вдоль путей нашего автомата, идущих из q_i в q_j. Кроме этого, ясно, что

$$L = T(\mathfrak{A}) = \bigcup_{q_j \in F} R(1, j, n).$$

Так как регулярные языки замкнуты относительно объединения, то достаточно доказать, что все $R(i, j, k)$ регулярны. Докажем это утверждение индукцией по k.

1^0. При $k = 0$ множество $R(i, j, 0)$ — это все слова, читаемые вдоль одной дуги, ведущей из q_i в q_j. Возможны три случая. Если такая дуга существует и она помечена символом a, то $R(i, j, 0) = \{a\}$. Если такой дуги нет и $q_i = q_j$, то $R(i, j, 0) = \{\Lambda\}$. Если такой дуги нет и $q_i \neq q_j$, то $R(i, j, 0) = \varnothing$. По определению все такие языки регулярны.

2^0. Допустим, что утверждение доказано для $k - 1$, т. е. все языки $R(i, j, k - 1)$ регулярны. Рассмотрим произвольное слово $w \in R(i, j, k)$, оно читается вдоль пути, который начинается в q_i, несколько раз (может и 0 раз) заходит в q_k и заканчивается в q_j. Если этот путь не заходит в q_k, то вдоль него читается слово из $R(i, j, k - 1)$, т. е. $w \in R(i, j, k - 1)$. Если же этот путь заходит в q_k, то пусть u — подслово w, которое читается вдоль участка пути, начинающегося в q_i и заканчивающегося в первом попадании в состояние q_k; w_1 — подслово w, которое читается вдоль участка, начинающегося в первом попадании в q_k и заканчивающегося во втором попадании в q_k; ...; w_s — подслово w, которое читается вдоль участка, начинающегося в предпоследнем попадании в q_k и заканчивающегося в последнем попадании в q_k; и, наконец, пусть v — подслово w, которое читается вдоль участка, начинающегося в последнем попадании в q_k и заканчивающегося в q_j.



Тогда $u \in R(i, k, k - 1)$, $w_1, \ldots, w_s \in R(k, k, k - 1)$ и $v \in R(k, j, k - 1)$. Отсюда следует, что слово $w = uw_1 \ldots w_s v \in R(i, k, k - 1)R(k, k, k - 1)^* R(k, j, k - 1)$. Объединяя оба случая, заключаем, что имеет место включение $R(i, j, k) \subseteq R(i, j, k - 1) \cup R(i, k, k - 1)R(k, k, k - 1)^* R(k, j, k - 1)$. Нетрудно проверить, что обратное включение тоже верно. Таким образом, мы доказали равенство

$$R(i, j, k) = R(i, j, k - 1) \cup R(i, k, k - 1)R(k, k, k - 1)^* R(k, j, k - 1).$$

Следовательно, в силу индукционного предположения и замкнутости регулярных языков относительно объединения, конкатенации и звездочки Клини окончательно получаем, что $R(i, j, k)$ — регулярный язык. Что и требовалось доказать.

□

Следствие 10. *Язык является регулярным тогда и только тогда, когда он является обобщенно регулярным.*

Доказательство. Если язык L — регулярный, то, очевидно, L — обобщенно регулярный, поскольку любое регулярное выражение является обобщенно регулярным.

С другой стороны, если L — обобщенно регулярный, то, используя теорему 5 (дополнение), можно так же, как и в первой части доказательства предыдущей теоремы, показать, что L — автоматный. Следовательно, L — регулярный. □

Таким образом, регулярные и обобщенно регулярные выражения задают одни и те же языки. Однако этот факт не стал поводом для прекращения изучения свойств обобщенно регулярных выражений. Более того, с обобщенно регулярными выражениями связана одна достаточно известная математическая проблема, которая имеет простую формулировку, но до сих пор является открытой.

Определение. Определим отображение sh из множества всех обобщенно регулярных выражений над алфавитом $\mathcal{A}$ в множество ω следующим образом:

$$\begin{aligned}
&\mathrm{sh}(\varnothing) = 0,\\
&\mathrm{sh}(\Lambda) = 0,\\
&\mathrm{sh}(a) = 0, \text{ для любого } a \in \mathcal{A},\\
&\mathrm{sh}(\alpha\beta) = \max\{\mathrm{sh}(\alpha), \mathrm{sh}(\beta)\},\\
&\mathrm{sh}(\alpha \cup \beta) = \max\{\mathrm{sh}(\alpha), \mathrm{sh}(\beta)\},\\
&\mathrm{sh}(\alpha^*) = \mathrm{sh}(\alpha) + 1,\\
&\mathrm{sh}(\overline{\alpha}) = \mathrm{sh}(\alpha).
\end{aligned}$$

Число $\mathrm{sh}(\alpha)$ называется *звездной высотой* выражения α.

Определение. Для каждого регулярного языка L над алфавитом $\mathcal{A}$ определим:

$$\begin{aligned}
&\mathrm{sh}(L) = \min\{\mathrm{sh}(\alpha) \mid \alpha \text{ — регулярное выражение такое, что } \mathcal{L}(\alpha) = L\},\\
&\mathrm{gsh}(L) = \min\{\mathrm{sh}(\alpha) \mid \alpha \text{ — обобщенно регулярное выражение такое, что } \mathcal{L}(\alpha) = L\}.
\end{aligned}$$

Число $\mathrm{sh}(L)$ называется *звездной высотой* языка L, а $\mathrm{gsh}(L)$ — *обобщенно звездной высотой* языка L. Нетрудно видеть, что $\mathrm{gsh}(L) \leqslant \mathrm{sh}(L)$.

Пример. Найдем звездную и обобщенно звездную высоту языка $L = \{w \in \{0,1\}^* \mid w$ содержит подслово $11\}$. Сначала заметим, что конкатенация и объединение конечных языков являются конечными языками. Поэтому, в силу бесконечности языка L, заключаем, что любое регулярное выражение, задающее L, должно содержать звездочку Клини. Отсюда следует, что $\mathrm{sh}(L) \geqslant 1$. С другой стороны, язык L можно задать регулярным выражением $(0 \cup 1)^*11(0 \cup 1)^*$, звездная высота которого равна 1. Таким образом, $\mathrm{sh}(L) = 1$.

Обобщенно звездная высота $\mathrm{gsh}(L) = 0$, поскольку L можно задать обобщенно регулярным выражением $\overline{\varnothing}11\overline{\varnothing}$, не содержащим звездочек Клини.

Замечание. Известно, что для любого $n \in \omega$ существует регулярный язык L такой, что его звездная высота $\mathrm{sh}(L) = n$. Однако вопрос о существовании примеров языков L таких, что $\mathrm{gsh}(L) > 1$, до сих пор является нерешенным.

§ 8. Определение формальных грамматик

Часто формальные языки описываются как совокупности слов, полученных с помощью правил замены одних цепочек символов на другие. Такой подход лежит в основе понятия формальной грамматики. В отличие от конечных автоматов, формальные грамматики не распознают слова, а порождают их. Любую формальную грамматику можно представлять как некоторый генератор языка. Запуск такого генератора производится с помощью «стартового сигнала», после чего начинается порождение слов языка. Действия генератора на каждом этапе порождения слов недетерминированны, но ограничены конечным списком правил. Время от времени этот процесс прерывается для того, чтобы выдать очередное выходное слово. Множество всех слов, появившихся на выходе в процессе работы, образует язык, порожденный данной грамматикой. Процесс порождения слов по правилам формальной грамматики безусловно является алгоритмическим.

Пример. Термин «формальная грамматика» неслучайно имеет такое лингвистическое происхождение. Процесс порождения слов в латинском языке может служить примером использования некоторой формальной грамматики.

Рассмотрим определенный фрагмент латинского языка, который неформально описывается следующими правилами словообразования:

1) Любое слово имеет приставку, корень, суффикс и окончание, причем они расположены в слове именно в указанном здесь порядке.

2) Приставка либо отсутствует, либо присутствует в единственном числе и является элементом множества {`ad, de, dis, in, prae, suc, ...`}.

3) В слове может быть либо один, либо два корня из множества {`albi, capill, cub, fect, it, laborat, ...`}.

4) Суффикс либо отсутствует, либо присутствует в единственном числе и является элементом множества {`bu, re, ūr, ...`}.

5) В слове может быть только одно окончание из множества {`ā, o, ōrum, nt, s, us, ...`}.

Теперь перепишем правила (1)–(5) формально, используя так называемые продукции:

1) (слово)$\longrightarrow$(приставка)(корень)(суффикс)(окончание)

2) (приставка)$\longrightarrow\Lambda$ (приставка)$\longrightarrow$`in`
(приставка)$\longrightarrow$`ad` (приставка)$\longrightarrow$`prae`
(приставка)$\longrightarrow$`de` (приставка)$\longrightarrow$`suc`
(приставка)$\longrightarrow$`dis` $\cdots$

3) (корень)$\longrightarrow$(корень1)(корень2) (корень2)$\longrightarrow\Lambda$
(корень1)$\longrightarrow$`albi` (корень2)$\longrightarrow$`albi`
(корень1)$\longrightarrow$`capill` (корень2)$\longrightarrow$`capill`
(корень1)$\longrightarrow$`cub` (корень2)$\longrightarrow$`cub`
(корень1)$\longrightarrow$`fect` (корень2)$\longrightarrow$`fect`
$\cdots$ $\cdots$

4) (суффикс)$\longrightarrow\Lambda$ (суффикс)$\longrightarrow$`re`
(суффикс)$\longrightarrow$`bu` (суффикс)$\longrightarrow$`ūr`
$\cdots$ $\cdots$

5) (окончание)$\longrightarrow$`ā` (окончание)$\longrightarrow$`o`
(окончание)$\longrightarrow$`ōrum` (окончание)$\longrightarrow$`nt`
(окончание)$\longrightarrow$`s` (окончание)$\longrightarrow$`us`
... ...

Например, процесс порождения слова `albicapillus` (*седовласый*) по данным правилам выглядит следующим образом:

(слово) $\to$ (приставка)(корень)(суффикс)(окончание) $\to$
(корень)(суффикс)(окончание) $\to$ (корень1)(корень2)(суффикс)(окончание) $\to$
`albi`(корень2)(суффикс)(окончание) $\to$ `albicapill`(суффикс)(окончание) $\to$
`albicapill`(окончание) $\to$ `albicapillus`.

Слово `aditūrus` (*намеревающийся подойти*) порождается так:

(слово) $\to$ (приставка)(корень)(суффикс)(окончание) $\to$
`ad`(корень)(суффикс)(окончание) $\to$ `ad`(корень1)(корень2)(суффикс)(окончание) $\to$
`adit`(корень2)(суффикс)(окончание) $\to$ `adit`(суффикс)(окончание) $\to$
`aditūr`(окончание) $\to$ `aditūrus`.

Слово `incubus` (*инкуб*) порождается следующим образом:

(слово) $\to$ (приставка)(корень)(суффикс)(окончание) $\to$
`in`(корень)(суффикс)(окончание) $\to$ `in`(корень1)(корень2)(суффикс)(окончание) $\to$
`incub`(корень2)(суффикс)(окончание) $\to$ `incub`(суффикс)(окончание) $\to$
`incub`(окончание) $\to$ `incubus`.

Слово `succubus` (*суккуб*) имеет схожую цепочку порождения:

(слово) $\to$ (приставка)(корень)(суффикс)(окончание) $\to$
`suc`(корень)(суффикс)(окончание) $\to$
`suc`(корень1)(корень2)(суффикс)(окончание) $\to$
`succub`(корень2)(суффикс)(окончание) $\to$ `succub`(суффикс)(окончание) $\to$
`succub`(окончание) $\to$ `succubus`.

Определение. *Формальной грамматикой* называется упорядоченная четверка $\Gamma = \langle V, T, P, S\rangle$, состоящая из следующих объектов:

а) V — конечное множество *нетерминальных (вспомогательных)* символов;

б) T — конечное множество *терминальных* символов такое, что $V \cap T = \varnothing$;

в) P — конечное множество *продукций*, т. е. цепочек вида $\alpha \longrightarrow \beta$, где $\alpha, \beta \in (V \cup T)^*$, и в α содержится хотя бы один элемент из V;

г) S — *начальный символ*, $S \in V$.

Определение. Пусть $\Gamma = \langle V, T, P, S\rangle$ — формальная грамматика, $\alpha, \beta \in (V \cup T)^*$. Говорят, что *слово β непосредственно получается из слова α в грамматике* Γ, и пишут $\alpha \underset{\Gamma}{\longrightarrow} \beta$, если существуют слова α_0, α_1, γ, δ такие, что $\alpha = \alpha_0\gamma\alpha_1$, $\beta = \alpha_0\delta\alpha_1$, и продукция $\gamma \longrightarrow \delta$ принадлежит множеству P.

Определение. Говорят, что *слово β выводимо из слова α в грамматике* Γ, и пишут $\alpha \xrightarrow[\Gamma]{*} \beta$, если существует конечная последовательность слов $\beta_0, \ldots, \beta_k$ такая, что $\beta_k = \beta$ и имеет место

$$\alpha \underset{\Gamma}{\longrightarrow} \beta_0 \underset{\Gamma}{\longrightarrow} \beta_1 \underset{\Gamma}{\longrightarrow} \ldots \underset{\Gamma}{\longrightarrow} \beta_k. \qquad (*)$$

Цепочка $(*)$ называется *выводом слова β из слова α в грамматике* Γ.

Определение. Пусть $\Gamma = \langle V, T, P, S\rangle$ — формальная грамматика. Язык $L(\Gamma) = \{w \in T^* \mid S \xrightarrow[\Gamma]{*} w\}$ называется *языком, порожденным грамматикой* Γ.

Пример. Вернемся к предыдущему примеру. С формальной точки зрения в данной грамматике множество нетерминальных символов имеет вид $V = \{$ (слово), (приставка), (корень), (корень1), (корень2), (суффикс), (окончание) $\}$, множеством терминальных символов является множество $T = \{$ `ad, de, dis, in, prae, suc, albi, capill, cub, bu, ūr, ā, nt, o,` $\ldots\}$. Продукциями являются цепочки из пунктов (1)–(5). Начальный символ — это символ (слово).

Пример. Рассмотрим теперь пример формальной грамматики, которая порождает язык всех десятичных записей натуральных чисел в алфавите из десяти терминальных цифр $T = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Каждая запись натурального числа есть либо 0, либо получается приписыванием ненулевой цифры слева к некоторой последовательности цифр. В свою очередь каждая последовательность цифр есть либо пустое слово Λ, либо получается приписыванием любой цифры (включая 0) слева к некоторому слову, про которое уже известно, что оно является последовательностью цифр. Эти рассуждения позволяют формально выписать множество продукций P:

$$\begin{array}{ll} S \longrightarrow 0 & A \longrightarrow \Lambda \\ S \longrightarrow 1A & A \longrightarrow 0A \\ S \longrightarrow 2A & A \longrightarrow 1A \\ \ldots & \ldots \\ S \longrightarrow 9A & A \longrightarrow 9A \end{array}$$

Таким образом, алфавит нетерминальных символов $V = \{S, A\}$ состоит из двух букв. Символ S — начальный, он соответствует термину «запись натурального числа». Символ A соответствует термину «последовательность цифр».

Определение (типы грамматик)**.** Формальная грамматика $\Gamma = \langle V, T, P, S\rangle$ называется:

а) *неукорачивающей*, если для любой ее продукции $\alpha \to \beta$ выполняется $|\alpha| \leqslant |\beta|$;

б) *контекстно-свободной*, если все ее продукции имеют вид $A \longrightarrow \beta$, где $A \in V$, $\beta \in (V \cup T)^*$, $|\beta| \geqslant 1$;

в) *регулярной*, если все ее продукции имеют вид $A \longrightarrow aB$ или $A \longrightarrow a$, где $A, B \in V$, $a \in T$.

Замечание. Любая регулярная грамматика является контекстно-свободной. Любая контекстно-свободная грамматика является неукорачивающей. Регулярные грамматики также иногда называют *праволинейными*, поскольку при порождении слов в таких грамматиках каждый новый терминальный символ появляется справа от предыдущего, т. е. слово порождается слева направо.

§ 9. Свойства формальных грамматик

Как было замечено выше, в регулярных грамматиках слова порождаются слева направо без возможности возврата в начальные символы слова. Иначе говоря, если на промежуточном шаге было выведено слово $a_1a_2\dots a_kA$, где $a_1,\dots,a_k$ — терминальные символы, A — нетерминальный символ, то на всех последующих шагах префикс $a_1a_2\dots a_k$ останется неизменным. Похожим свойством обладают конечные автоматы, которые не имеют памяти вне процессора и в которых символы слов также прочитываются слева направо. Это наблюдение лежит в основе следующей теоремы.

Теорема 11. *Имеют место следующие два утверждения:*

1) Если язык L порождается регулярной грамматикой, то L — автоматный.

2) Если L — автоматный язык, то $L\setminus\{\Lambda\}$ порождается регулярной грамматикой.

Доказательство. Сначала докажем первое утверждение теоремы. Пусть язык L порождается регулярной грамматикой $\Gamma=\langle V,T,P,S\rangle$. Определим недетерминированный конечный автомат $\mathfrak{A}$ следующим образом:

а) Множеством состояний автомата является множество $V\cup\{q\}$, где q — новый символ, т. е. $q\notin V$.

б) Внешним алфавитом автомата будет множество T.

в) Начальным состоянием является S.

г) Единственным выделенным состоянием будет q.

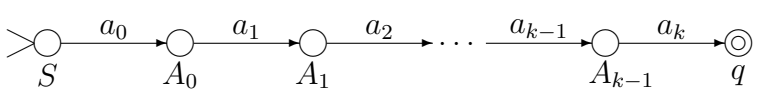
д) Переходы в автомате $\mathfrak{A}$ определяются по следующему принципу. Для каждой продукции вида $A\longrightarrow aB$ добавляем дугу $\underset{A}{\bigcirc}\xrightarrow{a}\underset{B}{\bigcirc}$. Для каждой продукции $A\longrightarrow a$ добавляем дугу $\underset{A}{\bigcirc}\xrightarrow{a}\underset{q}{\circledcirc}$.

Покажем, что $L=T(\mathfrak{A})$. Пусть $w=a_0a_1\dots a_k\in T^*$ — произвольное слово. Тогда имеет место следующая цепочка эквивалентных утверждений:

$w\in L\Longleftrightarrow$ Существует вывод $S\underset{\Gamma}{\longrightarrow}a_0A_0\underset{\Gamma}{\longrightarrow}a_0a_1A_1\underset{\Gamma}{\longrightarrow}\dots\underset{\Gamma}{\longrightarrow}a_0\dots a_{k-1}A_{k-1}\underset{\Gamma}{\longrightarrow}a_0\dots a_{k-1}a_k$ слова w в грамматике Γ. $\Longleftrightarrow$ В множестве P имеются продукции вида

$$\begin{aligned}S&\longrightarrow a_0A_0\\A_0&\longrightarrow a_1A_1\\\dots\quad&\quad\dots\quad\dots\\A_{k-2}&\longrightarrow a_{k-1}A_{k-1}\\A_{k-1}&\longrightarrow a_k\end{aligned}$$

$\Longleftrightarrow$ В построенном автомате $\mathfrak{A}$ имеется путь вида



$\Longleftrightarrow w = a_0 a_1 \dots a_k \in T(\mathfrak{A})$. Что и требовалось доказать.

Теперь докажем второе утверждение теоремы. Пусть L распознается конечным автоматом $\mathfrak{A}$. В силу леммы о вахтере можно считать, что $\mathfrak{A}$ обладает свойством вахтера. Преобразуем автомат $\mathfrak{A}$ в автомат $\mathfrak{A}'$, сделав начальное состояние невыделенным, если оно было выделенным в $\mathfrak{A}$. Ясно, что $L \setminus \{\Lambda\} = T(\mathfrak{A}')$. Пусть $\mathfrak{A}' = \langle Q, \mathcal{A}, \delta, q_0, F\rangle$. Определим регулярную грамматику Γ следующим образом:

а) Множеством нетерминальных символов грамматики является множество Q.

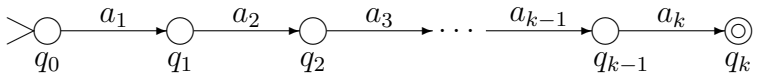
б) Множеством терминальных символов будет множество $\mathcal{A}$.

в) Начальным символом является q_0.

г) Множество продукций определяется по следующему принципу. Для каждой дуги автомата вида $\underset{q_1}{\bigcirc} \xrightarrow{a} \underset{q_2}{\bigcirc}$ добавляем продукцию $q_1 \longrightarrow aq_2$. Кроме этого, если состояние q_2 является выделенным, то добавляем дополнительно продукцию $q_1 \longrightarrow a$.

Покажем, что $L \setminus \{\Lambda\} = L(\Gamma)$. Ясно, что пустое слово не принадлежит обеим частям этого тождества. Пусть $w = a_1 \dots a_k \in \mathcal{A}^*$ — произвольное непустое слово. Тогда имеет место следующая цепочка эквивалентных утверждений:

$w \in L \Longleftrightarrow$ В автомате $\mathfrak{A}'$ существует путь



вдоль которого распознается слово w. $\Longleftrightarrow$ В построенной грамматике Γ имеются продукции

$$\begin{aligned} q_0 &\longrightarrow a_1 q_1 \\ q_1 &\longrightarrow a_2 q_2 \\ \dots \quad &\dots \quad \dots \\ q_{k-2} &\longrightarrow a_{k-1} q_{k-1} \\ q_{k-1} &\longrightarrow a_k \end{aligned}$$

$\Longleftrightarrow$ Существует вывод $q_0 \underset{\Gamma}{\longrightarrow} a_1 q_1 \underset{\Gamma}{\longrightarrow} a_1 a_2 q_2 \underset{\Gamma}{\longrightarrow} \dots \underset{\Gamma}{\longrightarrow} a_1 \dots a_{k-1} q_{k-1} \underset{\Gamma}{\longrightarrow} a_1 \dots a_{k-1} a_k$ слова w в грамматике Γ. $\Longleftrightarrow w \in L(\Gamma)$. Что и требовалось показать. □

Замечание. Предыдущая теорема позволяет утверждать, что класс автоматных языков *почти* совпадает с классом языков, порождаемых регулярными грамматиками. Иногда в литературе дается несколько другое определение регулярной грамматики, в котором допускается возможность вывода пустого слова. Это делается для того, чтобы избавиться от слова *почти* в предыдущем утверждении. При таком расширенном определении класс автоматных языков в точности равен классу языков, порождаемых регулярными грамматиками.

Теперь мы докажем важное свойство разрешимости любого языка, порождаемого неукорачивающей грамматикой. Под *разрешимостью языка* L над алфавитом $\mathcal{A}$ мы понимаем существование алгоритма, который по любому слову $w \in \mathcal{A}^*$ за конечное число шагов позволяет определить, принадлежит ли w языку L или нет. Ниже мы

приведем лишь неформальное описание разрешающей процедуры для языка, порожденного неукорачивающей грамматикой. В наших рассуждениях мы будем опираться на интуитивную эффективность таких процедур, как непосредственное получение одного слова из другого по правилам заданной формальной грамматики; проверка принадлежности заданного слова конечному множеству, имеющему эффективное описание; и др.

В формулировке следующего утверждения через $|V \cup T|$ мы обозначаем количество элементов в конечном множестве $V \cup T$.

Предложение 12. *Пусть $\Gamma = \langle V, T, P, S\rangle$ — неукорачивающая грамматика. Если $\alpha \xrightarrow[\Gamma]{*} \beta$, то в Γ существует вывод $\alpha \xrightarrow[\Gamma]{} \ldots \xrightarrow[\Gamma]{} \beta$, содержащий не более $|\beta|\cdot|V\cup T|^{|\beta|}$ слов.*

Доказательство. Пусть имеется вывод $\alpha = \alpha_0 \xrightarrow[\Gamma]{} \alpha_1 \xrightarrow[\Gamma]{} \ldots \xrightarrow[\Gamma]{} \alpha_n = \beta$. Так как Γ — неукорачивающая, то $1 \leqslant |\alpha_0| \leqslant |\alpha_1| \leqslant \ldots \leqslant |\alpha_n|$. Разобъем данный вывод на несколько участков, каждый из которых состоит из слов одинаковой длины:

$$\underbrace{\alpha_0 \longrightarrow \ldots \longrightarrow \alpha_{i_0}}_{\text{слова длины } |\alpha_0|} \longrightarrow \underbrace{\alpha_{i_0+1} \longrightarrow \ldots \longrightarrow \alpha_{i_1}}_{\text{слова длины } |\alpha_0|+1} \longrightarrow \ldots \longrightarrow \underbrace{\alpha_{i_{k-1}+1} \longrightarrow \ldots \longrightarrow \alpha_{i_k}}_{\text{слова длины } |\alpha_n|},$$

где $k = |\alpha_n| - |\alpha_0|$ и $i_k = n$.

Рассмотрим произвольный участок, состоящий из всех слов длины m. Заметим, что количество слов длины m над алфавитом $V \cup T$ равно $|V \cup T|^m$. Поэтому, если рассматриваемый участок содержит более, чем $|V \cup T|^m$ элементов, то среди них найдутся одинаковые слова $\alpha_i = \alpha_j$, $i < j$. Следовательно, вывод можно укоротить, удалив участок $\alpha_i \longrightarrow \ldots \longrightarrow \alpha_j$. Повторив, если потребуется, аналогичные рассуждения несколько раз, мы добьемся того, что слов длины m в нашем выводе окажется не более, чем $|V \cup T|^m$ штук.

Применив описанную процедуру ко всем m, мы получим новый вывод слова β из α, в котором суммарное количество слов не превосходит

$$\begin{aligned} &|V \cup T|^{|\alpha_0|} + |V \cup T|^{|\alpha_0|+1} + \ldots + |V \cup T|^{|\alpha_n|} \leqslant \\ &\leqslant \underbrace{|V \cup T|^{|\alpha_n|} + |V \cup T|^{|\alpha_n|} + \ldots + |V \cup T|^{|\alpha_n|}}_{|\alpha_n|-|\alpha_0|+1} = \\ &= (|\alpha_n| - |\alpha_0| + 1) \cdot |V \cup T|^{|\alpha_n|} \leqslant |\alpha_n| \cdot |V \cup T|^{|\alpha_n|}. \end{aligned}$$

Что и требовалось доказать. □

Теорема 13. *Пусть $\Gamma = \langle V, T, P, S\rangle$ — неукорачивающая грамматика. Тогда язык $L(\Gamma)$ разрешим, т. е. существует алгоритм, который по любому слову $w \in T^*$ за конечное число шагов позволяет определить, порождается ли w в грамматике Γ или не порождается.*

Доказательство. Пусть w — произвольное слово в алфавите T. Алгоритм имеет следующее описание. Сначала вычисляем натуральное число $k = |w| \cdot |V \cup T|^{|w|}$. Затем строим конечную последовательность $A_1, A_2, \ldots, A_k$ конечных множеств слов по индукции:

$$A_1 = \{S\}, \quad A_{n+1} = A_n \cup \{v \in (V \cup T)^* \mid v \text{ непосредственно получается из некоторого } u \in A_n \text{ в грамматике } \Gamma\}.$$

Заметим, что для любого $1 \leqslant n \leqslant k$ множество A_n состоит из всех слов, имеющих вывод в грамматике Γ, длина которого не превосходит n. В силу предыдущего предложения имеет место эквивалентность $w \in L(\Gamma) \iff w \in A_k$. Следовательно, если условие $w \in A_k$ выполнено, то w порождается заданной грамматикой, в противном случае — не порождается. □

Следствие 14. *Если язык L принадлежит одному из следующих классов, то L разрешим:*

а) класс языков, порождаемых контекстно-свободными грамматиками;

б) класс языков, порождаемых регулярными грамматиками;

в) класс автоматных языков.

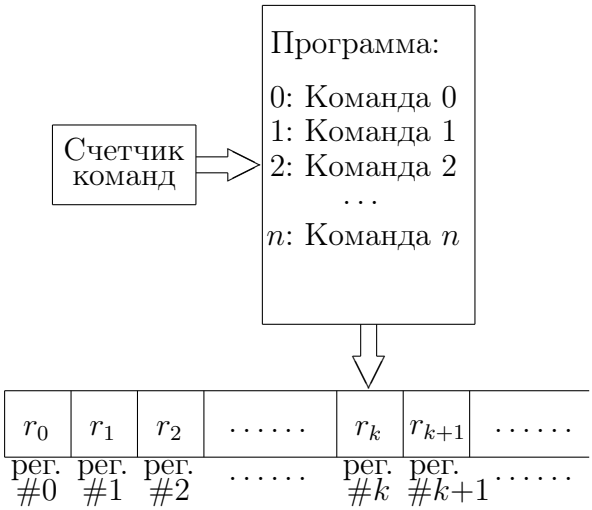
Доказательство. Разрешимость языков из пунктов (а) и (б) следует из теоремы 13. Если L — автоматный язык, не содержащий пустого слова, то его разрешимость следует из пункта (б) и теоремы 11. Если же L — автоматный и $\Lambda \in L$, то в силу пункта (б) и теоремы 11 язык $L_1 = L \setminus \{\Lambda\}$ разрешим. Следовательно, язык $L = L_1 \cup \{\Lambda\}$ также разрешим, поскольку он получен из L_1 добавлением пустого слова. □

Глава III

Формализации понятия вычислимой функции

Основной целью данной главы является формализация понятия вычислимой функции, действующей на натуральных числах. Напомним, что *n-местной частичной функцией на ω* мы называем любую функцию вида $f : X \to \omega$, где $X \subseteq \omega^n$, $n \in \omega$. Существует несколько различных подходов, приводящих к тому или иному определению частичной вычислимой функции. Мы рассмотрим четыре подобных подхода и покажем, что все они эквивалентны. Первая из формализаций, с которой мы познакомимся в следующем параграфе, связана с машинами Шёнфилда [8], [13].

§ 10. Машины Шёнфилда



Машина Шёнфилда — это механическое вычислительное устройство, хотя окончательная его формализация является совершенно математической. Тем не менее, мы ограничимся только «механическим» описанием.
Машина Шёнфилда однозначно задается:

1) Бесконечным множеством *регистров*, пронумерованных натуральными числами $0, 1, 2, \ldots$ Каждый регистр — это ячейка памяти, способная содержать любое натуральное число.

Содержимое регистров может меняться по ходу вычислений. Отметим, что каждая конкретная машина Шёнфилда использует в своих вычислениях только конечное число регистров. Основное назначение регистровой памяти — это хранение входных, промежуточных и выходных данных.

2) *Счетчиком команд*, являющимся особой ячейкой памяти, которая в каждый данный момент содержит некоторое натуральное число. Счетчик команд указывает на номер команды в программе, которая исполняется в данный момент. В начальный момент в счетчик команд заносится 0.

3) *Программой*, содержащейся в отдельной выделенной памяти машины. Программа — это конечный список команд, последовательно пронумерованных натуральными числами от 0 до некоторого $n \in \omega$. В ходе вычислений программа не меняется, она фиксирована.

Шаг машины состоит в исполнении той команды из программы, номер которой указан в данный момент в счетчике команд. Если команды с таким номером в программе нет, то машина останавливается.

Существует только два типа *команд:*

1) $\mathsf{INC}\ i$
   При исполнении данной команды машина увеличивает содержимое i-го регистра и счетчик команд на 1, после чего переходит к следующему шагу в своей работе.

2) $\mathsf{DEC}\ i, n$
   Если к началу исполнения данной команды содержимое i-го регистра больше 0, то машина уменьшает его на 1 и заносит n в счетчик команд. Если же в данный момент содержимое i-го регистра равно 0, то машина увеличивает счетчик команд на 1.

Еще раз подчеркнем, что машина останавливается тогда и только тогда, когда в счетчик команд заносится номер несуществующей в программе команды. Однако не исключены ситуации, в которых машина работает бесконечно, т. е. не останавливается.

Для наших дальнейших рассуждений будет удобно расширить язык машин Шёнфилда и ввести дополнительные команды, отличные от команд типа $\mathsf{INC}\ i$ и $\mathsf{DEC}\ i, n$. Чтобы вычислительные возможности исходных машин не изменились, мы будем добавлять только те команды, которые допускают реализацию в виде программы, использующей только команды типа $\mathsf{INC}\ i$ и $\mathsf{DEC}\ i, n$. Любую дополнительную команду, обладающую таким свойством, будем называть *макросом.*

Таким образом, любой макрос получается по следующему принципу: если P — обычная программа, то ей сопоставляется оператор P^*, который называется *макросом* и который может исполняться как отдельная команда. Программы, содержащие макросы, будем называть *макропрограммами.*

Исполнение макроса P^*, находящегося в макропрограмме в строке под номером m, происходит следующим образом:

1) Если в счетчике команд находится m, то в машину (в качестве подпрограммы) вызывается программа P.

2) P исполняется с теми данными в регистрах, которые там содержались в данный момент.

3) Если работа программы P закончилась, то в счетчик команд заносится $m + 1$ и исполнение макроса на этом заканчивается. Если же P работает бесконечно, то макропрограмма также не останавливается.

Любой макрос может использовать конечный набор параметров, каждый из которых является номером некоторого регистра (эти номера обычно присутствуют в имени макроса). Макросы не могут использовать в качестве своих параметров номера строк программы.

Пример. Исполнение последовательной пары команд $0 : \mathsf{INC}\ 0$ и $1 : \mathsf{DEC}\ 0, n$ просто заносит n в счетчик команд, при этом содержимые регистров не меняются. Однако мы не можем сопоставить этой короткой программе макрос, использующий параметр n, поскольку n здесь является номером строки.

Пример. Программа с простейшим циклом $0 : \mathsf{DEC}\ i, 0$ обнуляет содержимое i-го регистра. Обозначим макрос, соответствующий этой программе, через $\mathsf{ZERO}\ i$.

