

МОСКОВСКИЙ
ГОСУДАРСТВЕННЫЙ
ЗАОЧНЫЙ
ПЕДАГОГИЧЕСКИЙ
ИНСТИТУТ

ГЛАВНОЕ УПРАВЛЕНИЕ ВЫСШИХ И СРЕДНИХ
ПЕДАГОГИЧЕСКИХ УЧЕБНЫХ ЗАВЕДЕНИЙ
МИНИСТЕРСТВА ПРОСВЕЩЕНИЯ РСФСР

Ф. Л. ВАРПАХОВСКИЙ

ЭЛЕМЕНТЫ
ТЕОРИИ АЛГОРИТМОВ

ПРОСВЕЩЕНИЕ

1970

ГЛАВНОЕ УПРАВЛЕНИЕ ВЫСШИХ И СРЕДНИХ
ПЕДАГОГИЧЕСКИХ УЧЕБНЫХ ЗАВЕДЕНИЙ
МИНИСТЕРСТВА ПРОСВЕЩЕНИЯ РСФСР

Московский государственный заочный
педагогический институт

Ф. Л. Варпаховский

**ЭЛЕМЕНТЫ
ТЕОРИИ АЛГОРИТМОВ**

*Учебное пособие для заочных отделений
физико-математических факультетов
педагогических институтов*

ИЗДАТЕЛЬСТВО «ПРОСВЕЩЕНИЕ»

Москва — 1970

СОДЕРЖАНИЕ

Предисловие : :	3
§ 1. Понятие алгоритма	4
§ 2. Машины Тьюринга и тезис Чёрча	10
§ 3. Нумерация машин и пример алгоритмической неразрешимости	17
§ 4. Нормально вычислимые функции	20

Редактор О. А. Павлович
Редактор издательства Л. М. Котова
Технический редактор Л. К. Кухаревич
Корректор Р. Грошева

Сдано в набор 13/IV 1970 г. Подписано к печати 26/VIII 1970 г. 84×108^{1/32}
Бумага типографская № 3. Печ. л. 0,75 Услов. л. 1,26 Уч.-изд. л. 1,15
Тираж 10 тыс. экз. (План 1970 г) А-08659,

Издательство «Просвещение» Комитета по печати при Совете Министров РСФСР
Москва, 3-й проезд Марьиной рощи, 41
Типография № 2 Росглаволиграфпрома, г. Рыбинск, ул. Чкалова, 8. Зак. 1636
Цена 3 коп.

ПРЕДИСЛОВИЕ

В настоящее время на русском языке имеется достаточное число книг и монографий, посвященных теории алгоритмов. Укажем, например, на обстоятельную монографию А. А. Маркова «Теория алгорифмов», книги А. И. Мальцева «Алгоритмы и рекурсивные функции» и В. А. Успенского «Лекции о вычислимых функциях», наконец, книгу С. К. Клини «Введение в математику», в которой содержится подробное изложение теории алгоритмов. Все перечисленные монографии, однако, ориентированы, главным образом, на специалистов в соответствующей области (или родственных областях), в них подробно развивается общая теория алгоритмов с использованием громоздкого технического аппарата.

Между тем для студентов-заочников педагогических институтов требуется руководство, которое, давая ясное представление об общем понятии алгоритма, машине Тьюринга, тезисе Чёрча, алгоритмической неразрешимости, не было бы при этом перегружено техническими деталями. В известной степени указанным требованиям удовлетворяет прекрасная брошюра Б. А. Трахтенброта «Алгоритмы и машинное решение задач». Но эта увлекательная книга ставит своей целью скорее заинтересовать, чем научить, не говоря уже о том, что она стала почти библиографической редкостью.

Настоящее пособие представляет собой попытку элементарного изложения основ теории алгоритмов, которое могло бы служить требуемым руководством для студентов педвузов. Общий план изложения заимствован из лекций, прочитанных П. С. Новиковым на курсах усовершенствования учителей при Московском государственном педагогическом институте им. В. И. Ленина.

Читателю, которого интересует предмет, можно рекомендовать перечисленные выше книги по теории алгоритмов.

Автор

§ 1. ПОНЯТИЕ АЛГОРИТМА

Понятие алгоритма не является точным математическим понятием и не допускает поэтому математического определения. Можно, следовательно, только описать и объяснить это понятие, и соответствующее «описательное определение» будет приведено в этом параграфе. Вначале, однако, полезно рассмотреть несколько примеров.

Введем прежде всего необходимое для дальнейшего понятие *алфавита*. Под алфавитом будем понимать конечный набор различных символов (*букв*), причем запись $A = \{a_1, \dots, a_s\}$ означает, что алфавит A состоит из букв a_1, \dots, a_s ($s \geq 1$). Словами алфавита называются конечные последовательности букв алфавита. Например, для алфавита $A = \{\square, |, +\}$ словами будут последовательности $\square++$, $|||||$, $+\square+|$, $+$, $\square\square||||$ и т. д. Для удобства вводится также *пустое слово*, которое состоит из 0 букв и считается словом любого алфавита. Слова будем обозначать буквами греческого алфавита, пустое слово — символом Λ . Переходим теперь к примерам.

Пример 1. Указать правило, которое для каждого натурального числа n позволяло бы установить за конечное число шагов, делится это число на данное натуральное число $p > 0$ или нет¹.

Можно, например, поступать следующим образом. Если n меньше p , то n делится на p в случае, когда $n = 0$, и не делится в противном случае. Если же n не меньше p , то вычтем p из n и пусть $n_1 = n - p$. Если n_1 меньше p , то n делится на p в случае, когда $n_1 = 0$, и не делится в противном случае. Если же n_1 не меньше p , то вычтем p из n_1 , получим $n_2 = n_1 - p$ и т. д. Ясно, что после конечного числа вычитаний мы получим число n_k , меньшее p , и по тому,

¹ Под натуральными числами здесь и в дальнейшем понимаются нуль и все целые положительные числа.

равно n_k нулю или нет, сможем заключить, делится n на p или нет.

Рассмотрим этот пример более детально. Отметим прежде всего, что поставленная «общая» задача включает в себя бесконечное (счетное) множество «частных» задач. Так, если число p фиксировано, например положено $p=23$, то можно говорить о следующих частных задачах: делится ли 0 на 23 (задача 1), делится ли 1 на 23 (задача 2), делится ли 2 на 23 (задача 3) и т. д. Приведенное выше правило одинаково пригодно для решения любой из этих задач, и решение каждой из них получается по данному правилу после проведения конечного числа операций (шагов). В нашем случае шаги эти включают в себя операции вычитания и сравнения чисел по их величине. Эти операции, в свою очередь, могут быть «разложены» на более простые операции также с конечным числом шагов. Вопрос о том, какие же операции следует считать наиболее простыми, не поддающимися дальнейшему разложению, мы сейчас оставляем в стороне, довольствуясь уверенностью в «конечности» каждой из применяемых здесь операций.

