## Теория алгоритмов

Полный конспект лекций по курсу<sup>1</sup> Доцент кафедры ДМИ, к. ф.-м. н. С. Ю. Подзоров HГУ, 2003 – 2004.

## 1 Введение

Слово "алгоритм" возникло довольно поздно (оно образовано от имени арабского математика аль-Хорезми, жившего в 9 веке н. э.) Несмотря на это, понятие алгоритма является одним из базовых понятий математики. Прежде чем вести разговор об алгоритмах, следует определить, что мы вкладываем в это слово.

Определение 1 *Алгоритм* — это конечный набор четких, недвусмысленных инструкций, следуя которым можно по входным данным определенного вида получать на выходе некоторый результат.

В качестве иллюстрации приведем несколько примеров:

- 1. Алгоритм сложения чисел в столбик. Если даны два натуральных числа в десятичной записи, то, следуя известному алгоритму, изучаемому в младших классах средней школы, можно получить десятичную запись их суммы.
- 2. Различные алгоритмы построения циркулем и линейкой, изучаемые в средней школе. Например, алгоритм нахождения середины отрезка или алгоритм построения биссектрисы заданного угла.
- 3. Алгоритм решения квадратных уравнений. Подставляя в известную со школы формулу коэффициенты уравнения, можно вычислить количество его корней и сами корни (если они есть).

На ранних этапах своего развития математика была ничем иным, как набором алгоритмов "на все случаи жизни". Сегодняшняя математика — это скорее набор теорем, а не набор алгоритмов, однако так было не

 $<sup>^{1}{\</sup>rm Kypc}$  подготовлен при поддержке Министерства образования РФ, грант КЦФЕ PD02-1.1-475

всегда: доказательства (а вместе с ними и теоремы) появились только в древней Греции, однако еще до греков вавилоняне, шумеры и египтяне знали, как вычислять различные геометрические характеристики объектов и решать простейшие уравнения. Эти рецепты измерений и вычислений не имели обоснования в современном смысле этого слова (сейчас обоснование подразумевает наличие доказательства: например, в курсе алгебры мы сначала доказываем формулу Крамера, а потом пользуемся ею при решении систем линейных уравнений). Откуда древнейшие математики черпали уверенность в справедливости своих алгоритмов — вопрос темный; в основном они занимались хранением имеющихся знаний и их применением, хотя нельзя отрицать, что они эти знания все же понемногу приумножали.

Современная ситуация во многом принципиально отличается от той, что была в древнем мире, однако есть нечто, оставшееся неизменным. Для самих математиков то, чем они занимаются, имеет мало общего с тем, чем занимались их коллеги в древности, однако для конечного потребителя "математического продукта" математика по-прежнему является набором алгоритмов "на все случаи жизни". Алгоритмы — это "продукт" математики, как древней, так и современной, то, что она дает на выходе. Чтобы проиллюстрировать этот тезис, рассмотрим следующую воображаемую ситуацию.

Артиллерист приходит к математику и говорит: "Физик мне сказал, что полет тела в атмосфере подчиняется таким-то и таким-то законам. Дайте мне формулу, при помощи которой я мог бы, подставив в нее массу снаряда, начальную скорость его полета и угол наклона ствола, определить, куда он попадет." Математик садится, составляет систему дифференциальных уравнений, исследует ее, решает, наконец, приходит к артиллеристу и говорит: "Вынужден тебя огорчить, формулы не существует, но ты не отчаивайся. Вот тебе алгоритм, иди к программисту, он напишет программу. Будешь вводить в нее начальные данные, а она тебе будет вычислять место падения снаряда с любой наперед заданной точностью." Артиллерист, в принципе, доволен. Его не интересует ни теория дифференциальных уравнений, ни такая математическая дисциплина, как методы вычислений. Математик сделал свое дело. Он сидел, погружаясь в абстракции, строил гипотезы, доказал пару теорем, а в результате выдал ... алгоритм.

Если принять тезис о том, что алгоритмы — это "конечный продукт" математики, то становится понятным, насколько они важны для мате-

матики в целом. Они оправдывают присутствие этой науки во "внешнем мире". Теоремы и леммы никому, кроме самих математиков, не нужны: если бы еще и алгоритмы никого кроме них не интересовали, то тогда математикой смогли бы заниматься лишь обеспеченные люди, которые могут позволить себе эту роскошь<sup>2</sup>.

Итак, в общих чертах ситуация выглядит следующим образом: некто приносит математику задачу и просит, чтобы он предоставил ему алгоритм ее решения. Математик, подумав, может дать ему этот алгоритм. Может случится так, что он через некоторое время скажет: "Ничего не могу придумать. Задача что-то очень сложная. Буду думать еще." Но возможен и третий вариант: математик, подумав, скажет: "Я тут провел исследования и доказал, что алгоритма для решения вашей задачи не существует." Этот вариант возможен. Известно множество примеров таких задач, для которых не существует алгоритма их решения. Приведем три из них:

- 1. Классические неразрешимые задачи на построение при помощи циркуля и линейки: удвоение куба, триссекция угла, квадратура круга.
- 2. Решение диофантовых уравнений (т. н. 10-ая проблема Гильберта). Доказано, что не существует алгоритма, который по многочлену от нескольких переменных с целыми коэффициентами определял бы, есть ли у него целочисленные корни.
- 3. Неразрешимость элементарной теории арифметики. Не существует алгоритма, который бы по утверждению о натуральных числах выдавал ответ на вопрос: истинно оно или ложно?