Пример. Пусть i, j, k — фиксированные натуральные числа такие, что $i \neq k$ и $j \neq k$. Обозначим через $[i] \to [j], (k)$ макрос, который копирует содержимое i-го регистра в j-й регистр, используя k-й регистр как вспомогательный. Содержимое i-го регистра не изменяется. Программа, реализующая данный макрос при $i \neq j$, имеет следующий вид:

$$\begin{aligned}
0 &: \ \mathsf{DEC}\ j, 0 \\
1 &: \ \mathsf{DEC}\ k, 1 \\
2 &: \ \mathsf{INC}\ 0 \\
3 &: \ \mathsf{DEC}\ 0, 6 \\
4 &: \ \mathsf{INC}\ j \\
5 &: \ \mathsf{INC}\ k \\
6 &: \ \mathsf{DEC}\ i, 4 \\
7 &: \ \mathsf{INC}\ 0 \\
8 &: \ \mathsf{DEC}\ 0, 10 \\
9 &: \ \mathsf{INC}\ i \\
10 &: \ \mathsf{DEC}\ k, 9
\end{aligned}$$

При $i = j$ можно взять любую программу, которая ничего не меняет, например

$$\begin{aligned}
0 &: \ \mathsf{INC}\ 0 \\
1 &: \ \mathsf{DEC}\ 0, 2
\end{aligned}$$

Определение. Макропрограммы P и Q называются *эквивалентными*, если запуски машин Шёнфилда с этими макропрограммами на одних и тех же начальных содержимых регистров либо приведут к остановке обеих машин с одинаковыми содержимыми всех регистров, либо обе машины никогда не остановятся.

Следующая теорема показывает, что введение макросов не расширяет класс алгоритмических задач, которые допускают решение на машинах Шёнфилда. Таким образом, макросы — это лишь удобный инструмент для сокращения текстов программ.

Теорема 15 (об элиминации макросов)**.** *Любая макропрограмма эквивалентна программе, не содержащей макросы.*

Доказательство. Опишем, как следует преобразовать произвольную макропрограмму Q в эквивалентную ей макропрограмму Q', содержащую на один макрос меньше, чем в Q. Повторяя эту процедуру необходимое число раз, мы в конце концов получим программу без макросов, эквивалентную исходной.

Пусть P^* — некоторый макрос в Q. Чтобы получить искомую макропрограмму Q', заменим вхождение P^* на список команд P, которые реализуют данный макрос, при этом необходимо проделать следующее:

а) заново перенумеровать все команды полученной макропрограммы, начиная с номера 0;

б) если DEC i, n — старая команда из Q или P, n — номер существующей команды C из Q или P соответственно, то заменить ее на DEC i, m, где m — новый номер команды C;

в) если DEC i, n — старая команда из Q, n — номер несуществующей в Q команды, то заменить ее на DEC i, m, где m больше, чем число команд новой макропрограммы;

г) если DEC i, n — старая команда из P, n — номер несуществующей в P команды, то заменить ее на DEC i, m, где m — номер команды, следующей после списка команд P в новой макропрограмме.

□

Пример. Обозначим через ADD [1] TO [0], (3) макрос, при исполнении которого содержимое 1-го регистра прибавляется к содержимому 0-го регистра. При этом содержимое 1-го регистра сохраняется, а 3-й регистр используется как вспомогательный. Программа, реализующая данный макрос, имеет следующий вид:

$0:$ DEC $3, 0$
$1:$ INC 0
$2:$ DEC $0, 5$
$3:$ INC 0
$4:$ INC 3
$5:$ DEC $1, 3$
$6:$ INC 0
$7:$ DEC $0, 9$
$8:$ INC 1
$9:$ DEC $3, 8$

Запишем теперь с помощью введенного макроса макропрограмму, перемножающую содержимые 1-го и 2-го регистров и записывающую полученное произведение в 0-й регистр:

$0:$ DEC $0, 0$
$1:$ INC 0
$2:$ DEC $0, 4$
$3:$ ADD [1] TO [0], (3)
$4:$ DEC $2, 3$

Если применить к данной макропрограмме процедуру преобразования, описанную в доказательстве теоремы об элиминации макросов, то получим эквивалентную ей программу без макросов:

$$
\begin{array}{rl}
0: & \mathsf{DEC}\ 0, 0 \\
1: & \mathsf{INC}\ 0 \\
2: & \mathsf{DEC}\ 0, 13 \\
\hline
3: & \mathsf{DEC}\ 3, 3 \\
4: & \mathsf{INC}\ 0 \\
5: & \mathsf{DEC}\ 0, 8 \\
6: & \mathsf{INC}\ 0 \\
7: & \mathsf{INC}\ 3 \\
8: & \mathsf{DEC}\ 1, 6 \\
9: & \mathsf{INC}\ 0 \\
10: & \mathsf{DEC}\ 0, 12 \\
11: & \mathsf{INC}\ 1 \\
12: & \mathsf{DEC}\ 3, 11 \\
\hline
13: & \mathsf{DEC}\ 2, 3
\end{array}
$$

Определение. k-местная частичная функция $f : M \subseteq \omega^k \longrightarrow \omega$ называется *вычислимой на машине Шёнфилда*, если существует машина Шёнфилда с программой P такая, что для любых $x_1, \ldots, x_k \in \omega$ выполняется:

а) если $f(x_1, \ldots, x_k)$ определено, то после запуска машины с начальными значениями $x_1, \ldots, x_k$ в регистрах $1, \ldots, k$ соответственно и с нулями в остальных регистрах машина после конечного числа шагов останавливается, и после остановки в 0-ом регистре находится $f(x_1, \ldots, x_k)$;

б) если $f(x_1, \ldots, x_k)$ не определено, то после запуска машины с начальными значениями $x_1, \ldots, x_k$ в регистрах $1, \ldots, k$ соответственно и с нулями в остальных регистрах машина никогда не останавливается и работает бесконечно.

Замечание. Если в определении выше f — нульместная функция, то входные данные $x_1, \ldots, x_k$ отсутствуют, а результат вычислений по программе P не зависит от начальных содержимых регистров.

Определение. Пусть f — k-местная частичная функция, вычислимая на машине Шёнфилда с программой P. Обозначим через

$$f([i_1], \ldots, [i_k]) \to [j], s$$

макрос, который вычисляет значение функции f от содержимых регистров с номерами $i_1, \ldots, i_k$, записывает его в j-й регистр и при этом не изменяет содержимое всех регистров до s-го включительно, кроме j-го регистра. Если упомянутое значение функции не определено, то исполнение данного макроса приводит к бесконечной работе программы.

Нам необходимо построить макропрограмму, реализующую введенный выше макрос $f([i_1], \ldots, [i_k]) \to [j], s$. Обозначим, как обычно, через P^* макрос, соответствующий программе P. Пусть m — произвольное число, превосходящее числа $i_1, \ldots, i_k, j, s$ и номера всех регистров, входящих в программу P. Отсюда, в частности, следует, что результат исполнения макроса P^* не зависит от содержимых регистров, номера которых больше, чем $m-1$. Тогда искомая макропрограмма имеет вид (номера строк опущены):

$[0] \to [m+1], (m)$ $\dots$ $[m-1] \to [m+m], (m)$	сохраняем содержимое регистров с 0-го до $(m-1)$-го в регистрах с номерами от $m+1$ до $2m$
$[m+1+i_1] \to [1], (m)$ $\dots$ $[m+1+i_k] \to [k], (m)$ ZERO 0 ZERO $k+1$ $\dots$ ZERO $m-1$	подготавливаем входные данные для исполнения макроса P^*
P^*	вычисляем необходимое значение функции f
$[m+2] \to [1], (m)$ $\dots$ $[m+m] \to [m-1], (m)$	восстанавливаем содержимые регистров с номерами от 1 до $m-1$
$[0] \to [j], (m)$	заносим результат вычислений в j-й регистр
$[m+1] \to [0], (m)$	если $j \neq 0$, то восстанавливаем содержимое 0-го регистра (если $j = 0$, то эта срока не включается в программу!)

В дальнейшем мы будем опускать параметр s, поскольку его всегда можно заранее выбрать достаточно большим, чтобы не влиять на работу внешней макропрограммы.

§ 11. Частично рекурсивные функции

В этом параграфе мы рассмотрим другой подход к формализации понятия вычислимой функции. Его можно назвать алгебраическим, поскольку определяемый здесь класс функций будет порождаться из некоторых простейших функций с помощью определенных операций. Совокупность функций, возникающих в результате подобного алгебраического порождения, называется классом частично рекурсивных функций.

В рамках данного параграфа рассматриваются только частичные функции на ω, т. е. всевозможные функции вида $f : X \to \omega$, где множество $X \subseteq \omega^n$ является областью определения функции, а число $n \in \omega$ — ее местностью. Любую 0-местную всюду определенную функцию $f = a$ мы отождествляем с константой $a \in \omega$. Нигде не определенная функция единственна и имеет вид $f = \varnothing$, причем любое натуральное число является местностью нигде не определенной функции.

Если f — n-местная, а g — m-местная частичная функция, то для любых значений $x_1, \dots, x_n, y_1, \dots, y_m \in \omega$ мы пишем $f(x_1, \dots, x_n) = g(y_1, \dots, y_m)$ тогда и только тогда, когда либо эти значения одновременно не определены, либо они оба определены и совпадают.

Определение. *Простейшими* функциями называются нульместная функция 0, всюду определенные одноместные функции $o(x) = 0$, $s(x) = x + 1$ и n-местные функции $I^n_m(x_1, \dots, x_n) = x_m$ для всех m, n таких, что $1 \leqslant m \leqslant n$.

Определение. Говорят, что функция $f(x_1, \dots, x_n)$ получается с помощью *оператора суперпозиции* S из функций $h(y_1, \dots, y_m), g_1(x_1, \dots, x_n), \dots, g_m(x_1, \dots, x_n)$, если для любых $x_1, \dots, x_n$ выполняется:

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

Замечание. С неформальной точки зрения оператор суперпозиции соответствует принципу последовательного запуска вычислительных устройств, когда результаты вычислений одних устройств служат входными данными для других. Если f получена с помощью суперпозиции из $h, g_1, \ldots, g_m$, то вычисление значения $f(x_1, \ldots, x_n)$ происходит следующим образом. Сначала на входных данных $\overline{x} = \langle x_1, \ldots, x_n \rangle$ вычисляются значения $y_1 = g_1(\overline{x}), \ldots, y_m = g_m(\overline{x})$. Если хотя бы одно из этих значений не определено, то значение $f(\overline{x})$ тоже не определено. Если же все $y_1, \ldots, y_m$ определены, то затем они подаются на вход функции h. Если выходное значение $h(y_1, \ldots, y_m)$ не определено, то $f(\overline{x})$ считается неопределенным, иначе $f(\overline{x}) = h(y_1, \ldots, y_m)$ определено и является окончательным выходным значением.

Определение. Говорят, что функция $f(x_1, \ldots, x_n, y)$ получается с помощью *оператора примитивной рекурсии* R из функций $g(x_1, \ldots, x_n)$ и $h(x_1, \ldots, x_n, y, z)$, если для любых $x_1, \ldots, x_n, y$ выполняется схема примитивной рекурсии:

$$\begin{cases} f(x_1, \ldots, x_n, 0) = g(x_1, \ldots, x_n) \\ f(x_1, \ldots, x_n, y+1) = h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y)) \end{cases}$$

Замечание. Оператор примитивной рекурсии формализует циклическую структуру, вычисляющую функции, заданные рекуррентными соотношениями (или по индукции). Если f получена с помощью примитивной рекурсии из g и h, то вычисление значения $f(\overline{x}, y)$ происходит следующим образом. Сначала на входных данных $\overline{x}$ вычисляется значение $g(\overline{x})$, которое совпадает с $f(\overline{x}, 0)$. Если это значение не определено, то $f(\overline{x}, y)$ тоже не определено, иначе, подав $\overline{x}$, 0 и $f(\overline{x}, 0)$ на вход функции h, мы вычисляем $h(\overline{x}, 0, f(\overline{x}, 0))$, которое совпадает с $f(\overline{x}, 1)$. Если последнее значение не определено, то $f(\overline{x}, y)$ не определено, иначе мы подаем $\overline{x}$, 1 и $f(\overline{x}, 1)$ на вход функции h, и так далее. Данные вычисления продолжаются до тех пор, пока мы не достигнем значения $f(\overline{x}, y)$. Если при этом на промежуточных шагах хоты бы одно вычисляемое значение функции h не определено, то $f(\overline{x}, y)$ считается неопределенным.

Определение. Говорят, что функция $f(x_1, \ldots, x_n)$ получается с помощью *оператора минимизации* M из функции $g(x_1, \ldots, x_n, y)$ и обозначается $f(x_1, \ldots, x_n) = \mu y[g(x_1, \ldots, x_n, y) = 0]$, если для любых $x_1, \ldots, x_n$ выполняется:

$$f(x_1, \ldots, x_n) = \begin{cases} y, & \text{если } g(x_1, \ldots, x_n, 0), \ldots, g(x_1, \ldots, x_n, y-1) \\ & \text{определены и не равны 0, а значение} \\ & g(x_1, \ldots, x_n, y) \text{ определено и равно 0,} \\ \text{не определено} & \text{в противном случае.} \end{cases}$$

Замечание. Оператор минимизации формально описывает такой вычислительный прием, как эффективный перебор: последовательно перебирая числа из натурального ряда, мы проверяем, удовлетворяет ли текущее число фиксированному эффективному условию. Эффективное условие записывается в виде уравнения $g(\overline{x}, y) = 0$. Вычисление значения $f(\overline{x})$ происходит следующим образом. Сначала на входных данных $\overline{x}, 0$ вычисляется значение $g(\overline{x}, 0)$. Если это значение не определено, то $f(\overline{x})$ тоже не определено. Иначе если $g(\overline{x}, 0) = 0$, то вычисления заканчиваются – число 0 будет подано на выход. Если же $g(\overline{x}, 0) \neq 0$, то на входных данных $\overline{x}, 1$ вычисляется значение $g(\overline{x}, 1)$. Если это значение не определено, то $f(\overline{x})$ тоже не определено. Иначе если $g(\overline{x}, 1) = 0$, то вычисления заканчиваются – число 1 будет подано на выход. Если же $g(\overline{x}, 1) \neq 0$, то данный процесс продолжается. В результате мы либо вычислим

наименьшее решение уравнения $g(\overline{x}, y) = 0$, либо ничего не вычислим. Последнее возможно по двум причинам: либо некоторое вычисляемое значение функции g окажется неопределенным, либо $g(\overline{x}, y)$ определено для всех y, но не равно нулю.

Определение. Частичная функция $f(x_1, \ldots, x_n)$ называется *частично рекурсивной (примитивно рекурсивной)*, если существует конечная последовательность функций $f_0, \ldots, f_m = f$ такая, что для любого $i \leqslant m$ функция f_i либо простейшая, либо получается из некоторых предыдущих с помощью одного из операторов S, R или M (операторов S или R).

Определение. Всюду определенные частично рекурсивные функции называтся *рекурсивными.*

Замечание. Будем использовать сокращения: ч.р.ф. — для частично рекурсивных функций, п.р.ф. — для примитивно рекурсивных функций, р.ф. — для рекурсивных функций.

Из определения следует, что если функция f получается из некоторых ч.р.ф. (п.р.ф.) с помощью оператора S, R или M (S или R), то f тоже является ч.р.ф. (п.р.ф.).

Кроме этого, очевидно, между введенными классами функций имеют место следующие теоретико-множественные включения:

$$\text{ПРФ} \subseteq \text{РФ} \subseteq \text{ЧРФ}.$$

Замечание. Заметим, что одноместную функцию $o(x)$ можно исключить из числа простейших функций — класс частично рекурсивных функций при этом не изменится. Это следует из того, что $o(x)$ можно получить из 0-местной функции 0 и 2-местной функции $I_2^2(x, y)$ с помощью оператора примитивной рекурсии.

Теорема 16 (о вычислимости ч.р.ф. на машинах Шёнфилда)**.** *Любая частично рекурсивная функция является вычислимой на машине Шёнфилда.*

Доказательство. Пусть f — ч.р.ф. Следовательно, по определению существует конечная последовательность функций $f_0, \ldots, f_k = f$ такая, что для любого $i \leqslant k$ функция f_i либо простейшая, либо получается из некоторых предыдущих с помощью оператора S, R или M. Индукцией по числу $k \in \omega$ докажем, что f вычислима на некоторой машине Шёнфилда.

Если $k = 0$, то f является простейшей, т. е. либо константой 0, либо функцией $o(x)$, либо функцией $s(x)$, либо функцией вида $I_m^n(x_1, \ldots, x_n)$. Следующие три макропрограммы вычисляют эти функции на машинах Шёнфилда (0 и $o(x)$ вычисляются одной программой):

0 и $o(x)$:	0:ZERO 0	$s(x)$:	0:INC 1 1:[1] → [0]	$I_m^n(\overline{x})$:	0:[m] → [0]

Пусть $k > 0$. Тогда f получена из некоторых ч.р.ф. с помощью одного из трех операторов. Рассмотрим соответствующие три случая.

Если $f(x_1, \ldots, x_n)$ получена с помощью оператора суперпозиции из $h(y_1, \ldots, y_m)$, $g_1(x_1, \ldots, x_n)$, $\ldots$, $g_m(x_1, \ldots, x_n)$, то в силу индукционного предположения функции $h, g_1, \ldots, g_m$ вычислимы по Шёнфилду. Тогда следующая макропрограмма вычисляет функцию f:

$$\begin{array}{rl} 0: & g_1([1],\ldots,[n]) \to [n+1] \\ \ldots & \ldots \\ m{-}1: & g_m([1],\ldots,[n]) \to [n+m] \\ m: & h([n+1],\ldots,[n+m]) \to [0]. \end{array}$$

Если $f(x_1,\ldots,x_n,y)$ получена с помощью оператора примитивной рекурсии из частичных функций $g(x_1,\ldots,x_n)$ и $h(x_1,\ldots,x_n,y,z)$, которые мы считаем вычислимыми по Шёнфилду, то следующая макропрограмма будет вычислять f:

$$\begin{array}{rl} 0: & g([1],\ldots,[n]) \to [0] \\ 1: & [n+1] \to [n+2] \\ 2: & \mathsf{ZERO}\ n+1 \\ 3: & \mathsf{INC}\ 0 \\ 4: & \mathsf{DEC}\ 0,8 \\ 5: & h([1],\ldots,[n],[n+1],[0]) \to [n+3] \\ 6: & [n+3] \to [0] \\ 7: & \mathsf{INC}\ n+1 \\ 8: & \mathsf{DEC}\ n+2,5 \end{array}$$

Если функция $f(x_1,\ldots,x_n)$ получена с помощью оператора минимизации из вычислимой по Шёнфилду функции $g(x_1,\ldots,x_n,y)$, то следующая макропрограмма будет вычислять f:

$$\begin{array}{rl} 0: & \mathsf{INC}\ 0 \\ 1: & \mathsf{DEC}\ 0,3 \\ 2: & \mathsf{INC}\ 0 \\ 3: & g([1],\ldots,[n],[0]) \to [n+1] \\ 4: & \mathsf{DEC}\ n+1,2 \end{array}$$

Теорема доказана. □

§ 12. Рекурсивность некоторых функций и отношений

Нашей дальнейшей целью является доказательство утверждения, обратного теореме 16. Для этого нам потребуется целая серия предварительных фактов и новых понятий. Всюду далее через $\overline{x}$ мы обозначаем кортеж $\langle x_1,\ldots,x_n\rangle$, при $n=0$ этот кортеж считается пустым.

Лемма 17. *Все нульместные всюду определенные функции являются примитивно рекурсивными.*

Доказательство. Пусть $a \in \omega$ — произвольная константа. Возможны два случая. Если $a = 0$, то это по определению простейшая функция, и значит она является п.р.ф. Если же $a > 0$, то $a = \underbrace{s(s\ldots s}_{a}(0)\ldots)$, т. е. a получена из простейших функций 0 и $s(x)$ с помощью a-кратного применения оператора S. □

Лемма 18. *Следующие функции являются примитивно рекурсивными:*

а) $f(x,y) = x+y$;

б) $f(x,y) = x\cdot y$;

в) $f(x,y)=x^y$ (здесь $0^0=1$);

г) $\operatorname{sg}(x)=\begin{cases}0, & x=0;\\ 1, & x>0;\end{cases}$

д) $\overline{\operatorname{sg}}(x)=\begin{cases}1, & x=0;\\ 0, & x>0;\end{cases}$

е) $f(x)=x\dot{-}1=\begin{cases}0, & x=0;\\ x-1, & x>0;\end{cases}$

ж) $f(x,y)=x\dot{-}y=\begin{cases}0, & x\leqslant y;\\ x-y, & x>y;\end{cases}$

з) $f(x,y)=|x-y|$

Доказательство. а) Для функции $f(x,y)=x+y$ имеет место следующая схема примитивной рекурсии:

$$\begin{cases}f(x,0)=x,\\ f(x,y+1)=(x+y)+1=s(f(x,y))=s(I_3^3(x,y,f(x,y))),\end{cases}$$

т. е. $f(x,y)$ получена с помощью оператора R из п.р.ф. $g(x)=x=I_1^1(x)$ и п.р.ф. $h(x,y,z)=s(I_3^3(x,y,z))$, которая, в свою очередь, получена из простейших функций s и I_3^3 с помощью оператора S.

б) Для функции $f(x,y)=x\cdot y$ имеет место следующая схема примитивной рекурсии:

$$\begin{cases}f(x,0)=0,\\ f(x,y+1)=xy+x=\varphi(xy,x)=\varphi(I_3^3(x,y,f(x,y)),I_1^3(x,y,f(x,y))),\end{cases}$$

где $\varphi(u,v)=u+v$ — п.р.ф. из предыдущего пункта, т. е. $f(x,y)$ получена с помощью оператора R из п.р.ф. $g(x)=0=o(x)$ и п.р.ф. $h(x,y,z)=\varphi(I_3^3(x,y,z),I_1^3(x,y,z))$, которая, в свою очередь, получена из п.р.ф. φ, I_3^3, I_1^3 с помощью оператора S.

в) Доказывается аналогично путем выписывания соответствующей схемы примитивной рекурсии.

г) Выписываем схему примитивной рекурсии:

$$\begin{cases}\operatorname{sg}(0)=0,\\ \operatorname{sg}(x+1)=1=s(0)=s(o(I_1^2(x,\operatorname{sg}(x)))).\end{cases}$$

Другими словами, $\operatorname{sg}(x)$ получена с помощью оператора R из 0-местной функции 0, которая является простейшей, и 2-местной функции $s(o(I_1^2(x,y)))$, которая является п.р.ф., поскольку получена суперпозициями из простейших.

д) Выписываем схему примитивной рекурсии:

$$\begin{cases}\overline{\operatorname{sg}}(0)=1,\\ \overline{\operatorname{sg}}(x+1)=0=o(I_1^2(x,\overline{\operatorname{sg}}(x))).\end{cases}$$

Таким образом, $\overline{\operatorname{sg}}(x)$ получена с помощью оператора R из 0-местной функции 1, которая является п.р.ф. в силу предыдущей леммы, и 2-местной функции $o(I_1^2(x,y))$, которая является п.р.ф., так как получена суперпозициями из простейших функций.

е) Доказывается аналогично.

ж) Обозначим $f(x,y) = x \dot{-} y$. Тогда имеет место схема примитивной рекурсии:

$$\begin{cases} f(x,0) = x = I_1^1(x), \\ f(x,y+1) = x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1 = \psi(f(x,y)), \end{cases}$$

где $\psi(u) = u \dot{-} 1$ — п.р.ф. из предыдущего пункта. Выше мы воспользовались тождеством $x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1$, которое верно для любых $x, y \in \omega$.

з) Заметим, что $|x-y| = (x \dot{-} y) + (y \dot{-} x)$ — суперпозиция примитивно рекурсивных функций. □

Лемма 19. *Если функция $g(\overline{x}, y)$ является ч.р.ф. (п.р.ф.), то функции $f(\overline{x}, y) = \sum\limits_{i=0}^{y} g(\overline{x}, i)$ и $h(\overline{x}, y) = \prod\limits_{i=0}^{y} g(\overline{x}, i)$ тоже являются ч.р.ф. (п.р.ф.).*

Доказательство. Частичная (примитивная) рекурсивность функции f вытекает из пункта (а) леммы 18 и следующей схемы примитивной рекурсии:

$$\begin{cases} f(\overline{x}, 0) = g(\overline{x}, 0), \\ f(\overline{x}, y+1) = f(\overline{x}, y) + g(\overline{x}, y+1). \end{cases}$$

Утверждение для функции h доказывается аналогично с использованием пункта (б) леммы 18. □

Определение. Говорят, что функция $f(\overline{x})$ получается с помощью *ограниченной минимизации* из всюду определенных функций $g(\overline{x}, y)$ и $h(\overline{x})$, и обозначается $f(\overline{x}) = \mu y \leqslant h(\overline{x})[g(\overline{x}, y) = 0]$, если для любых значений $\overline{x}$ выполняется:

$$f(\overline{x}) = \begin{cases} y, & \text{если } g(\overline{x}, i) \neq 0 \text{ для всех } i < y, \text{ и } g(\overline{x}, y) = 0, \text{ и } y \leqslant h(\overline{x}), \\ h(\overline{x}) + 1, & \text{в противном случае.} \end{cases}$$

Предложение 20 (об ограниченной минимизации)**.** *Если функции g и h примитивно рекурсивны и функция f получена из g и h с помощью ограниченной минимизации, то f тоже примитивно рекурсивна.*

Доказательство. Следует из лемм 18, 19 и тождества $f(\overline{x}) = \sum\limits_{i=0}^{h(\overline{x})} \operatorname{sg}\Big(\prod\limits_{j=0}^{i} g(\overline{x}, j)\Big)$. □

Распространим понятие рекурсивности на класс всех отношений, заданных на ω. Для этого достаточно связать с каждым отношением некоторую частичную функцию, которая однозначно его задает, и назвать отношение рекурсивным, если таковой является данная функция.

Определение. Отношение (предикат) $R \subseteq \omega^n$ называется *рекурсивным (примитивно рекурсивным)*, если его характеристическая функция

$$\mathcal{X}_R(x_1, \ldots, x_n) = \begin{cases} 1, & \text{если } \langle x_1, \ldots, x_n \rangle \in R, \\ 0, & \text{если } \langle x_1, \ldots, x_n \rangle \notin R \end{cases}$$

является рекурсивной (примитивно рекурсивной).

Напомним, что вместо $\langle x_1, \ldots, x_n\rangle \in R$ иногда пишут $R(x_1, \ldots, x_n)$ и говорят, что предикат R истиннен на элементах $x_1, \ldots, x_n$. Если же $\langle x_1, \ldots, x_n\rangle \notin R$, то пишут $\neg R(x_1, \ldots, x_n)$ и говорят, что предикат R ложен на элементах $x_1, \ldots, x_n$.

Определение. Для $P, Q \subseteq \omega^n$ введем следующие обозначения для n-местных отношений:

$$P\&Q = \{\overline{x} \in \omega^n \mid P(\overline{x}) \text{ истинно, и } Q(\overline{x}) \text{ истинно}\},$$
$$P \vee Q = \{\overline{x} \in \omega^n \mid P(\overline{x}) \text{ истинно, или } Q(\overline{x}) \text{ истинно}\},$$
$$P \to Q = \{\overline{x} \in \omega^n \mid \text{если } P(\overline{x}) \text{ истинно, то } Q(\overline{x}) \text{ истинно}\},$$
$$\neg P = \{\overline{x} \in \omega^n \mid P(\overline{x}) \text{ ложно}\}.$$

Кроме этого, для $R \subseteq \omega^{n+1}$ введем следующие обозначения для $(n+1)$-местных отношений:

$$\exists i \leqslant y\, R(\overline{x}, i) = \{\langle \overline{x}, y\rangle \in \omega^{n+1} \mid \text{существует } i \leqslant y \text{ такой, что } R(\overline{x}, i) \text{ истинно}\},$$
$$\forall i \leqslant y\, R(\overline{x}, i) = \{\langle \overline{x}, y\rangle \in \omega^{n+1} \mid \text{для всех } i \leqslant y \text{ отношение } R(\overline{x}, i) \text{ истинно}\},$$
$$\exists i < y\, R(\overline{x}, i) = \{\langle \overline{x}, y\rangle \in \omega^{n+1} \mid \text{существует } i < y \text{ такой, что } R(\overline{x}, i) \text{ истинно}\},$$
$$\forall i < y\, R(\overline{x}, i) = \{\langle \overline{x}, y\rangle \in \omega^{n+1} \mid \text{для всех } i < y \text{ отношение } R(\overline{x}, i) \text{ истинно}\}.$$

Замечание. С теоретико-множественной точки зрения отношение $P\&Q$ совпадает с пересечением $P \cap Q$, отношение $P \vee Q$ совпадает с объединением $P \cup Q$, а отношение $\neg P$ совпадает с дополнением $\omega^n \setminus P$. Кроме этого, имеет место тождество $P \to Q = \neg P \vee Q$.

Предложение 21. *Если отношения $P(\overline{x})$ и $Q(\overline{x})$ рекурсивны (примитивно рекурсивны), то отношения $P(\overline{x})\&Q(\overline{x})$, $P(\overline{x}) \vee Q(\overline{x})$, $\neg P(\overline{x})$, $P(\overline{x}) \to Q(\overline{x})$ тоже рекурсивны (примитивно рекурсивны).*

Доказательство. Следует из леммы 18 и следующих тождеств:

$$\mathcal{X}_{P\&Q}(\overline{x}) = \mathcal{X}_P(\overline{x}) \cdot \mathcal{X}_Q(\overline{x}), \qquad \mathcal{X}_{P\vee Q}(\overline{x}) = \operatorname{sg}(\mathcal{X}_P(\overline{x}) + \mathcal{X}_Q(\overline{x})), \qquad \mathcal{X}_{\neg P}(\overline{x}) = \overline{\operatorname{sg}}(\mathcal{X}_P(\overline{x})),$$
$$\mathcal{X}_{P\to Q}(\overline{x}) = \mathcal{X}_{\neg P\vee Q}(\overline{x}) = \operatorname{sg}(\overline{\operatorname{sg}}(\mathcal{X}_P(\overline{x})) + \mathcal{X}_Q(\overline{x})).$$

□

Предложение 22. *Бинарные отношения $=$, $\neq$, $<$, $>$, $\leqslant$, $\geqslant$ являются примитивно рекурсивными.*

Доказательство. Примитивная рекурсивность данных отношений следует из леммы 18 и тождеств

$$\mathcal{X}_{=}(x, y) = \overline{\operatorname{sg}}|x - y|, \qquad \mathcal{X}_{\neq}(x, y) = \operatorname{sg}|x - y|,$$
$$\mathcal{X}_{<}(x, y) = \operatorname{sg}(y \dot{-} x), \qquad \mathcal{X}_{>}(x, y) = \operatorname{sg}(x \dot{-} y),$$
$$\mathcal{X}_{\leqslant}(x, y) = \overline{\operatorname{sg}}(x \dot{-} y), \qquad \mathcal{X}_{\geqslant}(x, y) = \overline{\operatorname{sg}}(y \dot{-} x).$$

□

Предложение 23. *Если отношение $R(\overline{x}, i)$ рекурсивно (примитивно рекурсивно), то отношения $\exists i \leqslant y\, R(\overline{x}, i)$, $\forall i \leqslant y\, R(\overline{x}, i)$, $\exists i < y\, R(\overline{x}, i)$, $\forall i < y\, R(\overline{x}, i)$ тоже рекурсивны (примитивно рекурсивны).*

Доказательство. Утверждение для отношения $P(\overline{x}, y) = \exists i \leqslant y\, R(\overline{x}, i)$ следует из леммы 19 и тождества

$$\mathcal{X}_P(\overline{x}, y) = \operatorname{sg}\Big(\sum_{i=0}^{y} \mathcal{X}_R(\overline{x}, i)\Big).$$

Утверждение для остальных отношений вытекает из предложений 21, 22 и эквивалентностей

$$\begin{gathered}\forall i \leqslant y\, R(\overline{x}, i) \Longleftrightarrow \neg\exists i \leqslant y\, \neg R(\overline{x}, i),\\ \exists i < y\, R(\overline{x}, i) \Longleftrightarrow \exists i \leqslant y (R(\overline{x}, i)\,\&\, i \neq y),\\ \forall i < y\, R(\overline{x}, i) \Longleftrightarrow \neg\exists i < y\, \neg R(\overline{x}, i).\end{gathered}$$

□

Предложение 24 (о кусочном задании функции)**.** *Пусть $R_0, \ldots, R_k \subseteq \omega^n$ — рекурсивные (примитивно рекурсивные) отношения, такие, что $R_0 \cup \ldots \cup R_k = \omega^n$ и $R_i \cap R_j = \varnothing$ при $i \neq j$. Пусть далее $f_0(x_1, \ldots, x_n), \ldots, f_k(x_1, \ldots, x_n)$ — рекурсивные (примитивно рекурсивные) функции. Тогда функция*

$$F(\overline{x}) = \begin{cases} f_0(\overline{x}), & \text{если } R_0(\overline{x}),\\ \ldots & \ldots\\ f_k(\overline{x}), & \text{если } R_k(\overline{x}) \end{cases}$$

тоже рекурсивна (примитивно рекурсивна).

Доказательство. Достаточно заметить, что $F(\overline{x}) = f_0(\overline{x}) \cdot \mathcal{X}_{R_0}(\overline{x}) + \ldots + f_k(\overline{x}) \cdot \mathcal{X}_{R_k}(\overline{x})$. □

Определение. Пусть $R(\overline{x}, y)$ — отношение, $h(\overline{x})$ — всюду определенная функция. Обозначим через $\mu y[R(\overline{x}, y)]$ функцию $\mu y[|\mathcal{X}_R(\overline{x}, y) - 1| = 0]$, а через $\mu y \leqslant h(\overline{x})[R(\overline{x}, y)]$ обозначим функцию $\mu y \leqslant h(\overline{x})[|\mathcal{X}_R(\overline{x}, y) - 1| = 0]$. Ясно, что если $R(\overline{x}, y)$ рекурсивно, то $\mu y[R(\overline{x}, y)]$ — ч.р.ф. Кроме этого, из предложения 20 следует, что если $R(\overline{x}, y)$ и $h(\overline{x})$ примитивно рекурсивны, то $\mu y \leqslant h(\overline{x})[R(\overline{x}, y)]$ — п.р.ф.