Следующее важное замечание относится к вопросу «кодирования» условий частных задач и их решений. Именно, речь идет о возможности задания такого алфавита A с буквами a_1, \dots, a_s , словами которого можно было бы выразить как условия всех частных задач, так и их решения (т. е. ответы к задачам), причем по кодирующему слову соответствующее условие или ответ должны восстанавливаться однозначно. Оказывается, что в рассмотренном случае такой алфавит возможен — достаточно даже просто однобуквенного алфавита, скажем алфавита A_1 , состоящего из единственной буквы $|$: $A = \{ | \}$ (в алфавите A_1 возможны только слова $\underbrace{|| \dots ||}_k$,

состоящие из k последовательных «палочек», которые мы будем называть словами длины k ; в частности, пустое слово называется словом длины нуль). Закодируем, например, ответ *делится* словом $|$, ответ *не делится* — пустым словом, а условие частной задачи, в которой требуется определить, делится ли число k на p , — словом длины k .

У п р а ж н е н и е 1. Вывести из данного примера правило (конечную процедуру) для решения следующей за-

дачи: для каждого числа n узнать, каков остаток от деления этого числа на данное число p . Как могут быть закодированы условия и решения соответствующих частных задач?

У п р а ж н е н и е 2. Не прибегая к процессу последовательного вычитания, а пользуясь известным признаком делимости на 9, указать правило, которое для каждого натурального числа n позволяло бы определить, делится это число на 9 или нет. Как обосновать «конечность» применяемой процедуры?

П р и м е р 2. Найти правило, позволяющее для любых двух натуральных чисел a и b , не равных одновременно нулю, определить за конечное число шагов их наибольший общий делитель.

Правило нахождения наибольшего общего делителя, известное со времен древности и называемое алгоритмом Евклида, состоит в следующем. Пусть $a \geq b > 0$ ¹. Делим a на b , получаем остаток $r_1 < b$. Если $r_1 = 0$, то b есть наибольший общий делитель, если же $r_1 > 0$, то искомый делитель совпадает с наибольшим общим делителем чисел b и r_1 . Поэтому делим b на r_1 , и если новый остаток $r_2 = 0$, то r_1 — искомый делитель, если $r_1 > r_2 > 0$, то делим r_1 на r_2 . Продолжая этот процесс, получим:

$$a \geq b > r_1 > r_2 > \dots > r_n > r_{n+1} = 0.$$

Последнее отличное от нуля число в этой цепочке (r_n) и будет искомым. При этом весь процесс закончится в конечном числе шагов, поскольку натуральных чисел, меньших числа b , — конечное число.

Приведенная здесь процедура применима к любой паре чисел, не равных одновременно нулю, и для каждой такой пары дает ответ после выполнения конечного числа операций. Если, например, взять пару 124, 44, то получим:

$$r_1 = 36, r_2 = 8, r_3 = 4, r_4 = 0.$$

Искомый делитель равен 4.

Условия и ответы к частным задачам снова могут быть выражены в подходящем алфавите, например в алфавите $A = \{ |, 0 \}$. Так, условие рассмотренной числовой

¹ Если в точности одно число из пары a, b отлично от нуля, то оно и будет искомым общим делителем.

задачи кодируется словом $\underbrace{|| \dots ||}_{124} | 0 | \underbrace{|| \dots ||}_{44}$, а ее решение — словом $||||$, если число n кодировать с помощью слова алфавита A_1 длины n .

У п р а ж н е н и е 3. Найти правило, которое для любого натурального числа n позволяет конечным способом определить, с каким показателем данное простое число p входит в разложение числа n на простые множители (если n не делится на p , то считается, что p входит в указанное разложение с показателем нуль).

П р и м е р 3. Пусть задан какой-нибудь алфавит $A = \{a_1, \dots, a_s\}$, состоящий из $s \geq 1$ букв, и некоторое слово α в этом алфавите. Требуется указать конечную процедуру, позволяющую для любого слова β узнать, содержит ли оно в качестве своей связной части слово α («входит» ли слово α в слово β).

Требуемая процедура получается из следующих соображений: связных частей слова β , имеющих столько же букв, сколько α , — конечное число. Поэтому потребуется только конечное число шагов для побуквенного сравнения всех таких частей с α , и в ходе этого сравнения мы либо обнаружим тождественность какой-нибудь такой части и слова α , либо убедимся, что все эти части отличны от α . Скажем, если $A = \{ |, +, a \}$, $\alpha = || +$, $\beta = | + | a a | +$, то требуется испытать на совпадение с α следующие части β : $| + |$, $| + | a$, $| a a$, $| a a |$, $| a | +$. Ни одна из этих частей с α не совпадает, поэтому β не содержит α .

В этом примере условие каждой частной задачи можно кодировать словом β уже введенного алфавита, утвердительный ответ — словом $|$, отрицательный — словом $||$ (ответ к задаче $| + | a a | +$ будет тогда $||$).

П р и м е р 4. Определим на множестве натуральных чисел следующую функцию $f(n)$: если какая-нибудь цифра встречается в десятичном разложении числа n в точности $n+1$ раз подряд, то $f(n) = 1$, в противном случае $f(n) = 0$. Требуется указать процедуру, позволяющую для каждого n вычислить в конечное число шагов $f(n)$.

Ясно, что эта задача принадлежит тому же типу, что и рассматривавшиеся ранее. В частности, условия и решения соответствующих частных задач могут быть закодированы словами конечного алфавита. Однако в настоящее время никакое решение указанной задачи неизвестно: с одной стороны, не обнаружено требуемой конечной

процедуры, с другой стороны, не доказана и невозможность подобной процедуры.

Пример 5. Известно, что вещественные числа подразделяются на алгебраические и трансцендентные¹. Можно было бы попытаться поставить такую задачу: указать правило, позволяющее для каждого вещественного числа x в конечное число шагов определить, является это число алгебраическим или трансцендентным. То обстоятельство, что трансцендентность некоторых конкретных чисел (таких, как π или e) была установлена ценой больших усилий, не служит, понятно, принципиальным препятствием для постановки подобной задачи. Нетрудно, однако, заметить, что задача эта существенно отличается от предыдущих. В самом деле, частных задач здесь столько, сколько вещественных чисел, т. е. континуум, в то время как слов любого конечного алфавита — счетное множество. Поэтому кодирование условий задач словами какого-либо конечного алфавита в данном случае невозможно. Но если невозможно конечное кодирование условий задач, то тем более бессмысленно искать конечную процедуру для их решения. В дальнейшем такие «некорректные» задачи рассматриваться не будут.