Возможно, кого-то эти результаты оставят равнодушным, но у автора этого курса (да и у многих других людей) они всегда вызывали странное чувство, которое точнее всего можно охарактеризовать как чувство немого восхищения. Понятно, как можно найти или пытаться искать алгоритм для решения какой-нибудь конкретной задачи. Но как можно

<sup>&</sup>lt;sup>2</sup>Справедливости ради следует заметить, что ситуация здесь представлена в намеренно упрощенном виде. Например, для физика математика дает не только алгоритмы — она предоставляет ему также язык, на котором он может записывать свои законы. Не желая подробно останавливаться на этих сложных и, в общем-то, не имеющих прямого отношения к нашему курсу вопросах, все же признаем, что в сказанном выше содержится немалая доля истины. А раз так, то алгоритмы — одна из самых важных вещей в математике (если не самая важная).

доказать, что алгоритма не существует? Ведь алгоритмов бесконечно много, перебрать все не удастся!

Очевидно, что должна существовать какая-то мощная теория, которая глубоко проникла в сущность понятия алгоритма (гораздо глубже, чем мы это сделали в определении 1). Такая теория есть, она называется "теория алгоритмов". Именно она и составляет предмет этого курса<sup>3</sup>.

Сразу же следует сказать, что название предмета (равно как и название теории) — не очень удачное. Дело в том, что та теория алгоритмов, которую мы будем изучать, рассматривает далеко не все алгоритмы. Так, например, алгоритмы геометрических построений при помощи циркуля и линейки не имеют к ней никакого отношения. Чтобы точнее определить круг вопросов, которыми занимается теория алгоритмов, будет уместно воспользоваться понятиями, относящимися к современной технике.

В текстах, относящихся к современной электронной технике, часто встречается пара противоположных понятий "цифровой/аналоговый". Можно (достаточно условно) сказать, что аналоговые данные — это данные о значениях какой-то "непрерывной" величины (например, набор действительных чисел). Напротив, цифровые данные носят "дискретный" характер (как правило, это различные последовательности нулей и единиц). Можно также сказать, что цифровые данные — это данные, которые могут быть описаны при помощи конечной последовательности символов. Аналоговые устройства — это устройства для обработки аналоговых данных, цифровые — цифровых. По аналогии можно рассматривать аналоговые и цифровые алгоритмы, соответственно характеру тех данных, с которыми они работают.

Теория алгоритмов, которую мы изучаем, имеет дело только с цифровыми алгоритмами. Так, вторая и третья задачи, упомянутые выше в списке алгоритмически неразрешимых проблем, относятся к теории алгоритмов, а первая — нет. Если обратиться к списку примеров, который приводится в начале этого введения, то первый алгоритм в этом списке является предметом рассмотрения теории алгоритмов, а второй — нет. Третий также не является, если речь идет о точном нахождении корней квадратного уравнения с произвольными действительными коэффициентами, но может являться, если мы рассматриваем уравнения

 $<sup>^3</sup>$ Теория алгоритмов является составной частью большой математической дисциплины, которая называется "математическая логика". Курс математической логики читается на ММФ НГУ во втором и третьем семестрах.

с рациональными коэффициентами и ищем их решения в виде рациональных чисел, которые отличаются от точных решений не более чем на произвольную наперед заданную рациональную величину  $\varepsilon$ .

Для того, чтобы еще точнее задать предмет теории алгоритмов, введем несколько определений.

**Определение 2** *Конструктивным объектом* называется такой объект, который может быть полностью описан при помощи конечной последовательности символов.

## Приведем ряд примеров.

- 1. Натуральные числа являются конструктивными объектами. В качестве описания натурального числа можно рассматривать, например, его десятичную запись. Десятичная запись числа полностью определяет само число; имея десятичную запись, мы обладаем всей информацией о числе, с которым она соотносится.
- 2. Упорядоченные пары натуральных чисел являются конструктивными объектами. Чтобы полностью описать пару, достаточно перечислить через запятую описания последовательно первой и второй компонентов этой пары.
- 3. Конечные последовательности натуральных чисел и конечные подмножества натурального ряда являются конструктивными объектами. Описаниями в данном случае служат перечисления через запятую элементов соответствующих последовательностей в том порядке, в котором они следуют в них (либо перечисления через запятую элементов соответствующих множеств).
- 4. Рациональные числа являются конструктивными объектами. Чтобы полностью описать рациональное число, необходимо всего лишь задать его числитель и знаменатель.
- 5. Иррациональные действительные числа не являются конструктивными объектами. Чтобы задать такое число, необходимо перечислить бесконечное число знаков после запятой, а это нельзя сделать при помощи конечной последовательности символов.