Лемма 25. *а) Функция $[\frac{x}{y}]$, равная целой части от частного $\frac{x}{y}$, примитивно рекурсивна (по определению считаем, что $[\frac{x}{0}] = x$).*

б) Отношение $\mathrm{Div}(x, y)$*, истинное тогда и только тогда, когда x делит y, примитивно рекурсивно.*

в) Отношение $\mathrm{Prime}(x)$*, истинное тогда и только тогда, когда x — простое число, примитивно рекурсивно.*

Доказательство. Докажем утверждение пункта (а). Имеет место цепочка эквивалентностей $[x/y] = z \iff (y = 0\ \&\ x = z) \vee (y \neq 0\ \&\ z \leqslant x/y < z + 1) \iff (y = 0\ \&\ x = z) \vee (y \neq 0\ \&\ zy \leqslant x < (z + 1)y) \iff (y = 0\ \&\ x = z) \vee (y \neq 0\ \&\ z$ – наименьшее, такое, что $x < (z + 1)y)$. Кроме этого, ясно, что $[x/y] \leqslant x$. Отсюда получаем:

$$[x/y] = \mu z \leqslant x\Big[(y = 0\ \&\ x = z) \vee (y \neq 0\ \&\ x < (z + 1)y)\Big].$$

Таким образом, функция $[x/y]$ получена ограниченной минимизацией из примитивно рекурсивных функций, а значит, является п.р.ф.

Утверждения пунктов (б) и (в) следуют из эквивалентностей:

$$\mathrm{Div}(x,y) \Longleftrightarrow \exists z \leqslant y(xz = y)$$
$$\mathrm{Prime}(x) \Longleftrightarrow (x \geqslant 2)\ \&\ \forall y \leqslant x\Big(\mathrm{Div}(y,x) \longrightarrow (y = 1 \vee y = x)\Big).$$

□

Лемма 26. *а) Функция $f(x) = p_x$, где $p_0 = 2, p_1 = 3, p_2 = 5, \ldots$ — перечисление всех простых чисел в порядке возрастания, является примитивно рекурсивной.*

б) Функция $\mathrm{ex}(i,x)$, равная показателю степени p_i в каноническом разложении числа x на простые множители, является примитивно рекурсивной (здесь $\mathrm{ex}(i,0) = 0$).

Доказательство. а) Сначала индукцией по $n \in \omega$ докажем, что $p_n \leqslant 2^{2^n}$. Действительно, пусть для всех $i \leqslant n$ уже доказано, что $p_i \leqslant 2^{2^i}$. Тогда имеет место цепочка неравенств

$$p_0 p_1 \ldots p_n + 1 \leqslant 2^{2^0} \cdot 2^{2^1} \cdot \ldots \cdot 2^{2^n} + 1 = 2^{1+2+\ldots+2^n} + 1 =$$
$$= 2^{2^{n+1}-1} + 1 \leqslant 2^{2^{n+1}-1} + 2^{2^{n+1}-1} = 2^{2^{n+1}}.$$

Таким образом, число $N = p_0 p_1 \ldots p_n + 1$ не превосходит $2^{2^{n+1}}$ и, очевидно, не делится на простые числа $p_0, p_1, \ldots, p_n$. Следовательно, N должно делиться на некоторое простое число p_k для $k \geqslant n+1$. Но тогда, очевидно, $p_{n+1} \leqslant p_k \leqslant N \leqslant 2^{2^{n+1}}$. Что и требовалось доказать.

Отсюда следует, что функцию $f(x) = p_x$ можно получить по схеме примитивной рекурсии

$$\begin{cases} p_0 = 2, \\ p_{x+1} = \mu y \leqslant 2^{2^{x+1}}\big[\mathrm{Prime}(y)\ \&\ y > p_x\big]. \end{cases}$$

Так как участвующие в схеме функции $g = 2$ и $h(x,z) = \mu y \leqslant 2^{2^{x+1}}\big[\mathrm{Prime}(y)\ \&\ y > z\big]$ примитивно рекурсивны, то $f(x)$ является п.р.ф.

б) Функция $\mathrm{ex}(i,x)$ получается с помощью ограниченной минимизации

$$\mathrm{ex}(i,x) = \mu y \leqslant x[\neg\mathrm{Div}(p_i^{y+1}, x) \vee x = 0].$$

□

Замечание. В доказательстве пункта (а) предыдущей леммы было использовано неравенство $p_x \leqslant 2^{2^x}$. Используя теорему Чебышева, утверждающую, что для любого $n \geqslant 1$ среди чисел $n+1, n+2, \ldots, 2n$ найдется хотя бы одно простое, можно доказать более точную оценку $p_x \leqslant 2^{x+1}$. Однако для нас точность оценки не имеет никакого значения. Достаточно было найти любую п.р.ф. $\varphi(x)$ со свойством $p_x \leqslant \varphi(x)$.

§ 13. Кодирование машин Шёнфилда

Мы постепенно приближаемся к доказательству теоремы о том, что любая функция, вычислимая на машине Шёнфилда, является частично рекурсивной. Основная идея доказательства этой теоремы заключается в следующем: мы закодируем все вычисления на машинах Шёнфилда натуральными числами таким образом, что все необходимые операции с полученными кодами (т. е. операции, имитирующие работу машины) можно будет производить с помощью частично рекурсивных функций. В этом параграфе будет формально описан математический аппарат, реализующий эту идею. Нам предстоит закодировать натуральными числами команды, программы, наборы входных данных, состояния счетчика команд, состояния регистров и, наконец, вычисления на машинах Шёнфилда.

Существует много различных способов кодирования объектов натуральными числами. Мы остановимся на способе, в основе которого лежит возможность представления любого положительного числа $x \in \omega$ в виде $x = p_0^{a_0} \cdot p_1^{a_1} \cdot \ldots \cdot p_n^{a_n}$ (каноническое разложение). Прежде всего укажем способ кодирования произвольных конечных последовательностей натуральных чисел.

Определение. *Кодом последовательности* натуральных чисел $x_0, \ldots, x_{k-1}$ назовем натуральное число $p_0^{x_0+1} \cdot p_1^{x_1+1} \cdot \ldots \cdot p_{k-1}^{x_{k-1}+1}$, которое будем обозначать через $< x_0, \ldots, x_{k-1} >$. *Кодом пустой последовательности* $\varnothing$ будем считать число 1.

Замечание. Если $x =< x_0, \ldots, x_{k-1} >$ — код последовательности, то длина этой последовательности вычисляется с помощью примитивно рекурсивной функции

$$\mathrm{lh}(x) = \mu i \leqslant x[\mathrm{ex}(i, x) = 0],$$

а i-ая координата в последовательности, т. е. x_i, вычисляется с помощью примитивно рекурсивной функции

$$(x)_i = \mathrm{ex}(i, x) \dot{-} 1.$$

Лемма 27. *Предикат* $\mathrm{Seq}(x)$*, истинный тогда и только тогда, когда* x *— код последовательности, является примитивно рекурсивным.*

Доказательство. Произвольное ненулевое число x является кодом последовательности тогда и только тогда, когда в каноническом разложении x на простые множители все простые числа «идут подряд», т. е. если p_i входит в разложение с ненулевым показателем, то все числа $p_0, \ldots, p_{i-1}$ тоже входят в разложение с ненулевыми показателями. Таким образом, окончательно получаем

$$\mathrm{Seq}(x) \iff (x \neq 0) \,\&\, \forall i \leqslant x \Big(\mathrm{ex}(i, x) \neq 0 \longrightarrow \forall j \leqslant i (\mathrm{ex}(j, x) \neq 0) \Big).$$

Предикат, находящийся справа в этой эквивалентности, примитивно рекурсивен. Следовательно, предикат $\mathrm{Seq}(x)$ тоже примитивно рекурсивен. □

Определение. Определим *код команды* машины Шёнфилда следующим образом:

$$\text{код}(\mathsf{INC}\ i) =< 0, i >$$
$$\text{код}(\mathsf{DEC}\ i, j) =< 1, i, j >$$

Определим *код программы*, состоящей из команд $c_0, \ldots, c_{k-1}$, как следующее натуральное число:

$$< \text{код}(c_0), \ldots, \text{код}(c_{k-1}) >$$

Лемма 28. *Предикат* $\mathrm{Com}(x)$*, истинный тогда и только тогда, когда* x *— код команды, является примитивно рекурсивным.*

Доказательство. $\mathrm{Com}(x) \iff \mathrm{Seq}(x)\ \&\ \Big(((x)_0{=}0\ \&\ \mathrm{lh}(x){=}2) \vee ((x)_0{=}1\ \&\ \mathrm{lh}(x){=}3)\Big)$. Последний предикат является примитивно рекурсивным. Следовательно, предикат $\mathrm{Com}(x)$ тоже примитивно рекурсивен. □

Лемма 29. *Предикат* $\mathrm{Prog}(x)$*, истинный тогда и только тогда, когда* x *— код программы, является примитивно рекурсивным.*

Доказательство. Число x является кодом программы тогда и только тогда, когда x — код последовательности натуральных чисел, каждое из которых является кодом команды. Следовательно,

$$\mathrm{Prog}(x) \iff \mathrm{Seq}(x)\ \&\ \forall i < \mathrm{lh}(x)\big(\mathrm{Com}((x)_i)\big).$$

Последний предикат, в силу леммы 27 и леммы 28, является примитивно рекурсивным. □

Лемма 30 (о совместной рекурсии)**.** *Пусть функции* $f_0(\overline{x}, y), f_1(\overline{x}, y)$ *определены по схеме совместной рекурсии:*

$$\begin{cases} f_0(\overline{x}, 0) = g_0(\overline{x}) \\ f_1(\overline{x}, 0) = g_1(\overline{x}) \\ f_0(\overline{x}, y+1) = h_0(\overline{x}, y, f_0(\overline{x}, y), f_1(\overline{x}, y)) \\ f_1(\overline{x}, y+1) = h_1(\overline{x}, y, f_0(\overline{x}, y), f_1(\overline{x}, y)). \end{cases}$$

Если функции $g_i(\overline{x}), h_i(\overline{x}, y, z_0, z_1)$*, где* $i = 0, 1$*, примитивно рекурсивны, то функции* $f_0(\overline{x}, y)$*,* $f_1(\overline{x}, y)$ *тоже примитивно рекурсивны.*

Доказательство. Закодируем значения функций $f_0(\overline{x}, y)$ и $f_1(\overline{x}, y)$ одним числом, определив функцию

$$F(\overline{x}, y) =< f_0(\overline{x}, y), f_1(\overline{x}, y) >= 2^{f_0(\overline{x}, y)+1} \cdot 3^{f_1(\overline{x}, y)+1}.$$

Если мы докажем, что $F(\overline{x}, y)$ — п.р.ф., то, очевидно, функции $f_0(\overline{x}, y) = (F(\overline{x}, y))_0$, $f_1(\overline{x}, y) = (F(\overline{x}, y))_1$ тоже будут п.р.ф.

Докажем, что $F(\overline{x}, y)$ — п.р.ф., выписав для нее схему примитивной рекурсии:

$$\begin{cases} F(\overline{x}, 0) =< g_0(\overline{x}), g_1(\overline{x}) > \\ F(\overline{x}, y+1) =< h_0(\overline{x}, y, (F(\overline{x}, y))_0, (F(\overline{x}, y))_1), h_1(\overline{x}, y, (F(\overline{x}, y))_0, (F(\overline{x}, y))_1) > \end{cases}$$

В этой схеме участвуют только примитивно рекурсивные функции. Следовательно, $F(\overline{x}, y)$ — п.р.ф. Что и требовалось доказать. □

Замечание. Если в формулировке леммы о совместной рекурсии заменить всюду слова «примитивно рекурсивны» на слова «частично рекурсивны» (или на слова «рекурсивны»), то утверждение по-прежнему будет верным. Однако нам понадобится только случай примитивно рекурсивных функций.

Теперь мы введем две важные функции, кодирующие полностью всю информацию о ходе вычисления на машине Шёнфилда. Чтобы целиком охватить поток данных, изменяющихся в ходе такого вычисления, необходимо знать на каждом шаге содержимое счетчика команд и содержимые всех регистров, которые влияют на ход вычислений. Оценим, насколько большим может быть номер регистра, существенно влияющего на ход работы заданной машины Шёнфилда. Для этого заметим, что для любого $x \in \omega$ справедливо неравенство $(x)_i < x$. Действительно, имеет место цепочка неравенств:

$$(x)_i = \mathrm{ex}(i,x) \dot{-} 1 \leqslant \mathrm{ex}(i,x) < 2^{\mathrm{ex}(i,x)} \leqslant p_i^{\mathrm{ex}(i,x)} \leqslant x.$$

Отсюда следует, что если e — код программы, m — номер упомянутого в этой программе регистра, то e больше кода любой команды из этой программы, и тем более $e > m$. Кроме этого, ход вычислений по программе с кодом e зависит от входных данных, которые могут быть записаны в регистрах с 1-го по k-й, где k, вообще говоря, — произвольное натуральное число.

Таким образом, можно утверждать, что на ход вычислений по программе с кодом e, начатых с входными данными, записанными в регистрах с 1-го по k-й, могут влиять только регистры, номера которых не превосходят $e+k$. По крайней мере, регистры с номерами большими, чем $e+k$, не оказывают никакого влияния на работу машины, а их содержимое остается неизменным.

Определение. Определим две 3-местные частичные функции:

$$\mathrm{ct}(e,x,n) = \begin{cases} y, & \text{если выполняются следующие условия:} \\ & \text{(а) } e \text{ — код некоторой программы } P, \\ & \text{(б) } x =< x_1, \ldots, x_k > \text{ — код некоторой} \\ & \text{последовательности натуральных чисел,} \\ & \text{(в) } y \text{ — содержимое счетчика команд после} \\ & n \text{ шагов выполнения программы } P, \text{ начатой с} \\ & \text{содержимыми регистров } 0, x_1, \ldots, x_k, 0, 0, \ldots, \\ 0 & \text{в противном случае.} \end{cases}$$

$$\mathrm{rg}(e,x,n) = \begin{cases} < r_0, \ldots, r_{e+k} >, & \text{если выполняются следующие условия:} \\ & \text{(а) } e \text{ — код некоторой программы } P, \\ & \text{(б) } x =< x_1, \ldots, x_k > \text{ — код некоторой} \\ & \text{последовательности натуральных чисел,} \\ & \text{(в) } r_i \ (0 \leqslant i \leqslant e+k) \text{ — содержимое } i\text{-го} \\ & \text{регистра после } n \text{ шагов выполнения} \\ & \text{программы } P, \text{ начатой с содержимыми} \\ & \text{регистров } 0, x_1, \ldots, x_k, 0, 0, \ldots, \\ 0 & \text{в противном случае.} \end{cases}$$

Функция $\mathrm{ct}(e,x,n)$ называется *содержимым счетчика команд* после n шагов вычислений, а функция $\mathrm{rg}(e,x,n)$ называется *кодом состояния регистров* после n шагов вычислений.

Лемма 31. *Функции* $\mathrm{ct}(e,x,n)$ *и* $\mathrm{rg}(e,x,n)$ *являются примитивно рекурсивными.*

Доказательство. Достаточно показать, что $\mathrm{ct}(e,x,n)$ и $\mathrm{rg}(e,x,n)$ получается по схеме совместной рекурсии из некоторых п.р.ф. Совместная рекурсия будет производиться по переменной n, т. е. по количеству уже выполненных шагов в вычислении, упомянутом в определении выше.

При $n = 0$ очевидно $\mathrm{ct}(e,x,0) = 0$, а значение $\mathrm{rg}(e,x,0)$ можно вычислить по следующей неформальной схеме:

$$\mathrm{rg}(e,x,0) = \begin{cases} <0,x_1,\ldots,x_k,0\ldots,0>, & \text{если } e \text{ — код некоторой программы, и} \\ & x =< x_1,\ldots,x_k > \text{ — код некоторой} \\ & \text{последовательности натуральных чисел,} \\ 0 & \text{в противном случае.} \end{cases}$$

Определим вспомогательную примитивно рекурсивную функцию:

$$\alpha(i,x) = \begin{cases} \mathrm{ex}(i \dot{-} 1, x), & \text{если } 1 \leqslant i \leqslant \mathrm{lh}(x), \\ 1 & \text{иначе.} \end{cases}$$

Тогда формальная схема для вычисления $\mathrm{rg}(e,x,0)$ (т. е. схема, использующая только примитивно рекурсивные функции от e и x) имеет вид:

$$\mathrm{rg}(e,x,0) = \begin{cases} \prod\limits_{i=0}^{e+\mathrm{lh}(x)} p_i^{\alpha(i,x)}, & \text{если } \mathrm{Prog}(e) \,\&\, \mathrm{Seq}(x), \\ 0, & \text{если } \neg\mathrm{Prog}(e) \vee \neg\mathrm{Seq}(x). \end{cases}$$

Предположим, что значения $\mathrm{ct}(e,x,n)$ и $\mathrm{rg}(e,x,n)$ уже вычислены. Обозначим $y = \mathrm{ct}(e,x,n)$. Тогда значение $\mathrm{ct}(e,x,n+1)$ можно вычислить с помощью следующей неформальной схемы:

$$\mathrm{ct}(e,x,n+1) = \begin{cases} y+1, & \text{если } \mathrm{Prog}(e), \mathrm{Seq}(x), \text{ и команда с} \\ & \text{номером } \mathrm{ct}(e,x,n) \text{ имеет вид } \mathsf{INC}\ i, \\ j, & \text{если } \mathrm{Prog}(e), \mathrm{Seq}(x), \text{ команда с} \\ & \text{номером } \mathrm{ct}(e,x,n) \text{ имеет вид } \mathsf{DEC}\ i,j \\ & \text{и содержимое } i\text{-го регистра в данный момент } > 0, \\ y+1, & \text{если } \mathrm{Prog}(e), \mathrm{Seq}(x), \text{ команда с} \\ & \text{номером } \mathrm{ct}(e,x,n) \text{ имеет вид } \mathsf{DEC}\ i,j \\ & \text{и содержимое } i\text{-го регистра в данный момент } = 0, \\ y, & \text{если } \mathrm{Prog}(e), \mathrm{Seq}(x), \text{ и в программе} \\ & \text{нет команды с номером } \mathrm{ct}(e,x,n), \\ 0 & \text{во всех остальных случаях.} \end{cases}$$

Теперь перепишем эту схему формально, используя только примитивно рекурсивные функции, зависящие от e, x, $\mathrm{ct}(e,x,n)$ и $\mathrm{rg}(e,x,n)$:

$$\mathrm{ct}(e,x,n+1)=\begin{cases}\mathrm{ct}(e,x,n)+1, & \text{если } \mathrm{Prog}(e),\mathrm{Seq}(x),\mathrm{ct}(e,x,n)<\mathrm{lh}(e),\\ & \text{и } \left((e)_{\mathrm{ct}(e,x,n)}\right)_0=0,\\ \left((e)_{\mathrm{ct}(e,x,n)}\right)_2, & \text{если } \mathrm{Prog}(e),\mathrm{Seq}(x),\mathrm{ct}(e,x,n)<\mathrm{lh}(e),\\ & \left((e)_{\mathrm{ct}(e,x,n)}\right)_0=1 \text{ и } (\mathrm{rg}(e,x,n))_{\left((e)_{\mathrm{ct}(e,x,n)}\right)_1}>0,\\ \mathrm{ct}(e,x,n)+1, & \text{если } \mathrm{Prog}(e),\mathrm{Seq}(x),\mathrm{ct}(e,x,n)<\mathrm{lh}(e),\\ & \left((e)_{\mathrm{ct}(e,x,n)}\right)_0=1 \text{ и } (\mathrm{rg}(e,x,n))_{\left((e)_{\mathrm{ct}(e,x,n)}\right)_1}=0,\\ \mathrm{ct}(e,x,n), & \text{если } \mathrm{Prog}(e),\mathrm{Seq}(x) \text{ и } \mathrm{ct}(e,x,n)\geqslant\mathrm{lh}(e),\\ 0 & \text{в остальных случаях.}\end{cases}$$

В данной схеме если e — код программы, x — код последовательности, то условие $\mathrm{ct}(e,x,n)<\mathrm{lh}(e)$ равносильно тому, что в указанной программе существует команда с номером $\mathrm{ct}(e,x,n)$, и в этом случае $(e)_{\mathrm{ct}(e,x,n)}$ — это код команды, исполняемой на $(n+1)$-ом шаге. Если эта команда имеет вид $\mathsf{INC}\ i$, то $\left((e)_{\mathrm{ct}(e,x,n)}\right)_0=0$, при этом параметр i вычисляется следующим образом: $i=\left((e)_{\mathrm{ct}(e,x,n)}\right)_1$. Если же эта команда имеет вид $\mathsf{DEC}\ i,j$, то $\left((e)_{\mathrm{ct}(e,x,n)}\right)_0=1$, и тогда параметры i,j можно вычислить так: $i=\left((e)_{\mathrm{ct}(e,x,n)}\right)_1$, $j=\left((e)_{\mathrm{ct}(e,x,n)}\right)_2$. Наконец, проверка условия положительности содержимого i-го регистра после n шагов вычислений равносильна проверке выполнимости условия $(\mathrm{rg}(e,x,n))_{\left((e)_{\mathrm{ct}(e,x,n)}\right)_1}>0$.

Далее, значение $\mathrm{rg}(e,x,n+1)$ можно вычислить с помощью следующей неформальной схемы:

$$\mathrm{rg}(e,x,n+1)=\begin{cases}<r_0,\ldots,r_i+1,\ldots,r_{e+k}>, & \text{если } \mathrm{Prog}(e),\mathrm{Seq}(x),\text{ команда с}\\ & \text{номером } \mathrm{ct}(e,x,n) \text{ имеет вид } \mathsf{INC}\ i,\\ & \text{и } \mathrm{rg}(e,x,n)=<r_0,\ldots,r_{e+k}>,\\ <r_0,\ldots,r_i\dot{-}1,\ldots,r_{e+k}>, & \text{если } \mathrm{Prog}(e),\mathrm{Seq}(x),\text{ команда с}\\ & \text{номером } \mathrm{ct}(e,x,n) \text{ имеет вид } \mathsf{DEC}\ i,j,\\ & \mathrm{rg}(e,x,n)=<r_0,\ldots,r_{e+k}>, \text{ и } r_i>0,\\ <r_0,\ldots,r_i,\ldots,r_{e+k}>, & \text{если } \mathrm{Prog}(e),\mathrm{Seq}(x),\text{ команда с}\\ & \text{номером } \mathrm{ct}(e,x,n) \text{ имеет вид } \mathsf{DEC}\ i,j,\\ & \mathrm{rg}(e,x,n)=<r_0,\ldots,r_{e+k}>, \text{ и } r_i=0,\\ <r_0,\ldots,r_{e+k}>, & \text{если } \mathrm{Prog}(e),\mathrm{Seq}(x), \text{в программе}\\ & \text{нет команды с номером } \mathrm{ct}(e,x,n),\\ & \text{и } \mathrm{rg}(e,x,n)=<r_0,\ldots,r_{e+k}>,\\ 0 & \text{во всех остальных случаях.}\end{cases}$$

Заметим, что в этой схеме можно объединить второй и третий случаи, поскольку если $r_i=0$, то $r_i\dot{-}1=r_i$.

Введем вспомогательную примитивно рекурсивную функцию

$$\beta(i,x,y)=\left[\frac{x}{p_i^{\mathrm{ex}(i,x)}}\right]\cdot p_i^{y+1},$$

которая удовлетворяет условию: если $x =< x_0,\dots,x_{i-1},x_i,x_{i+1},\dots,x_k >$ — код последовательности и $i\leqslant k$, то $\beta(i,x,y) =< x_0,\dots,x_{i-1},y,x_{i+1},\dots,x_k >$.

Тогда окончательно получаем следующую формальную схему для вычисления $\mathrm{rg}(e,x,n+1)=$

$$=\begin{cases}\beta\Big(\Big((e)_{\mathrm{ct}(e,x,n)}\Big)_1,\mathrm{rg}(e,x,n),\Big(\mathrm{rg}(e,x,n)\Big)_{\Big((e)_{\mathrm{ct}(e,x,n)}\Big)_1}+1\Big), & \text{если } \mathrm{Prog}(e),\mathrm{Seq}(x),\\ & \mathrm{ct}(e,x,n)<\mathrm{lh}(e),\\ & \text{и } \Big((e)_{\mathrm{ct}(e,x,n)}\Big)_0=0,\\ \beta\Big(\Big((e)_{\mathrm{ct}(e,x,n)}\Big)_1,\mathrm{rg}(e,x,n),\Big(\mathrm{rg}(e,x,n)\Big)_{\Big((e)_{\mathrm{ct}(e,x,n)}\Big)_1}\dot{-}1\Big), & \text{если } \mathrm{Prog}(e),\mathrm{Seq}(x),\\ & \mathrm{ct}(e,x,n)<\mathrm{lh}(e),\\ & \text{и } \Big((e)_{\mathrm{ct}(e,x,n)}\Big)_0=1,\\ \mathrm{rg}(e,x,n), & \text{если } \mathrm{Prog}(e),\mathrm{Seq}(x),\\ & \text{и } \mathrm{ct}(e,x,n)\geqslant\mathrm{lh}(e),\\ 0 & \text{в остальных случаях.}\end{cases}$$

Таким образом, используя предложение 24, мы полностью выписали требуемую схему совместной рекурсии, в которой участвуют только примитивно рекурсивные функции и отношения. Следовательно, ct и rg — п.р.ф. □

Определение. Определим трехместный предикат:

$\mathrm{Stop}(e,x,n)\iff$ Выполняются следующие три условия:

(а) e — код некоторой программы P,

(б) $x =< x_1,\dots,x_k >$ — код некоторой последовательности натуральных чисел,

(в) программа P, начав работу с содержимыми регистров $0,x_1,\dots,x_k,0,0,\dots,$ останавливается после n-го шага.

Лемма 32. *Предикат* $\mathrm{Stop}(e,x,n)$ *примитивно рекурсивен.*

Доказательство. Заметим, что если выполняются условия (а) и (б) из определения выше, то предикат $\mathrm{Stop}(e,x,n)$ истиннен тогда и только тогда, когда машина Шёнфилда с кодом e не найдет команду для исполнения впервые только после n-го шага. Следовательно, имеет место эквивалентноть:

$$\mathrm{Stop}(e,x,n)\iff \mathrm{Prog}(e)\ \&\ \mathrm{Seq}(x)\ \&\ \big(\mathrm{ct}(e,x,n)\geqslant\mathrm{lh}(e)\big)\ \&\ \forall j<n\big(\mathrm{ct}(e,x,j)<\mathrm{lh}(e)\big).$$

Отсюда, используя леммы 27, 29, 31, заключаем, что отношение $\mathrm{Stop}(e,x,n)$ примитивно рекурсивно. □

Определение. Пусть натуральные числа e и x удовлетворяют следующим трем условиям:

а) e — код некоторой программы P,

б) $x =< x_1, \ldots, x_k >$ — код некоторой последовательности натуральных чисел,

в) программа P, начав работу с содержимыми регистров $0, x_1, \ldots, x_k, 0, 0, \ldots$, останавливается.

Тогда *кодом вычисления* на машине Шёнфилда с номером e и начальными содержимыми регистров $0, x_1, \ldots, x_k, 0, 0, \ldots$ будем называть код последовательности $< \mathrm{rg}(e, x, 0), \ldots, \mathrm{rg}(e, x, m) >$, состоящей из всех последовательно записанных кодов состояния регистров по ходу вычислений, причем считаем, что в последнем из этих состояний произошла остановка машины.

Если y — код вычисления, то результат этого вычисления содержится в 0-ом регистре после остановки и может быть вычислен с помощью примитивно рекурсивной функции

$$U(y) = \Big((y)_{\mathrm{lh}(y) \dot{-} 1}\Big)_0.$$

Если для $e, x \in \omega$ хотя бы одно из условий (а), (б), (в) не выполнено, то код вычисления не определен.

Определение. Пусть $k \in \omega$, определим $(k{+}2)$-местный *T-предикат Клини*:

$$T_k(e, x_1, \ldots, x_k, y) \iff \begin{array}{l} e \text{ — код некоторой программы, а } y \text{ — код} \\ \text{вычисления по этой программе с начальными} \\ \text{содержимыми регистров } 0, x_1, \ldots, x_k, 0, 0, \ldots \end{array}$$

Лемма 33. *Предикат $T_k(e, x_1, \ldots, x_k, y)$ примитивно рекурсивен.*

Доказательство. Для фиксированного $k \in \omega$ имеет место эквивалентность

$$\begin{aligned} T_k(e, x_1, \ldots, x_k, y) \iff & \mathrm{Seq}(y) \;\&\; \mathrm{Stop}(e, < x_1, \ldots, x_k >, \mathrm{lh}(y) \dot{-} 1) \;\& \\ & \forall i < \mathrm{lh}(y) \Big((y)_i = \mathrm{rg}(e, < x_1, \ldots, x_k >, i) \Big) \end{aligned}$$

(в правой части эквивалентности опущен конъюнктивный член $\mathrm{Prog}(e)$, поскольку он уже учтен в определении предиката Stop).

Отсюда, в силу лемм 27, 31, 32, заключаем, что T_k — примитивно рекурсивный предикат. □

§ 14. Машины Шёнфилда vs Частично рекурсивные функции

Мы, наконец, готовы к тому, чтобы привести доказательство теоремы о частичной рекурсивности функций, вычислимых на машинах Шёнфилда.

Теорема 34. *Любая функция, вычислимая по Шёнфилду, является частично рекурсивной.*

Доказательство. Пусть $f(x_1, \ldots, x_k)$ — произвольная частичная функция, вычислимая по Шёнфилду. Следовательно, существует программа P для машины, которая вычисляет $f(x_1, \ldots, x_k)$. Пусть e — код программы P, а $x_1, \ldots, x_k$ — произвольные значения аргументов f.

Если $f(x_1, \ldots, x_k)$ определено, то P на входных данных $x_1, \ldots, x_k$ останавливается. Следовательно, определен код вычисления y на машине с программой P и с начальными содержимыми регистров $0, x_1, \ldots, x_k, 0, \ldots$, причем $U(y) = f(x_1, \ldots, x_k)$. Другими словами, существует y такой, что предикат $T_k(e, x_1, \ldots, x_k, y)$ истиннен. Заметим, что тогда данный y является единственным со свойством $T_k(e, x_1, \ldots, x_k, y)$, а значит, он является наименьшим с такими свойствами. Отсюда заключаем, что $f(x_1, \ldots, x_k) = U(\mu y[T_k(e, x_1, \ldots, x_k, y)])$.

Если же $f(x_1, \ldots, x_k)$ не определено, то P на входных данных $x_1, \ldots, x_k$ никогда не остановится. Следовательно, не существует кода вычисления y на машине P с начальными содержимыми регистров $0, x_1, \ldots, x_k, 0, \ldots$ Следовательно, для любого y предикат $T_k(e, x_1, \ldots, x_k, y)$ ложен, и значит значение $U(\mu y[T_k(e, x_1, \ldots, x_k, y)])$ тоже не определено. Таким образом, опять выполняется соотношение $f(x_1, \ldots, x_k) = U(\mu y[T_k(e, x_1, \ldots, x_k, y)])$.

Так как T_k — примитивно рекурсивный предикат, то $\mu y[T_k(e, x_1, \ldots, x_k, y)]$ — частично рекурсивная функция. Поскольку U — примитивно рекурсивная функция, то функция $f(x_1, \ldots, x_k) = U(\mu y[T_k(e, x_1, \ldots, x_k, y)])$ частично рекурсивна. □

Из теоремы 34 можно вывести несколько важных следствий. Следующие четыре результата безусловно можно считать достаточным оправданием той трудоемкой работы, которую мы проделали в двух предыдущих параграфах.

Следствие 35. *Частичная функция вычислима на машине Шёнфилда тогда и только тогда, когда она является частично рекурсивной.*

Доказательство. Следует из теоремы 16 и теоремы 34. □

Теорема 36 (теорема Клини о нормальной форме)**.** *Для любой k-местной частично рекурсивной функции $f(x_1, \ldots, x_k)$ существует натуральное число e такое, что*

$$f(x_1, \ldots, x_k) = U(\mu y[T_k(e, x_1, \ldots, x_k, y)]).$$

Доказательство. Поскольку $f(x_1, \ldots, x_k)$ частично рекурсивна, то она вычислима по Шёнфилду. Следовательно, найдется e — код программы, которая вычисляет функцию $f(x_1, \ldots, x_k)$. Из доказательства теоремы 34 следует, что e — искомое. □

Следствие 37. *Любая ч.р.ф. может быть получена из простейших функций с помощью конечной последовательности применений операторов S, R или M, в которой общее число применений оператора M не превосходит 1.*

Доказательство. Функция U и предикат T_k примитивно рекурсивны и, следовательно, могут быть получены из простейших функций только с помощью операторов S и R. Таким образом, доказываемое утверждение следует из теоремы Клини о нормальной форме. □

Определение. Пусть $k \in \omega$, K — некоторое семейство k-местных частичных функций. $(k{+}1)$-местная частичная функция $F(x_0, x_1, \ldots, x_k)$ называется *универсальной для семейства K*, если выполняются следующие условия:

1) для любого $n \in \omega$ существует $f \in K$ такая, что $F(n, x_1, \ldots, x_k) = f(x_1, \ldots, x_k)$;

2) для любой $f \in K$ существует $n \in \omega$ такое, что $F(n, x_1, \ldots, x_k) = f(x_1, \ldots, x_k)$.

Другими словами, K совпадает с множеством функций $\{F(0, \overline{x}), F(1, \overline{x}), F(2, \overline{x}), \ldots\}$.