Пример 6. При определении предикатной формулы исходными символами считаются: а) большие латинские буквы с индексами или без них (предикатные буквы); б) малые латинские буквы с индексами или без них (символы предметных переменных); в) знаки $\neg, \supset, \vee, \&, \neg, \exists, (, \prime$. Если теперь несколько видоизменить это определение, считая, что группа а) состоит из наборов $P, P|, P||, \dots$, а группа б) — из наборов $x, x0, x00, \dots$, то окажется возможным каждую предикатную формулу записывать в конечном алфавите $A = \{P, |, x, 0, (, \neg, \supset, \&, \neg, \exists, \prime\}$. Формулу

$$(((P(x, y)) \vee (\forall x(Q(x)))) \vee (\exists y(R(y))))$$

можно, например, записать следующим образом:

$$(((P(x, x0)) \vee (\forall x(P|(x)))) \vee (\exists x0(P|| (x0))))$$

Ясно, что в алфавите A можно записать предикатную формулу с любым количеством различных букв и различных предметных переменных.

¹ Алгебраическими называются числа, являющиеся корнями многочленов с целыми коэффициентами, все прочие числа называются трансцендентными.

Поставим теперь следующую задачу: указать способ, которой позволил бы для любой предикатной формулы установить в конечное число шагов, является эта формула тождественно истинной или нет. Каждая частная задача, соответствующая какой-либо предикатной формуле, кодируется словом, представляющим эту формулу в алфавите A ; положительные и отрицательные ответы можно кодировать, скажем, словами P и x .

Современные исследования показали, что способа, о котором идет речь, не существует! Тонкое и кропотливое доказательство этого замечательного факта мы здесь привести не можем. Однако в дальнейшем будет рассмотрена сходная задача, решение которой позволит выяснить сущность применяемого здесь метода.

У п р а ж н е н и е 4. Существует ли способ для выяснения в конечное число шагов тождественной истинности произвольной пропозициональной формулы? Как следует видоизменить определение пропозициональной формулы, чтобы все такие формулы можно было бы записывать словами конечного алфавита?

Переходим к рассмотрению понятия алгоритма. Пусть задано бесконечное множество задач и каким-нибудь образом указано, что понимается под решением каждой из них. Пусть далее все задачи и решения к ним записываются (кодируются) словами подходящего алфавита A . Будем говорить, что существует алгоритм для решения данного бесконечного множества задач, если существует е д и н ы й способ, позволяющий для каждой задачи этого множества в конечное число шагов найти ее решение. Иными словами, существование алгоритма означает наличие единого способа, который для каждого слова алфавита A , кодирующего какую-нибудь задачу данного множества, позволяет в конечное число шагов найти слово алфавита A , кодирующее решение этой задачи.

Подчеркнем еще раз, что приведенное здесь описание никоим образом не является математическим определением алгоритма, поскольку лишены математической точности выражения «единый», «конечное число шагов», фигурирующие в данном описании.

На понятие алгоритма можно взглянуть и с несколько иной точки зрения. Мы требовали, чтобы задачи данного множества задач и их решения выражались словами не-

которого алфавита A . Поставим теперь в соответствие каждому слову, кодирующему какую-нибудь из данных задач, слово, кодирующее решение этой задачи. Тем самым на подмножестве множества слов алфавита A будет определена функция, значениями которой также будут слова алфавита A . Область определения этой функции совпадает с множеством слов, кодирующих данные задачи, а множество значений — с множеством слов, кодирующих решения этих задач. Поэтому алгоритм можно понимать как единый способ, позволяющий в конечное число шагов «вычислять» значения построенной функции для любых значений аргумента из ее области определения.

Рекомендуем читателю, освоившись с приведенным описанием, еще раз внимательно просмотреть примеры, приведенные ранее. При этом полезно уяснить, о каком именно бесконечном множестве задач идет речь в каждом примере, как обосновывается конечность применяемой процедуры и т. д.

У п р а ж н е н и е 5. Бесконечное множество задач, о котором говорится в описании алгоритма, на самом деле счетно. Почему?

§ 2. МАШИНЫ ТЬЮРИНГА И ТЕЗИС ЧЕРЧА

В этом параграфе будет изучен один важный специальный класс алгоритмов, так называемые машины Тьюринга. Рассматриваемый здесь вариант машины является некоторой модификацией первоначального варианта, предложенного английским ученым Тьюрингом.

Машина Тьюринга включает в себя *ленту*, бесконечную в обе стороны и разбитую на *ячейки*, в каждой из которых записана ровно одна буква алфавита $A = \{a_0, a_1, \dots, a_n\}$ ($n \geq 1$), называемого *алфавитом машины*. Предполагается, что машина может находиться в одном из конечного числа состояний q_0, q_1, \dots, q_k ($k \geq 1$) и что в каждый момент времени машина «обозревает» в точности одну ячейку ленты.

Машина может совершать следующие *элементарные действия*: 1) изменить свое состояние и заменить букву обозреваемой ею ячейки ленты на любую другую букву своего алфавита; 2) изменить свое состояние и передвинуться на одну ячейку вправо, т. е. начать обозревать ближайшую справа ячейку ленты; 3) изменить свое со-

стояние и передвинуться на одну ячейку влево. Никаких других действий машина производить не может.

Работа машины заключается в последовательном выполнении элементарных действий, причем каждое такое действие однозначно определяется состоянием машины q_i в данный момент и буквой a_j , находящейся в обозреваемой ею ячейке ленты. Поэтому работа машины полностью описывается совокупностью *приказов* или *команд*, которые для каждой пары $q_i a_j$ указывают, каково именно элементарное действие должна произвести машина, находящаяся в состоянии q_i и обозревающая букву a_j . Три возможных типа команд, соответствующих элементарным действиям 1), 2), 3), будут обозначаться соответственно:

$$1) q_i a_j q_l a_m, \quad 2) q_i a_j q_l \Pi, \quad 3) q_i a_j q_l \Lambda.$$

(Запись $q_i a_j q_l a_m$, например, означает команду: машине, находящейся в состоянии q_i и обозревающей ячейку с буквой a_j , перейти в состояние q_l и одновременно заменить в обозреваемой ячейке букву a_j на букву a_m).