- 6. Бесконечные подмножества натурального ряда не являются конструктивными объектами. Чтобы полностью задать бесконечное подмножество, надо перечислить все его элементы.
- Функции из N в N не являются конструктивными объектами. Чтобы определить функцию, надо задать ее значения на всех аргументах.
- 8. Алгоритмы являются конструктивными объектами. Согласно определению 1, алгоритм это конечный набор инструкций. Сам список этих инструкций будет являться описанием соответствующего алгоритма.

Определение 3 Конструктивным пространством называется множество всех конструктивных объектов одинакового вида.

Примеры конструктивных пространств — множества  $\mathbb{N}$  и  $\mathbb{Q}$ , множество  $\mathbb{N}^2$ , множество конечных последовательностей натуральных чисел и множество конечных подмножеств натурального ряда. Множество же  $\mathbb{N} \cup \mathbb{N}^2$  не является конструктивным пространством: хотя оно и состоит только из конструктивных объектов, в нем встречаются объекты двух разных видов. Также не будет конструктивным пространством и произвольное подмножество  $\mathbb{N}$ , например, множество простых чисел: хотя оно и состоит только из конструктивных объектов одного вида, однако содержит не все такие объекты.

Определение 4 Вычислимой функцией называется функция из одного конструктивного пространства в другое конструктивное пространство, для которой существует алгоритм, позволяющий по описанию любого аргумента получить описание значения функции на этом аргументе.

Предмет теории алгоритмов можно теперь сформулировать следующим образом: "Исследование функций из одного конструктивного пространства в другое конструктивное пространство в связи с вопросами их вычислимости". В связи с этим, поскольку мы ограничиваемся только "цифровыми" алгоритмами (то есть такими алгоритмами, которые работают с конструктивными объектами), было бы более уместно назвать наш курс не "теория алгоритмов", а "теория вычислимости". Однако название "теория алгоритмов" уже является исторически сложившимся и менять его не стоит.

Прежде чем переходить к содержательной части курса необходимо сделать еще одно замечание. Оно опять же касается того, чем мы будем заниматься, а точнее — чем не будем.

Все вопросы, касающиеся алгоритмов преобразования объектов одного конструктивного пространства в объекты другого конструктивного пространства, можно разбить на четыре категориии.

- 1. Вопросы, относящиеся собственно к теории вычислимости. Это вопросы о том, какие функции являются вычислимыми, а какие нет, и вопросы типа: "Какие функции станут вычислимыми, если мы объявим вычислимой какую-нибудь конкретную функцию?"
- 2. Вопросы, относящиеся к теории сложности. Это вопросы о том, как быстро (за какое количество шагов) можно вычислить ту или иную вычислимую функцию, то есть вопросы о сложности оптимальных алгоритмов.
- 3. Вопросы, относящиеся к программированию или, более широко, к computer science. Это вопросы относятся к конкретным алгоритмам вычислений и к их реализации.
- 4. Вопросы, относящиеся к такой математической дисциплине, как методы вычислений. Это вопросы о нахождении приближенных решений различных дифференциальных и интегральных уравнений при помощи цифровых алгоритмов.

Основная область наших интересов относится к первой категории, ей мы посвятим большую часть нашего курса. Немножко затронем и вторую. Третьей и четвертой категориями вопросов мы заниматься не будем, это не наша задача. Данный курс теории алгоритмов не имеет прямого отношения к программированию. Мы занимаемся абстрактными задачами о существовании тех или иных алгоритмов, а не их конкретной реализацией. Что касается четвертой категории вопросов, то по ним на ММФ НГУ читается отдельный курс (даже несколько), нас же они совершенно не касаются.

## 2 Конечные автоматы и языки

Пусть  $\Sigma$  — некоторое непустое конечное множество, которое мы будем называть *алфавитом*, а элементы которого — *символами* или *буквами*.

Под словом алфавита  $\Sigma$  мы понимаем произвольную конечную последовательность элементов из  $\Sigma$ . Каждое слово имеет  $\partial$ лину, которая является некоторым натуральным числом<sup>4</sup> и равна количеству букв в этом слове. Длина слова w обозначается через |w|. Для слова w и натурального числа  $0 < i \le |w|$  через  $(w)_i$  мы обозначаем i-ую букву в слове w. Множество всех слов алфавита  $\Sigma$  мы обозначаем через  $\Sigma^*$ . В  $\Sigma^*$  имеется одно слово нулевой длины, которое мы обозначаем через e и называем nустым словом (при этом мы считаем, что символ e не входит в алфавит).

Для каждого конечного  $\Sigma$  слова алфавита  $\Sigma$  являются конструктивными объектами, а множество  $\Sigma^*$  — конструктивным пространством (в смысле определений 2 и 3). Действительно, в качестве описания слова алфавита  $\Sigma$  можно рассматривать само это слово (либо символ e для пустого слова).

На множестве слов  $\Sigma^*$  определена бинарная операция, которая называется операцией конкатенации. Конкатенация слов w и v — это такое слово, в котором сначала идут все символы слова w, а прямо следом за ними все символы слова v. Конкатенация слов w и v обозначается через wv.