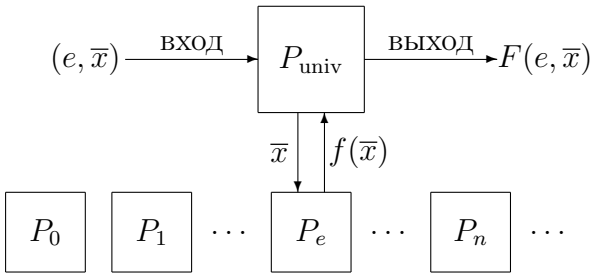
Теорема 38 (об универсальной ч.р.ф.)**.** *Существует* $(k{+}1)$*-местная ч.р.ф.* $F(x_0, x_1, \ldots, x_k)$*, универсальная для семейства всех* k*-местных ч.р.ф.*

Доказательство. В качестве F можно взять $(k{+}1)$-местную частично рекурсивную функцию

$$F(x_0, x_1, \ldots, x_k) = U(\mu y[T_k(x_0, x_1, \ldots, x_k, y)]).$$

Из теоремы Клини о нормальной форме следует, что F — универсальная для всех k-местных частично рекурсивных функций. □

Из теоремы об универсальной ч.р.ф. следует, что существует *универсальная машина Шёнфилда* P_{univ}, которая в определенном смысле способна заменить бесконечное семейство всех машин Шёнфилда. Работу P_{univ} можно описать следующим образом (см. рисунок): на вход универсальной машины подается кортеж данных $(e, \overline{x})$; универсальная машина по коду e конструирует программу P_e, соответствующую этому коду, и запускает ее на входных данных $\overline{x}$; результат $f(\overline{x})$ работы программы P_e выдается на выход P_{univ} и является окончательным результатом ее работы, т. е. $F(e, \overline{x}) = f(\overline{x})$.



Определение. Пусть $k \in \omega$. Для каждого $e \in \omega$ введем обозначение

$$æ_e^k(x_1, \ldots, x_k) = U(\mu y[T_k(e, x_1, \ldots, x_k, y)]).$$

Функцию $æ_e^k(x_1, \ldots, x_k)$ будем называть k*-местной частично вычислимой функцией с клиниевским номером* e. В случае одноместных функций будем просто писать $æ_e(x)$ без верхнего индекса.

Замечание. Если e — номер машины Шёнфилда, которая вычисляет функцию f, то e также будет клиниевским номером этой функции. Но обратное, вообще говоря, неверно! Не любой клиниевский номер функции является номером машины Шёнфилда, вычисляющей эту функцию. Более того, если e — клиниевский номер, не являющийся номером машины, то предикат $T_k(e, \overline{x}, y)$ тождественно ложен, и значит функция $æ_e^k(\overline{x})$ нигде не определена.

Следствие 39. *Существуют частичные функции, не являющиеся частично рекурсивными.*

Доказательство. Рассмотрим семейство $\{æ_0(x), æ_1(x), æ_2(x), \ldots\}$. В силу теоремы об универсальной ч.р.ф. оно совпадает с семейством всех одноместных ч.р.ф. Введем следующую всюду определенную одноместную функцию

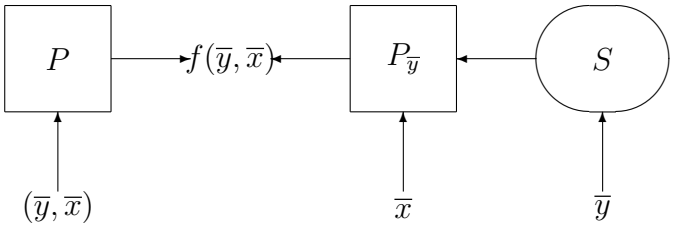
$$f(x) = \begin{cases} æ_x(x) + 1, & \text{если } æ_x(x) \text{ определено,} \\ 0 & \text{иначе.} \end{cases}$$

Допустим, f — ч.р.ф. Тогда найдется $e \in \omega$ такое, что $f = æ_e$. Следовательно, $æ_e$ тоже всюду определена. В частности, значение $æ_e(e)$ определено. Но тогда $æ_e(e) = f(e) = æ_e(e) + 1$. Противоречие. Следовательно, f не является частично рекурсивной. □

В заключение параграфа мы докажем еще одну фундаментальную теорему теории алгоритмов — теорему о параметризации. Интуитивный смысл данной теоремы состоит в следующем. Если задана некоторая ч.р.ф. $f(\overline{y}, \overline{x})$, переменные которой условно разделены на две группы, то к вычислению этой функции можно подойти двумя способами.

Первый способ — стандартный: мы целиком подаем на вход машины P, вычисляющей f, весь набор $\langle \overline{y}, \overline{x} \rangle$ и получаем на выходе значение $f(\overline{y}, \overline{x})$.

Второй способ: мы фиксируем значения переменных $\overline{y}$, считая их параметрами, и преобразуем нашу программу P с помощью специального *параметризатора* в программу $P_{\overline{y}}$, которая уже будет вычислять функцию f как функцию от оставшихся переменных $\overline{x}$. Код программы $P_{\overline{y}}$ зависит от значений $\overline{y}$, но основное свойство заключается в том, что этот код может быть вычислен по $\overline{y}$ с помощью некоторой п.р.ф. $s(\overline{y})$. Строго говоря, параметризатор — это и есть функция s, которая, используя значения $\overline{y}$, преобразует код e программы P в код $s(\overline{y})$ программы $P_{\overline{y}}$.



Теорема 40 (о параметризации)**.** *Для любой ч.р.ф.* $f(y_1, \ldots, y_m, x_1, \ldots, x_n)$ *существует разнозначная п.р.ф.* $s(y_1, \ldots, y_m)$ *такая, что*

$$f(y_1, \ldots, y_m, x_1, \ldots, x_n) = æ^n_{s(y_1,\ldots,y_m)}(x_1, \ldots, x_n).$$

Доказательство. Обозначим для краткости $\overline{y} = \langle y_1, \ldots, y_m \rangle$, $\overline{x} = \langle x_1, \ldots, x_n \rangle$ и введем функцию $f'(\overline{x}, \overline{y})$, положив по определению $f'(\overline{x}, \overline{y}) = f(\overline{y}, \overline{x})$. Ясно, что $f'(\overline{x}, \overline{y})$ — ч.р.ф. Следовательно, существует машина Шёнфилда P с кодом e, которая ее вычисляет. Обозначим через P^* макрос, соответствующий программе P.

Зафиксируем значения $\overline{y}$, считая их параметрами, и рассмотрим функцию $g(\overline{x}) = f'(\overline{x}, \overline{y})$. Данная функция вычисляется с помощью следующей макропрограммы:

$$\left.\begin{array}{l} \mathsf{INC}\ n+1 \\ \cdots \\ \mathsf{INC}\ n+1 \end{array}\right\} y_1$$
$$\cdots$$
$$\left.\begin{array}{l} \mathsf{INC}\ n+m \\ \cdots \\ \mathsf{INC}\ n+m \end{array}\right\} y_m$$
$$P^*$$

По теореме об элиминации макросов эта макропрограмма эквивалентна программе $P_{\overline{y}}$, которая не использует макросы и код которой зависит от параметров $\overline{y}$. Ясно, что

если l — число команд в программе P, то число команд в $P_{\overline{y}}$ вычисляется с помощью п.р.ф. $h(\overline{y}) = y_1 + \ldots + y_m + l$.

Определим функцию

$$G(k,\overline{y}) = \begin{cases} \text{код } k\text{-ой команды } P_{\overline{y}}, & \text{если в } P_{\overline{y}} \text{ есть команда с номером } k, \\ 0 & \text{иначе.} \end{cases}$$

Тогда код программы $P_{\overline{y}}$ задается функцией

$$s(\overline{y}) = \prod_{k=0}^{h(\overline{y}) \dot{-} 1} p_k^{G(k,\overline{y})+1}.$$

Отсюда видно, что если доказать примитивную рекурсивность функции $G(k,\overline{y})$, то функция $s(\overline{y})$ также будет примитивно рекурсивной. Запишем определение функции $G(k,\overline{y})$ в виде следующей неформальной схемы:

$$G(k,\overline{y}) = \begin{cases} \text{код}(\mathsf{INC}\ n+1), & \text{если } 0 \leqslant k < y_1, \\ \text{код}(\mathsf{INC}\ n+2), & \text{если } y_1 \leqslant k < y_1 + y_2, \\ \ldots & \ldots \\ \text{код}(\mathsf{INC}\ n+m), & \text{если } y_1 + \ldots + y_{m-1} \leqslant k < y_1 + \ldots + y_m, \\ \text{код}(\mathsf{INC}\ i), & \text{если } y_1 + \ldots + y_m \leqslant k < y_1 + \ldots + y_m + l, \\ & \text{и команда с номером } k \dot{-} (y_1 + \ldots + y_m) \\ & \text{из } P \text{ имеет вид } \mathsf{INC}\ i, \\ \text{код}(\mathsf{DEC}\ i, j + y_1 + \ldots + y_m), & \text{если } y_1 + \ldots + y_m \leqslant k < y_1 + \ldots + y_m + l, \\ & \text{и команда с номером } k \dot{-} (y_1 + \ldots + y_m) \\ & \text{из } P \text{ имеет вид } \mathsf{DEC}\ i, j, \\ 0 & \text{в остальных случаях.} \end{cases}$$

Теперь перепишем эту схему формально, используя только примитивно рекурсивные функции и предикаты. Определим вспомогательные п.р.ф. $\alpha(\overline{y}) = y_1 + \ldots + y_m$ и $\beta(k,\overline{y}) = k \dot{-} \alpha(\overline{y})$. Тогда

$$G(k,\overline{y}) = \begin{cases} < 0, n+1 >, & \text{если } 0 \leqslant k < y_1, \\ < 0, n+2 >, & \text{если } y_1 \leqslant k < y_1 + y_2, \\ \ldots & \ldots \\ < 0, n+m >, & \text{если } y_1 + \ldots + y_{m-1} \leqslant k < y_1 + \ldots + y_m, \\ < 0, \left((e)_{\beta(k,\overline{y})}\right)_1 >, & \text{если } y_1 + \ldots + y_m \leqslant k < y_1 + \ldots + y_m + l \\ & \text{и } \left((e)_{\beta(k,\overline{y})}\right)_0 = 0, \\ < 1, \left((e)_{\beta(k,\overline{y})}\right)_1, \left((e)_{\beta(k,\overline{y})}\right)_2 + \alpha(\overline{y}) >, & \text{если } y_1 + \ldots + y_m \leqslant k < y_1 + \ldots + y_m + l \\ & \text{и } \left((e)_{\beta(k,\overline{y})}\right)_0 = 1, \\ 0, & \text{если } k \geqslant y_1 + \ldots + y_m + l. \end{cases}$$

Отсюда, в силу предложения 24, заключаем, что $G(k,\overline{y})$ — п.р.ф. По построению $æ^n_{s(\overline{y})}(\overline{x}) = g(\overline{x}) = f'(\overline{x},\overline{y}) = f(\overline{y},\overline{x})$. Таким образом, функция $s(\overline{y})$ — искомая.

Нам осталось только убедиться в том, что s — инъективная функция. Для этого необходимо доказать, что для любого $z \in \mathrm{range}(s)$ существует единственный набор $\overline{y} \in \omega^m$ такой, что $s(\overline{y}) = z$. Другими словами, требуется доказать, что существует обратная функция $s^{-1} : \mathrm{range}(s) \to \omega^m$. (В этой части доказательства не требуется показывать, что s^{-1} частично рекурсивна!)

Действительно, если задано $z \in \mathrm{range}(s)$, то z является кодом программы вида

$$\left.\begin{array}{l}\mathsf{INC}\ n+1\\ \cdots\\ \mathsf{INC}\ n+1\end{array}\right\} y_1 \text{ строк}$$
$$\cdots$$
$$\left.\begin{array}{l}\mathsf{INC}\ n+m\\ \cdots\\ \mathsf{INC}\ n+m\end{array}\right\} y_m \text{ строк}$$
$$\left.\begin{array}{l}P\end{array}\right\} l \text{ строк}$$

для некоторых чисел $y_1, \ldots, y_m$ и известного нам числа l (l — это число строк в программе P). Числа $y_1, \ldots, y_m$ можно однозначно восстановить по виду программы, для этого нужно удалить из этой программы последние l строк, т. е. часть P. Тогда количество команд $\mathsf{INC}\ n{+}1$ в оставшейся части — это y_1, количество команд $\mathsf{INC}\ n{+}2$ в оставшейся части — это y_2, и так далее. Окончательно получаем, что $s^{-1}(z) = \langle y_1, \ldots, y_m \rangle$. □

Следствие 41 (s-m-n-теорема)**.** *Для любых $m, n \in \omega$ существует разнозначная $(m{+}1)$-местная примитивно рекурсивная функция $s^m_n(e, y_1, \ldots, y_m)$ такая, что*

$$æ^{m+n}_e(y_1, \ldots, y_m, x_1, \ldots, x_n) = æ^n_{s^m_n(e, y_1, \ldots, y_m)}(x_1, \ldots, x_n).$$

Доказательство. Рассмотрим $(m{+}n{+}1)$-местную ч.р.ф. $f(e, \overline{y}, \overline{x}) = æ^{m+n}_e(\overline{y}, \overline{x})$. По теореме о параметризации существует разнозначная п.р.ф. $s(e, \overline{y})$ такая, что имеет место $f(e, \overline{y}, \overline{x}) = æ^n_{s(e,\overline{y})}(\overline{x})$. Функция s является искомой s-m-n-функцией. □

§ 15. Машины Тьюринга

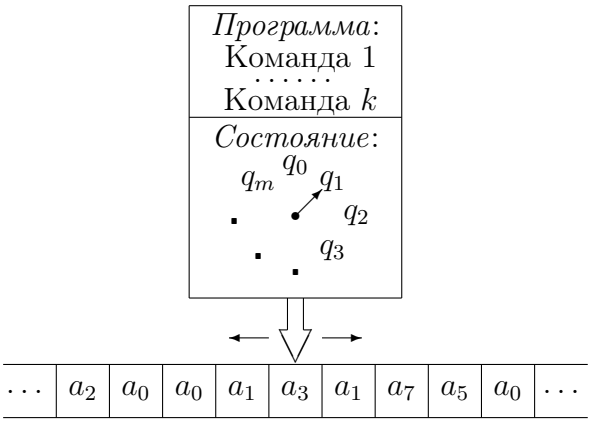
Машины Тьюринга — это еще один подход к формализации понятия вычислимой функции. Так же как и для машин Шёнфилда, модель для машины Тьюринга имеет механическое описание, но в отличие от первых машины Тьюринга более удобны для непосредственной реализации в виде электротехнических устройств.

Существует как минимум две причины, по которым машины Тьюринга можно считать более «реалистичными», чем машины Шёнфилда. Во-первых, в модели Тьюринга входные и выходные данные имеют символьный формат, т. е. данные — это слова, составленные из букв, записанных в ячейках ленты. В машинах Шёнфилда, как мы знаем, данные в регистрах могут быть записаны только в виде натуральных чисел. С инженерной точки зрения реализация типа данных «натуральные числа» является нетривиальной задачей и в конечном счете сводится к кодированию в виде тех же самых слов. Во-вторых, машины Тьюринга — это устройства с *последовательным* доступом к памяти: чтобы получить доступ к содержащейся в некоторой

ячейке информации, машине приходится перебирать одну за другой все ячейки между текущей и требуемой. В отличие от этого, машины Шёнфилда имеют память со *случайным* доступом: обращение к каждой ячейке осуществляется напрямую за один шаг.

Исторически машины Тьюринга возникли раньше, чем машины Шёнфилда, но, не смотря на то, что они являются более естественными и классическими устройствами, при доказательстве некоторых теорем удобнее оперировать с машинами Шёнфилда. В частности, теорему Клини о нормальной форме и теорему о параметризации можно доказать, используя терминологию и кодирование машин Тьюринга. Однако такой вариант доказательства этих теорем был бы намного сложнее технически, чем тот, что был предложен в предыдущем параграфе.

Перейдем к неформальному описанию машин Тьюринга.



Машина Тьюринга состоит из *ленты*, разбитой на одинаковые ячейки. В ячейках могут содержаться символы из внешнего алфавита $\mathcal{A}$. Содержимое ячеек может меняться. Лента — это механизм входов и выходов машины. Перед запуском машины входные данные записываются в конечном участке ленты, и в дальнейшем на каждом шаге вычислений используется только конечное множество ячеек. Однако в процессе работы лента может достраиваться новыми ячейками как слева, так и справа. Другими словами, лента является потенциально бесконечной в обе стороны.

Также машина Тьюринга обладает *считывающей головкой*, которая в процессе работы машины может обозревать только одну ячейку и распознавать содержимое этой ячейки. В зависимости от исполняемой команды головка способна изменять содержимое обозреваемой ячейки и сдвигаться за один шаг на одну ячейку влево или вправо.

Каждая машина Тьюринга имеет конечное число *состояний* $q_0, q_1, \ldots, q_m$, в которых она может находиться. В процессе работы машина может несколько раз возвращаться в одно и то же состояние. Работа всегда начинается с выделенного *начального состояния*. Кроме этого, есть выделенное *конечное состояние*. Если когда-нибудь машина попадает в конечное состояние, то она останавливается, иначе работает бесконечно.

Любая конкретная машина Тьюринга имеет свою уникальную программу. *Программа* — это конечный список команд, каждая из которых есть директива вида $q_i a_j \to q_k a_l S$, где q_i, q_k — состояния, a_j, a_l — символы внешнего алфавита, $S \in \{R, L, \Lambda\}$, причем для каждого неконечного состояния q_i и любого символа a_j в программе имеется ровно одна команда, начинающаяся с символов $q_i a_j \to \ldots$ Если же q_i — конечное состояние, то команда вида $q_i a_j \to \ldots$ отсутствует в программе.

Опишем один шаг работы машины Тьюринга. Пусть машина находится в некотором состоянии q_i, а головка в текущий момент обозревает ячейку с символом a_j. Если q_i — конечное состояние, то машина останавливается. В противном случае в программе находится единственная команда вида $q_i a_j \to q_k a_l S$, которая затем испол-

няется следующим образом: в обозреваемую ячейку записывается символ a_l, затем головка сдвигается по ленте на одну ячейку вправо (если $S = R$), или влево (если $S = L$), или вообще не двигается (если S — пустое слово), и затем машина переходит в состояние q_k.

Если при исполнении некоторой команды головке необходимо сдвинуться за левый край ленты, то в этот момент происходит автоматическое достраивание новой ячейки слева, в нее записывается выделенный символ (как правило, это символ 0) и головка передвигается на достроенную ячейку. Аналогичным образом происходит достраивание ячеек справа.

Теперь мы введем строгие, формальные определения.

Определение. *Машина Тьюринга* — это пятерка $T = \langle \mathcal{A}, Q, P, q_1, q_0 \rangle$, где:

а) $\mathcal{A} = \{a_0, a_1, \ldots, a_n\}$ — конечный *внешний алфавит* (мы будем всегда предполагать, что $n \geqslant 1$ и $a_0 = 0, a_1 = 1$);

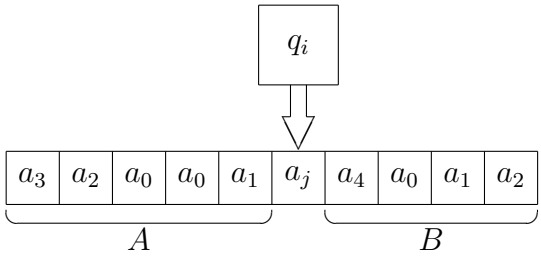
б) $Q = \{q_0, q_1, \ldots, q_m\}$ — конечный алфавит *внутренних состояний*;

в) $P = \{T(i,j) \mid 1 \leqslant i \leqslant m, 0 \leqslant j \leqslant n\}$ — *программа*, состоящая из команд $T(i,j)$, каждая из которых есть слово вида: $q_i a_j \to q_k a_l$, или $q_i a_j \to q_k a_l R$, или $q_i a_j \to q_k a_l L$, где $0 \leqslant k \leqslant m$, $0 \leqslant l \leqslant n$;

г) q_1 — *начальное состояние*;

д) q_0 — *конечное состояние*.

Определение. *Машинным словом (конфигурацией)* называется любое слово вида $Aq_i a_j B$, где $q_i \in Q$, $a_j \in \mathcal{A}$, A и B — слова в алфавите $\mathcal{A}$ (возможно, пустые).



Машинное слово $Aq_i a_j B$ — это формальное представление текущего положения всех деталей машины: q_i — текущее состояние, a_j — содержимое ячейки, обозреваемой головкой в данный момент, слово A состоит из всех символов, содержащихся в ячейках слева от обозреваемой, слово B состоит из всех символов, записанных в ячейках справа от обозреваемой. Исходя из такого интуитивного смысла, несложно определить формально, как преобразуются машинные слова за один шаг работы машины.

Определение. Пусть $M = Aq_i a_j B$ — машинное слово. Говорят, что машинное слово M'_T *получается из* M *за один шаг работы машины Тьюринга* T, если выполняется одно из условий:

1) $i = 0$, $M'_T = M$.

2) $i > 0$ и выполняется один из случаев:

   а) $T(i,j)$ имеет вид $q_i a_j \to q_k a_l$, и $M'_T = Aq_k a_l B$;

   б) $T(i,j)$ имеет вид $q_i a_j \to q_k a_l R$, $B = a_s B'$, и $M'_T = Aa_l q_k a_s B'$;

в) $T(i,j)$ имеет вид $q_ia_j \to q_ka_lR$, $B = \Lambda$, и $M'_T = Aa_lq_ka_0$ (достраивается новая ячейка справа);

г) $T(i,j)$ имеет вид $q_ia_j \to q_ka_lL$, $A = A'a_s$, и $M'_T = A'q_ka_sa_lB$;

д) $T(i,j)$ имеет вид $q_ia_j \to q_ka_lL$, $A = \Lambda$, и $M'_T = q_ka_0a_lB$ (достраивается новая ячейка слева).

Определение. Пусть $M = Aq_ia_jB$ — машинное слово. Положим $M_T^{(0)} = M$, $M_T^{(k+1)} = \left(M_T^{(k)}\right)'$ для всех $k \in \omega$.

Говорят, что машина T *перерабатывает слово M в слово M_1 без достраивания ячеек слева*, и пишут $M \underset{T}{\Longrightarrow} M_1$, если существует $k \in \omega$ такое, что $M_T^{(k)} = M_1$ и при этом не используется пункт (д).

Говорят, что машина T *перерабатывает слово M в слово M_1 без достраивания ячеек слева и справа*, и пишут $M \underset{T}{\models\!\Longrightarrow} M_1$, если существует $k \in \omega$ такое, что $M_T^{(k)} = M_1$ и при этом не используются пункты (в) и (д).

Определение. Говорят, что машина Тьюринга T *(правильно) вычисляет* частичную n-местную функцию $f : \mathrm{dom}(f) \subseteq \omega^n \longrightarrow \omega$, если для любых $x_1, \ldots, x_n \in \omega$ выполняются условия:

а) если $\langle x_1, \ldots, x_n\rangle \in \mathrm{dom}(f)$, то $q_101^{x_1}0 \ldots 01^{x_n}0 \underset{T}{\Longrightarrow} q_001^{f(x_1,\ldots,x_n)}00^s$, для некоторого $s \geqslant 0$;

б) если $\langle x_1, \ldots, x_n\rangle \notin \mathrm{dom}(f)$, то машина T, начав работу с машинного слова $M = q_101^{x_1}0 \ldots 01^{x_n}0$, работает бесконечно, т. е. q_0 не входит в $M_T^{(k)}$ ни для какого $k \in \omega$.

Определение. Частичная функция f называется *вычислимой по Тьюрингу*, если существует машина Тьюринга T, которая ее правильно вычисляет.

Теорема 42. *Частичная функция вычислима по Тьюрингу тогда и только тогда, когда она является частично рекурсивной.*

Доказательство. Мы приведем лишь схему доказательства. Более подробное и детальное доказательство этой важнейшей теоремы будет предложено в курсе по математической логике.

Чтобы доказать, что любая ч.р.ф. вычислима по Тьюрингу, нужно построить машины, вычисляющие все простейшие функции, и доказать, что функции, полученные из вычислимых по Тьюрингу функций с помощью операторов S, R, или M, также вычислимы по Тьюрингу.

Доказательство частичной рекурсивности любой функции, вычислимой по Тьюрингу, состоит из нескольких этапов:

а) Сначала необходимо закодировать все команды:

$$\begin{aligned}
\text{код}(q_ia_j \to q_ka_l) &=< i,j,k,l,0 >,\\
\text{код}(q_ia_j \to q_ka_lR) &=< i,j,k,l,1 >,\\
\text{код}(q_ia_j \to q_ka_lL) &=< i,j,k,l,2 > .
\end{aligned}$$

б) Затем нужно закодировать все программы. Если P — программа, состоящая из команд $c_0, \ldots, c_s$ (порядок расположения команд в программе не важен), то

$$\text{код}(P) =< \text{код}(c_0), \ldots, \text{код}(c_s) > .$$

в) Далее кодируем все машинные слова. Если $M = a_{j_0} \ldots a_{j_{p-1}} q_i a_{j_p} a_{j_{p+1}} \ldots a_{j_r}$ — машинное слово, то

$$\text{код}(M) =< j_0, \ldots, j_r, i, p > .$$

Кодировки, введенные в пунктах (а), (б), (в), являются эффективными, по коду команды (программы, машинного слова) можно однозначно восстановить вид самой команды (программы, машинного слова).

г) *Кодом вычисления на машине Тьюринга T, начатого в конфигурации M*, назовем натуральное число $< \text{код}(M_0), \ldots, \text{код}(M_t) >$, где $M = M_0, M_1, \ldots, M_t$ — машинные слова, такие, что

$$M_0 \underset{T}{\Longrightarrow} M_1 \underset{T}{\Longrightarrow} \ldots \underset{T}{\Longrightarrow} M_t,$$

причем $M_k = M_T^{(k)}$ для каждого $k \in \{0, \ldots, t\}$, слово M_t содержит вхождение q_0, а остальные слова M_i $(i < t)$ не содержат q_0 (другими словами, в конфигурации M_t произошла остановка машины).

Если для любого $k \in \omega$ слово $M_T^{(k)}$ не содержит q_0, т. е. машина не останавливается, то код вычисления не определен.

д) Для фиксированного $k \in \omega$ определим $(k{+}2)$-местный *T-предикат Клини*:

$$T_k(e, x_1, \ldots, x_k, y) \iff \begin{array}{l} e \text{ — код некоторой программы } P, \text{ а } y \text{ — код} \\ \text{вычисления на машине Тьюринга с программой } P, \\ \text{начатого в конфигурации } q_1 01^{x_1} 0 \ldots 01^{x_k} 0. \end{array}$$

Необходимо доказать, что предикат T_k рекурсивен.

е) Определим одноместную функцию U следующим образом:

$$U(y) = z \iff \begin{array}{l} y \text{ — код вычисления, оканчивающегося на машинном} \\ \text{слове вида } q_0 01^z 00^s \text{ для некоторого } s \geqslant 0, \text{ т. е. } z \text{ — результат} \\ \text{этого вычисления.} \end{array}$$

Необходимо доказать, что функция U рекурсивна.

ж) Наконец, если $f(x_1, \ldots, x_k)$ вычислима на машине Тьюринга по программе с кодом e, то нужно доказать, что имеет место представление в нормальной форме, т. е.

$$f(x_1, \ldots, x_k) = U(\mu y[T_k(e, x_1, \ldots, x_k, y)]).$$

Отсюда будет следовать, что f — ч.р.ф.

□

Существует несколько модернизаций и обобщений машин Тьюринга. Наиболее известной и естественной модернизацией является многоленточная машина Тьюринга, т. е. машина, которая имеет k ($k \geqslant 1$) лент и такое же число считывающих головок. Однако вычислительные возможности таких машин не отличаются от возможностей обычных машин Тьюринга.

Другое обобщение — недетерминированные машины Тьюринга, т. е. машины, в программе которых для некоторых состояний q_i и некоторых внешних символов a_j могут присутствовать несколько различных команд, начинающихся с символов $q_i a_j \to \ldots$ К машинам такого типа мы еще вернемся в главе, посвященной теории сложности вычислений.

§ 16. Нормальные алгорифмы Маркова

В данном параграфе будет предложено краткое введение в теорию нормальных алгорифмов Маркова, которые являются еще одной (в нашем курсе — уже четвертой) формализацией понятия вычислимости. Этот подход основан на том, что любые наборы данных и любые программы для алгоритмов, работающих с этими данными, могут быть представлены в виде некоторых текстов и даже в виде слов (если считать пробелы между словами, знаки препинания и символы перехода на новую строку полноправными буквами алфавита), а сам алгоритм может быть рассмотрен как процесс преобразования слов.

Таким образом, алгорифмы Маркова — это словарные алгоритмы, принцип действия которых напоминает работу формальных грамматик: так же как и в грамматиках, алгорифмы Маркова преобразуют слова путем замены одних подслов на другие. Однако есть ряд существенных отличий. По сравнению с формальными грамматиками нормальные алгорифмы Маркова имеют входные данные и зависящие от них выходные данные. Кроме этого, действия нормального алгорифма на каждом шаге строго детерминированны. Эта детерминированность достигается с помощью следующих двух принципов. Во-первых, на каждом шаге работы алгорифма для слова w однозначно определены подслово u и то вхождение u в w, которое будет подвергаться замене на некоторое другое слово. Во-вторых, однозначно определено то правило подстановки $u \to v$, с помощью которого осуществляется подобная замена. Заметим также, что на некоторых входных словах нормальные алгорифмы Маркова могут работать бесконечно, никогда не останавливаясь.

Перейдем теперь к точным формулировкам (см. [5], [6]).

Определение. Упорядоченная пара $\mathfrak{A} = \langle A, \Pi \rangle$ называется *нормальным алгорифмом в алфавите* A, если выполняются следующие условия:

1) A — конечный непустой алфавит, не содержащий символы $\to$ и $\to\cdot$

2) $\Pi = \langle \Phi_1, \ldots, \Phi_n \rangle$ — конечный упорядоченный список так называемых *формул подстановки (редукций)* в алфавите A, т. е. цепочек вида $P \to Q$ или $P \to\cdot Q$, где P, Q — некоторые (возможно, пустые) слова в алфавите A.

Редукция вида $P \to Q$ называется *простой*. Редукция вида $P \to\cdot Q$ называется *заключительной*.

Список П называется *схемой алгорифма* $\mathfrak{A}$, которую обычно записывают в виде

$$\begin{array}{c} P_1 \to (\cdot)Q_1 \\ \vdots \quad \vdots \\ P_n \to (\cdot)Q_n, \end{array}$$

где $P_i \to (\cdot)Q_i$ есть либо редукция $P_i \to Q_i$, либо редукция $P_i \to\cdot\, Q_i$. Порядок записи редукций в схеме алгорифма имеет значение!

Определение (шаг работы нормального алгорифма)**.** Пусть $\mathfrak{A}$ — нормальный алгорифм в алфавите A, P — слово в алфавите A. Возможны следующие два случая:

1) Ни одно слово $P_1, \ldots, P_n$ не является подсловом в P. Тогда пишем $\mathfrak{A}{:}P \sqsupset$ и говорим, что P *не поддается* нормальному алгорифму $\mathfrak{A}$.

2) Среди слов $P_1, \ldots, P_n$ существуют такие, которые являются подсловами в P. Пусть m — наименьшее такое, что $1 \leqslant m \leqslant n$ и P_m — подслово P. Определим слово R, которое получается из P заменой самого левого вхождения P_m на слово Q_m. При этом:

   а) если Φ_m имела вид $P_m \to Q_m$, то пишем $\mathfrak{A}{:}P \vdash R$ и говорим, что $\mathfrak{A}$ *просто переводит* P в R;

   б) если Φ_m имела вид $P_m \to\cdot\, Q_m$, то пишем $\mathfrak{A}{:}P \vdash\cdot\, R$ и говорим, что $\mathfrak{A}$ *заключительно переводит* P в R.

Определение. Пусть $\mathfrak{A}$ — нормальный алгорифм в алфавите A, P — слово в алфавите A. Говорят, что $\mathfrak{A}$ *преобразует слово* P *в слово* R и пишут $\mathfrak{A}(P) = R$, если существует конечная последовательность $R_0, R_1, \ldots, R_k$ слов в алфавите A такая, что $R_0 = P, R_k = R$ и выполняется одно из двух условий:

1) либо для любого $i = 0, 1, \ldots, k-1$ выполняется $\mathfrak{A}{:}R_i \vdash R_{i+1}$ и $\mathfrak{A}{:}R_k \sqsupset$;

2) либо для любого $i = 0, 1, \ldots, k-2$ выполняется $\mathfrak{A}{:}R_i \vdash R_{i+1}$ и $\mathfrak{A}{:}R_{k-1} \vdash\cdot\, R_k$.

Обозначим $\operatorname{dom}\mathfrak{A} = \{P \in A^* \mid \exists R\ \mathfrak{A}(P) = R\}$. Говорят, что $\mathfrak{A}$ *применим к слову* P, если $P \in \operatorname{dom}\mathfrak{A}$; в противном случае говорят, что $\mathfrak{A}$ *не применим к слову* P.

Замечание. Алгорифм $\mathfrak{A}$ не применим к слову P тогда и только тогда, когда существует бесконечная цепочка $\mathfrak{A} : P \vdash P_1 \vdash P_2 \vdash \ldots$

Определение. Частичная функция $f : \operatorname{dom}(f) \subseteq \omega^n \longrightarrow \omega$ называется *вычислимой по Маркову*, если существует нормальный алгорифм $\mathfrak{A}$ в некотором алфавите $A \supseteq \{0, 1\}$ такой, что:

а) если $(x_1, \ldots, x_n) \in \operatorname{dom}(f)$, то $\mathfrak{A}$ применим к слову $1^{x_1+1}01^{x_2+1}0\ldots01^{x_n+1}$ и при этом выполняется $\mathfrak{A}(1^{x_1+1}01^{x_2+1}0\ldots01^{x_n+1}) = 1^{f(x_1,\ldots,x_n)+1}$;

б) если $(x_1, \ldots, x_n) \notin \operatorname{dom}(f)$, то $\mathfrak{A}$ не применим к слову $1^{x_1+1}01^{x_2+1}0\ldots01^{x_n+1}$;

в) алгорифм $\mathfrak{A}$ не применим к словам в алфавите $\{0, 1\}$, отличным от слов вида $1^{x_1+1}01^{x_2+1}0\ldots01^{x_n+1}$.

(Применимость алгорифма $\mathfrak{A}$ ко всем остальным словам в алфавите A не имеет значения.)