Условимся состояние q_0 называть *заключительным* состоянием машины и будем считать, что, оказавшись в состоянии q_0 , машина останавливается, т. е. никаких дальнейших действий не совершает (поэтому ни одна из команд 1), 2), 3) не начинается символом q_0). Состояние q_1 будет называться *начальным* состоянием машины. Букву a_0 будем считать символом «пустой» ячейки, т. е. ячейки, в которой ничего не записано. Совокупность команд, определяющих работу машины, будем называть *программой*. Программа машины Тьюринга с алфавитом $\{a_0, a_1, \dots, a_n\}$ и состояниями q_0, q_1, \dots, q_k содержит равно $k(n+1)$ команд — по числу всевозможных различных пар $q_i a_j$, не начинающихся символом q_0 .

Рассмотрим в виде примера машину Тьюринга, имеющую два состояния q_0, q_1 и алфавит $A = \{a_0, \}$. Программа машины состоит из двух команд $q_1 a_0 q_0 |, q_1 | q_1 \Pi$. Посмотрим, как работает эта машина. Начальное положение машины изобразим следующей схемой:

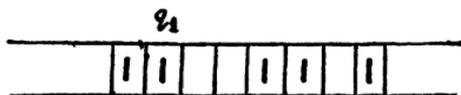
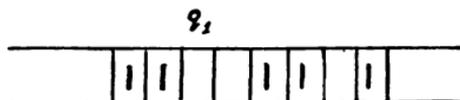
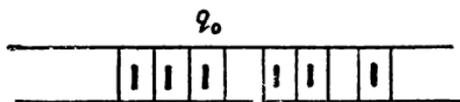


Схема эта означает, что машина, находясь в начальном состоянии q_1 , обозревает ячейку, в которой записана буква |, та же буква записана в соседней слева ячейке, а в соседней справа ячейке ничего не записано (т. е. по нашему соглашению записана буква a_0) и т. д. В непоказанных ячейках также ничего не записано. В соответствии со второй командой машина должна сохранить свое состояние и передвинуться на ячейку вправо. Поэтому положение машины после выполнения первого элементарного действия изобразится схемой:

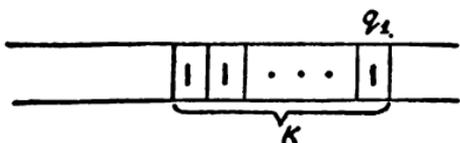


Теперь машина, находясь в состоянии q_1 , обозревает пустую ячейку (ячейку с буквой a_0); поэтому, согласно первой команде, машина переходит в состояние q_0 и «записывает» в пустую ячейку букву | (меняет букву a_0 на букву |):

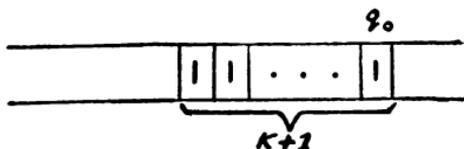


Состояние q_0 — заключительное, и машина останавливается.

У п р а ж н е н и е 1. Убедиться, что, исходя из положения



(начальное положение), данная машина переходит в следующее (заключительное) положение:



У п р а ж н е н и е 2. При выполнении команды $q_i a_j q_i a_m$ машина, вообще говоря, производит одновременно две операции: меняет свое состояние и букву обозреваемой

ячейки. Показать, что с помощью одного дополнительно введенного состояния можно заменить указанное элементарное действие машины последовательностью действий, каждое из которых меняет либо только состояние машины, либо только букву обозреваемой ячейки.

Введем теперь следующие определения. Будем говорить, что некоторое непустое слово α в алфавите $A—a_0 = \{a_1, \dots, a_k\}$ воспринимается машиной в *стандартном положении*, если это слово записано в последовательных ячейках ленты, все другие ячейки ленты пусты (записана буква a_0) и машина обозревает ячейку с последней буквой слова α . Пустое слово воспринимается в стандартном положении, если все ячейки ленты пусты. Стандартное положение называется *начальным*, соответственно *заключительным*, если машина, воспринимающая слово в стандартном положении, находится в начальном, соответственно *заключительном*, состоянии. Так, в начальном положении упражнения 1 слово $\underbrace{|| \dots ||}_k$ воспринимается

машиной в начальном стандартном положении. Аналогично *заключительное* положение машины того же упражнения дает пример *заклучительного* стандартного положения слова $\underbrace{|| \dots ||}_{k+1}$.

Будем говорить, что слово α *перерабатывается* машиной в слово β , если от слова α , воспринимаемого в начальном стандартном положении, машина после выполнения конечного числа элементарных действий (команд) приходит к слову β , воспринимаемому в *заклучительном* стандартном положении.

Основное определение. Пусть на некотором множестве слов алфавита $A = \{a_1, \dots, a_l\}$ определена функция, значениями которой являются слова в том же алфавите. Будем называть эту функцию *вычислимой*, если существует машина Тьюринга с алфавитом $B = \{a_0, a_1, \dots, a_l, \dots, a_k\}$ ($k \geq l$), которая каждое слово из области определения этой функции перерабатывает в слово, являющееся значением функции. Про такую машину Тьюринга говорят, что она *вычисляет* данную функцию.

Рассмотрим, например, функцию, следующим образом определенную на множестве N натуральных чисел: для каждого $n \in N$ $f(n) = n + 1$. Воспользуемся уже применявшимся однобуквенным алфавитом $A_1 = \{|\}$ и будем каж-

дое натуральное число n изображать словом длины n этого алфавита. При таком способе изображения чисел функция $f(n) = n + 1$ переходит в следующую функцию, определенную на множестве всех слов алфавита A_1 : каждому слову длины n ставится в соответствие слово длины $n + 1$. Нетрудно убедиться, что эта функция является вычислимой, поскольку рассмотренная выше машина Тьюринга вычисляет ее (см. программу этой машины и упражнение 1).

Конечно, функция $f(n) = n + 1$ и вычисляющая ее машина Тьюринга весьма просты и не дают сколько-нибудь полного представления о возможностях машин Тьюринга. В этой связи полезно разобрать пример более сложной функции и посмотреть, как строится для нее соответствующая машина Тьюринга. Пусть функция $f_p(n)$ ($p > 0$) определена на N следующим образом:

$$f_p(n) = \begin{cases} 1, & \text{если } n \text{ делится на } p, \\ 0, & \text{если } n \text{ не делится на } p. \end{cases}$$

Через $f_p(\alpha)$ обозначается соответствующая функция в алфавите $A_1 = \{ | \}$.

$$f_p(\underbrace{| \ | \ \dots \ |}_n) = \begin{cases} |, & \text{если } n \text{ делится на } p, \\ \text{пустому слову,} & \text{если } n \text{ не делится на } p. \end{cases}$$