Операция конкатенации не коммутативна, но ассоциативна (то есть wv не всегда равно vw, но для произвольных слов u, v и w всегда справедливо равенство u(vw) = (uv)w). Пустое слово играет для операции конкатенации роль единицы при умножении: действительно, для любого слова w справедливо равенство we = ew = w.

Мы говорим, что слово w является haчaлом слова v, если существует слово u, такое что v=wu. Если слово w является началом слова v, то мы пишем  $w \preccurlyeq v$ .

Под *языком* алфавита  $\Sigma$  мы понимаем произвольное подмножество множества  $\Sigma^*$ .

**Определение 5** Детерминированным конечным автоматом называется произвольная пятерка  $\langle Q, \Sigma, \delta, s, F \rangle$ , такая что Q и  $\Sigma$  — конечные множества,  $s \in Q$ ,  $F \subseteq Q$ ,  $\delta : Q \times \Sigma \to Q$  — функция<sup>5</sup> из  $Q \times \Sigma$  в Q.

 $<sup>^4</sup>$ В нашем курсе натуральный ряд начинается с нуля. Таким образом, множество натуральных чисел  $\mathbb{N}$  — это  $\{0,1,2,\ldots\}$ .

 $<sup>^5</sup>$ Следуя общепринятому в таких случаях соглашению, мы рассматриваем  $\delta$  как функцию от двух аргументов и для  $q\in Q,\,a\in \Sigma$  пишем  $\delta(q,a)$  вместо  $\delta(\langle q,a\rangle)$ 

Пусть  $M = \langle Q, \Sigma, \delta, s, F \rangle$  — детерминированный конечный автомат. Множество  $\Sigma$  называется алфавитом автомата M (ниже мы увидим, что автоматы являются устройствами для распознавания языков; если автомат служит для распознавания языка некоторого алфавита  $\Sigma$ , то то же самое  $\Sigma$  будет алфавитом этого автомата). Множество Q называется множеством состояний автомата M, а элементы этого множества — состояниями. Поскольку в определение автомата входит s — элемент Q, то множество состояний не может быть пустым. Сам элемент s из определения M называется начальным состоянием автомата M. Множество F называется множеством конечных состояний автомата M, а его элементы — конечными состояниями. Наконец,  $\delta$  называется функцией переходов; если для  $q_1, q_2 \in Q$  и  $a \in \Sigma$   $q_2 = \delta(q_1, a)$ , то мы говорим, что автомат M переходит из состояния  $q_1$  в состояние  $q_2$  под действием символа a.

Конечные автоматы принято изображать в виде диаграмм, на которых кружочками отмечены состояния автоматов, а стрелками с надписанными над ними символами — переходы между состояниями. Начальное состояние автомата отмечается на диаграмме в виде маленького треугольника напротив соответствующего кружочка. Конечные состояния обводятся двойными кружками.

В качестве примера рассмотрим следующую диаграмму:

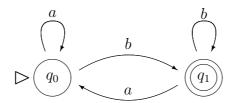


Рис. 1:

На этой диаграмме изображен конечный автомат, алфавит которого состоит из двух символов a и b, множество состояний равно  $\{q_0,q_1\}$ ,  $q_0$  является начальным состоянием, множество конечных состояний состоит из одного элемента и равно  $\{q_1\}$ , а функция перехода задается при

помощи следующей таблицы:

$\overline{q}$	$\overline{x}$	$\delta(q,x)$
$q_0$	a	$q_0$
$q_0$	b	$q_1$
$q_1$	a	$q_0$
$q_1$	b	$q_1$

Определим для конечного автомата  $M = \langle Q, \Sigma, \delta, s, F \rangle$  функцию  $p_M$ :  $Q \times \Sigma^* \to Q$ , которая сопоставляет каждой паре, состоящей из состояния автомата M и слова алфавита  $\Sigma$  некоторое состояние автомата M. Для каждого  $q \in Q$  и  $w \in \Sigma^*$  значение  $p_M(q,w)$  определяется индукцией по длине слова w согласно следующей схеме:

- 1.  $p_M(q, e) = q$ ;
- 2. для любых  $w \in \Sigma^*$  и  $a \in \Sigma$   $p_M(q, wa) = \delta(p_M(q, w), a)$ .

Ясно, что существует единственная функция  $p_M$ , удовлетворяющая этим двум свойствам.

Если для  $q_1, q_2 \in Q$  и  $w \in \Sigma^*$   $p_M(q_1, w) = q_2$ , то мы говорим, что автомат M переходит из состояния  $q_1$  в состояние  $q_2$  под действием слова w. Смысл функции  $p_M$  заключается в следующем: значение  $p_M(q, w)$  равно состоянию, в которое перейдет автомат, если он, находясь в состоянии q, последовательно начнет читать символы слова w и осуществлять под действием этих символов переходы между состояниями согласно функции переходов  $\delta$ . Так, например, если M — это автомат, изображенный на рисунке 1, то, стартуя из состояния  $q_0$  он под действием слова baba будет менять состояния следующим образом: сначала, прочитав символ b, он перейдет из состояния  $q_0$  в состояние  $q_1$ , затем под действием символа a вернется из  $q_1$  в  $q_0$ , потом опять, прочитав символ b, перейдет в  $q_1$ , а затем опять вернется в  $q_0$ . Таким образом,  $p_M(q_0, baba) = q_0$ . Аналогично  $p_M(q_1, baba) = q_0$ ,  $p_M(q_0, abb) = q_1$  и т. д.