Пример. Докажем, что функция $\mathrm{sg}(x)$ вычислима по Маркову. Для этого определим нормальный алгорифм $\mathfrak{A}$ в алфавите $\{0, 1, \alpha\}$, где $\alpha \neq 0$, $\alpha \neq 1$, по следующей схеме:

$$\begin{gathered} 0 \to 0 \\ \alpha 111 \to \alpha 11 \\ \alpha 11 \to\cdot\, 11 \\ \alpha 1 \to\cdot\, 1 \\ \Lambda \to \alpha \end{gathered}$$

Если $P = 1^{x+1}$, где $x > 0$, то $\mathfrak{A} : 1^{x+1} \vdash \alpha 1^{x+1} \vdash \alpha 1^x \vdash \ldots \vdash \alpha 11 \vdash\cdot\, 11$, т. е. $\mathfrak{A}$ применим к слову 1^{x+1}, и $\mathfrak{A}(1^{x+1}) = 11 = 1^{\mathrm{sg}(x)+1}$.

Если $P = 1$, то $\mathfrak{A} : 1 \vdash \alpha 1 \vdash\cdot\, 1$, т. е. $\mathfrak{A}$ применим к слову 1, и $\mathfrak{A}(1) = 1$.

Если P — слово в алфавите $\{0, 1\}$, отличное от 1^{x+1}, то либо $P = \Lambda$, либо P содержит 0. Покажем, что в обоих случаях $\mathfrak{A}$ не применим к P.

Если $P = \Lambda$, то $\mathfrak{A} : \Lambda \vdash \alpha \vdash \alpha\alpha \vdash \alpha\alpha\alpha \vdash \ldots$, т. е. будет бесконечно часто применяться последняя редукция.

Если P содержит 0, то $P = 1^n 0 w$, где $n \in \omega$, $w \in \{0, 1\}^*$. Тогда $\mathfrak{A} : 1^n 0 w \vdash 1^n 0 w \vdash 1^n 0 w \vdash \ldots$, т. е. будет бесконечно часто применяться первая редукция.

Таким образом, предложенный выше нормальный алгорифм $\mathfrak{A}$ вычисляет функцию $\mathrm{sg}(x)$.

Сформулируем теперь основной интересующий нас результат из теории нормальных алгорифмов.

Теорема 43. *Частичная функция вычислима по Маркову тогда и только тогда, когда она является частично рекурсивной.*

Общая схема доказательства теоремы 43 такая же, как для машин Шёнфилда и машин Тьюринга, поэтому здесь мы его не приводим. Подробности можно найти, например, в [6].

В заключение, заметим, что употребление буквы «ф» в термине «нормальный алгорифм» является традиционным при изложении теории нормальных алгоритмов. Эта традиция основана на произношении слова «algorithm». Тем не менее, в некоторых источниках встречается термин «нормальный алгоритм».

§ 17. Тезис Чёрча

Итак, мы рассмотрели четыре различные формализации понятия вычислимой функции: частично рекурсивные функции, машины Шёнфилда, машины Тьюринга и нормальные алгорифмы Маркова. Данные подходы отталкиваются от разных принципов и имеют не похожие друг на друга реализации (за исключением, быть может, подходов Тьюринга и Шёнфилда). Более того, существуют другие, менее распространенные формализации понятия вычислимости. Однако все известные формальные подходы порождают один и тот же класс функций, в чем мы уже успели частично убедиться. Сформулируем основное утверждение данной главы.

Теорема 44. *Для произвольной частичной функции* $f : \mathrm{dom}(f) \subseteq \omega^n \longrightarrow \omega$ *следующие условия эквивалентны:*

1) f *является частично рекурсивной функцией;*
2) f *вычислима на машине Шёнфилда;*
3) f *вычислима по Тьюрингу;*
4) f *вычислима по Маркову.*

Доказательство. Следует из теорем 16, 34, 42, 43. □

Поэтому в дальнейшем для нас не будет иметь значения, какая из данных формализаций выбрана. Объединим рассмотренные нами четыре формализации следующим определением.

Определение. Частичную функцию f, удовлетворяющую условиям (1)–(4) теоремы 44, будем называть *частично вычислимой (ч.в.ф.)*. Всюду определенную функцию f, удовлетворяющую условиям (1)–(4) теоремы 44, будем называть *вычислимой (в.ф.)*.

Таким образом, можно считать, что первоначальная цель, которая послужила отправной точкой для развития теории вычислимости, достигнута — найдено формальное определение вычислимой функции. Теорему 44 можно воспринимать как подтверждение высказывания, которое обычно называют тезисом Чёрча.

Тезис Чёрча. *Класс интуитивно вычислимых функций совпадает с классом всех частично рекурсивных функций.*

Тезис Чёрча — это высказывание, которое говорит, что предложенная формализация понятия вычислимой функции адекватно отражает наши интуитивные представления о вычислимости. Исторически именно этот тезис был первым точным определением частично вычислимой функции.

Тезис Чёрча не является формальным математическим утверждением. Однако при доказательстве теорем он часто используется в неявном виде. Как правило, это происходит следующим образом: чтобы доказать существование частично рекурсивной функции с определенными свойствами, в математических рассуждениях может быть показана ее интуитивная вычислимость, а строгое доказательство частичной рекурсивности при этом опускается. Такой прием позволяет сократить объем доказательства, но автор любого такого доказательства должен быть готов представить формальный вариант доказательства, не использующий тезис Чёрча.

Глава IV

Теория вычислимости

Данная глава является *введением* в теорию вычислимости и содержит фундаментальные понятия и теоремы из нескольких, ставших уже классическими, разделов этой богатейшей теории. Для дальнейшего изучения теории вычислимости можно порекомендовать книги [1], [5], [9], [10].

Следует особо подчеркнуть, что все результаты данной главы не зависят от выбора конкретного формального определения частично вычислимой функции. Для нас будет иметь значение только факт существования *эффективной* клиниевской нумерации $æ_e^k$ для класса всех k-местных частично вычислимых функций. Под эффективностью здесь мы понимаем справедливость s-m-n-теоремы, которая связывает нумерации $æ_e^k$ для различных k.

Можно интуитивно воспринимать $æ_n$ как частичную функцию, вычисляемую n-м алгоритмом при некотором эффективном перечислении всех алгоритмов. Однако нужно помнить, что подобный неформальный подход, так же как и рассуждения, использующие тезис Чёрча, всегда требуют формального обоснования. Мы чаще всего будем использовать формализм частично рекурсивных функций.

§ 18. Теорема о неподвижной точке

Теорема о неподвижной точке (другое название — теорема о рекурсии) была доказана Клини и является одним из наиболее элегантных и важных результатов теории вычислимости. Ее короткое доказательство использует s-m-n-теорему и на первый взгляд кажется несколько мистическим. Переформулируем s-m-n-теорему в новых, введенных нами выше терминах.

s-m-n-теорема. *Для любых $m, n \in \omega$ существует разнозначная $(m+1)$-местная вычислимая функция $s_n^m(e, y_1, \ldots, y_m)$ такая, что для всех $e, y_1, \ldots, y_m, x_1, \ldots, x_n$ выполняется*

$$æ_e^{m+n}(y_1, \ldots, y_m, x_1, \ldots, x_n) = æ^n_{s_n^m(e, y_1, \ldots, y_m)}(x_1, \ldots, x_n).$$

Конечно, в первоначальном варианте s-m-n-теоремы утверждалось, что функцию s_n^m можно выбрать даже примитивно рекурсивной. Однако для наших дальнейших целей потребуется только ее вычислимость. Как и раньше, через $\overline{x}$ мы обозначаем кортеж $\langle x_1, \ldots, x_k \rangle$.

Теорема 45 (о неподвижной точке)**.** *Справедливы следующие два утверждения:*

1) Для любой частично вычислимой функции $f(\overline{x}, y)$ существует вычислимая функция $g(\overline{x})$ такая, что

$$æ_{f(\overline{x}, g(\overline{x}))} = æ_{g(\overline{x})}.$$

2) Для любой частично вычислимой функции $f(x)$ существует $a \in \omega$ такое, что

$$æ_{f(a)} = æ_a.$$

Доказательство. Для доказательства первого утверждения рассмотрим частично вычислимую функцию $F(e, y, \overline{x}, z) = æ_e^{k+2}(y, \overline{x}, z)$, которая является универсальной для семейства всех $(k{+}2)$-местных ч.в.ф. По s-m-n-теореме существует вычислимая функция $s(e, y, \overline{x})$ такая, что

$$æ_e^{k+2}(y, \overline{x}, z) = æ_{s(e,y,\overline{x})}(z). \qquad (*)$$

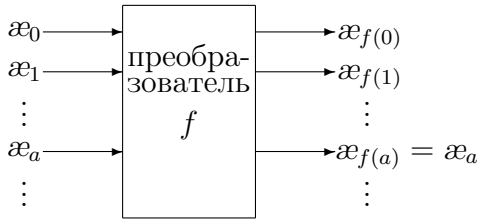
Функция $æ_{f(\overline{x},s(y,y,\overline{x}))}(z)$ является $(k{+}2)$-местной ч.в.ф. от переменных $\langle y, \overline{x}, z\rangle$. Следовательно, в силу универсальности найдется клиниевский номер $n \in \omega$ такой, что

$$æ_{f(\overline{x},s(y,y,\overline{x}))}(z) = æ_n^{k+2}(y, \overline{x}, z). \qquad (**)$$

Из $(*)$ и $(**)$ при $e = n$ следует, что $æ_{f(\overline{x},s(y,y,\overline{x}))}(z) = æ_{s(n,y,\overline{x})}(z)$. Подставив в полученное тождество $y = n$, получим $æ_{f(\overline{x},s(n,n,\overline{x}))}(z) = æ_{s(n,n,\overline{x})}(z)$. Отсюда видно, что вычислимая функция $g(\overline{x}) = s(n, n, \overline{x})$ является искомой.

Второе утверждение является вариантом первого в случае, когда кортеж $\overline{x}$ пуст. Его доказательство дословно повторяет рассуждения, проделанные выше. Разница лишь в том, что в итоге вместо функции $g(\overline{x}) = s(n, n, \overline{x})$ мы получим число $a = s(n, n)$. □

В утверждении (1) доказывается существование неподвижной точки с параметрами, утверждение (2) — классическая теорема о неподвижной точке. Функцию $f(x)$ из пункта (2) можно неформально представлять как эффективный преобразователь программ, который любую программу n, реализующую процедуру $æ_n$, перерабатывает в программу $f(n)$, реализующую, вообще говоря, какую-то другую процедуру $æ_{f(n)}$. Интуитивный смысл теоремы о неподвижной точке заключается в том, что для любого такого преобразователя найдется хотя бы одна процедура $æ_a$, которая не изменяется под его действием, т. е. $æ_{f(a)} = æ_a$. Другими словами, у любого такого преобразователя найдется неподвижная точка-процедура.



Пример. В программировании известна популярная задача-головоломка о написании программ, воспроизводящих свой собственный текст (такие программы называют словом «quine»). Например, текст такой программы на языке Паскаль выглядит следующим образом:

```
PROGRAM FIXEDPOINT;
TYPE
  STR=STRING[255];
VAR
```

```
  C:ARRAY[0..15] OF STR;
  I:INTEGER;
BEGIN
  C[ 0]:='''';
  C[ 1]:='  C[';
  C[ 2]:=']:=';
  C[ 3]:=';';
  C[ 4]:='PROGRAM FIXEDPOINT;';
  C[ 5]:='TYPE';
  C[ 6]:='  STR=STRING[255];';
  C[ 7]:='VAR';
  C[ 8]:='  C:ARRAY[0..15] OF STR;';
  C[ 9]:='  I:INTEGER;';
  C[10]:='BEGIN';
  C[11]:='  FOR I:=4 TO 10 DO WRITELN(C[I]);';
  C[12]:='  WRITELN(C[1],0:2,C[2],C[0],C[0],C[0],C[0],C[3]);';
  C[13]:='  FOR I:=1 TO 15 DO WRITELN(C[1],I:2,C[2],C[0],C[I],C[0],C[3]);';
  C[14]:='  FOR I:=11 TO 15 DO WRITELN(C[I]);';
  C[15]:='END.';
  FOR I:=4 TO 10 DO WRITELN(C[I]);
  WRITELN(C[1],0:2,C[2],C[0],C[0],C[0],C[0],C[3]);
  FOR I:=1 TO 15 DO WRITELN(C[1],I:2,C[2],C[0],C[I],C[0],C[3]);
  FOR I:=11 TO 15 DO WRITELN(C[I]);
END.
```

Оказывается, существование подобных программ близко связано с теоремой о неподвижной точке. В определенном смысле, если для языка программирования справедлива теорема о параметризации, то можно написать программу на данном языке, которая выводит свой собственный код.

Докажем существование подобной программы для языка машин Шёнфилда, т. е. необходимо показать, что существует программа P с кодом e, которая после запуска всегда останавливается и выдает (в нулевом регистре) свой собственный код e. Для этого рассмотрим вычислимую функцию $g(x, y) = x$ и применим к ней теорему о параметризации — получим некоторую вычислимую функцию $f(x)$ такую, что $g(x, y) = æ_{f(x)}(y)$. Далее применим теорему о неподвижной точке к функции $f(x)$ — получим число $a \in \omega$ такое, что $æ_a = æ_{f(a)}$. Из доказательства теоремы о неподвижной точке видно, что $a = s(n, n)$ для некоторой s-m-n-функции s, а из доказательства теоремы о параметризации вытекает, что любая s-m-n-функция всегда в качестве своих значений выдает коды некоторых программ. Отсюда заключаем, что найденное a является кодом программы, т. е. предикат $\mathrm{Prog}(a)$ истиннен, и программа с кодом a вычисляет функцию

$$æ_a(y) = æ_{f(a)}(y) = g(a, y) = a,$$

т. е. выдает свой собственный код a. □

Теорема о неподвижной точке имеет очень важное следствие, которое утверждает, что никакое нетривиальное (т. е. присущее некоторым, но не всем функциям) свойство частично вычислимых функций не может быть эффективно распознаваемым по их клиниевским номерам. Для доказательства данного утверждения нам понадобится следующее определение, которое является переформулировкой определения рекурсивного отношения в терминах вычислимых функций.

Определение. Множество $A \subseteq \omega^n$ называется *вычислимым*, если его характеристическая функция $\mathcal{X}_A(x_1, \ldots, x_n)$ является вычислимой.

Теорема 46 (теорема Райса)**.** *Пусть K — произвольное семейство одноместных частично вычислимых функций, такое, что $K \neq \varnothing$ и K не совпадает с семейством всех одноместных частично вычислимых функций. Тогда множество $\{x \in \omega \mid æ_x \in K\}$ всех клиниевских номеров функций, принадлежащих семейству K, невычислимо.*

Доказательство. Допустим, напротив, множество $A = \{x \in \omega \mid æ_x \in K\}$ вычислимо, т. е. вычислимой является его характеристическая функция

$$\mathcal{X}_A(x) = \begin{cases} 1, & \text{если } x \in A, \\ 0, & \text{если } x \notin A. \end{cases}$$

Так как $K \neq \varnothing$ и $K \neq \{f \mid f$ — 1-местная ч.в.ф.$\}$, то найдутся $a, b \in \omega$ такие, что $æ_a \in K$ и $æ_b \notin K$. Определим функцию

$$f(x) = \begin{cases} b, & \text{если } x \in A, \\ a, & \text{если } x \notin A. \end{cases}$$

Поскольку $f(x) = b \cdot \mathcal{X}_A(x) + a \cdot \overline{\text{sg}}(\mathcal{X}_A(x))$, то $f(x)$ вычислима. По теореме о неподвижной точке существует $n \in \omega$ такое, что $æ_{f(n)} = æ_n$. Тогда имеет место следующая цепочка эквивалентностей (вторая эквивалентность следует из выбора a и b, третья эквивалентность следует из определения функции $f(x)$):

$$æ_n \in K \iff æ_{f(n)} \in K \iff f(n) = a \iff n \notin A \iff æ_n \notin K.$$

Противоречие. Следовательно, наше предположение о вычислимости множества $\{x \in \omega \mid æ_x \in K\}$ неверно. □

Пример. Пусть $f(x)$ — произвольная одноместная ч.в.ф. Рассмотрим одноэлементное семейство функций $K = \{f\}$. Оно удовлетворяет условиям теоремы Райса, следовательно, множество $A = \{e \in \omega \mid æ_e = f\}$ не вычислимо. Заметим, что множество A — это в точности множество всех клиниевских номеров функции f. Таким образом, из теоремы Райса следует, что *множество всех клиниевских номеров фиксированной ч.в.ф. не вычислимо.* В частности, любая ч.в.ф. имеет бесконечно много клиниевских номеров (поскольку, очевидно, любое конечное множество вычислимо).

§ 19. Нумерации и алгоритмические проблемы

Объектами классической теории вычислимости являются функции вида $f : A \to \omega$, где $A \subseteq \omega^n$. Для таких объектов мы ввели понятие вычислимости и получили целый ряд важных результатов. Однако алгоритмические проблемы, возникающие в алгебре, математической логике и других разделах математики, формулируются для абстрактных совокупностей объектов, традиционно используемых в этих разделах. Чтобы говорить о вычислимости алгебраических структур, разрешимости теорий и конструктивизируемости топологических пространств, необходимо ввести понятие вычислимости для подобных объектов. Эта необходимость приводит нас к понятию нумерации, в основе которого лежит очень естественная идея: нужно занумеровать (если это возможно) все объекты из рассматриваемой совокупности, отождествив каждый объект с натуральным числом, и в дальнейшем говорить не о самих объектах, а об их номерах, для которых уже развита теория вычислимости.

Определение. Пусть S — не более чем счетное множество. Любое сюръективное отображение $\nu : \omega \xrightarrow{\text{на}} S$ называется *нумерацией* множества S.

Определение. Пусть ν — нумерация. Отношение $\eta_\nu = \{\langle x, y\rangle \in \omega^2 \mid \nu(x) = \nu(y)\}$ называется *нумерационной эквивалентностью* нумерации ν.

Определение (типы нумераций)**.** Нумерация $\nu : \omega \xrightarrow{\text{на}} S$ множества S называется:

а) *однозначной*, если $\eta_\nu = \{\langle x, x\rangle \mid x \in \omega\}$, т. е. ν — биекция;

б) *разрешимой*, если η_ν вычислимо;

в) *позитивной*, если существует вычислимая функция, область значений которой есть множество кодов пар из η_ν;

г) *негативной*, если существует вычислимая функция, область значений которой есть множество кодов пар из $\omega^2 \setminus \eta_\nu$, или $\omega^2 = \eta_\nu$ (т. е. S одноэлементно).

Замечание. Нумерация ν позитивна тогда и только тогда, когда множество η_ν вычислимо перечислимо. Нумерация ν негативна тогда и только тогда, когда множество $\omega^2 \setminus \eta_\nu$ вычислимо перечислимо (определение вычислимо перечислимого множества см. в следующем параграфе).

Пример. 1) Отображение $æ : \omega \to \{f \mid f$ — одноместная ч.в.ф.$\}$, определенное по правилу $æ(n) = æ_n$, является нумерацией множества всех одноместных частично вычислимых функций. Эта нумерация не является однозначной, поскольку любая ч.в.ф. имеет бесконечно много клиниевских номеров. Более того, данная нумерация не является ни разрешимой, ни позитивной, ни негативной. Нумерацию $æ$ называют *клиниевской*.

2) Отображение $\nu : \omega \to \omega^2$, определенное по правилу $\nu(n) = \langle (n)_0, (n)_1\rangle$, является нумерацией множества всех пар натуральных чисел. Данная нумерация не является однозначной, но является разрешимой.

3) Если $A = \{a_0, \ldots, a_{n-1}\}$ — конечное n-элементное множество, где $n \geqslant 1$, то отображение $\nu(x) = a_{\text{rest}(x,n)}$, где $\text{rest}(x, n)$ — остаток от деления x на n, является нумерацией множества A. Данная нумерация не является однозначной, но является разрешимой.

4) Пусть $P_{\text{fin}}(\omega)$ — множество всех конечных подмножеств натуральных чисел. Определим нумерацию $\gamma : \omega \to P_{\text{fin}}(\omega)$ следующим образом. Если $n = 0$, то полагаем $\gamma(0) = \varnothing$. Если же $n \geqslant 1$, то для него существует единственное двоичное представление, т. е. найдутся такие $x_k > \ldots > x_0$, что $n = 2^{x_k} + \ldots + 2^{x_0}$. Тогда полагаем $\gamma(n) = \{x_0, \ldots, x_k\}$. Нумерация γ является однозначной.

Определение. Пусть $\nu : \omega \xrightarrow{\text{на}} S$ — нумерация множества S. Любое подмножество $S_0 \subseteq S$ называется *алгоритмической проблемой* относительно нумерации ν. Эта проблема называется *разрешимой* относительно ν, если множество $\{x \in \omega \mid \nu(x) \in S_0\}$ вычислимо.

Пример. 1) Из теоремы Райса следует, что все нетривиальные алгоритмические проблемы на множестве $\{f \mid f$ — одноместная ч.в.ф.$\}$ неразрешимы относительно клиниевской нумерации.

2) Проблема $\{\langle x,y\rangle \mid x \leqslant y\}$ на множестве ω^2 разрешима относительно нумерации $\nu(n) = \langle (n)_0, (n)_1\rangle$, т. е. по номеру n пары натуральных чисел $\langle x,y\rangle$ можно эффективно проверить, выполняется ли условие $x \leqslant y$ или нет. Эта проверка осуществляется с помощью вычислимой функции $\mathcal{X}(n) = \overline{\mathrm{sg}}((n)_0 \dot{-} (n)_1)$.

3) Любая алгоритмическая проблема на конечном множестве $A = \{a_0, \ldots, a_{n-1}\}$ разрешима относительно нумерации $\nu(x) = a_{\mathrm{rest}(x,n)}$. Действительно, если $A_0 = \{a_{k_0}, \ldots, a_{k_s}\}$ непустое подмножество A, то множество $\{x \in \omega \mid \nu(x) \in A_0\}$ совпадает с множеством $\bigcup\limits_{i=0}^{s} \{n\cdot t + k_i \mid t \in \omega\}$, которое, очевидно, вычислимо.

4) Пусть $S_0 = \{X \subseteq \omega \mid X$ — одноэлементное$\} \subseteq P_{\mathrm{fin}}(\omega)$. Проблема S_0 разрешима относительно нумерации γ. Нетрудно видеть, что натуральное n является γ-номером одноэлементного множества тогда и только тогда, когда n является степенью двойки. Следовательно, характеристическую функцию множества $A = \{n \in \omega \mid \gamma(n)$ — одноэлементное$\}$ можно записать в виде $\mathcal{X}_A(n) = \overline{\mathrm{sg}}(|n - 2^{\mathrm{ex}(0,n)}|)$, т. е. A вычислимо.

Заметим, что любое подмножество $S_0 \subseteq \omega$ можно рассматривать как алгоритмическую проблему относительно тождественной нумерации $\nu(x) = x$. В этом случае разрешимость проблемы S_0 эквивалентна вычислимости множества S_0. В следующей важной теореме мы как раз рассматриваем проблему на множестве ω относительно тождественной нумерации.

Напомним, что для частичной функции f запись $f(x)\downarrow$ означает, что значение $f(x)$ определено, а запись $f(x)\uparrow$ означает, что $f(x)$ не определено.

Теорема 47. *Множество $K = \{x \in \omega \mid æ_x(x)\downarrow\}$ невычислимо.*

Доказательство. Допустим, напротив, множество K вычислимо. Следовательно, $\mathcal{X}_K(x)$ вычислима. Рассмотрим одноместную функцию

$$f(x) = \begin{cases} 0, & \text{если } æ_x(x)\uparrow \\ \text{не определено}, & \text{если } æ_x(x)\downarrow \end{cases}$$

Так как $f(x) = \mu y[\mathcal{X}_K(x) = 0 \;\&\; y = 0]$, то $f(x)$ частично вычислима. По теореме об универсальной ч.р.ф. существует клиниевский номер $n \in \omega$ такой, что $f(x) = æ_n(x)$. Рассмотрим значение аргумента $x = n$, получим следующую цепочку эквивалентных условий:

$$æ_n(n)\downarrow \iff f(n)\downarrow \iff æ_n(n)\uparrow .$$

Противоречие. □

Пусть S — множество всех 1-местных ч.в.ф. Определим нумерацию $\nu : \omega \to S\times\omega$, положив $\nu(n) = \langle æ_{(n)_0}, (n)_1\rangle$ для каждого $n \in \omega$. *Проблемой остановки* мы будем называть множество $\{\langle f,x\rangle \in S \times \omega \mid f(x)\downarrow\}$. Нетрудно понять, что разрешимость проблемы остановки относительно введенной нумерации ν равносильна вычислимости множества $\{\langle x,y\rangle \in \omega^2 \mid æ_x(y)\downarrow\}$.

Следствие 48 (о неразрешимости проблемы остановки)**.** *Множество $\{\langle x,y\rangle \in \omega^2 \mid æ_x(y)\downarrow\}$ невычислимо.*

Доказательство. Допустим $A = \{\langle x,y\rangle \mid æ_x(y)\downarrow\}$ вычислимо. Тогда $\mathcal{X}_A(x,y)$ вычислима. Следовательно, функция $f(x) = \mathcal{X}_A(x,x)$ тоже вычислима. Заметим, что $f(x) = 1 \iff \mathcal{X}_A(x,x) = 1 \iff \langle x,x\rangle \in A \iff æ_x(x)\downarrow \iff x \in K$. Таким образом, функция $\mathcal{X}_K(x) = f(x)$ вычислима, что противоречит теореме 47. □

Замечание. Заметим, что T-предикат Клини $T_1(e,x,y)$ ложен, если e не является кодом программы. Поэтому из условия $æ_e(x)\downarrow$ вытекает, что e должен быть кодом программы для машины Шёнфилда. Отсюда следует, что множество из формулировки следствия 48 совпадает с множеством

$$\{\langle e,x\rangle \mid e \text{ — код программы для машины Шёнфилда, которая останавливается на входе } x\},$$

а утверждение следствия 48 означает, что не существует алгоритма, который по заданному коду программы и входным данным определяет, останавливается ли данная программа на заданном входе или нет.

В различных разделах математики очень часто одну задачу сводят к другой задаче так, что если удается решить вторую задачу, то полученное решение некоторым эффективным образом трансформируется в решение первой задачи.

В терминах теории нумераций сводимость одной нумерации ν_0 к другой нумерации ν означает, что нумерацию ν можно эффективно преобразовать в нумерацию ν_0 так, что из разрешимости любой проблемы в нумерации ν будет следовать разрешимость этой же проблемы в нумерации ν_0.

Формальное определение сводимости нумераций выглядит следующим образом.

Определение. Пусть заданы две нумерации $\nu_0 : \omega \xrightarrow{\text{на}} S_0$ и $\nu : \omega \xrightarrow{\text{на}} S$, где $S_0 \subseteq S$. Говорят, что ν_0 *сводится* к ν, и пишут $\nu_0 \leqslant \nu$, если существует вычислимая функция $f(x)$ такая, что $\nu_0(x) = \nu(f(x))$ для каждого $x \in \omega$. Другими словами, коммутативна следующая диаграмма

$$\begin{array}{ccc} S_0 & \xrightarrow{\text{id}} & S \\ {\scriptstyle\nu_0}\uparrow & & \uparrow{\scriptstyle\nu} \\ \omega & \xrightarrow[f]{} & \omega \end{array}$$

Говорят, что нумерации ν_0 и ν *эквивалентны*, и пишут $\nu_0 \equiv \nu$, если $\nu_0 \leqslant \nu$ и $\nu \leqslant \nu_0$. (В частности, если $\nu_0 \equiv \nu$, то $S_0 = S$.)

Теорема 49 (о разрешимых нумерациях бесконечных множеств)**.** *Любая разрешимая нумерация бесконечного множества эквивалентна некоторой однозначной нумерации.*

Доказательство. Пусть $\nu : \omega \xrightarrow{\text{на}} S$ — разрешимая нумерация бесконечного множества S. Следовательно, нумерационная эквивалентность $A = \{\langle x,y\rangle \mid \nu(x) = \nu(y)\}$ вычислима, т. е. функция $\mathcal{X}_A(x,y)$ вычислима.

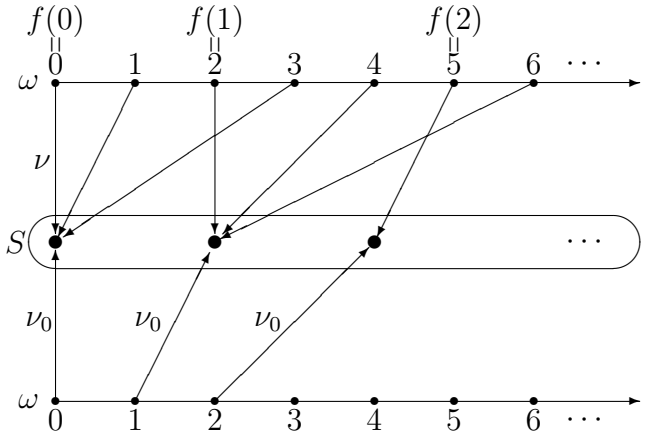
Определим функцию f по следующей схеме (см. рисунок):

$$\begin{cases} f(0) = 0, \\ f(n+1) = \mu x[\nu(x) \neq \nu f(0)\ \&\ \ldots\ \&\ \nu(x) \neq \nu f(n)]. \end{cases}$$

В силу бесконечности S функция f всюду определена. Запишем схему более формально:

$$\begin{cases} f(0) = 0, \\ f(n+1) = \mu x[\sum\limits_{i=0}^{n} \mathcal{X}_A(x, f(i)) = 0]. \end{cases}$$

Поэтому из вычислимости функции $\mathcal{X}_A$ следует, что f тоже вычислима. (Упражнение: доказать, что введенная схема сводится к схеме примитивной рекурсии.)



Определим нумерацию $\nu_0(n) = \nu f(n)$. Ясно, что $\mathrm{range}(\nu_0) \subseteq S$. По построению для любого $n \in \omega$ выполняются условия $\nu_0(n+1) \neq \nu_0(0), \ldots, \nu_0(n+1) \neq \nu_0(n)$, следовательно, ν_0 однозначна. Кроме этого, очевидно, что $\nu_0 \leqslant \nu$.

Докажем, что $S \subseteq \mathrm{range}(\nu_0)$. Для этого индукцией по n покажем, что имеет место включение $\{\nu(0), \ldots, \nu(n)\} \subseteq \{\nu_0(0), \ldots, \nu_0(n)\}$. При $n = 0$ по построению $\nu_0(0) = \nu(f(0)) = \nu(0)$.

Пусть включение $\{\nu(0), \ldots, \nu(n)\} \subseteq \{\nu_0(0), \ldots, \nu_0(n)\}$ уже доказано. Рассмотрим элемент $s = \nu(n+1)$. Если $s \in \{\nu_0(0), \ldots, \nu_0(n)\} \subseteq \{\nu_0(0), \ldots, \nu_0(n), \nu_0(n+1)\}$, то доказывать нечего. Если же $s \notin \{\nu_0(0), \ldots, \nu_0(n)\}$, то $\nu(n+1) \neq \nu f(0), \ldots, \nu(n+1) \neq \nu f(n)$ и по индукционному предположению $n+1$ является наименьшим натуральным числом с таким условием. Следовательно, по определению функции f заключаем, что $f(n+1) = n+1$, а значит, $\nu(n+1) = \nu_0(n+1)$ и $\{\nu(0), \ldots, \nu(n+1)\} \subseteq \{\nu_0(0), \ldots, \nu_0(n+1)\}$. Что и требовалось доказать.

Итак, $\{\nu(0), \ldots, \nu(n)\} \subseteq \{\nu_0(0), \ldots, \nu_0(n)\}$ для всех $n \in \omega$. А поскольку $\mathrm{range}(\nu) = S$, то отсюда следует, что $S \subseteq \mathrm{range}(\nu_0)$. Таким образом, $S = \mathrm{range}(\nu_0)$.

Докажем, наконец, что $\nu \leqslant \nu_0$. Определим вычислимую функцию

$$g(x) = \mu n[\nu(f(n)) = \nu(x)] = \mu n[\mathcal{X}_A(f(n), x) = 1].$$

Пусть $x \in \omega$. Тогда найдется минимальный n такой, что $\nu(x) = \nu_0(n)$. Но $\nu_0(n) = \nu(f(n))$, следовательно, $g(x) = n$. Отсюда заключаем, что $\nu(x) = \nu_0(g(x))$. Таким образом, $\nu \leqslant \nu_0$. □

§ 20. Вычислимо перечислимые множества

В этом параграфе мы введем понятие вычислимо перечислимого множества (сокращенно в.п. множества) и изучим некоторые свойства таких множеств. С интуитивной точки зрения множество является вычислимо перечислимым, если существует алгоритм эффективного перечисления всех его элементов. При этом мы допускаем, что это перечисление может иметь повторения и не обязано быть перечислением в каком-то строго определенном порядке.

Определение. Пусть $k \geqslant 1$. Множество $A \subseteq \omega^k$ называется *вычислимо перечислимым (в.п.)*, если $A = \varnothing$ или $A = \{\langle f_1(x), \ldots, f_k(x)\rangle \mid x \in \omega\}$ для некоторых вычислимых функций $f_1(x), \ldots, f_k(x)$.

Другими словами, вычислимые функции $f_1, \ldots, f_k$ покоординатно перечисляют множество A. В случае $k = 1$ определение выглядит проще: множество $A \subseteq \omega$ является в.п. тогда и только тогда, когда $A = \varnothing$ или $A = \mathrm{range}(f)$ для некоторой вычислимой функции $f(x)$. Введенное определение является формальным описанием интуитивного понятия перечислимости. Однако существует несколько эквивалентных описаний в.п. множеств, каждое из которых оказывается полезным при изучении тех или иных свойств в.п. множеств.

Теорема 50 (об эквивалентных определениях в.п. множеств)**.**
Для произвольного множества $A \subseteq \omega^k$ следующие условия эквивалентны:

1) A вычислимо перечислимо.

2) Существует вычислимое отношение $R \subseteq \omega^{k+1}$ такое, что

$$\langle x_1, \ldots, x_k \rangle \in A \iff \exists y R(x_1, \ldots, x_k, y).$$

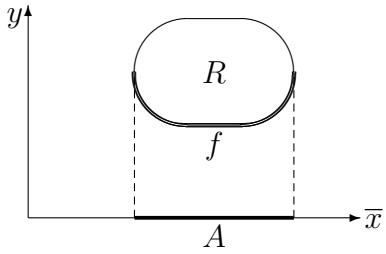
3) Существует частично вычислимая функция $f(x_1, \ldots, x_k)$ такая, что $A = \mathrm{dom}(f)$.