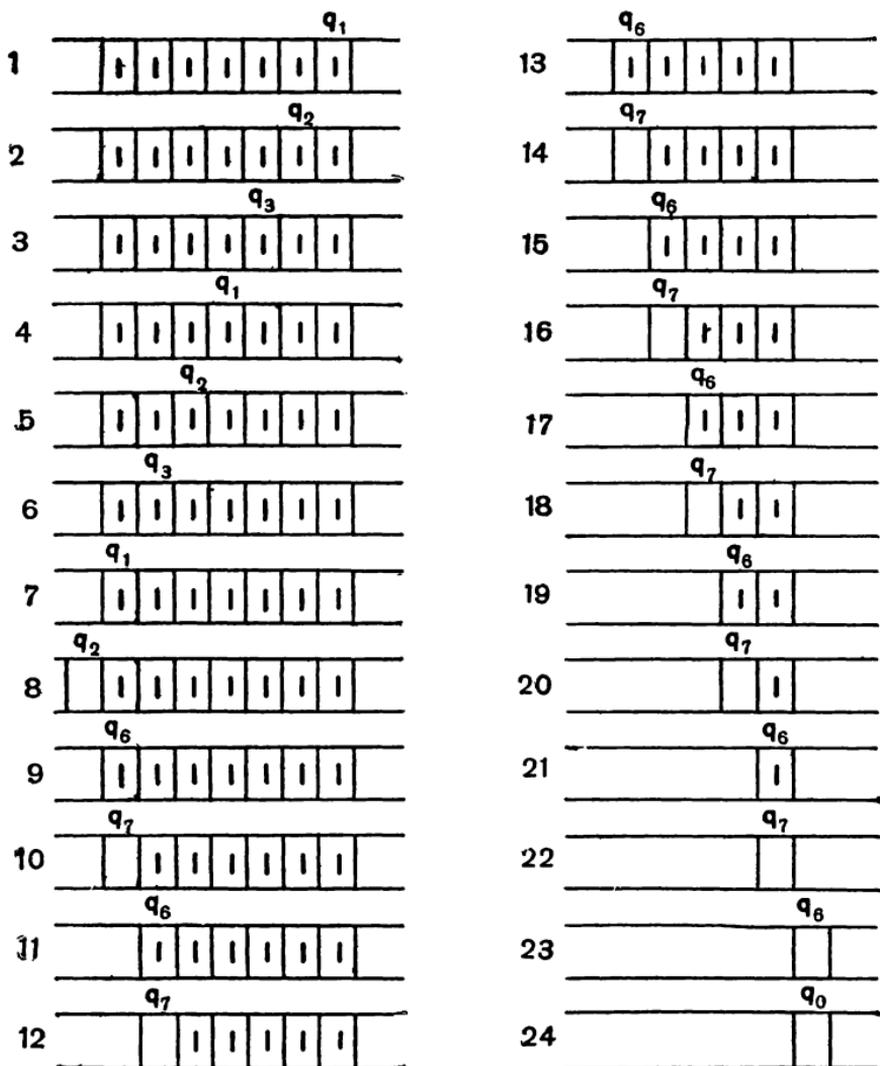
Является ли функция $f_p(n)$ вычислимой? Оказывается, является. Ниже приводится программа машины T_3 с алфавитом $\{a_0, | \}$, вычисляющей функцию $f_3(\alpha)$:

	q_1	q_2	q_3	q_4	q_5	q_6	q_7
a_0	$q_4\Pi$	$q_6\Pi$	$q_6\Pi$	$q_0 $	$q_4\Pi$	$q_0 a_0$	$q_6\Pi$
	$q_2Л$	$q_3Л$	$q_1Л$	$q_5 a_0$		$q_7 a_0$	

Программа записана в виде таблицы, в каждой клетке которой указано, что должна делать машина в зависимости от своего состояния и обозреваемой буквы. Например, запись $q_1Л$ в клетке третьего столбца второй строки таблицы означает команду: если машина находится в состоянии q_3 , в обозреваемой ею ячейке ленты записана буква | (машина «видит» букву |), то она должна перейти в состояние q_1 и передвинуться на ячейку влево. В двух клет-

как ничего не записано, так как соответствующие команды не влияют на работу машины (т. е. какие бы команды здесь ни были записаны, машина все равно вычисляла бы $f_3(\alpha)$). Таким образом, можно подразумевать, что в этих клетках записаны просто какие-нибудь команды.

Посмотрим, как машина T_3 перерабатывает слово $|||||$, исходя из начального стандартного положения. Вот соответствующая последовательность положений ленты и состояний машины:



Вообще, машина «работает» так. Сначала она двигается влево, меняя состояния q_1 на q_2 , q_2 на q_3 , q_3 на q_1 и т. д., пока не дойдет до первой пустой ячейки. Тогда возможны два случая: 1) Машина увидит пустую ячейку в состоянии q_1 (это значит, что число непустых ячеек было кратно 3). Тогда машина переходит в состояние q_4 и и далее «стирает» все «палочки», ставит одну «палочку» и останавливается (переходит в состояние q_0). 2) Машина увидит пустую ячейку в состоянии q_2 или q_3 (это значит, что число непустых ячеек было не кратно 3). Тогда машина переходит в состояние q_6 и далее («стирает» все «палочки» и останавливается. В рассмотренном примере имел место случай 2).

Упражнение 3. Убедиться, что машина T_p с алфавитом $\{a_0, |\}$ и программой

	q_1	q_2	...	q_{p-1}	q_p	q_{p+1}	q_{p+2}	q_{p+3}	q_{p+4}
a_0	$q_{p+1}\Pi$	$q_{p+3}\Pi$...	$q_{p+3}\Pi$	$q_{p+3}\Pi$	$q_0 $	$q_{p+1}\Pi$	$q_0 a_0$	$q_{p+3}\Pi$
	$q_2\Lambda$	$q_3\Lambda$...	$q_p\Lambda$	$q_1\Lambda$	$q_{p+2} a_0$		$q_{p+4} a_0$	

вычисляет функцию $f_p(\alpha)$ в алфавите $A_1 = \{|\}$.

Упражнение 4. Убедиться, что машина с алфавитом $\{\alpha_0, |\}$ и программой

	q_1	q_2	q_3	\bar{q}_1	q_4	\bar{q}_2	q_5	\bar{q}_3	q_6	q_7
a_0	$\bar{q}_1\Pi$	$\bar{q}_2\Pi$	$\bar{q}_3\Pi$	$q_0 a_0$	$\bar{q}_1\Pi$	$q_0 $	$\bar{q}_2\Pi$	$q_7 $	$\bar{q}_3\Pi$	$q_0 $
	$q_2\Lambda$	$q_3\Lambda$	$q_1\Lambda$	$q_4 a_0$		$q_5 a_0$		$q_6 a_0$		$q_7\Pi$

каждое слово длины n алфавита $A_1 = \{|\}$ перерабатывает в слово длины r , где r — остаток от деления n на 3.