Для функции  $p_M$  справедливо следующее: если  $w,v\in \Sigma^*$  и q — состояние автомата M, то  $p_M(q,wv)=p_M(p_M(q,w),v)$ . Доказать это простое равенство можно, например, индукцией по длине слова v.

Мы говорим, что слово w алфавита  $\Sigma$  распознается автоматом  $M = \langle Q, \Sigma, \delta, s, F \rangle$ , если  $p_M(s, w) \in F$ , то есть если автомат M под действием

слова w переходит из начального состояния в одно из конечных состояний. Множество всех слов, распознаваемых автоматом M, мы обозначаем через L(M) и называем языком, распознаваемым автоматом M.

Таким образом, автоматы — это устройства для распознавания языков. Автомат можно рассматривать как представление алгоритма для вычисления функции, действующей из конструктивного пространства  $\Sigma^*$  в конструктивное пространство возможных ответов {да, нет}. Другими словами, автомат M — это алгоритм, который, получив на входе слово w, дает ответ на вопрос: "Верно ли, что  $w \in L(M)$ ?"

Детерминированные конечные автоматы — это первая формализация понятия алгоритма, которую мы рассматриваем в нашем курсе. Нельзя сказать, что она очень удачная. Алгоритмы, которые мы собирались рассматривать, не обязательно должны выбирать конечный результат из двух возможных альтернатив; в определении вычислимой функции не сказано, что число ее возможных значений должно быть не более двух. Но дело даже не в этом. Если ограничиться только функциями из  $\Sigma^*$  в двухэлементное множество, то и тут не все в порядке. Конечно, не подлежит сомнению, что каждая такая функция является вычислимой, если она задается конечным автоматом. Но обратное, к сожалению, не верно. Позже мы увидим, что существует множество языков, для которых явно существует алгоритм их распознавания, но которые не распознаются никаким конечным автоматом.

Тем не менее, конечные автоматы очень важны. Во-первых, существует множество важных языков (например, в программировании), отдельные фрагменты которых распознаются конечными автоматами. Поскольку конечный автомат очень легко запрограммировать и программа будет работать очень быстро, то использование конечных автоматов при создании разных компьютерных программ очень уместно. Во-вторых, изучение конечных автоматов позволит нам глубже разобраться в природе трудностей, возникающих при попытках формализации общего понятия алгоритма. Ну а третья причина ... она носит скорее философский, нежели практический характер, но, возможно, для тех, кто склонен к философии, она и будет самой важной. Дело в том, что хотя мы и можем рассматривать разные идеальные устройства, которые будут гораздо мощнее, чем конечные автоматы, на практике ничего лучше конечных автоматов сконструировать все равно не удастся. Каждый реальнечных автоматов сконструировать все равно не удастся.

но существующий компьютер — это конечный автомат  $^6$ . Конечно, число состояний такого автомата очень велико (если, например, компьютер имеет 256 мегабайт оперативной памяти, то число состояний будет порядка  $2^{8\cdot256\cdot2^{20}}$  — это намного больше, чем число атомов во вселенной). Имея дело с таким огромным числом состояний, на практике гораздо естественнее представлять себе компьютер как устройство с бесконечной памятью, однако в принципе все что он делает — это переходит по стрелкам из одного состояния в другое в зависимости от входных данных.

Прежде чем идти дальше, мы должны ввести несколько очень важных понятий, которые используются не только в теории алгоритмов, но и практически во всех областях современной математики.

**Определение 6** *Бинарным отношением* на множестве A называется произвольное подмножество декартова квадрата  $A \times A$ .

Примеры бинарных отношений всем хорошо известны. Это отношения равенства, неравенств (строгого и нестрогого), сравнения по модулю и т. д. Например, отношение строгого неравенства на множестве действительных чисел — это множество таких пар  $\langle x,y\rangle$  действительных чисел x и y, что x строго меньше чем y. Если R — бинарное отношение на множестве A и два элемента  $a,b\in A$  связаны этим отношением, то этот факт обычно записывают следующим образом: aRb. Конечно, можно было бы обойтись и обычной записью  $\langle a,b\rangle\in R$ , но для часто используемых отношений уже сложился первый вариант. Так, например, для бинарного отношения нестрогого неравенства на множестве целых чисел мы пишем  $2\leqslant 3$  вместо формально корректной, но выглядящей достаточно нелепо записи  $\langle 2,3\rangle\in \leqslant$ .