Доказательство. (1) $\Rightarrow$ (2) Если $A = \varnothing$, то вычислимое множество $R = \varnothing$ очевидно удовлетворяет условию (2). Если же $A \neq \varnothing$ и $A = \{\langle f_1(x), \ldots, f_k(x) \rangle \mid x \in \omega\}$, где $f_1, \ldots, f_k$ — вычислимые функции, то имеет место эквивалентность

$$\langle x_1, \ldots, x_k \rangle \in A \iff \exists y \big(f_1(y) = x_1 \,\&\, \ldots \,\&\, f_k(y) = x_k \big).$$

Тогда множество $R = \{\langle x_1, \ldots, x_k, y \rangle \mid f_1(y) = x_1 \,\&\, \ldots \,\&\, f_k(y) = x_k\}$ является искомым $(k{+}1)$-местным вычислимым отношением.

(2) $\Rightarrow$ (3) Пусть $R \subseteq \omega^{k+1}$ — вычислимое отношение такое, что A является его проекцией (см. рисунок), т. е. имеет место эквивалентность $\overline{x} \in A \iff \exists y R(\overline{x}, y)$.



Определим частичную k-местную функцию $f(\overline{x}) = \mu y[R(\overline{x}, y)]$. Так как R вычислимо, то $f(\overline{x})$ — ч.в.ф. Кроме этого, имеет место

$$f(\overline{x}) \downarrow \iff \exists y R(\overline{x}, y) \iff \overline{x} \in A.$$

Другими словами, $\mathrm{dom}(f) = A$.

(3) $\Rightarrow$ (1) Если $A = \varnothing$, то доказывать нечего. Пусть $A \neq \varnothing$. Следовательно, найдется кортеж $\overline{a} = \langle a_1, \ldots, a_k \rangle \in A$. По условию $A = \mathrm{dom}(f)$, где f — ч.в.ф. Тогда f вычислима на некоторой машине Шёнфилда с кодом e. По теореме Клини о нормальной форме $f(\overline{x}) = U(\mu y[T_k(e, \overline{x}, y)])$, где U и T_k — вычислимые. Для каждого $i \in \{1, \ldots, k\}$ определим вычислимую функцию

$$f_i(n) = \begin{cases} (n)_i, & \text{если } T_k(e, (n)_1, \ldots, (n)_k, (n)_0), \\ a_i, & \text{если } \neg T_k(e, (n)_1, \ldots, (n)_k, (n)_0). \end{cases}$$

Покажем, что набор функций $f_1, \ldots, f_k$ — искомый, т. е. $A = \{\langle f_1(x), \ldots, f_k(x)\rangle \mid x \in \omega\}$. Для этого докажем сначала включение $A \subseteq \{\langle f_1(x), \ldots, f_k(x)\rangle \mid x \in \omega\}$. Пусть $\overline{x} = \langle x_1, \ldots, x_k\rangle \in A$. Тогда $f(\overline{x}) = U(\mu y[T_k(e, \overline{x}, y)])$ определено. Следовательно, значение $\mu y[T_k(e, \overline{x}, y)]$ определено. Следовательно, существует $y \in \omega$ такой, что $T_k(e, \overline{x}, y)$ истинно. Положим $n =< y, x_1, \ldots, x_k >$. Тогда $T_k(e, (n)_1, \ldots, (n)_k, (n)_0)$ истинно и, значит, $f_i(n) = (n)_i = x_i$ для всех $i \in \{1, \ldots, k\}$. Таким образом, $\overline{x} \in \{\langle f_1(x), \ldots, f_k(x)\rangle \mid x \in \omega\}$.

Докажем теперь включение $\{\langle f_1(x), \ldots, f_k(x)\rangle \mid x \in \omega\} \subseteq A$. Рассмотрим произвольный набор $\langle f_1(n), \ldots, f_k(n)\rangle$, где $n \in \omega$. Если $T_k(e, (n)_1, \ldots, (n)_k, (n)_0)$ ложно, то $\langle f_1(n), \ldots, f_k(n)\rangle = \overline{a} \in A$ и все доказано. Пусть $T_k(e, (n)_1, \ldots, (n)_k, (n)_0)$ истинно. Тогда по определению T-предиката Клини $z = (n)_0$ является кодом вычисления по программе с кодом e и входом $\overline{x} = \langle (n)_1, \ldots, (n)_k\rangle$, причем этот z является единственным, удовлетворяющим этому условию. Следовательно, $z = \mu y[T_k(e, \overline{x}, y)]$. Отсюда заключаем, что значение $f(\overline{x}) = U(\mu y[T_k(e, \overline{x}, y)])$ определено. Таким образом, $\langle f_1(n), \ldots, f_k(n)\rangle = \langle (n)_1, \ldots, (n)_k\rangle = \overline{x} \in \mathrm{dom}(f) = A$. □

Переформулируем отдельно теорему об эквивалентных определениях в.п. множеств для 1-местных отношений, добавив еще одно, четвертое эквивалентное условие.

Следствие 51. *Для произвольного множества $A \subseteq \omega$ следующие условия эквивалентны:*

1) A вычислимо перечислимо.

2) Существует вычислимое отношение $R \subseteq \omega^2$ такое, что

$$x \in A \iff \exists y R(x, y).$$

3) Существует частично вычислимая функция $f(x)$ такая, что $A = \mathrm{dom}(f)$.

4) Существует частично вычислимая функция $f(x)$ такая, что $A = \mathrm{range}(f)$.

Доказательство. Эквивалентность условий (1), (2) и (3) доказана в теореме 50. Покажем, что условие (4) эквивалентно условиям (1)–(3).

Пусть A удовлетворяет условию (3), т.е. $A = \mathrm{dom}(f)$ для некоторой ч.в.ф. $f(x)$. Определим частично вычислимую функцию $g(x) = x + o(f(x))$. Тогда $\mathrm{range}(g) = \mathrm{dom}(g) = \mathrm{dom}(f) = A$, т. е. A удовлетворяет условию (4).

Пусть теперь A удовлетворяет условию (4), т. е. $A = \mathrm{range}(f)$, где f — ч.в.ф. Тогда f вычислима на некоторой машине Шёнфилда с кодом e. По теореме Клини о нормальной форме $f(x) = U(\mu t[T_1(e, x, t)])$, где U и T_1 — вычислимые. Отсюда получаем:

$$y \in A \iff \exists x(f(x) \downarrow = y) \iff \exists x \exists t\big(T_1(e, x, t)\ \&\ U(t) = y\big).$$

Закодируем пару $\langle x, t\rangle$ одним числом $z =< x, t >$. Тогда имеет место эквивалентность

$$y \in A \iff \exists z\big(T_1(e, (z)_0, (z)_1)\ \&\ U((z)_1) = y\big).$$

Поскольку отношение $R = \{\langle y, z\rangle \mid T_1(e, (z)_0, (z)_1)\ \&\ U((z)_1) = y\}$ является вычислимым, то отсюда следует, что A удовлетворяет условию (2). □

Выясним теперь, как соотносятся между собой семейство всех в.п. множеств и семейство всех вычислимых множеств.

Предложение 52. *Если $A \subseteq \omega^k$ вычислимо, то A вычислимо перечислимо.*

Доказательство. Пусть A вычислимо, следовательно, функция $\mathcal{X}_A(x_1, \ldots, x_k)$ вычислима. Определим частичную функцию $f(\overline{x}) = \mu y[|\mathcal{X}_A(\overline{x}) - 1| = 0]$. Тогда f — ч.в.ф. и $\mathrm{dom}(f) = A$. Следовательно, в силу пункта (3) теоремы 50 A является в.п. множеством. □

Утверждение, обратное к предложению 52, вообще говоря, неверно.

Предложение 53. *Существует в.п. множество $K \subseteq \omega$, которое не является вычислимым.*

Доказательство. Рассмотрим множество $K = \{x \in \omega \mid æ_x(x) \downarrow\}$. По теореме 47 оно не вычислимо. Докажем, что K вычислимо перечислимо. По теореме Клини о нормальной форме $æ_x(y) = U(\mu t[T_1(x, y, t)])$. Тогда имеем:

$$x \in K \iff æ_x(x) \downarrow \iff U(\mu t[T_1(x, x, t)]) \downarrow \iff \exists t\, T_1(x, x, t).$$

Отсюда по пункту (2) теоремы 50 заключаем, что K вычислимо перечислимо. □

Далее мы исследуем свойства замкнутости семейства в.п. множеств относительно теоретико-множественных операций. Напомним, что для вычислимых множеств справедливо следующее

Предложение 54. *Пусть $A, B \subseteq \omega^k$ — вычислимые множества. Тогда множества $A \cup B$, $A \cap B$ и $\omega^k \setminus A$ тоже вычислимы.*

Доказательство. См. доказательство предложения 21. □

Для семейства в.п. множеств замкнутость относительно объединения и пересечения остается справедливой. Однако, в отличие от вычислимых множеств, в.п. множества незамкнуты относительно дополнения.

Предложение 55. *Пусть $A, B \subseteq \omega^k$ — вычислимо перечислимые множества. Тогда множества $A \cup B$ и $A \cap B$ тоже вычислимо перечислимы.*

Доказательство. Пусть $P, R \subseteq \omega^{k+1}$ такие вычислимые множества, что

$$\overline{x} \in A \iff \exists y P(\overline{x}, y),$$
$$\overline{x} \in B \iff \exists y R(\overline{x}, y).$$

Тогда для объединения получаем:

$$\overline{x} \in A \cup B \iff \Big(\exists y P(\overline{x}, y) \vee \exists y R(\overline{x}, y)\Big) \iff \exists y \Big(P(\overline{x}, y) \vee R(\overline{x}, y)\Big) \iff \exists y Q(\overline{x}, y),$$

где $Q(\overline{x}, y) = P(\overline{x}, y) \vee R(\overline{x}, y)$ — вычислимый предикат. Следовательно, $A \cup B$ вычислимо перечислимо.

Для пересечения имеем:

$$\overline{x} \in A \cap B \iff \Big(\exists y P(\overline{x}, y) \,\&\, \exists z R(\overline{x}, z)\Big) \iff \exists y \exists z \Big(P(\overline{x}, y) \,\&\, R(\overline{x}, z)\Big) \iff$$
$$\iff \exists t \Big(P(\overline{x}, (t)_0) \,\&\, R(\overline{x}, (t)_1)\Big) \iff \exists t Q(\overline{x}, t),$$

где $Q(\overline{x}, t) = P(\overline{x}, (t)_0) \,\&\, R(\overline{x}, (t)_1)$ — вычислимый предикат. Следовательно, $A \cap B$ вычислимо перечислимо. □

Теорема 56 (теорема Поста)**.** *Множество $A \subseteq \omega^k$ вычислимо тогда и только тогда, когда A и $\omega^k \setminus A$ вычислимо перечислимы.*

Доказательство. ($\Longrightarrow$) Следует из замкнутости вычислимых множеств относительно дополнения и предложения 52.

($\Longleftarrow$) Пусть $P, R \subseteq \omega^{k+1}$ такие вычислимые множества, что

$$\overline{x} \in A \iff \exists y P(\overline{x}, y),$$
$$\overline{x} \notin A \iff \exists y R(\overline{x}, y).$$

Определим вычислимую функцию $f(\overline{x}) = \mu y[P(\overline{x}, y) \vee R(\overline{x}, y)]$. Тогда получаем: $\overline{x} \in A \iff \exists y P(\overline{x}, y)\ \&\ \forall y \neg R(\overline{x}, y) \iff P(\overline{x}, f(\overline{x}))$. Поэтому $\mathcal{X}_A(\overline{x}) = \mathcal{X}_P(\overline{x}, f(\overline{x}))$ является вычислимой функцией. Таким образом, A вычислимо. □

Следствие 57. *Существует множество $A \subseteq \omega$ такое, что A вычислимо перечислимо, но $\omega \setminus A$ не является вычислимо перечислимым.*

Доказательство. Рассмотрим множество $K = \{x \in \omega \mid æ_x(x)\downarrow\}$. По предложению 53 K — в.п. Если бы $\omega \setminus K$ было в.п., то по теореме Поста K было бы вычислимым, что невозможно. □

Теорема 58 (теорема о графике)**.** *Частичная функция $f(x_1, \ldots, x_k)$ является частично вычислимой тогда и только тогда, когда ее график*

$$\Gamma_f = \{\langle x_1, \ldots, x_k, y\rangle \mid \langle x_1, \ldots, x_k\rangle \in \mathrm{dom}(f), f(x_1, \ldots, x_k) = y\}$$

является вычислимо перечислимым.

Доказательство. ($\Longrightarrow$) Пусть f — ч.в.ф. По теореме Клини о нормальной форме $f(x_1, \ldots, x_k) = U(\mu t[T_k(e, x_1, \ldots, x_k, t)])$, где e — код программы для машины Шёнфилда, вычисляющей f. Тогда получаем

$$f(\overline{x})\downarrow = y \iff \exists t\big(T_k(e, \overline{x}, t)\ \&\ U(t) = y\big) \iff \exists t R(\overline{x}, y, t),$$

где $R = \{\langle \overline{x}, y, t\rangle \mid T_k(e, \overline{x}, t)\ \&\ U(t) = y\}$. Поскольку отношение R является вычислимым, то в силу пункта (2) теоремы 50 заключаем, что Γ_f — в.п.

($\Longleftarrow$) Пусть Γ_f — в.п. В соответствии с пунктом (2) теоремы 50 существует вычислимое отношение $R \subseteq \omega^{k+2}$ такое, что справедлива эквивалентность $\langle \overline{x}, y\rangle \in \Gamma_f \iff \exists z R(\overline{x}, y, z)$. Отсюда заключаем:

$$\overline{x} \in \mathrm{dom}(f) \iff \exists y \langle \overline{x}, y\rangle \in \Gamma_f \iff \exists y \exists z R(\overline{x}, y, z) \iff$$
$$\iff \exists t R(\overline{x}, (t)_0, (t)_1) \iff \mu t[R(\overline{x}, (t)_0, (t)_1)] \text{ определено.}$$

Поэтому $f(\overline{x}) = \Big(\mu t[R(\overline{x}, (t)_0, (t)_1)]\Big)_0$ является ч.в.ф. □

В заключение параграфа мы определим нумерацию всех в.п. подмножеств ω, которая является аналогом клиниевской нумерации всех одноместных ч.в.ф. В частности, для данной нумерации справедливы собственные варианты s-m-n-теоремы, теоремы о неподвижной точке и теоремы Райса.

Определение. Для каждого $n \in \omega$ введем обозначение $W_n = \mathrm{dom}(æ_n)$. Число n называется *в.п.-индексом* множества W_n. В силу пункта (3) следствия 51 заключаем, что отображение $n \mapsto W_n$ является нумерацией семейства всех в.п. подмножеств ω. Эта нумерация называется *клиниевской нумерацией в.п. множеств.*

Определение. Пусть S – некоторое семейство подмножеств ω. Нумерация $\nu : \omega \xrightarrow{\text{на}} S$ называется *вычислимой*, если множество $\{\langle x, y\rangle | x \in \nu(y)\}$ вычислимо перечислимо.

Предложение 59. *Клиниевская нумерация в.п. множеств является вычислимой.*

Доказательство. Пусть S_0 — семейство всех в.п. подмножеств ω, $\nu_0 : \omega \xrightarrow{\text{на}} S_0$ — клиниевская нумерация, т. е. $\nu_0(n) = W_n$. Тогда для любых $x, y \in \omega$ имеем:

$$x \in \nu_0(y) \iff x \in W_y \iff x \in \mathrm{dom}(æ_y) \iff$$
$$\iff æ_y(x) \downarrow \iff U(\mu t[T_1(y, x, t)]) \downarrow \iff \exists t T_1(y, x, t).$$

Отсюда в силу вычислимости T-предиката Клини и в силу пункта (2) теоремы 50 заключаем, что $\{\langle x, y\rangle \mid x \in \nu_0(y)\}$ вычислимо перечислимо. □

Следующее утверждение говорит о том, что клиниевская нумерация является наибольшей среди всех вычислимых нумераций (относительно предпорядка сводимости нумераций).

Предложение 60. *Любая вычислимая нумерация семейства подмножеств ω сводится к клиниевской.*

Доказательство. Пусть S — некоторое семейство подмножеств ω, $\nu : \omega \to S$ — вычислимая нумерация. Пусть далее S_0 — семейство всех в.п. подмножеств ω, $\nu_0 : \omega \to S_0$ — клиниевская нумерация.

Так как ν вычислима, то существует вычислимое множество $R \subseteq \omega^3$ такое, что имеет место свойство $x \in \nu(y) \iff \exists z R(x, y, z)$. Тогда для любого фиксированного $n \in \omega$ множество $\nu(n) = \{x \mid \exists z R(x, n, z)\}$ является вычислимо перечислимым и, значит, $S \subseteq S_0$.

Определим двухместную частичную функцию

$$g(x, y) = \begin{cases} 0, & \text{если } x \in \nu(y), \\ \text{не определено} & \text{иначе.} \end{cases}$$

Поскольку $g(x, y) = o(\mu z[R(x, y, z)])$, то g — ч.в.ф. По s-m-n-теореме найдется вычислимая функция $f(y)$ такая, что $g(x, y) = æ_{f(y)}(x)$. Тогда для любых $x, y \in \omega$ получаем:

$$x \in \nu(y) \iff g(x, y) \downarrow \iff æ_{f(y)}(x) \downarrow \iff$$
$$\iff x \in \mathrm{dom}(æ_{f(y)}) \iff x \in W_{f(y)} \iff x \in \nu_0(f(y)).$$

Другими словами, $\nu = \nu_0 \circ f$, т. е. $\nu \leqslant \nu_0$. Что и требовалось доказать. □

§ 21. Универсальные функции

Напомним, что $(k{+}1)$-местная частичная функция $F(x_0, x_1, \ldots, x_k)$ называется *универсальной* для некоторого семейства K, состоящего из k-местных частичных функций, если выполняеются следующие условия:

1) для любого $n \in \omega$ существует $f \in K$ такая, что $F(n, x_1, \ldots, x_k) = f(x_1, \ldots, x_k)$;

2) для любой $f \in K$ найдется $n \in \omega$ такой, что $F(n, x_1, \ldots, x_k) = f(x_1, \ldots, x_k)$.

Мы уже убедились в том, что для семейства всех k-местных ч.р.ф. существует универсальная $(k{+}1)$-местная функция, которая сама является ч.р.ф. Однако для семейства всех k-местных п.р.ф. и для семейства всех k-местных р.ф. аналогичное свойство не имеет места.

Предложение 61. *Пусть $k \geqslant 1$. Не существует $(k{+}1)$-местной п.р.ф., универсальной для семейства всех k-местных п.р.ф.*

Доказательство. Допустим, напротив, $F(x_0, x_1, \ldots, x_k)$ — п.р.ф., универсальная для семейства всех k-местных п.р.ф., т. е. $\{F(0, x_1, \ldots, x_k), F(1, x_1, \ldots, x_k), \ldots\}$ — класс всех k-местных п.р.ф. Определим k-местную функцию:

$$f(x_1, \ldots, x_k) = F(x_1, x_1, x_2, \ldots, x_k) + 1.$$

Тогда $f(x_1, \ldots, x_k)$ — п.р.ф. Следовательно, в силу универсальности F найдется $n \in \omega$ такой, что для всех $x_1, \ldots, x_k \in \omega$ выполняется $F(n, x_1, \ldots, x_k) = f(x_1, \ldots, x_k)$. Рассмотрим значения $x_1 = n, x_2 = \ldots = x_k = 0$. Тогда получаем:

$$F(n, n, 0, \ldots, 0) = f(n, 0, \ldots, 0) = F(n, n, 0, \ldots, 0) + 1.$$

Противоречие. □

Предложение 62. *Пусть $k \geqslant 1$. Не существует $(k{+}1)$-местной р.ф., универсальной для семейства всех k-местных р.ф.*

Доказательство. Повторяет доказательство предложения 61. □

Замечание. Условие $k \geqslant 1$ в формулировках предложений 61 и 62 присутствует не случайно. Для семейства всех 0-местных п.р.ф. (которое совпадает с семейством всех 0-местных р.ф.) существует универсальная п.р.ф. — это, очевидно, 1-местная функция $F(x) = x$.

Диагональный метод доказательства предложения 61 существенно опирается на предположение о примитивной рекурсивности функции $F(x_0, x_1, \ldots, x_k)$. Если это предположение ослабить и считать, что $F(x_0, x_1, \ldots, x_k)$ — рекурсивная, то диагональные рассуждения уже ни к чему не приводят, т. е. не доказывают отсутствия р.ф., универсальной для семейства всех п.р.ф. Более того, справедливо следующее утверждение, которое мы приводим без доказательства.

Предложение 63. *Существует $(k{+}1)$-местная р.ф., универсальная для семейства всех k-местных п.р.ф.* □

Полное доказательство предложения 63 изложено в [5]. В основе этого доказательства лежат теорема Робинсона о том, что все 1-местные п.р.ф. можно получить

из функций $s(x) = x+1$ и $q(x) = x \dot{-} [\sqrt{x}]^2$ операциями сложения, суперпозиции и итерирования функций, и теорема о рекурсивности функций, полученных из некоторых п.р.ф. с помощью рекурсии 2-й ступени.

Из предложения 63 вытекает важное следствие, утверждающее, что класс всех п.р.ф. не совпадает с классом всех р.ф. Напомним, что справедливы следующие теоретико-множественные включения:

$$\text{ПРФ} \subseteq \text{РФ} \subseteq \text{ЧРФ}.$$

Каждое из этих включений — строгое. Второе включение является строгим, поскольку, очевидно, существуют не всюду определенные ч.р.ф. Например, нигде не определенная функция $f = \mu x[s(x) = 0]$ является ч.р.ф. Докажем, что первое включение является строгим.

Следствие 64. *Существует рекурсивная функция, не являющаяся примитивно рекурсивной.*

Доказательство. Пусть $F(x, y)$ — р.ф., универсальная для семейства всех 1-местных п.р.ф., которая существует в силу предложения 63. Если бы $F(x, y)$ была примитивно рекурсивной, то она была бы п.р.ф., универсальной для семейства всех 1-местных п.р.ф., что невозможно в силу предложения 61. □

Существует другой известный пример рекурсивной функции, которая не является примитивно рекурсивной, — это так называемая *функция Аккермана*. Определение и свойства функции Аккермана можно найти в [5].

Вопрос о существовании универсальной функции можно исследовать не только для семейств всех k-местных ч.р.ф., р.ф. или п.р.ф. Для дальнейших целей нам потребуется универсальная функция для семейства всех полиномов.

Предложение 65. *Существует 2-местная вычислимая функция, универсальная для семейства всех полиномов от одной переменной с натуральными коэффициентами.*

Доказательство. Если $f(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ — полином, $a_i \in \omega$, $n \in \omega$, то сопоставим ему код$(f) =< a_0, a_1, \ldots, a_n >= p_0^{a_0+1} \cdot p_1^{a_1+1} \cdot \ldots \cdot p_n^{a_n+1}$, где $p_0 = 2, p_1 = 3, p_2 = 5, \ldots$ — перечисление простых чисел в порядке возрастания.

Заметим, что если $k =$ код(f), то степень полинома f находится с помощью вычислимой функции $\text{lh}(k) \dot{-} 1 = \mu i[\text{ex}(i, k) = 0] \dot{-} 1$, а i-й коэффициент f находится с помощью вычислимой функции $(k)_i = \text{ex}(i, k) \dot{-} 1$.

Определим двухместную вычислимую функцию

$$F(y, x) = \sum_{i=0}^{\text{lh}(y) \dot{-} 1} (y)_i \cdot x^i.$$

Тогда, с одной стороны, если у функции $F(y, x)$ зафиксировать значение y, то получится некоторый полином от переменной x с натуральными коэффициентами. С другой стороны, если f — произвольный полином, то $F(\text{код}(f), x) = f(x)$. Таким образом, F — искомая универсальная функция. □

Теперь мы более детально изучим свойства универсальной функции для семейства всех одноместных частично вычислимых функций. Для данного семейства мы, используя кодирование машин Шёнфилда, построили универсальную клиниевскую функцию $æ_n(x)$. Клиниевская универсальная функция обладает важным свойством — она сама является частично вычислимой функцией от переменных n, x. Но, кроме частичной вычислимости, функция $æ_n(x)$ обладает еще другим, пожалуй, более важным свойством — для клиниевской нумерации одноместных ч.в.ф. справедлива теорема о параметризации: *для любой двухместной ч.в.ф. $f(n,x)$ существует разнозначная в.ф. $s(n)$ такая, что $f(n,x) = æ_{s(n)}(x)$.*

Это наблюдение лежит в основе следующего определения.

Определение. Двухместная ч.в.ф. $\psi_n(x)$ называется *сильно универсальной*, если для любой ч.в.ф. $f(n,x)$ существует разнозначная в.ф. $s(n)$ такая, что $f(n,x) = \psi_{s(n)}(x)$.

Предложение 66. *Любая сильно универсальная ч.в.ф. является универсальной для семейства всех одноместных ч.в.ф.*

Доказательство. Пусть $\psi_n(x)$ — сильно универсальная ч.в.ф., $f(x)$ — произвольная одноместная ч.в.ф. Рассмотрим двухместную ч.в.ф. $F(n,x) = f(x)$. В силу сильной универсальности найдется разнозначная в.ф. $s(n)$ такая, что $\psi_{s(n)}(x) = F(n,x) = f(x)$. В частности, для любого $m = s(n)$ имеет место $\psi_m(x) = f(x)$. Таким образом, $\psi_n(x)$ — универсальная для семейства всех одноместных ч.в.ф. □

Предложение 67. *Существует двухместная ч.в.ф., которая является универсальной для семейства всех одноместных ч.в.ф., но не является сильно универсальной.*

Доказательство. Определим двухместную частичную функцию

$$\psi_x(y) = \begin{cases} \text{не определено}, & \text{если } y = 0 \text{ и } (x)_0 = 0, \\ (x)_0 - 1, & \text{если } y = 0 \text{ и } (x)_0 > 0, \\ æ_{(x)_1}(y), & \text{если } y > 0. \end{cases}$$

Докажем, что $\psi_x(y)$ частично вычислима. Для этого заметим, что для произвольных $x, y, z \in \omega$ имеет место:

$$\psi_x(y) \downarrow = z \iff \Big((x)_0{>}0 \;\&\; y{=}0 \;\&\; z{=}(x)_0 - 1\Big) \vee \Big(y{>}0 \;\&\; æ_{(x)_1}(y) \downarrow \;\&\; z{=}æ_{(x)_1}(y)\Big)$$
$$\iff \Big((x)_0 > 0 \;\&\; y = 0 \;\&\; z = (x)_0 - 1\Big) \vee \exists t\Big(y > 0 \;\&\; T_1((x)_1, y, t) \;\&\; z = U(t)\Big).$$

Отсюда в силу теоремы об эквивалентных определениях в.п. множеств заключаем, что график функции $\psi_x(y)$ вычислимо перечислим. Следовательно, $\psi_x(y)$ частично вычислима.

Докажем, что $\psi_x(y)$ — универсальная. Пусть $æ_e(y)$ — произвольная одноместная ч.в.ф. Если $æ_e(0) \uparrow$, то, положив $n =< 0, e >$, заключаем, что $\psi_n(y) = æ_e(y)$. Если же $æ_e(0) \downarrow$ и $æ_e(0) = m$, то, определив $n =< m+1, e >$, получаем, что $\psi_n(y) = æ_e(y)$. Что и требовалось доказать.

Допустим теперь, что $\psi_x(y)$ — сильно универсальная. Рассмотрим двухместную клиниевскую ч.в.ф. $æ_x(y)$. В силу сильной универсальности ψ заключаем, что существует разнозначная в.ф. $s(x)$ такая, что $\psi_{s(x)}(y) = æ_x(y)$. Тогда имеет место эквивалентность

$$æ_x(0) \downarrow \iff \psi_{s(x)}(0) \downarrow \iff \big(s(x)\big)_0 > 0.$$

Множество $\{x \in \omega \mid \big(s(x)\big)_0 > 0\}$ очевидно вычислимо. Однако множество $\{x \in \omega \mid æ_x(0)\downarrow\}$ в силу теоремы Райса невычислимо. Противоречие. Таким образом, $\psi_x(y)$ не является сильно универсальной. □

Итак, совокупность всех универсальных ч.в.ф можно разбить на две непустые части. Первая часть — это все *сильно универсальные функции*, сюда попадает клиниевская универсальная функция $æ_n(x)$. Вторая часть — это все *слабо универсальные функции*, т. е. функции, которые являются универсальными, но не являются сильно универсальными. Оказывается, все функции из первой части в определенном смысле эквивалентны друг другу, и доказательству этого свойства посвящен следующий параграф.

§ 22. Единственность сильно универсальной функции

Единственный пример сильно универсальной функции, который нам пока известен, — это клиниевская универсальная функция $æ_n(x)$. Мы определили ее, используя технику кодирования вычислений на машинах Шёнфилда. Вообще говоря, клиниевскую универсальную функцию можно было построить другим способом, используя терминологию машин Тьюринга или нормальных алгорифмов Маркова. Кроме этого, можно было использовать другую кодировку кортежей, команд и машин. Каждый из описанных способов в результате приводит к той или иной сильно универсальной функции. Поэтому возникает естественное опасение: возможно, какая-нибудь универсальная функция из данного многообразия более приемлема, чем остальные. Насколько различными могут получиться универсальные функции?

Оказывается, все универсальные функции, для которых справедлива теорема о параметризации, т. е. все сильно универсальные функции, практически не отличаются в алгоритмическом смысле. Каждую сильно универсальную функцию можно получить из любой другой с помощью вычислимого изоморфизма. Можно даже утверждать, что с точностью до вычислимого изоморфизма сильно универсальная функция единственна, а построенная нами конкретная функция $æ_n(x)$ ничем не хуже и не лучше остальных.

Для доказательства теоремы о вычислимом изоморфизме сильно универсальных функций нам потребуются следующие три леммы.

Лемма 68. *Пусть $\psi_n(x)$ — сильно универсальная функция. Тогда для любой ч.в.ф. $f(x,y,z)$ существует разнозначная в.ф. $g(x,y)$ такая, что $f(x,y,z) = \psi_{g(x,y)}(z)$.*

Доказательство. Рассмотрим двухместную ч.в.ф. $f'(t,z) = f((t)_0,(t)_1,z)$. В силу сильной универсальности ψ существует разнозначная в.ф. $s(t)$ такая, что $f'(t,z) = \psi_{s(t)}(z)$. Подставим $t =< x,y >$, получим $f(x,y,z) = f'(< x,y >,z) = \psi_{s(<x,y>)}(z)$.

Обозначим $g(x,y) = s(< x,y >)$. Так как s и $< \cdot,\cdot >$ — разнозначные, то g тоже разнозначная. Так как s и $< \cdot,\cdot >$ — вычислимые, то g тоже вычислимая. Следовательно, g — искомая. □

Лемма 69. *Пусть $\psi_n(x)$ — сильно универсальная функция. Тогда существует разнозначная в.ф. $g(x,t)$ такая, что для всех $x,y,t \in \omega$ выполняется $\psi_x(y) = \psi_{g(x,t)}(y)$.*

В частности, для одноместной функции ψ_x существует бесконечно много чисел $n \in \omega$ таких, что $\psi_x = \psi_n$.

Доказательство. Рассмотрим ч.в.ф. $f(x,t,y) = \psi_x(y) + 0\cdot t$ (t — фиктивная). По лемме 68 существует разнозначная в.ф. $g(x,t)$ такая, что $f(x,t,y) = \psi_{g(x,t)}(y)$. Следовательно, $\psi_x(y) = \psi_{g(x,t)}(y)$. □

Лемма 70. *Существует двухместная ч.в.ф. $k_x(y)$ такая, что если $æ_x(y)$ — вычислимая перестановка на ω, то $k_x(y) = æ_x^{-1}(y)$.*

Доказательство. $k_x(y) = \mu z[|æ_x(z) - y| = 0]$ — искомая ч.в.ф. от переменных $\langle x, y\rangle$. □

Определение. Любая вычислимая биекция $p : \omega \xrightarrow[\text{на}]{1-1} \omega$ называется *вычислимой перестановкой.*

Определение. Двухместные частичные функции $\psi_n(x)$ и $\theta_n(x)$ называются *вычислимо изоморфными*, если существует вычислимая перестановка $p : \omega \xrightarrow[\text{на}]{1-1} \omega$ такая, что для любых $n, x \in \omega$ выполняется

$$\psi_{p(n)}(p(x)) = p(\theta_n(x)).$$

Другими словами, ψ и θ вычислимо изоморфны, если существует вычислимая перестановка p, которая отображает график $\Gamma_\theta = \{\langle n, x, \theta_n(x)\rangle \mid \theta_n(x)\downarrow\}$ покоординатно на график $\Gamma_\psi = \{\langle n, x, \psi_n(x)\rangle \mid \psi_n(x)\downarrow\}$.

$$\begin{array}{ccc} \langle n, x\rangle & \xrightarrow{\ \theta\ } & \theta_n(x) \\ p\big\downarrow & & \big\downarrow p \\ \langle p(n), p(x)\rangle & \xrightarrow[\ \psi\ ]{} & \psi_{p(n)}(p(x)) \end{array}$$

Теорема 71 (о вычислимом изоморфизме сильно универсальных функций)**.** *Любые две сильно универсальные ч.в.ф. $\psi_n(x)$ и $\theta_n(x)$ вычислимо изоморфны.*

Доказательство. Нам нужно доказать, что существует вычислимая перестановка $p : \omega \xrightarrow[\text{на}]{1-1} \omega$ такая, что для любых $x, y \in \omega$ выполняется

$$\psi_{p(x)}(p(y)) = p(\theta_x(y)).$$

Докажем, что существует двухместная вычислимая функция $h_z(x)$ такая, что для любого $z \in \omega$ функция $h_z : \omega \to \omega$ является вычислимой перестановкой, такой, что для любого $x \in \omega$ выполняется хотя бы одно из следующих двух условий:

$$\psi_{h_z(x)} = æ_z \theta_x k_z \qquad (*)$$
$$k_z \psi_{h_z(x)} æ_z = \theta_x \qquad (**)$$

Допустим, мы уже доказали существование такой функции $h_z(x)$. Тогда по теореме о параметризации существует в.ф. $s(z)$ такая, что $h_z(x) = æ_{s(z)}(x)$. Далее по теореме о неподвижной точке найдется число $n \in \omega$ такое, что $æ_{s(n)} = æ_n = h_n$. Обозначим $p(x) = h_n(x) = æ_n(x)$. Так как h_n — вычислимая перестановка, то $æ_n$ тоже вычислимая перестановка, и, следовательно, в силу леммы 70 $k_n = æ_n^{-1} = p^{-1}$. Тогда из $(*)$ и $(**)$ заключаем, что для любого $x \in \omega$ выполняется хотя бы одно из условий:

$$\psi_{p(x)} = p\theta_x p^{-1} \qquad (*)$$
$$p^{-1}\psi_{p(x)} p = \theta_x \qquad (**)$$

В любом случае, получаем $\psi_{p(x)}p = p\theta_x$, т. е. $\psi_{p(x)}(p(y)) = p(\theta_x(y))$, и наша теорема доказана.