В § 1 было показано, что любая алгоритмическая задача есть просто задача о вычислении значений функции, заданной в некотором алфавите. Ясно, что алгоритм заведомо существует (в том понимании, о котором говорилось в § 1), если соответствующая функция вычислима, т. е. существует вычисляющая ее машина Тьюринга. В самом

деле, по ка ж д о м у слову, являющемуся условием одной из задач данной серии, одна и та же машина через конечное число шагов выдаст слово, являющееся решением данной задачи. Так, алгоритмическая задача примера 1 § 1 решается положительно уже потому, что существует машина T_p , вычисляющая функцию $f_p(\alpha)$ (см. определение $f_p(\alpha)$ и упражнение 3).

С другой стороны, многочисленные исследования ученых показали, что во всех известных случаях существования алгоритма для вычисления значений функции, заданной в некотором алфавите, эта функция оказывалась вычислимой, т. е. для нее находилась подходящая вычисляющая машина Тьюринга. Это обстоятельство дало повод американскому ученому Алонсо Чёрчу высказать следующую гипотезу, носящую название тезиса Чёрча:

Алгоритм для нахождения значений функции, заданной в некотором алфавите, существует тогда и только тогда, когда эта функция вычислима.

Тезис Чёрча принципиально не может быть доказан, поскольку понятие алгоритма, как уже говорилось, не является точным математическим понятием. Впрочем, в принципе не исключена возможность, что тезис этот будет опровергнут, хотя такая возможность и кажется весьма маловероятной.

Ценность тезиса Чёрча заключается в том, что он заменяет расплывчатое и неясное понятие алгоритма точным математическим понятием машины Тьюринга. Без этого тезиса доказательство несуществования алгоритма было бы невозможно¹, в то время как уже в § 3 мы сможем, опираясь на гипотезу Чёрча, доказать несуществование алгоритма в конкретной алгоритмической задаче.

§ 3. НУМЕРАЦИЯ МАШИН И ПРИМЕР АЛГОРИТМИЧЕСКОЙ НЕРАЗРЕШИМОСТИ

Каждая машина Тьюринга полностью определяется своей программой, т. е. при заданном алфавите $\{a_0, \dots, a_n\}$ и заданных состояниях q_0, \dots, q_k — совокупностью $(n+1)k$ команд, соответствующих всевозможным парам $q_i a_j$. Две машины Тьюринга, отличающиеся лишь обозначениями букв своих алфавитов и состояний, напри-

¹В математике не может быть доказано отсутствие объекта или конструкции, не имеющих точного математического определения.

мер машины с программами $q_1 a_0 q_0 |$, $q_1 | q_1 \Pi$ и $q a_0 q_0 \times$, $q \times q \Pi$, весьма несущественно разнятся друг от друга, поскольку работают они совершенно одинаково. Так как обозначения состояний машин и букв их алфавитов не играют роли, то их можно черпать из каких-нибудь единых фиксированных последовательностей символов, скажем

$$q_0, q_1, q_2, \dots, q_r, \dots, \\ a_0, |, a_2, \dots, a_s, \dots$$

Теперь можно сделать следующий шаг и выразить все символы этих последовательностей словами подходящего конечного алфавита, например алфавита $\{a_0, |, q\}$. При этом q_i ($i \geq 0$) будем обозначать словом $\underbrace{qq \dots q}_{i+1}$,

a_j ($j \geq 2$) — словом $\underbrace{|| \dots |}_j$. Получатся такие последовательности:

$$q, qq, qqq, \dots, \\ a_0, |, ||, \dots$$

Присоединим к алфавиту $\{a_0, q, |\}$ символы $\Pi, \text{Л}$. Словами нового алфавита $\{a_0, q, |, \Pi, \text{Л}\}$ могут быть выражены как команды машин Тьюринга, так и их программы (последовательности команд). Например, программа машины Тьюринга $q_1 a_0 q_0 |$, $q_1 | q_1 \Pi$ запишется в виде слова:

$$\underbrace{qq a_0 q}_1 | \underbrace{qq}_2 | qq \Pi.$$

Общее правило записи программы машин Тьюринга в алфавите $\{a_0, q, |, \Pi, \text{Л}\}$ заключается в следующем. Сначала записываются все команды, для чего в принятых ранее записях этих команд $q_i a_j q_l a_m$, $q_i a_j q_l \Pi$, $q_i a_j q_l \text{Л}$ символы q_i, a_j, q_l, a_m заменяются подходящими словами алфавита $\{a_0, q, |\}$ по только что приведенному правилу.

Затем программа записывается в виде слова, получающегося в результате последовательного сочленения слов, представляющих команды (в любом порядке).

Упражнение 1. Убедиться, что существует алгоритм, который для любого слова алфавита $\{a_0, q, |, \Pi, \text{Л}\}$ распознает, является это слово записью программы неко-

торой машины Тьюринга или нет. В чем заключается этот алгоритм?

Перенумеруем теперь все машины Тьюринга. Для этого расположим все слова алфавита $\{a_0, q, |, П, Л\}$, представляющие программы машин Тьюринга, в виде фиксированной счетно-бесконечной последовательности, составленной по такому правилу: сначала выписываются в какой-нибудь фиксированной последовательности все однобуквенные слова $\alpha_0, \dots, \alpha_p$, представляющие программы машин Тьюринга, потом выписываются все такие двубуквенные слова $\alpha_{p+1}, \dots, \alpha_r$, потом — трехбуквенные и т. д. Получится последовательность всех программ:

$$\alpha_0, \alpha_1, \dots, \alpha_n, \dots$$

Число k будет называться номером машины T , если программа машины T записывается словом α_k .

У п р а ж н е н и е 2. Убедиться, что каждая машина Тьюринга получит по крайней мере один номер, все натуральные числа окажутся номерами машин Тьюринга и существует алгоритм, позволяющий по каждому такому числу восстановить программу той машины Тьюринга, номером которой оно является. В чем заключается этот алгоритм?

Приведенная нумерация машин Тьюринга и тезис Чёрча, сформулированный в § 2, позволяют построить функцию, для вычисления которой не существует никакого алгоритма. Само существование такой функции, определенной, скажем, в алфавите $A_1 = \{|\}$, вытекает уже из того, что множество функций в этом алфавите несчетно, в то время как из нашей нумерации следует, что множество машин Тьюринга счетно. Нетрудно, однако, указать и конкретную невычислимую функцию в алфавите A_1 . Определим, например, функцию по такому правилу: если слово α длины n в алфавите A_1 перерабатывается машиной Тьюринга с номером n в слово β_n алфавита A_1 ¹, то положим $\psi(\alpha) = \beta_n |$, в противном случае будем считать, что $\psi(\alpha) = |$.