<sup>&</sup>lt;sup>6</sup>Типичный настольный компьютер состоит из процессора, оперативной памяти и различных периферийных устройств. В компьютере также есть тактовый генератор, который с очень большой частотой генерирует сигналы — такты работы процессора. Процессор выполняет команды и анализирует сигналы, поступающие от внешних устройств, по тактам: каждое действие процессора занимает целое число тактов. Работа компьютера четко детерминированна: состояние процессора и оперативной памяти компьютера на следующем такте работы однозначно зависит от состояния этих элементов на предыдущем такте и от того, какие сигналы поступали в процессор от периферийных устройств в тот момент. Можно считать, что всевозможные состояния памяти и процессора — это внутренние состояния компьютера, а последовательность сигналов от периферийных устройств — это последовательность символов некоторого конечного алфавита, которые компьютер получает на входе, такт за тактом.

Пусть R — бинарное отношение на множестве A. Мы говорим, что отношение R:

- 1.  $pe \phi$ лексивно, если для любого  $a \in A$  имеет место aRa;
- 2. *таких что аRb* и *bRc*, имеет место aRc;
- 3. симметрично, если для любых  $a, b \in A$  из aRb следует bRa;
- 4. aнтисимметрично, если для любых  $a,b \in A$  из aRb и bRa следует a=b.

В приведенных выше примерах все отношения будут рефлексивными и транзитивными. Отношения равенства и сравнения по модулю будут к тому же симметричными, а отношение нестрогого порядка — антисимметричным. Отношение равенства также является антисимметричным (это единственное отношение, удовлетворяющее всем четырем свойствам). Можно легко придумать примеры отношений, которые будут не рефлексивными или не транзитивными; вообще, для большинства комбинаций этих четырех свойств можно придумать отношение, которое удовлетворяет в точности свойствам из данной комбинации.

**Определение 7** Бинарное отношение называется *отношением эквивалентности*, если оно рефлексивно, транзитивно и симметрично.

В качестве примеров отношения эквивалентности можно привести отношение равенства и отношение сравнения целых чисел по модулю какого-нибудь фиксированного положительного числа. Существует и множество других примеров.

**Теорема 1** Пусть R — отношение эквивалентности на непустом множестве A. Существует единственное множество B, такое что:

- 1. элементами множества B являются непустые подмножества A;
- 2. для любых  $b_1, b_2 \in B$  если  $b_1 \neq b_2$ , то  $b_1 \cap b_2 = \emptyset$ ;

 $<sup>^{7}</sup>$ Из симметричности и антисимметричности следует транзитивность. Можно доказать, что это единственное ограничение, которому должно удовлетворять произвольное бинарное отношение.

- 3.  $A = \bigcup_{b \in B} b$ ;
- 4. для любых  $a_1, a_2 \in A$  выполнено  $a_1Ra_2$  тогда и только тогда, когда для некоторого  $b \in B$  справедливо  $a_1, a_2 \in b$ .

Доказательство. Пусть  $a \in A$ . Через [a] обозначим подмножество A, состоящее из всех таких  $x \in A$ , что x связано с a отношением R, то есть множество  $\{x \in A : aRx\}$ . Положим  $B = \{[a] : a \in A\}$ . Покажем, что B удовлетворяет требуемым свойствам.

Элементы B непусты, так как если  $[a] \in B$ , то, в силу рефлексивности R,  $a \in [a]$  и  $[a] \neq \emptyset$ . Сразу же можно отметить и справедливость третьего пункта теоремы: действительно, каждый элемент  $a \in A$  является элементом некоторого элемента B (например, [a]), а то, что все элементы B содержат только элементы A, следует из определения B.

Пусть для  $a_1, a_2 \in A$   $a_1 \in [a_2]$ ; докажем, что  $[a_1] = [a_2]$ . Пусть  $a \in [a_1]$ . Тогда  $a_1Ra$ . Так как  $a_1 \in [a_2]$ , то  $a_2Ra_1$ . Из транзитивности R заключаем, что  $a_2Ra$ , то есть  $a \in [a_2]$ . Пусть, наоборот,  $a \in [a_2]$ . Тогда  $a_2Ra$ . Так как  $a_1 \in [a_2]$ , то  $a_2Ra_1$  и, в силу симметричности R,  $a_1Ra_2$ . Из транзитивности R получаем  $a_1Ra$ , то есть  $a \in [a_1]$ .

Докажем свойство из второго пункта теоремы. Пусть  $[a_1], [a_2] \in B$  и  $[a_1] \cap [a_2] \neq \emptyset$ . Тогда существует  $a \in [a_1] \cap [a_2]$ . Но тогда  $[a_1] = [a]$  и  $[a] = [a_2]$ ; следовательно,  $[a_1] = [a_2]$ .

Докажем четвертый пункт. Если справедливо  $a_1Ra_2$ , то имеем  $a_2 \in [a_1], [a_1] = [a_2]$  и  $a_1, a_2 \in [a_1]$ . Обратно, пусть  $a_1, a_2 \in [a]$ . Тогда  $[a_1] = [a]$ ,  $[a_2] = [a]$  и  $[a_1] = [a_2]$ . Так как  $a_2 \in [a_2]$ , то  $a_2 \in [a_1]$ , а это значит, что  $a_1Ra_2$ .