Таким образом, осталось доказать существование функции $h_z(x)$ с нужными свойствами. Поскольку график функции однозначно задает саму функцию, то мы будем строить график функции $\Gamma_h = \{\langle z, x, h_z(x)\rangle \mid z, x \in \omega\}$. Поскольку в силу теоремы о графике вычислимость h равносильна вычислимой перечислимости Γ_h, то мы опишем алгоритм перечисления троек из графика Γ_h.

Для каждого $z \in \omega$ мы опишем равномерный пошаговый алгоритм перечисления пар $\langle x, u\rangle$ таких, что тройка $\langle z, x, u\rangle \in \Gamma_h$. На каждом шаге будем добавлять новую тройку в график Γ_h. Шаги будут делиться на нечетные и четные. На шаге n будем строить конечное множество M_n такое, что $M_0 \subseteq M_1 \subseteq \ldots \subseteq M_n \subseteq \ldots$, и каждое M_n состоит из пар $\langle x, u\rangle$ с условием $\langle z, x, u\rangle \in \Gamma_h$.

Перейдем к описанию пошаговой конструкции.

Шаг 0. Полагаем $M_0 = \varnothing$.

Шаг $2t+1$ (добавление нового элемента в $\mathrm{dom}(h_z)$).

а) Ищем наименьшее x, не содержащееся среди левых координат пар из M_{2t}.

б) Ищем u, не содержащееся среди правых координат пар из M_{2t}, такое, что

$$\forall y \in \omega \quad \psi_u(y) = æ_z\theta_x k_z(y) \qquad (*).$$

Данное u находится эффективно следующим образом. По лемме 68 существует в.ф. $g(z, x)$ такая, что $\psi_{g(z,x)}(y) = æ_z\theta_x k_z(y)$. Для заданных значений z и x число $v = g(z, x)$ является фиксированным. По лемме 69 существует разнозначная в.ф. $r(v, s)$ такая, что для любого $s \in \omega$ выполняется $\psi_v(y) = \psi_{r(v,s)}(y)$. Так как множество $\{r(v, 0), r(v, 1), \ldots\}$ бесконечно, а множество M_{2t} конечно, то найдется наименьшее s такое, что число $u = r(v, s)$ не содержится среди правых координат пар из M_{2t}, и $\psi_u = \psi_v = æ_z\theta_x k_z$. Данное u — искомое.

в) Полагаем $M_{2t+1} = M_{2t} \cup \{\langle x, u\rangle\}$, $h_z(x) = u$. Перечисляем тройку $\langle z, x, u\rangle$ в графике Γ_h.

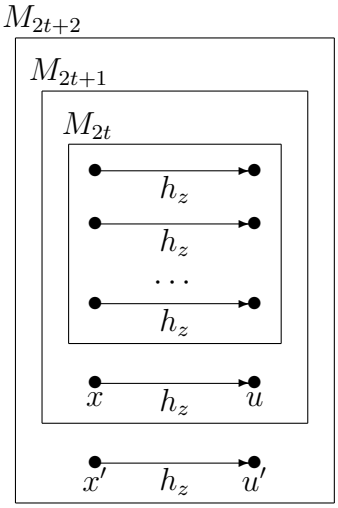
Шаг $2t+2$ (добавление нового элемента в $\mathrm{range}(h_z)$).

а) Ищем наименьшее u, не содержащееся среди правых координат пар из M_{2t+1}.

б) Находим аналогично нечетному шагу число x, не содержащееся среди левых координат пар из M_{2t+1}, такое, что

$$\forall y \in \omega \quad k_z\psi_u æ_z(y) = \theta_x(y) \qquad (**).$$

в) Полагаем $M_{2t+2} = M_{2t+1} \cup \{\langle x, u\rangle\}$, $h_z(x) = u$. Перечисляем тройку $\langle z, x, u\rangle$ в графике Γ_h.



Определим $M = \bigcup\limits_{t\in\omega} M_t$. Положим: $h_z(x) = u \iff \langle x, u\rangle \in M$. По построению h_z является функцией. В силу пункта (а) нечетного шага $\mathrm{dom}(h_z) = \omega$. В силу пункта (а) четного шага $\mathrm{range}(h_z) = \omega$. Из построения также следует, что h_z — инъективное отображение. Так как M перечислимо, то по теореме о графике $h_z : \omega \xrightarrow[\text{на}]{1-1} \omega$ — вычислимая перестановка, и по построению для h_z выполняется $(*)$ или $(**)$ для любого значения x. □

Следствие 72. *Двухместная ч.в.ф. $\psi_x(y)$ является сильно универсальной тогда и только тогда, когда она вычислимо изоморфна клиниевской функции $æ_x(y)$.*

Доказательство. ($\Longrightarrow$) Следует из предыдущей теоремы и из сильной универсальности клиниевской функции $æ_x(y)$.

($\Longleftarrow$) Пусть $\psi_x(y)$ вычислимо изоморфна $æ_x(y)$. Следовательно, существует вычислимая перестановка p такая, что для любого $x \in \omega$ выполняется $\psi_{p(x)} = p\cdot æ_x\cdot p^{-1}$. Пусть $f(x, y)$ — произвольная ч.в.ф. Рассмотрим двухместную ч.в.ф. $p^{-1}(f(x, p(z)))$. По s-m-n-теореме существует вычислимая разнозначная функция $s(x)$ такая, что $p^{-1}(f(x, p(z))) = æ_{s(x)}(z)$. Теперь, сделав замену $z = p^{-1}(y)$, заключаем, что $f(x, y) = p\cdot æ_{s(x)}\cdot p^{-1}(y) = \psi_{p(s(x))}(y)$. Вычислимая разнозначная функция $p(s(x))$ является искомой. □

Следствие 73. *Никакая слабо универсальная ч.в.ф. не является вычислимо изоморфной никакой сильно универсальной ч.в.ф.* □

Глава V

Теория сложности алгоритмов

§ 23. О вычислительной сложности

Классическая теория вычислимости, с введением в которую мы познакомились в двух предыдущих главах, подразумевает, что все абстрактные вычислительные устройства удовлетворяют *принципу потенциальной осуществимости*, т. е. предполагается, что при осуществлении процесса вычисления, определенного алгоритмом, мы имеем неограниченный запас времени и материалов. Однако количество простейших действий, необходимых для получения результата вычисления, может быть весьма большим. На первый взгляд, учитывая достижения индустрии компьютерных технологий, данная проблема кажется несущественной. Тем не менее, вычислительная практика показывает, что многие алгоритмические проблемы, которые в принципе разрешимы, невозможно решить на современных вычислительных устройствах, поскольку алгоритмы решения этих проблем требуют в буквальном смысле космических затрат времени и ресурсов. Примерами таких задач являются классические задачи из комбинаторики: задача о коммивояжере, задача о гамильтоновом цикле, задача выполнимости булевой формулы и др.

Пример. Рассмотрим, например, *задачу о коммивояжере*. Постановка этой задачи очень проста: на карте обозначены n городов и известны расстояния между ними, коммивояжеру (который находится в одном из этих городов) необходимо найти маршрут обхода всех городов, имеющий минимальную длину. Теоретически эта задача безусловно разрешима. Если необходимо посетить n городов, то число всевозможных маршрутов равно в точности $(n-1)!$. Поэтому достаточно найти длины всех маршрутов и выбрать из них минимальную. Однако если мы начнем реализовывать этот алгоритм на практике, то сразу придем к выводу о его непригодности. Допустим, коммивояжеру необходимо посетить 10 городов. Тогда потребуется перебрать $9! = 362880$ маршрутов, такая задача вполне по силам современному компьютеру. Если же, например, количество городов равно 40, то потребуется перебрать 39! маршрутов, а это уже больше, чем 10^{45}. Даже если мы сможем просчитывать 10^{15} маршрутов в секунду (что гораздо быстрее, чем быстродействие многих современных суперкомпьютеров), то время, необходимое для завершения вычислений, составит несколько миллиардов сроков жизни Вселенной!

Теория вычислительной сложности отказывается от принципа потенциальной осуществимости. Если в классической теории вычислимости совокупность всех алгоритмических проблем разбивалась на две категории — разрешимые и неразрешимые, то в теории сложности возникает более тонкая классификация алгоритмических проблем: в зависимости от затрачиваемых времени и ресурсов различают более приемлемые и менее приемлемые алгоритмы. Общепринято, что если задачу нельзя решить

быстрее, чем за экспоненциальное время, то ее следует рассматривать как трудно разрешимую. В нашем примере с задачей о коммивояжере функция $(n-1)!$ растет быстрее, чем 2^n, поэтому предложенный алгоритм перебора всех маршрутов оказывается непригодным. Поскольку экспоненциальная функция (такая, как 2^n) растет быстрее любой полиномиальной функции от n, то при таком соглашении задачи, решаемые алгоритмами полиномиальной сложности, будут легко разрешимыми.

Таким образом, постулируется следующий тезис: *алгоритмы, число операций которых оценивается некоторым полиномом $p(n)$ от длины входных данных n, являются реально пригодными.*

При формальном изложении основ теории сложности в качестве модели вычислений традиционно используются машины Тьюринга. При таком формализме время будет измеряться числом тактов машины Тьюринга, затраченных на вычисление, а память будет измеряться числом ячеек ленты, использованных при вычислении. По причинам, которые скоро станут ясными, ключевое понятие в теории сложности — это недетерминированная машина Тьюринга.

§ 24. Недетерминированные машины Тьюринга

Обычные машины Тьюринга, введенные нами ранее, называют также *детерминированными машинами Тьюринга*, подчеркивая тем самым, что в программе любой такой машины для каждого состояния q_i $(i > 0)$ и любого внешнего символа a_j существует ровно одна команда вида $q_i a_j \to \ldots$ Процесс вычисления на такой машине мы представляли как последовательность конфигураций, каждая из которых однозначно определялась предыдущей.

По аналогии с недетерминированными конечными автоматами мы введем обобщенное понятие *недетерминированной машины Тьюринга*, которая отличается от детерминированного варианта тем, что в ее программе для любого q_i $(i > 0)$ и любого a_j может быть несколько (или нисколько) команд вида $q_i a_j \to \ldots$ Это означает, что, находясь в состоянии q_i и обозревая на ленте символ a_j, машина может выполнить любую из этих команд. Если же в состоянии q_i, обозревая символ a_j, машина не находит в программе команду вида $q_i a_j \to \ldots$, то дальнейшие шаги в данной ветви вычислений невозможны. Таким образом, шаг работы машины уже не определен однозначно, а процесс вычисления можно представлять как дерево конфигураций, в котором из одной конфигурации может получиться уже несколько других. Каждая ветвь дерева конфигураций либо является бесконечной, либо заканчивается в заключительном состоянии, либо обрывается на некотором незаключительном состоянии. В последнем случае мы будем говорить, что произошла *неправильная остановка машины.*

Перейдем к формальным определениям.

Определение. *Недетерминированная машина Тьюринга* — это упорядоченная пятерка $T = \langle \mathcal{A}, Q, P, q_1, q_0 \rangle$, где $\mathcal{A}$, Q, q_1, q_0 имеют тот же смысл, что и у детерминированных машин, а *программой* называется любое подмножество

$$P \subseteq (Q \setminus \{q_0\}) \times \mathcal{A} \times Q \times \mathcal{A} \times \{\Lambda, R, L\}.$$

Как и раньше, элементы программы P называются *командами*, и каждую команду $\langle q_i, a_j, q_k, a_l, S \rangle \in P$ мы записываем в виде слова $q_i a_j \to q_k a_l S$.

Замечание. Таким образом, в программе P недетерминированной машины для любых q_i и a_j может быть несколько команд вида $q_ia_j \to \ldots$ Допускается также случай, когда в P команды, начинающиеся на $q_ia_j \to \ldots$, отсутствуют. Если в программе для любого q_i $(i > 0)$ и любого a_j существует ровно одна команда вида $q_ia_j \to \ldots$, то машина T детерминирована.

Конфигурации (машинные слова) для недетерминированных машин Тьюринга определяются так же, как и раньше. Переопределим отношение преобразования конфигураций за один шаг работы машины Тьюринга в недетерминированном случае.

Определение. Пусть $M = Aq_ia_jB$ — конфигурация недетерминированной машины T. Говорят, что конфигурация M' *получается из* M *за один шаг работы машины* T, и пишут $M \vdash_T M'$, если выполняется одно из условий:

а) В программе P существует команда вида $q_ia_j \to q_ka_l$, и $M' = Aq_ka_lB$.

б) В программе P существует команда вида $q_ia_j \to q_ka_lR$, $B = a_sB'$, и $M' = Aa_lq_ka_sB'$.

в) В программе P существует команда вида $q_ia_j \to q_ka_lR$, $B = \Lambda$, и $M' = Aa_lq_ka_0$ (достраивается новая ячейка справа).

г) В программе P существует команда вида $q_ia_j \to q_ka_lL$, $A = A'a_s$, и $M' = A'q_ka_sa_lB$.

д) В программе P существует команда вида $q_ia_j \to q_ka_lL$, $A = \Lambda$, и $M' = q_ka_0a_lB$ (достраивается новая ячейка слева).

Замечание. В силу недетерминированности для фиксированной конфигурации M может существовать несколько конфигураций M', таких, что $M \vdash_T M'$. Если $M = Aq_ia_jB$ и в программе нет команд вида $q_ia_j \to \ldots$ (в частности, если $i = 0$), то вообще не существует конфигурации M' такой, что $M \vdash_T M'$.

Определение. *Вычислением* на недетерминированной машине Тьюринга T называется любая конечная последовательность конфигураций $M_0, M_1, \ldots, M_n$ для некоторого $n \in \omega$, такая, что

$$M_0 \vdash_T M_1 \vdash_T \ldots \vdash_T M_n.$$

При этом говорят, что вычисление *начинается в конфигурации* M_0*, заканчивается в конфигурации* M_n *и имеет длину* n (или состоит из n *шагов*), и пишут $M_0 \vdash_T^n M_n$.

Если $M \vdash_T^n M'$ для некоторого $n \in \omega$, то пишут $M \vdash_T^* M'$. Если в вычислении $M_0 \vdash_T \ldots \vdash_T M_n$ последнее слово $M_n = Aq_0a_jB$, то говорят, что в данном вычислении произошла *правильная остановка*. Если $M_n = Aq_ia_jB$, где $i > 0$, и в программе нет команды вида $q_ia_j \to \ldots$, то говорят, что в данном вычислении произошла *неправильная остановка*.

Замечание. Если T — детерминированная машина Тьюринга, то элементы вычисления однозначно определяются по начальной конфигурации M_0. Поэтому условие $M_0 \vdash_T^n M_n$ равносильно условию $M_n = \big(M_0\big)_T^{(n)}$ и можно говорить, что машина T перерабатывает конфигурацию M_0 в конфигурацию M_n.

Определение. Пусть $T = \langle \mathcal{A}, Q, P, q_1, q_0 \rangle$ — детерминированная машина Тьюринга. Говорят, что язык L над алфавитом $\mathcal{A}_0 \subseteq \mathcal{A} \setminus \{0\}$ *распознается машиной* T, если для любого слова $w \in \mathcal{A}_0^*$ выполняются следующие условия:

а) машина T в процессе переработки входной конфигурации $q_1 0w0$ не достраивает новых ячеек слева;

б) если $w \in L$, то машина T перерабатывает конфигурацию $q_1 0w0$ в конфигурацию $uq_0 1v$ для некоторых слов $u, v \in \mathcal{A}^*$;

в) если $w \notin L$, то машина T перерабатывает конфигурацию $q_1 0w0$ в конфигурацию $uq_0 0v$ для некоторых слов $u, v \in \mathcal{A}^*$.

Таким образом, детерминированная машина Тьюринга, распознающая язык L, всегда правильно останавливается на входных словах $w \in \mathcal{A}_0^*$ и после остановки выдает 1, если $w \in L$, и выдает 0, если $w \notin L$. Заметим, что в определении нет никаких требований на поведение машины при запуске с входными словами, содержащими символы, не принадлежащие $\mathcal{A}_0$.

С помощью недетерминированных машин Тьюринга также можно распознавать языки. Однако соответствующее определение выглядит необычным и несимметричным.

Определение. Пусть $T = \langle \mathcal{A}, Q, P, q_1, q_0 \rangle$ — недетерминированная машина Тьюринга. Говорят, что язык L над алфавитом $\mathcal{A}_0 \subseteq \mathcal{A} \setminus \{0\}$ *распознается машиной* T, если для любого слова $w \in \mathcal{A}_0^*$ выполняются следующие условия:

а) в любом вычислении машины T с входной конфигурацией $q_1 0w0$ не достраиваются новые ячейки слева;

б) существует натуральное число N, зависящее от T и w, такое, что не существует конфигурации M с условием $q_1 0w0 \vdash_T^N M$;

в) $w \in L$ тогда и только тогда, когда $q_1 0w0 \vdash_T^* uq_0 1v$ для некоторых $u, v \in \mathcal{A}^*$.

Таким образом, если недетерминированная машина T распознает язык L, то все ее вычисления, начатые на входном слове $w \in \mathcal{A}_0^*$, заканчиваются (это может произойти по двум причинам: либо в вычислении происходит правильная остановка, либо происходит неправильная остановка). При этом если хотя бы одно вычисление заканчивается в состоянии q_0 и выдает 1, то считается, что слово w распозналось. Даже если в данной ситуации среди остальных ветвей вычислений найдутся такие, которые выдают 0 в состоянии q_0 или заканчиваются ввиду неправильной остановки, машина по определению распознает слово w.

Пример. Рассмотрим язык $L = \{a^n \mid n$ — составное число$\}$ над алфавитом $\mathcal{A}_0 = \{a\}$.

Обычная детерминированная машина Тьюринга, распознающая данный язык, действует следующим образом: если на вход подано слово a^n, $n > 0$, то машина последовательно перебирает числа $k = 2, 3, \ldots, [\sqrt{n}]$, и для каждого k проверяет, делится ли n на число k. Если хотя бы одно k делит n, то число n — составное, иначе — простое (или меньше двойки).

Мы построим недетерминированную машину Тьюринга, которая распознает тот же язык L, но устроена проще. Благодаря недетерминированности циклический перебор чисел $k = 2, 3, \ldots, [\sqrt{n}]$ можно разбить на несколько ветвей параллельных вычислений — для каждого значения k своя ветвь. Если хотя бы одна ветвь окажется успешной, то число — составное. Если все ветви неуспешные, то число — несоставное.

Внешний алфавит машины, кроме символов $a, 0, 1$, будет содержать вспомогательный символ b. Технически работа машины выглядит так. Допустим, на вход машины

подано слово $0aaaaaaaaaaaa0$ (т. е. $n = 12$ в данном частном случае). Машина пытается недетерминированно «угадать» нетривиальный делитель k числа n, для этого она, используя символ b, отмечает в входном слове начальный префикс длины k следующим образом: $0bb0aaaaaaaaa0$ (т. е. $k = 3$ в данном частном случае). После этого машина пытается «исчерпать» выбранным префиксом $bb0$ все буквы a, передвигая его слева направо примерно так:

$$0bb0aaaaaaaaa0 \vdash^* 0000bb0aaaaaa0 \vdash^* 0000000bb0aaa0 \vdash^* \ldots$$

Если входное слово удается покрыть кратным количеством слов вида $bb0$, то «догадка» машины оказалась правильной. Иначе, т. е. если n не делится нацело на k, данную ветвь вычислений можно считать неуспешной.

Программа имеет 13 состояний:

$$\begin{array}{ll}
q_1 0 \to q_2 0R & q_8 b \to q_8 bL \\
q_2 a \to q_3 bR & q_8 0 \to q_9 0L \\
q_3 a \to q_3 bR & q_9 b \to q_4 b \\
q_3 a \to q_4 0L & q_9 0 \to q_{10} 0R \\
q_4 b \to q_4 bL & q_{10} 0 \to q_{11} 0R \\
q_4 0 \to q_5 0R & q_{11} b \to q_{11} bR \\
q_5 b \to q_6 0R & q_{11} a \to q_{12} 0R \\
q_6 b \to q_6 bR & q_{12} 0 \to q_0 1 \\
q_6 0 \to q_7 0R & q_{12} a \to q_{13} aL \\
q_7 b \to q_7 bR & q_{13} 0 \to q_4 0L \\
q_7 a \to q_8 bL &
\end{array}$$

Вычисления по данной программе происходят следующим образом. Начальная конфигурация имеет вид $q_1 0a^n 0$. Если $n \leqslant 1$, то сразу происходит неправильная остановка. Если $n \geqslant 2$, то в состоянии q_3 машина недетерминированным образом выбирает число k, формально производя такие преобразования:

$$q_1 0a^n 0 \vdash^* 0b^{k-2} q_4 b0a^{n-k}0 = 00^{sk+t} b^{k-2-t} q_4 b0b^t a^{n-(s+1)k-t}0,$$

при значениях $s = 0$ и $t = 0$.

Далее запускается двойная циклическая структура. Внешний цикл имеет счетчик s, а его описанию соответствуют состояния q_4 – q_{13}. Вложенный в него внутренний цикл имеет счетчик t, ему соответствуют состояния q_4 – q_9. Одна циклическая итерация состоит из следующей цепочки преобразований:

$$\begin{gathered}
00^{sk+t} b^{k-2-t} q_4 b0b^t a^{n-(s+1)k-t}0 \vdash^* 00^{sk+t} q_5 b^{k-1-t} 0b^t a^{n-(s+1)k-t}0 \vdash \\
\vdash 00^{sk+t} 0q_6 b^{k-2-t} 0b^t a^{n-(s+1)k-t}0 \vdash^* 00^{sk+t} 0b^{k-2-t} 0q_7 b^t a^{n-(s+1)k-t}0.
\end{gathered}$$

Если в состоянии q_7 обозревается 0, то происходит неправильная остановка, это означает, что либо k не делит n, либо $k = n$. Иначе цепочка продолжается следующим образом:

$$00^{sk+t+1} b^{k-2-t} 0b^t q_7 a^{n-(s+1)k-t}0 \vdash^* 00^{sk+t+1} b^{k-2-t} q_8 0b^{t+1} a^{n-(s+1)k-(t+1)}0.$$

Далее из состояния q_8 машина переходит в состояние q_9 и смещается на ячейку влево. Если в состоянии q_9 обозревается b, то машина переходит на следующую итерацию

во внутреннем цикле. Если же в состоянии q_9 обозревается 0, то, значит, $t = k - 2$ и цепочка продолжается так:

$$00^{(s+1)k-2}q_900b^{k-1}a^{n-(s+2)k+1}0 \vdash^* 00^{(s+1)k}b^{k-1}q_{11}a^{n-(s+2)k+1}0.$$

Если в состоянии q_{11} обозревается 0, то происходит неправильная остановка, это означает, что $n = (s+2)k - 1$, т. е. не делится на k. Иначе цепочка продолжается так:

$$00^{(s+1)k}b^{k-1}q_{11}a^{n-(s+2)k+1}0 \vdash 00^{(s+1)k}b^{k-1}0q_{12}a^{n-(s+2)k}0.$$

Если в состоянии q_{12} обозревается 0, то, значит, k делит n, машина производит правильную остановку и выдает 1. Иначе машина после исполнения последних двух команд из программы переходит на следующую итерацию во внешнем цикле.

§ 25. Классы $\mathbb{P}$ и $\mathbb{NP}$

Поскольку детерминированные машины Тьюринга являются частным случаем недетерминированных, то следующее определение мы сформулируем для случая недетерминированных машин.

Определение. Недетерминированная машина Тьюринга $T = \langle \mathcal{A}, Q, P, q_1, q_0 \rangle$ называется *полиномиально ограниченной над алфавитом* $\mathcal{A}_0 \subseteq \mathcal{A} \setminus \{0\}$, если существует полином $p(n)$ с натуральными коэффициентами, такой, что для любого слова $w \in \mathcal{A}_0^*$ не существует конфигурации M с условием $q_10w0 \vdash_T^{p(|w|)+1} M$.

Другими словами, любое вычисление машины заканчивается не более чем за $p(|w|)$ шагов, где $|w|$ — длина входного слова.

Определение. Язык $L \subseteq \mathcal{A}_0^*$ называется *полиномиально распознаваемым на детерминированной машине Тьюринга*, если существует детерминированная полиномиально ограниченная над алфавитом $\mathcal{A}_0$ машина Тьюринга T, которая распознает язык L.

Через $\mathbb{P}$ обозначим класс всех языков, полиномиально распознаваемых на детерминированных машинах Тьюринга.

Определение. Язык $L \subseteq \mathcal{A}_0^*$ называется *полиномиально распознаваемым на недетерминированной машине Тьюринга*, если существует недетерминированная полиномиально ограниченная над алфавитом $\mathcal{A}_0$ машина Тьюринга T, которая распознает язык L.

Через $\mathbb{NP}$ обозначим класс всех языков, полиномиально распознаваемых на недетерминированных машинах Тьюринга.

Пример. Вернемся к примеру из предыдущего параграфа. Мы построили недетерминированную машину Тьюринга T, которая распознает язык $L = \{a^n \mid n$ — составное число $\}$ над алфавитом $\mathcal{A}_0 = \{a\}$.

Допустим, на вход машины T подана начальная конфигурация q_10a^n0, т. е. длина входного слова равна n. Оценим сверху максимально возможную длину вычисления на построенной машине с данной начальной конфигурацией.

Недетерминированная часть вычисления, т. е. цепочка преобразований:

$$q_10a^n0 \vdash^* 0b^{k-2}q_4b0a^{n-k}0 = 00^{sk+t}b^{k-2-t}q_4b0b^ta^{n-(s+1)k-t}0,$$

где $s = t = 0$, очевидно осуществляется за $k + 1$ шагов.

Далее, как мы уже говорили, запускается двойная циклическая структура. В цепочке преобразований

$$00^{sk+t}b^{k-2-t}q_4b0b^ta^{n-(s+1)k-t}0 \vdash^* 00^{sk+t}q_5b^{k-1-t}0b^ta^{n-(s+1)k-t}0 \vdash$$
$$\vdash 00^{sk+t}0q_6b^{k-2-t}0b^ta^{n-(s+1)k-t}0 \vdash^* 00^{sk+t}0b^{k-2-t}0q_7b^ta^{n-(s+1)k-t}0$$

использовано ровно $2(k-t)$ шагов. Если в состоянии q_7 обозревается 0, то происходит неправильная остановка, но длина такого вычисления не является максимальной. Иначе у нашей цепочки будет продолжение

$$00^{sk+t+1}b^{k-2-t}0q_7b^ta^{n-(s+1)k-t}0 \vdash^* 00^{sk+t+1}b^{k-2-t}q_80b^{t+1}a^{n-(s+1)k-(t+1)}0,$$

в которой использовано $2t + 1$ шагов. Далее из состояния q_8 машина переходит за 1 шаг в состояние q_9 и смещается на ячейку влево. Если в состоянии q_9 обозревается b, то машина переходит за 1 шаг на следующую итерацию во внутреннем цикле. Таким образом, одна итерация внутреннего цикла осуществляется за $2(k-t)+(2t+1)+2 = 2k + 3$ шага. Внутренний счетчик принимает значения $t = 0, \ldots, k - 2$.

Если же в состоянии q_9 обозревается 0, то, значит, $t = k - 2$ и цепочка имеет продолжение

$$00^{(s+1)k-2}q_900b^{k-1}a^{n-(s+2)k+1}0 \vdash^* 00^{(s+1)k}b^{k-1}q_{11}a^{n-(s+2)k+1}0,$$

в котором насчитывается $k+1$ шагов. Если в состоянии q_{11} обозревается 0, то происходит неправильная остановка, но длина такого вычисления не является максимальной. Иначе цепочка продолжается так (здесь использован 1 шаг):

$$00^{(s+1)k}b^{k-1}q_{11}a^{n-(s+2)k+1}0 \vdash 00^{(s+1)k}b^{k-1}0q_{12}a^{n-(s+2)k}0.$$

Если в состоянии q_{12} обозревается 0, то, значит, k делит n, машина производит правильную остановку и выдает 1. Иначе машина после исполнения последних двух команд, т. е. за 2 шага, переходит на следующую итерацию во внешнем цикле.

Таким образом, внешний цикл состоит из $k - 1$ повторений внутреннего цикла и из дополнительных $k + 5$ шагов. Всего в одной итерации внешнего цикла получается $(k - 1)(2k + 3) + k + 5 = 2k^2 + 2k + 2$ шагов.

Внешний счетчик в самом худшем случае принимает значения $s = 0, \ldots, [\frac{n}{k}]$. Следовательно, учитывая шаги из стартовой части вычислений, получаем, что общее число шагов в вычислении не превосходит

$$k + 1 + (2k^2 + 2k + 2)(\left[\frac{n}{k}\right] + 1) \leqslant n + 1 + (2n^2 + 2n + 2)(n + 1) = 2n^3 + 4n^2 + 5n + 3.$$

Следовательно, длина вычислений ограничена сверху полиномом $p(n) = 2n^3 + 4n^2 + 5n + 3$ и, значит, язык L полиномиально распознаваем на недетерминированной машине Тьюринга, т. е. $L \in \mathbb{NP}$.

Теперь мы сформулируем некоторые важные свойства классов $\mathbb{P}$ и $\mathbb{NP}$.

Предложение 74. *Имеет место включение* $\mathbb{P} \subseteq \mathbb{NP}$.

Доказательство. Очевидно, поскольку детерминированные машины Тьюринга являются частным случаем недетерминированных. □

Замечание. Вопрос о справедливости обратного включения $\mathbb{NP} \subseteq \mathbb{P}$ является одной из важнейших нерешенных проблем современной математики, которую очень часто обозначают как *ПРОБЛЕМА* «$\mathbb{P} = \mathbb{NP}$?».

Для машин Тьюринга справедлива теорема о детерминизации, т. е. *для любой недетерминированной машины Тьюринга, распознающей некоторый язык L, существует детерминированная машина Тьюринга, которая распознает тот же самый язык L.* Доказательство данной теоремы можно найти в [12]. Таким образом, класс всех языков, распознаваемых недетерминированными машинами Тьюринга, совпадает с классом всех языков, распознаваемых детерминированными машинами Тьюринга, но данный факт никак не решает проблему «$\mathbb{P} = \mathbb{NP}$?». Причина этого заключается в том, что процедура детерминизации недетерминированной машины Тьюринга очень сильно увеличивает временную сложность машины. Наименьшая верхняя граница сложности, которая возникает после применения процедуры детерминизации, экспоненциальная.

Предложение 75. *Класс $\mathbb{P}$ замкнут относительно дополнения.*

Доказательство. Пусть язык $L \subseteq \mathcal{A}_0^*$ принадлежит классу $\mathbb{P}$. Следовательно, найдется детерминированная полиномиально ограниченная над алфавитом $\mathcal{A}_0 \subseteq \mathcal{A} \setminus \{0\}$ машина Тьюринга $T = \langle \mathcal{A}, Q, P, q_1, q_0 \rangle$, которая распознает язык L.

Определим машину Тьюринга $T' = \langle \mathcal{A}, Q', P', q_1, q_0' \rangle$, где $Q' = Q \cup \{q_0'\}$, $P' = P \cup \{q_0 0 \to q_0' 1, q_0 1 \to q_0' 0, \dots\}$. Другими словами, мы добавили в машину T новое заключительное состояние q_0' и две команды $q_0 0 \to q_0' 1$ и $q_0 1 \to q_0' 0$ (конкретный вид команд $q_0 a_i \to \dots$, где $a_i \notin \{0, 1\}$, не имеет значения).

Ясно, что определенная таким образом машина T' полиномиально ограничена над алфавитом $\mathcal{A}_0$ и распознает дополнение $\mathcal{A}_0^* \setminus L$. □

Замечание. Отметим без доказательства, что класс $\mathbb{P}$ замкнут также относительно объединения, пересечения, конкатенации и звездочки Клини.

Класс $\mathbb{NP}$ также замкнут относительно объединения, пересечения, конкатенации и звездочки Клини. Однако вопрос о замкнутости класса $\mathbb{NP}$ относительно дополнения до сих пор является открытым и тесно связан с проблемой «$\mathbb{P} = \mathbb{NP}$?».

Предложение 76. *Если класс $\mathbb{NP}$ не замкнут относительно операции дополнения, то $\mathbb{P} \neq \mathbb{NP}$.*

Доказательство. Допустим, существует язык L такой, что $L \in \mathbb{NP}$, но его дополнение $\overline{L} \notin \mathbb{NP}$. Докажем, что в этом случае $L \notin \mathbb{P}$. Если, напротив, $L \in \mathbb{P}$, то в силу предложения 75 $\overline{L} \in \mathbb{P}$. Следовательно, в силу предложения 74 заключаем, что $\overline{L} \in \mathbb{NP}$, что противоречит нашему первоначальному предположению. □

В заключение параграфа докажем, что класс $\mathbb{P}$ нетривиален, т. е. является собственным подклассом в классе всех языков, распознаваемых (детерминированными или недетерминированными) машинами Тьюринга.