Т е о р е м а. Не существует алгоритма для вычисления значений $\psi(\alpha)$.

¹ Мы вправе говорить о переработке любого слова алфавита A_1 любой машиной Тьюринга, так как каждая машина по нашему соглашению содержит в своем алфавите букву $|$ (см. стр. 18).

Доказательство. Предположим, наоборот, что такой алгоритм существует. Тогда согласно тезису Чёрча функция $\psi(\alpha)$ должна быть вычислимой, т. е. существует вычисляющая ее машина Тьюринга с алфавитом $\{a_0, |, a_2, \dots, a_t\}$. Эта машина должна каждое слово алфавита A_1 переработать в слово того же алфавита (по определению функции $\psi(\alpha)$). Пусть k — какой-нибудь номер этой машины в нашей нумерации. Посмотрим, чему равно слово $\psi(\underbrace{|| \dots |}_k)$. С одной стороны, это слово есть резуль-

тат переработки слова длины k алфавита A_1 машиной с номером k и поэтому оно равно β_k . С другой же стороны, по самому определению функции $\psi(\alpha)$, это слово равно $\beta_k|$. Полученное противоречие доказывает теорему.

Хотя приведенное доказательство весьма просто, оно дает представление о методах, которые применяются для выяснения алгоритмической неразрешимости в значительно более сложных случаях.

Отметим еще, что алгоритмическая неразрешимость вовсе не означает невозможности решить ту или иную задачу из данной серии задач, а лишь отсутствие общего способа для решения всех задач данной серии. Это обстоятельство следует иметь в виду, с тем чтобы не принимать доказательства алгоритмической неразрешимости в качестве математического подтверждения философского тезиса о непознаваемости явлений.

§ 4. НОРМАЛЬНО ВЫЧИСЛИМЫЕ ФУНКЦИИ

Как уже было сказано, всякую алгоритмическую задачу можно понимать как задачу о вычислении значений некоторой функции, заданной в некотором алфавите. Когда возник вопрос о математическом уточнении понятия алгоритмически вычислимой функции, было предложено почти одновременно несколько различных определений, одним из которых является приведенное выше определение вычислимой функции по Тьюрингу. Другое важное определение — определение нормально вычислимой функции — принадлежит советскому математику А. А. Маркову.

В этом параграфе рассматривается определение Маркова, разбираются некоторые примеры и выясняется связь между понятиями вычислимости и нормальной вычислимости.

Для изложения указанных вопросов потребуются некоторые дополнительные обозначения и определения. Греческие буквы по-прежнему будут обозначать слова некоторого алфавита, а символ Λ — пустое слово. Запись $\alpha = \beta\gamma$ будет означать, что слово α получается приписыванием к слову β слова γ справа. Например, для алфавита $A = \{\square, |, +\}$ и слов $\beta = \square\square+|$ и $\gamma = |||+$ слово $\alpha = \beta\gamma = \square\square+|||+$. Говорят, что непустое слово α входит в слово β , если $\beta = a_1\alpha a_2$, где a_1, a_2 могут быть пустыми. При этом слова a_1, a_2 выделяют первое вхождение слова α в слово β , если a_1 пусто или если для каждого из a'_1, a'_2 , таких, что $\beta = a'_1\alpha a'_2$, слово a_1 входит в слово a'_1 . Например, в слове $\beta = +|||+$ первое вхождение слова $\alpha = ||$ выделяют слова $a_1 = +$ и $a_2 = |+$ ($\beta = \underbrace{+}_{a_1} \underbrace{||}_{\alpha} \underbrace{|+}_{a_2}$), но не слова

$$\alpha'_1 = + |, \alpha'_2 = + \left(\beta = \underbrace{+}_{a'_1} \underbrace{|}_{\alpha} \underbrace{|+}_{a'_2} \right).$$

Ясно, что если непустое слово α входит в слово β , то однозначно определяются и слова a_1, a_2 , выделяющие первое вхождение слова α в слово β (надо слова a_1, a_2 выбрать таким образом, чтобы $\beta = a_1\alpha a_2$ и чтобы слово a_1 было «наиболее коротким»).

Формулой подстановки называется выражение вида $\alpha \rightarrow \beta$ или вида $\alpha \rightarrow \cdot \beta$, где α, β — слова данного алфавита и α непусто, а знаки \rightarrow, \cdot не входят в алфавит. Формула $\alpha \rightarrow \beta$ называется *простой*, а формула $\alpha \rightarrow \cdot \beta$ — *заключительной*. Говорят, что формула подстановки (или просто подстановка) $\alpha \rightarrow \beta$ неприменима к слову γ , если α не входит в γ . Если же α входит в γ и a_1, a_2 выделяют первое вхождение α в γ ($\gamma = a_1\alpha a_2$), то *результатом применения подстановки* к слову γ называется слово $a_1\beta a_2$. Так, результатом применения подстановки $|| \rightarrow +$ к слову $+|||+$ будет слово $++|+$.

Распространим теперь приведенное определение на случай, когда в формуле подстановки $\alpha \rightarrow \beta$ слово α пусто. Подстановка $\Lambda \rightarrow \beta$ считается применимой к любому слову γ и ее результатом называется слово $\beta\gamma$. Скажем, результат применения подстановки $\Lambda \rightarrow +$ к слову $+|||+$ есть слово $++|||+$.

Схемой нормального алгоритма A в алфавите A назы-

вается упорядоченная последовательность формул подстановок:

$$\begin{aligned} \alpha_1 &\rightarrow \beta_1 \\ \alpha_2 &\rightarrow \cdot\beta_2 \\ &\dots \\ \alpha_n &\rightarrow \beta_n, \end{aligned}$$

где некоторые формулы простые, а некоторые — заключительные и все α_i, β_i — слова в алфавите A .

Процесс применения нормального алгоритма A к слову γ данного алфавита A заключается в следующем: в схеме алгоритма A ищем первую подстановку, применимую к γ , и применяем ее — получаем слово γ_1 . Если формула применяемой подстановки была заключительной, то процесс на этом оканчивается и считается, что алгоритм A перерабатывает слово γ в слово γ_1 : $A(\gamma) = \gamma_1$. Если ни одна из подстановок неприменима, то процесс также заканчивается и считается, что алгоритм A перерабатывает слово γ в слово γ : $A(\gamma) = \gamma$. Если же применялась простая подстановка, то ищем в схеме A первую подстановку, применимую к γ_1 , и применяем ее. Получаем слово γ_2 и т. д. Ясно, что возможны лишь два случая: 1) процесс на n -м шаге оборвется, тогда считается, что алгоритм перерабатывает слово γ в слово γ_n ($A(\gamma) = \gamma_n$); 2) процесс никогда не оборвется, тогда считаем, что алгоритм не действует на слово γ .