Осталось доказать единственность B. Пусть множество B' удовлетворяет свойствам 1-4; покажем, что B'=B.

Пусть  $b \in B'$ ,  $a \in A$  и  $a \in b$ . Покажем, что b = [a]. Если  $a' \in b$ , то, по свойству 4, aRa' и  $a' \in [a]$ . Пусть, наоборот,  $a' \in [a]$ . Тогда aRa' и, по свойству 4, a и a' оба принадлежат какому-то элементу множества B'. Однако, по свойству 2, a может принадлежать только одному элементу B', и это элемент b. Значит,  $a' \in b$ .

Докажем, что  $B' \subseteq B$ . Пусть  $b \in B'$ . По свойству 1 существует  $a \in A$ , такой что  $a \in b$ . По доказанному b = [a] и, следовательно,  $b \in B$ .

Докажем обратное включение. Пусть  $b \in B$ . Тогда b = [a] для некоторого  $a \in A$ . По свойству 3 существует  $b' \in B'$ , такой что  $a \in b'$ . Но тогда, по доказанному выше, b' = [a]. Получаем b = b' и  $b \in B'$ .  $\square$ 

Единственное множество B, существование которого доказано в теореме 1, называется  $\phi$ актор-множеством множества A по отношению R и обозначается A/R. Элементы фактор-множества называются классами эквивалентности. Для каждого  $a \in A$  существует единственный класс эквивалентности, содержащий a, который обозначается через  $[a]_R$  (или просто [a], если ясно, о каком отношении идет речь) и состоит из всех таких  $x \in A$ , что aRx. Два элемента множества A принадлежат одному и тому же классу тогда и только тогда, когда они связаны отношением R. Фактор-множество A/R можно мыслить как разбиение множества A на непустые непересекающиеся классы эквивалентности. В теореме 1 доказано, что каждое отношение эквивалентности задает разбиение. Можно доказать и обратную теорему: для каждого разбиения существует единственное отношение эквивалентности, которое его задает.

В качестве примера рассмотрим следующее отношение на множестве натуральных чисел:  $R = \{\langle n, m \rangle : n \equiv m \pmod{5}\}$ . Это отношение эквивалентности. Фактор-множество  $^{\mathbb{N}}/_{R}$  состоит из следующих пяти элементов:  $[0] = \{0, 5, 10, \ldots\}$ ,  $[1] = \{1, 6, 11, \ldots\}$ ,  $[2] = \{2, 7, 12, \ldots\}$ ,  $[3] = \{3, 8, 13, \ldots\}$ ,  $[4] = \{4, 9, 14, \ldots\}$ .

Вернемся к теории алгоритмов. Пусть  $M = \langle Q, \Sigma, \delta, s, F \rangle$  — детерминированный конечный автомат. На множестве  $\Sigma^*$  введем следующее отношение: для  $w, v \in \Sigma^*$  пусть  $w \sim_M v$  тогда и только тогда, когда  $p_M(s,w) = p_M(s,v)$ . Это отношение эквивалентности: действительно, оно рефлексивно, симметрично и транзитивно. Для слова  $w \in \Sigma^*$  класс эквивалентности  $[w]_{\sim_M}$  будет состоять из всех таких слов алфавита  $\Sigma$ , которые приводят автомат M из начального состояния в то же самое состояние, что и слово w. Соответствующее фактор-множество  $\Sigma^*/_{\sim_M}$  конечно: элементов в нем не больше, чем состояний у автомата M. Можно установить взаимно-однозначное соответствие между множеством  $\Sigma^*/_{\sim_M}$  и множеством состояний автомата M, которые достижимы из начального состояния при помощи хотя бы одного слова.

Введем еще одно бинарное отношение. Пусть  $L\subseteq \Sigma^*$  — произвольный язык алфавита  $\Sigma$ . На множестве  $\Sigma^*$  введем отношение  $\sim_L$  следующим образом:  $\sim_L = \{\langle w,v \rangle :$  для любого  $u \in \Sigma^*$   $wu \in L$  тогда и только тогда, когда  $vu \in L\}$ . Это опять отношение эквивалентности<sup>8</sup>. Число классов эквивалентности этого отношения (то есть количество элементов фак-

 $<sup>^{8}</sup>$ Тому, кто в этом сомневается, предлагается проверить свойства рефлексивности, транзитивности и симметричности самостоятельно.

тор-множества  $^{\Sigma^*}/_{\sim_L}$ ) обозначим через o(L). Для произвольного языка L o(L) может быть равно либо положительному натуральному числу, либо бесконечности.

**Лемма 1** Пусть L — произвольный язык алфавита  $\Sigma$ . Тогда верно следующее:

- 1. если  $l\in {\Sigma^*/_{\sim_L}}$  класс эквивалентности, то либо  $l\subseteq L$ , либо  $l\cap L=\varnothing;$
- 2. если w,v,u произвольные слова алфавита  $\Sigma$  и  $w \sim_L v$ , то  $wu \sim_L vu$ .