Теорема 77. *Существует распознаваемый машиной Тьюринга язык L, не принадлежащий классу $\mathbb{P}$.*

Доказательство. Для определения требуемого языка L нам необходимо закодировать каждую машину Тьюринга в виде слова над некоторым алфавитом. Пусть

$c = q_i a_j \to q_k a_l S$, где $S \in \{\Lambda, R, L\}$, — произвольная команда машины Тьюринга. Сопоставим ей слово

$$\widehat{c} = 1^i \mid 1^j \mid 1^k \mid 1^l \mid 1^s,$$

где $s = 0$, если $S = \Lambda$; $s = 1$, если $S = R$; и $s = 2$, если $S = L$. Если теперь T — произвольная детерминированная машина Тьюринга с программой $\{c_1, \ldots, c_m\}$, то сопоставим ей слово

$$\widehat{T} = \widehat{c_1} \parallel \widehat{c_2} \parallel \ldots \parallel \widehat{c_m}.$$

Зафиксируем алфавит $\mathcal{A}_0 = \{1, |, \|\}$ и рассмотрим над этим алфавитом следующий язык:

$$L = \{\widehat{T} \mid \text{машина Тьюринга } T \text{ перерабатывает конфигурацию } q_1 0 \widehat{T} 0 \text{ в конфигурацию } u q_0 1 v \text{ для некоторых } u \text{ и } v, \text{ не более, чем за } 2^{|\widehat{T}|} \text{ шагов}\}.$$

Язык L распознаваем машиной Тьюринга. Действительно, существует интуитивно вычислимая процедура (машина Тьюринга), которая по произвольному слову w над алфавитом $\mathcal{A}_0$ сначала проверяет, является ли оно словом вида $\widehat{T}$ для какой-либо детерминированной машины T. Затем, если w имеет вид $\widehat{T}$, данная процедура восстанавливает программу T и проверяет, действительно ли она перерабатывает конфигурацию $q_1 0 \widehat{T} 0$ в конфигурацию вида $u q_0 1 v$ не более, чем за $2^{|\widehat{T}|}$ шагов. Описанную процедуру можно смоделировать на машине Тьюринга.

Допустим теперь, что $L \in \mathbb{P}$. Следовательно, по предложению 75 язык $\overline{L}$ полиномиально распознается на некоторой детерминированной машине Тьюринга M. Пусть $p(x)$ — соответствующий полином из определения полиномиальной ограниченности M. Без ограничения общности можно считать, что

$$p(|\widehat{M}|) < 2^{|\widehat{M}|}.$$

Действительно, поскольку $\frac{p(n)}{2^n} \to 0$ при $n \to \infty$, найдется $n_0 \in \omega$ такое, что для всех $n \geqslant n_0$ выполняется $p(n) < 2^n$. Отсюда следует, что, добавив в машину M необходимое число фиктивных состояний и команд (т. е. команд, не влияющих на работу машины), мы можем увеличить число $|\widehat{M}|$ так, чтобы имело место неравенство $|\widehat{M}| \geqslant n_0$, из которого вытекает справедливость условия $p(|\widehat{M}|) < 2^{|\widehat{M}|}$.

Исследуем теперь вопрос принадлежности слова $\widehat{M}$ языку L. Условие $\widehat{M} \in L$ эквивалентно следующему условию:

1) *Машина M перерабатывает конфигурацию $q_1 0 \widehat{M} 0$ в конфигурацию $u q_0 1 v$ для некоторых u и v не более, чем за $2^{|\widehat{M}|}$ шагов.*

Поскольку машина M, будучи запущенная из конфигурации $q_1 0 \widehat{M} 0$, всегда останавливается не более, чем за $p(|\widehat{M}|) < 2^{|\widehat{M}|}$ шагов, то условие (1) равносильно следующему условию:

2) *Машина M перерабатывает конфигурацию $q_1 0 \widehat{M} 0$ в конфигурацию вида $u q_0 1 v$.*

Наконец, вспомним, что машина M распознает язык $\overline{L}$. Отсюда вытекает, что условие (2) эквивалентно условию $\widehat{M} \notin L$.

Таким образом, имеет место эквивалентность $\widehat{M} \in L \iff \widehat{M} \notin L$. Противоречие. Следовательно, $L \notin \mathbb{P}$. $\square$

§ 26. $\mathbb{NP}$-полные проблемы

Еще никому не удалось доказать, что существует язык из класса $\mathbb{NP}$, не принадлежащий классу $\mathbb{P}$. Проблема «$\mathbb{P} = \mathbb{NP}$?» по-прежнему остается открытой. Однако можно доказать, что некоторые языки не менее «трудны», чем любой язык из $\mathbb{NP}$, в том смысле, что если бы у нас был детерминированный алгоритм, распознающий один из этих «не менее трудных» языков за полиномиальное время, то можно было бы для каждого языка из класса $\mathbb{NP}$ найти детерминированный алгоритм, распознающий его за полиномиальное время. Такие языки называются $\mathbb{NP}$-*полными*.

Для определения $\mathbb{NP}$-полноты необходимо сначала ввести понятие полиномиальной сводимости, которое в свою очередь использует понятие полиномиально вычислимой функции.

Определение. Пусть $T = \langle \mathcal{A}, Q, P, q_1, q_0 \rangle$ — детерминированная машина Тьюринга и задан некоторый алфавит $\mathcal{A}_0 \subseteq \mathcal{A} \setminus \{0\}$. Говорят, что машина T *вычисляет* всюду определенную словарную функцию $f : \mathcal{A}_0^* \to \mathcal{A}_0^*$, если для любого слова $w \in \mathcal{A}_0^*$ имеет место

$$q_1 0 w 0 \underset{T}{\Longrightarrow} q_0 0 f(w) 00^s, \text{ для некоторого } s \geqslant 0.$$

Другими словами, машина T на входе $q_1 0 w 0$ всегда останавливается, преобразуя начальную конфигурацию без достраивания новых ячеек слева в конфигурацию $q_0 0 f(w) 00 \ldots 0$.

Функция $f : \mathcal{A}_0^* \to \mathcal{A}_0^*$ называется *полиномиально вычислимой*, если существует полиномиально ограниченная над алфавитом $\mathcal{A}_0$ детерминированная машина Тьюринга, которая вычисляет f.

Замечание. Для всюду определенных функций вида $f : \omega \to \omega$ предыдущее определение согласуется со старым определением вычислимой по Тьюрингу частичной функции. При этом мы, как обычно, заменяем числовую функцию $f : \omega \to \omega$ на ее словарный аналог $f : \{1\}^* \to \{1\}^*$ по правилу $f(1^n) = 1^{f(n)}$ и рассматриваем машины Тьюринга над алфавитом $\mathcal{A} = \{0, 1\}$.

Таким образом, любая полиномиально вычислимая функция $f : \omega \to \omega$ обязана быть частично вычислимой. Возникает естественный вопрос: существуют ли частично вычислимые функции, которые не вычислимы полиномиально? Интуитивные рассуждения, приведенные во введении в данную главу, дают нам основания для утвердительного ответа на этот вопрос. Ниже мы дадим более формальное доказательство этого факта.

Теорема 78. *Существует двухместная вычислимая функция $F(n, x)$, универсальная для семейства $\{f : \omega \to \omega \mid f$ — полиномиально вычислима$\}$.*

Доказательство. Мы изложим только идею доказательства. Пусть $p_m(x)$ — универсальная вычислимая функция для семейства всех полиномов с натуральными коэффициентами (см. предложение 65).

Неформальный алгоритм для вычисления функции $F(n, x)$ имеет следующее описание. Запускаем машину Тьюринга с номером $(n)_0$ на входной конфигурации $q_1 01^x 0$ и проделываем ровно $p_{(n)_1}(x)$ шагов работы машины. Если за данное количество шагов вычисление успешно завершилось, т. е. машина перешла без достраивания новых ячеек слева в конфигурацию вида $q_0 01^y 0 \ldots 0$, то выдаем y в качестве значения $F(n, x)$. В противном случае выдаем 0.

Определенная подобным образом функция F является искомой. □

Следствие 79. *Существует вычислимая функция, не являющаяся полиномиально вычислимой.*

Доказательство. Допустим, напротив, любая вычислимая функция вида $f : \omega \to \omega$ полиномиально вычислима. Следовательно, семейство всех одноместных вычислимых функций совпадает с семейством $\{f : \omega \to \omega \mid f$ — полиномиально вычислима$\}$. Тогда вычислимая функция $F(n, x)$, построенная в теореме 78, будет универсальной для семейства всех одноместных вычислимых функций. Последнее противоречит предложению 62. □

Определение. Говорят, что язык $L_1 \subseteq \mathcal{A}_0^*$ *полиномиально сводится* к языку $L_2 \subseteq \mathcal{A}_0^*$, если существует полиномиально вычислимая словарная функция $f : \mathcal{A}_0^* \to \mathcal{A}_0^*$, такая, что для любого слова $w \in \mathcal{A}_0^*$ выполняется:

$$w \in L_1 \iff f(w) \in L_2.$$

В теории сложности традиционно языки отождествляют с проблемами вхождения слов в эти языки. В данной терминологии полиномиальная сводимость языка L_1 к языку L_2 означает, что проблема L_1 не сложнее, чем проблема L_2. Если существует детерминированная машина, разрешающая проблему L_2 за полиномиальное время, то ее можно эффективно преобразовать в детерминированную машину, которая разрешает проблему L_2 также за полиномиальное время.

Определение. Язык $L_0 \subseteq \mathcal{A}_0^*$ называется $\mathbb{NP}$-*полным*, если $L_0 \in \mathbb{NP}$ и любой язык $L \in \mathbb{NP}$ полиномиально сводится к L_0.

Из определения следует, что все $\mathbb{NP}$-полные языки сводятся друг к другу. Семейство $\mathbb{NP}$-полных языков играет очень важную роль в теории сложности. Если хотя бы для одной из $\mathbb{NP}$-полных проблем удастся найти детерминированный алгоритм, решающий эту проблему за полиномиальное время, то будет справедливо равенство $\mathbb{P} = \mathbb{NP}$, и все остальные проблемы из этого семейства также будут решаться с помощью детерминированных полиномиальных алгоритмов.

Теорема 80. *Пусть L — произвольный $\mathbb{NP}$-полный язык. Равенство $\mathbb{P} = \mathbb{NP}$ выполняется тогда и только тогда, когда $L \in \mathbb{P}$.*

Доказательство. Докажем сначала необходимость. Пусть $\mathbb{P} = \mathbb{NP}$. Из определения $\mathbb{NP}$-полноты вытекает, что $L \in \mathbb{NP}$. Следовательно, $L \in \mathbb{P}$.

Теперь докажем достаточность. Пусть $L \in \mathbb{P}$. Следовательно, найдется детерминированная машина Тьюринга T_1, полиномиально ограниченная полиномом $p_1(n)$, которая распознает L. Пусть далее L' — произвольный язык из класса $\mathbb{NP}$. Требуется показать, что $L' \in \mathbb{P}$.

В силу $\mathbb{NP}$-полноты языка L язык L' полиномиально сводится к L посредством некоторой полиномиально вычислимой функции f. Следовательно, существует детерминированная машина Тьюринга T_2, полиномиально ограниченная полиномом $p_2(n)$, которая вычисляет f.

Рассмотрим композицию машин T_2T_1. Очевидно, что T_2T_1 является детерминированной машиной. Машина T_2T_1 распознает слово w тогда и только тогда, когда $f(w) \in L$. Условие $f(w) \in L$, в свою очередь, эквивалентно условию $w \in L'$. Таким образом, T_2T_1 в точности распознает язык L'.

Докажем, наконец, что машина T_2T_1 полиномиально ограничена. Работа машины T_2T_1 на произвольном входном слове w состоит из двух этапов. На первом этапе машина T_2 перерабатывает w в слово $f(w)$ за не более, чем $p_2(|w|)$ шагов. Поскольку машина T_2 за один такт работы способна записать на ленту не более одного символа, то длина слова $f(w)$ не превосходит $p_2(|w|) + |w|$. Следовательно, на втором этапе машина T_1 перерабатывает слово $f(w)$ в определенное выходное слово за не более, чем $p_1(p_2(|w|) + |w|)$ шагов. Суммируя все подсчеты, заключаем, что машина T_2T_1 останавливается на входе w за не более, чем $p_2(|w|) + p_1(p_2(|w|) + |w|)$ шагов, что является полиномом от $|w|$. □

§ 27. Теорема Кука

В предыдущем параграфе мы ввели понятие $\mathbb{NP}$-полного языка, но мы до сих пор не доказали, что такие языки существует. Теперь нам предстоит восполнить этот пробел. На самом деле, в настоящее время известно достаточно много $\mathbb{NP}$-полных проблем, которые имеют важное практическое значение и которые возникают в таких областях математики, как исследование операций, логика, комбинаторика, искусственный интеллект и др. Мы остановимся на *проблеме выполнимости булевых формул* и докажем ее $\mathbb{NP}$-полноту.

Определение. Пусть $V = \{P_i \mid i \in \omega\}$ — произвольное счетное множество, элементы которого мы будем называть *пропозициональными переменными*. Определим понятие *булевой формулы* от пропозициональных переменных из V следующим образом по индукции:

1) Любая пропозициональная переменная P_i является *булевой формулой*.

2) Если Φ и Ψ — булевы формулы, то слова $(\Phi\&\Psi)$, $(\Phi \vee \Psi)$, $\neg\Phi$ также являются *булевыми формулами*.

Замечание. Мы считаем, что символы $\&, \vee, \neg, (,)$ не являются элементами множества V. Таким образом, любая булева формула — это конечное слово в бесконечном алфавите $V \cup \{\&, \vee, \neg, (,)\}$. Будем считать, что булева формула Φ зависит от конечного списка переменных $\{P_{i_1}, \ldots, P_{i_n}\}$, если любая пропозициональная переменная, имеющая вхождение в слове Φ, является элементом множества $\{P_{i_1}, \ldots, P_{i_n}\}$.

Определение. Пусть $\gamma : V \to \{0, 1\}$ — произвольное отображение из множества пропозициональных переменных V в двухэлементное множество $\{0, 1\}$, которое мы будем называть *означиванием переменных*. Определим по индукции *значение* $\gamma(\Phi)$ *булевой формулы* Φ *при означивании* γ:

1) Если $\Phi = P_i$ — пропозициональная переменная, то $\gamma(\Phi) = \gamma(P_i)$.

2) Если $\Phi = (\Psi\&\Theta)$, или $\Phi = (\Psi\vee\Theta)$, или $\Phi = \neg\Psi$ для некоторых булевых формул Ψ, Θ, то значение $\gamma(\Phi)$ определяется в соответствии со следующей таблицей:

$\gamma(\Psi)$	$\gamma(\Theta)$	$\gamma(\Psi\&\Theta)$	$\gamma(\Psi \vee \Theta)$	$\gamma(\neg\Psi)$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Булева формула Φ называется *выполнимой*, если существует хотя бы одно означивание переменных $\gamma : V \to \{0, 1\}$, для которого $\gamma(\Phi) = 1$.

Замечание. Если булева формула Ф зависит от переменных $\{P_{i_1}, \ldots, P_{i_n}\}$, то, очевидно, значения переменных, не входящих в этот конечный список, никак не влияют на значение формулы Ф. Таким образом, для того, чтобы определить, является ли данная формула Ф выполнимой, достаточно перебрать всевозможные наборы значений n переменных $P_{i_1}, \ldots, P_{i_n}$ (таких наборов будет 2^n) и для каждого такого набора вычислить значение формулы. Если среди полученных значений найдется хотя бы одно значение 1, то Ф выполнима, в противном случае — невыполнима.

Замечание. Как было замечено выше, множество всех булевых формул — это язык в некотором бесконечном алфавите. Однако любая машина Тьюринга обладает лишь конечным внешним алфавитом. Поэтому для того, чтобы обработка булевых формул стала возможной на машинах Тьюринга, нам необходимо представить множество всех формул как язык над конечным алфавитом.

Примем следующие соглашение: *булевы формулы будем представлять в виде слов над конечным алфавитом* $\{0, 1, \ldots, 9, \&, \vee, \neg, (,)\}$, *в которых каждая пропозициональная переменная* P_i *представляется десятичной записью числа* i. *При этом символы* $\&, \vee, \neg, (,)$ *остаются в представлениях формул без изменений.*

Например, формула $\neg(P_{13}\&(P_2 \vee \neg P_5))$ представляется словом $\neg(13\&(2 \vee \neg 5))$.

Всюду далее язык, образованный всеми словами, представляющими выполнимые булевы формулы, будем называть *проблемой выполнимости булевых формул.*

Определение. Пусть $f(n)$ и $g(n)$ — произвольные всюду определенные функции на ω. Будем говорить, что функция $g(n)$ есть $\mathcal{O}(f(n))$, если существуют натуральные $c > 0$ и $d > 0$ такие, что для всех $n \in \omega$ имеет место $g(n) \leqslant c \cdot f(n) + d$.

Предложение 81. *Проблема выполнимости булевых формул принадлежит классу* $\mathbb{NP}$.

Доказательство. Мы приведем неформальное описание недетерминированной машины Тьюринга T, которая по произвольному входному слову w за полиномиальное время определяет, является ли w представлением выполнимой булевой формулы или нет.

Первая стадия работы машины T состоит в том, что машина проверяет, является ли слово w представлением некоторой булевой формулы (если нет, то T сразу выдает отрицательный ответ), и подсчитывает число n пропозициональных переменных, входящих в w.

На второй стадии машина T недетерминированным образом «угадывает» набор длины n из нулей и единиц, которые будут рассматриваться в качестве значений переменных формулы w. Другими словами, на данном этапе вычисление разветвляется на 2^n параллельных ветвей, каждая из которых соответствует одному набору значений переменных.

На третьей стадии угаданные значения (вдоль каждой ветви вычислений) подставляются вместо соответствующих вхождений переменных в слово w. Чтобы закрыть дыры, образующиеся в слове w в результате подстановки одного символа 0 или 1 вместо десятичного представления пропозициональной переменной, потребуется сделать сдвиги символов на ленте машины. Далее вычисляется значение формулы при данном наборе значений переменных.

Если хотя бы на одной ветви вычислений значение формулы равно 1, то машина выдает положительный ответ. Если на всех ветвях значения нулевые, то машина выдает отрицательный ответ. В целом, все три стадии работы машины T требуют $\mathcal{O}(|w|^2)$ шагов. $\square$

Теорема 82 (С. Кук, 1971)**.** *Проблема выполнимости булевых формул* $\mathbb{NP}$*-полна.*

Доказательство. Выше мы уже показали, что проблема выполнимости лежит в классе $\mathbb{NP}$. Осталось доказать, что любой язык L из класса $\mathbb{NP}$ полиномиально сводится к проблеме выполнимости булевых формул. Пусть T — недетерминированная машина Тьюринга, распознающая L за полиномиальное время. Мы опишем алгоритм, который по произвольному слову w, поданному на вход машины T, строит за полиномиальное время булеву формулу Φ такую, что Φ выполнима тогда и только тогда, когда T распознает w. Полином, который ограничивает время построения формулы Φ по слову w, будет зависеть от T.

Пусть $q_0, q_1, \ldots, q_s$ — внутренние состояния машины T, $a_0, a_1, \ldots, a_m$ — символы внешнего алфавита T, а полином $p(n)$ — временна́я сложность машины T. Предположим, что слово w, поданное на вход машины T, имеет длину n. Согласно определению, T распознает w тогда и только тогда, когда существует конечная последовательность конфигураций

$$M_0 \vdash_T M_1 \vdash_T \ldots \vdash_T M_q,$$

такая, что $M_0 = q_1 a_0 w a_0$, $M_q = u q_0 a_1 v$ для некоторых слов u, v; в данном вычислении не достраиваются новые ячейки слева; $q \leqslant p(n)$; и ни одна из указанных конфигураций не занимает более чем $p(n)$ ячеек на ленте.

Построим булеву формулу Φ, моделирующую последовательность конфигураций, проходимых машиной T. Каждый набор значений переменных формулы Φ будет соответствовать, самое большее, одной последовательности конфигураций, возможно, некорректной (т. е. такой, которая не может на самом деле реализоваться). Формула Φ будет принимать значение 1 тогда и только тогда, когда данный набор значений переменных будет соответствовать последовательности конфигураций $M_0, M_1, \ldots, M_q$, приводящей к распознаванию входа. В качестве пропозициональных переменных формулы Φ будем использовать следующие переменные (мы также указываем их подразумеваемую интерпретацию):

а) переменная $X\langle i, j, t\rangle$ принимает значение 1 тогда и только тогда, когда i-ая ячейка в текущей конфигурации машины T содержит символ a_j в момент времени t. Здесь $1 \leqslant i \leqslant p(n)$, $0 \leqslant j \leqslant m$, $0 \leqslant t \leqslant p(n)$;

б) переменная $Y\langle k, t\rangle$ принимает значение 1 тогда и только тогда, когда машина T в момент времени t находится в состоянии q_k. Здесь $0 \leqslant k \leqslant s$, $0 \leqslant t \leqslant p(n)$;

в) переменная $Z\langle i, t\rangle$ принимает значение 1 тогда и только тогда, когда головка машины в момент времени t обозревает i-ую ячейку текущей конфигурации. Здесь $1 \leqslant i \leqslant p(n)$, $0 \leqslant t \leqslant p(n)$.

Таким образом, пропозициональных переменных всего $\mathcal{O}(p^2(n))$, и их десятичные представления в записи формулы Φ будут содержать не более $c \log_{10} n$ разрядов, где c — константа, зависящая от полинома p. В дальнейшем удобно представлять себе каждую пропозициональную переменную как один символ, а не как слово из $c \log_{10} n$ символов. Эта потеря множителя $c \log_{10} n$ никак не отразится на полиномиальной ограниченности времени, затрачиваемого на вычисление той или иной функции.

Для построения формулы Φ нам потребуется вспомогательная булева формула $U(P_1, \ldots, P_r)$, которая принимает значение 1 тогда и только тогда, когда в точности

один из ее аргументов $P_1, \ldots, P_r$ принимает значение 1. Данную формулу можно выразить следующим образом:

$$U(P_1, \ldots, P_r) = (P_1 \vee \ldots \vee P_r) \,\&\, \Big(\mathop{\&}_{i \neq j}(\neg P_i \vee \neg P_j)\Big).$$

Формальная запись формулы $U(P_1, \ldots, P_r)$ имеет длину $\mathcal{O}(r^2)$ (напомним, что мы считаем переменную одним символом).

Не ограничивая общности, будем считать, что машина T модифицирована так, что на входе w она делает ровно $p(n)$ шагов, и все конфигурации, проходимые машиной, содержат ровно $p(n)$ ячеек. Этого можно добиться, дополнив все конфигурации нужным количеством фиктивных ячеек и заставив T, если нужно, работать после остановки, но не изменяя заключительной конфигурации и не сдвигая головки. Поэтому будем строить Ф как конъюнкцию семи формул A, B, C, D, E, F, G, которые утверждают, что на входе w машина T находилась последовательно в конфигурациях $M_0, \ldots, M_q$, причем каждое M_i имеет длину $p(n)$, $q = p(n)$, и в результате работы машина T распознала слово w. Данное утверждение равносильно совокупности следующих условий:

1) В каждой конфигурации головка обозревает ровно одну ячейку.

2) В каждой конфигурации в каждой ячейке ленты стоит ровно один символ.

3) Каждая конфигурация содержит ровно одно состояние.

4) При переходе от одной конфигурации к следующей может измениться содержимое только одной ячейки.

5) Изменение состояния, положения головки и содержимого ячейки при переходе от одной конфигурации к другой происходит в соответствии с программой машины T.

6) Первая конфигурация имеет вид $q_1 a_0 w a_0$.

7) Последняя конфигурация имеет вид $u q_0 a_1 v$ для некоторых слов u, v.

Перейдем к непосредственному построению формул A, B, C, D, E, F, G, которые соответствуют условиям (1)–(7).

1) A утверждает, что в каждый момент времени машина T обозревает ровно одну ячейку. Введем формулу

$$A_t = U(Z\langle 1, t\rangle, Z\langle 2, t\rangle, \ldots, Z\langle p(n), t\rangle),$$

которая утверждает, что в момент t обозревается в точности одна ячейка. Тогда $A = A_0 \& A_1 \& \ldots \& A_{p(n)}$. Заметим, что формула A имеет длину $\mathcal{O}(p^3(n))$ и ее можно записать за такое же время.

2) B утверждает, что каждая ячейка ленты в каждый момент времени содержит ровно один символ. Введем формулу

$$B_{it} = U(X\langle i, 0, t\rangle, X\langle i, 1, t\rangle, \ldots, X\langle i, m, t\rangle),$$

которая утверждает, что i-ая ячейка ленты в момент времени t содержит ровно один символ. Тогда $B = \mathop{\&}_{i,t} B_{it}$. Длина каждой формулы B_{it} не зависит от n, поскольку величина m зависит только от машины T. Следовательно, формула B имеет длину $\mathcal{O}(p^2(n))$.

3) C утверждает, что в каждый момент времени t машина T находится в одном и только одном состоянии, т. е.

$$C = \mathop{\&}_{0 \leqslant t \leqslant p(n)} U(Y\langle 0,t\rangle, Y\langle 1,t\rangle, \ldots, Y\langle s,t\rangle).$$

Так как число s зависит только от машины T, то C имеет длину $\mathcal{O}(p(n))$.

4) D утверждает, что в любой момент времени t может измениться содержимое не более, чем одной ячейки, т. е.

$$D = \mathop{\&}_{i,j,t}\Big(Z\langle i,t\rangle \vee (X\langle i,j,t\rangle \& X\langle i,j,t+1\rangle) \vee (\neg X\langle i,j,t\rangle \& \neg X\langle i,j,t+1\rangle)\Big).$$

Формула $Z\langle i,t\rangle \vee (X\langle i,j,t\rangle \& X\langle i,j,t+1\rangle) \vee (\neg X\langle i,j,t\rangle \& \neg X\langle i,j,t+1\rangle)$ утверждает, что либо

а) в момент t головка обозревает i-ую ячейку, либо

б) символ a_j записан в i-ой ячейке в момент $t+1$ тогда и только тогда, когда он был записан там в момент t.

Поскольку A и B утверждают, что в момент t головка обозревает только одну ячейку и i-ая ячейка содержит лишь один символ, то в момент времени t либо головка обозревает i-ую клетку, либо содержимое i-ой клетки не меняется. Длина формулы D равна $\mathcal{O}(p^2(n))$.

5) E утверждает, что очередная конфигурация машины T получается из предыдущей с помощью применения одной команды из программы машины T. Пусть E_{ijkt} означает одно из следующих утверждений:

а) в момент t i-ая ячейка не содержит символ a_j,

б) в момент t головка не обозревает i-ую ячейку,

в) в момент t машина не находится в состоянии q_k,

г) при переходе от момента t к моменту $t+1$ новая конфигурация машины получается из предыдущей с помощью применения одной команды из программы машины T.

Пусть для данных значений k и j списком

$$\{q_k a_j \to q_{k_1} a_{j_1} S_1, q_k a_j \to q_{k_2} a_{j_2} S_2, \ldots, q_k a_j \to q_{k_u} a_{j_u} S_u\}$$

исчерпываются все команды машины T, начинающиеся на $q_k a_j \to \ldots$ Для каждого $1 \leqslant l \leqslant u$ положим $i_l = i-1$, если $S_l = L$; $i_l = i$, если $S_l = \Lambda$; $i_l = i+1$, если $S_l = R$. Тогда формула E_{ijkt} имеет вид:

$$E_{ijkt} = \neg X\langle i,j,k\rangle \vee \neg Z\langle i,t\rangle \vee \neg Y\langle k,t\rangle \vee \bigvee_{l=1}^{u}\Big(X\langle i,j_l,t+1\rangle \& Y\langle k_l,t+1\rangle \& Z\langle i_l,t+1\rangle\Big).$$

Следовательно, формула E представляется в виде:

$$E = \mathop{\&}_{i,j,k,t} E_{ijkt}.$$

Каждая E_{ijkt} имеет ограниченную длину, не зависящую от n. Поэтому E имеет длину $\mathcal{O}(p^2(n))$.

6) F утверждает, что первая конфигурация имеет вид $q_1 a_0 w a_0 \dots a_0$ (напомним, что все конфигурации состоят ровно из $p(n)$ ячеек, поэтому первая конфигурация дополнена символами a_0), т. е.

$$F = Y\langle 1,0\rangle \& Z\langle 1,0\rangle \& X\langle 1,0,0\rangle \& \Big(\mathop{\&}_{1\leqslant i\leqslant n} X\langle i+1, j_i, 0\rangle\Big) \& \Big(\mathop{\&}_{n+2\leqslant i\leqslant p(n)} X\langle i,0,0\rangle\Big),$$

где $Y\langle 1,0\rangle$ утверждает, что T в момент $t=0$ находится в начальном состоянии q_1; $Z\langle 1,0\rangle$ утверждает, что T в момент $t=0$ обозревает самую левую ячейку ленты; $X\langle 1,0,0\rangle$ утверждает, что самая левая ячейка ленты вначале содержит символ a_0; $\mathop{\&}_{1\leqslant i\leqslant n} X\langle i+1, j_i, 0\rangle$ утверждает, что следующие n ячеек вначале содержат входное слово w (здесь $w = a_{j_1} a_{j_2} \dots a_{j_n}$); и, наконец, $\mathop{\&}_{n+2\leqslant i\leqslant p(n)} X\langle i,0,0\rangle$ утверждает, что в начальный момент оставшиеся ячейки входной ленты заполнены символами a_0. Очевидно, что длина F равна $\mathcal{O}(p(n))$.

7) G утверждает, что последняя конфигурация имеет вид uq_0a_1v для некоторых слов u, v. Для каждого $1 \leqslant i \leqslant p(n)$ введем формулу G_i, которая утверждает, что в момент времени $t = p(n)$ (т. е. после остановки) машина обозревает i-ую ячейку ленты, в которой находится символ a_1. Данная формула записывается следующим образом:

$$G_i = X\langle i, 1, p(n)\rangle \& Z\langle i, p(n)\rangle.$$

Тогда необходимая для нас формула G имеет следующий вид:

$$G = Y\langle 0, p(n)\rangle \& U(G_1, G_2, \dots, G_{p(n)}),$$

где $Y\langle 0, p(n)\rangle$ утверждает, что в момент времени $t = p(n)$ машина достигает заключительного состояния q_0, а $U(G_1, G_2, \dots, G_{p(n)})$ утверждает, что после остановки машина обозревает только одну ячейку ленты, в которой находится символ a_1. Формула G имеет длину $\mathcal{O}(p^2(n))$.

Итак, положим $\Phi = A\&B\&C\&D\&E\&F\&G$. Поскольку каждый из семи конъюнктивных членов состоит не более, чем из $\mathcal{O}(p^3(n))$ символов, сама формула Φ также состоит не более, чем из $\mathcal{O}(p^3(n))$ символов. Так как мы считали каждую пропозициональную переменную за один символ, а в действительности переменные представляются словами длины $\mathcal{O}(\log_{10} n)$, то на самом деле верхней границей для длины формулы Φ будет $\mathcal{O}(p^3(n)\log_{10} n)$, а это не превосходит $cnp^3(n)$, где c — некоторая константа. Таким образом, длина формулы Φ полиномиально зависит от длины слова w. Нетрудно понять, что если заданы слово w и полином p, то формулу Φ можно записать за время, пропорциональное ее длине.

По данной последовательности конфигураций $M_0, M_1, \dots, M_q$, приводящей к распознаванию слова w, можно, очевидно, найти такое присваивание значений 0 и 1 пропозициональным переменным $X\langle i,j,t\rangle$, $Y\langle k,t\rangle$ и $Z\langle i,t\rangle$, что формула Φ примет

значение 1. И наоборот, если задано означивание переменых формулы Ф, при котором она принимает значение 1, то легко найти последовательность конфигураций машины T, которая приводит к распознаванию слова w. Таким образом формула Ф выполнима тогда и только тогда, когда T распознает слово w.

Язык L был произвольным языком из класса $\mathbb{NP}$, и мы доказали, что L полиномиально сводится к проблеме выполнимости булевых формул. Отсюда заключаем, что проблема выполнимости $\mathbb{NP}$-полна. □

Список литературы

1. *Ершов Ю. Л.* Теория нумераций. М.: Наука, 1977.
2. *Ершов Ю. Л., Палютин Е. А.* Математическая логика. СПб.: Лань, 2004.
3. *Йех Т.* Теория множеств и метод форсинга. М.: Мир, 1973.
4. *Касьянов В. Н.* Лекции по теории формальных языков, автоматов и сложности вычислений. Новосибирск: НГУ, 1995.
5. *Мальцев А. И.* Алгоритмы и рекурсивные функции. М.: Наука, 1986.
6. *Мендельсон Э.* Введение в математическую логику. М.: Наука, 1971.
7. *Морозов А. С.* Введение в вычислимость, рукопись. [Электронный ресурс]. Режим доступа: http://math.nsc.ru/~asm256/lect/lect.html
8. *Морозов А. С.* Машины Шёнфилда: Методические указания. Новосибирск: НГУ, 1996.
9. *Роджерс Х.* Теория рекурсивных функций и эффективная вычислимость. М.: Мир, 1972.
10. *Соар Р. И.* Вычислимо перечислимые множества и степени. Казань: Казанское мат. общ-во, 2000.
11. *Khoussainov B., Nerode A.* Automata Theory and Its Applications. Boston: Birkhauser, 2001.
12. *Lewis H. R., Papadimitriou C. H.* Elements of the Theory of Computation. N. J.: Plentice Hall, 1998.
13. *Shoenfield J. R.* Recursion Theory, Lecture Notes in Logic. Berlin: Springer-Verlag, 1993.

Учебное издание

Когабаев Нурлан Талгатович

ЛЕКЦИИ ПО ТЕОРИИ АЛГОРИТМОВ

Учебное пособие

Редактор *Е. П. Войтенко*

Подписано в печать ??.??.2009 г.
Формат 60×84 1/8. Офсетная печать.
Усл. печ. л. ?,?. Уч.-изд. л. ?,?. Тираж 100 экз.
Заказ №

Редакционно-издательский цент НГУ
630090, Новосибирск-90, ул. Пирогова, 2