Пример 1. Пусть в алфавите $\{\square, +, |\}$ дана схема A , состоящая всего из одной подстановки: $\Lambda \rightarrow \square$. Первый шаг применения алгоритма к слову α заключается в приписывании к слову α слева буквы \square , получится слово $\square\alpha$. Точно так же на втором шаге получится слово $\square\square\alpha$, на третьем — слово $\square\square\square\alpha$ и т. д. Ясно, что процесс не оборвется ни на каком шаге, так как подстановка $\Lambda \rightarrow \square$ применима к любому слову и не является заключительной. Поэтому нормальный алгоритм A с данной схемой не действует ни на какое слово α . Нетрудно убедиться, что алгоритм со схемой $\Lambda \rightarrow \cdot\square$ перерабатывает всякое слово α в слово $\square\alpha$.

Пример 2. Пусть в алфавите $\{0, 1, 2\}$ нормальный алгоритм A задается схемой:

$$\begin{aligned} 20 &\rightarrow 02 \\ 21 &\rightarrow 12 \\ 2 &\rightarrow \cdot 110 \\ \Lambda &\rightarrow 2 \end{aligned}$$

Посмотрим, в какое слово перерабатывает алгоритм A слово $\alpha = 1010$, заданное в алфавите $\{0, 1\}$. В α отсутствует буква 2, поэтому на первом шаге первой применимой подстановкой окажется подстановка $\Lambda \rightarrow 2$. Следовательно, $\alpha_1 = 21010$. Теперь первой применимой подстановкой будет $21 \rightarrow 12$, поэтому $\alpha_2 = 12010$. Далее применима первая подстановка схемы, именно $20 \rightarrow 02$, так что $\alpha_3 = 10210$. Аналогично $\alpha_4 = 10120$, $\alpha_5 = 10102$. Теперь первая применимая подстановка будет $2 \rightarrow \cdot 110$. Поэтому $\alpha_6 = 1010110$. На этом процесс заканчивается, так как подстановка была заключительной, и получается, что $A(1010) = 1010110$. Вообще, к любому слову α алфавита $\{0, 1\}$ алгоритм «приписывает» слева букву 2 (получается 2α), «продвигает» букву 2 в конец слова (получается $\alpha 2$) и затем заключительно подставляет слово 110 вместо 2 (получается $\alpha 110$). Иначе говоря, алгоритм A каждое слово α алфавита $\{0, 1\}$ перерабатывает в слово $\alpha 110$.

Введем теперь основное определение нормально вычислимой функции. Функция f , заданная на некотором множестве слов алфавита $A = \{a_1, \dots, a_k\}$, называется *нормально вычислимой*, если найдется такой алфавит $A_1 = \{a_1, \dots, a_k, \dots, a_n\}$ ($n \geq k$) и такой нормальный алгоритм A в A_1 , что для каждого слова α алфавита A из области определения f выполняется условие $A(\alpha) = f(\alpha)$.

Пусть, например, функция f задана в алфавите $\{0, 1\}$ и каждому слову α этого алфавита ставит в соответствие слово $\alpha 110$. Тогда f нормально вычислима (см. пример 2).

У п р а ж н е н и е 1. Убедиться, что эта функция f является вычислимой.

В § 2 была рассмотрена функция $f_3(\alpha)$, следующим образом определенная в алфавите $\{|\}$:

$$f_3(\underbrace{|\ |\ \dots\ |}_n) = \begin{cases} |, & \text{если } n \text{ делится на } 3, \\ \Lambda, & \text{если } n \text{ не делится на } 3. \end{cases}$$

Было доказано, что $f_3(\alpha)$ вычислима. Покажем, что эта функция является также нормально вычислимой. Для этого построим в алфавите $\{|\}$ алгоритм A со следующей схемой:

$$\begin{array}{l} | \ | \ | \rightarrow \Lambda \\ | \ | \rightarrow \cdot \Lambda \\ | \rightarrow \cdot \Lambda \\ \Lambda \rightarrow \cdot | \end{array}$$

Алгоритм «работает» по такому принципу: пока число букв $|$ в слове не меньше 3, он «стирает» последовательно по три буквы. Если число букв меньше 3, но больше 0, то заключительно «стираются» оставшиеся буквы $|$; в случае, если слово пусто, оно заключительно переводится в слово $|$. Например, пусть $\alpha = |||||$. Тогда $\alpha_1 = |||$ (первая подстановка), $\alpha_2 = \Lambda$ (первая подстановка), $\alpha_3 = |$ (четвертая, заключительная подстановка) и $A(|||||) = |$.

У п р а ж н е н и е 2. Убедиться, что вычислимая функция упражнения 4 § 2 является нормально вычислимой.

У п р а ж н е н и е 3. Пусть задан алфавит $\{a_1, \dots, a_k\}$ ($k \geq 2$). Для каждого слова α этого алфавита определим слово $S_{a_1}^{a_k}(\alpha)$, которое получается из α заменой каждой буквы a_1 на букву a_k . Рассмотрим функцию $f(\alpha) = S_{a_1}^{a_k}(\alpha)$. Убедиться, что $f(\alpha)$ вычислима и нормально вычислима.

В рассмотренных выше примерах каждая вычислимая функция оказывалась нормально вычислимой и наоборот. Оказывается, это обстоятельство имеет общий характер, т. е. справедлива следующая теорема.

Т е о р е м а. Класс вычислимых функций совпадает с классом нормально вычислимых функций.

Доказательство этой теоремы совсем несложно, но громоздко, и мы поэтому его не приводим. Полезно, однако, уяснить смысл данной теоремы. Понятие нормально вычислимой функции было введено А. А. Марковым в качестве математического уточнения понятия алгоритмически вычислимой функции. Оно было введено независимо от понятия вычислимой функции и по своей исходной конструкции существенно от него отличается. Если бы класс нормально вычислимых функций оказался шире класса вычислимых функций, т. е. если бы существовала нормально вычислимая, но не вычислимая функция, то тезис Чёрча был бы опровергнут, поскольку такая функция была бы алгоритмически вычислимой (в обычном понимании), но не вычислимой по Тьюрингу. Однако ввиду справедливости приведенной теоремы класс тех алгоритмически вычислимых функций, которые нормально вычислимы по Маркову, содержится в классе вычислимых функций, что служит косвенным подтверждением тезиса Чёрча.