Доказательство. Пусть  $l \in {}^{\Sigma^*}/_{\sim_L}$ . Если для l первый пункт леммы неверен, то существуют  $w, v \in l$ , такие что  $w \in L$  и  $v \not\in L$ . Но тогда  $we \in L$ ,  $ve \not\in L$  и  $w \not\sim_L v$ . Получаем противоречие с тем, что w и v лежат в одном классе эквивалентности.

Докажем второй пункт. Пусть  $w \sim_L v$ . Требуется доказать, что для произвольного слова  $x \in \Sigma^*$   $(wu)x \in L \leftrightarrow (vu)x \in L$ . Это очевидно, так как (wu)x = w(ux) и (vu)x = v(ux).  $\square$ 

Лемма 1 показывает, что если l — класс эквивалентности отношения  $\sim_L$ , то при фиксированном u для всех слов  $w \in l$  слова wu будут лежать в одном и том же классе эквивалентности. Обозначим этот класс через lu. Введенная операция обладает следующим свойством ассоциативности: для  $w, v \in \Sigma^*$  (lw)v = l(wv). Действительно, рассмотрим произвольное слово  $x \in l$ . Тогда  $x(wv) \in l(wv)$ ,  $xw \in lw$  и (xw) $v \in (lw)v$ . Но x(wv) = (xw)v. Получаем, что классы эквивалентности (lw)v и l(wv) содержат общий элемент и, следовательно, равны.

**Теорема 2** (Майхила-Нероуда) Пусть L — произвольный язык некоторого конечного алфавита  $\Sigma$ . Тогда верно следующее.

- 1. Если язык L распознается некоторым детерминированным конечным автоматом, то число o(L) конечно и не превосходит числа состояний этого автомата.
- 2. Если  $o(L) < \infty$ , то существует детерминированный конечный автомат с числом состояний, равным o(L), который распознает язык L.

Доказательство. Пусть язык L распознается детерминированным конечным автоматом  $M = \langle Q, \Sigma, \delta, s, F \rangle$ . Тогда для любых двух слов  $w, v \in \Sigma^*$  из  $w \sim_M v$  следует  $w \sim_L v$ . Действительно, если  $p_M(s,w) = p_M(s,v)$ , то для любого слова  $u \in \Sigma^*$   $p_M(s,wu) = p_M(p_M(s,w),u) = p_M(p_M(s,v),u) = p_M(s,vu)$  и  $wu \in L \leftrightarrow p_M(s,wu) \in F \leftrightarrow p_M(s,vu) \in F \leftrightarrow vu \in L$ .

Из доказанного следует, что для каждого  $m \in {}^{\Sigma^*}/_{\sim_M}$  существует единственный  $l \in {}^{\Sigma^*}/_{\sim_L}$ , такой что  $m \subseteq l$ . Действительно, пусть  $m \in {}^{\Sigma^*}/_{\sim_M}$ . Существует  $w \in \Sigma^*$ , такой что  $w \in m$ . Для этого w существует единственный  $l \in {}^{\Sigma^*}/_{\sim_L}$ , такой что  $w \in l$ . Теперь если  $v \in m$ , то  $w \sim_M v$ ,  $w \sim_L v$  и  $v \in l$ .

Кроме того, если  $l \in {}^{\sum^*}/_{\sim_L}$ , то для него обязательно найдется хотя бы один  $m \in {}^{\sum^*}/_{\sim_M}$ , такой что  $m \subseteq l$  (поскольку  $l \neq \varnothing$ , то можно взять  $m = [w]_{\sim_M}$  для некоторого  $w \in l$ ). Из сказанного выше получаем, что число классов эквивалентности отношения  $\sim_L$  не больше, чем число классов эквивалентности отношения  $\sim_M$  Число же классов отношения  $\sim_M$  конечно; как мы уже отмечали, оно не превосходит числа состояний автомата M.

Первая часть теоремы доказана; докажем вторую. Пусть  $Q = {\sum^*}/_{\sim_L}$ . Через s обозначим элемент Q, равный  $[e]_{\sim_L}$ , через F — подмножество Q, равное  $\{q \in Q : q \subseteq L\}$ . Для  $q \in Q$  и  $a \in \Sigma$   $\delta(q,a)$  положим равным  $qa^{10}$ . Ясно, что  $M = \langle Q, \Sigma, \delta, s, F \rangle$  — детерминированный конечный автомат. Покажем, что M распознает язык L.

Для этого требуется доказать, что для любого  $w \in \Sigma^*$   $p_M(s,w) = sw$ . Доказательство проведем индукцией по длине w. Для w = e это так, поскольку  $se = s = p_M(s,e)$ . Пусть w = va для  $a \in \Sigma$  и для слова v это равенство выполнено. Тогда  $p_M(s,w) = \delta(p_M(s,v),a) = (sv)a = s(va) = sw$ .

Теперь мы можем доказать, что для произвольного  $w \in \Sigma^*$  автомат M распознает w тогда и только тогда, когда  $w \in L$ . Действительно,  $w \in L \leftrightarrow ew \in L \leftrightarrow sw \subseteq L$ , поскольку  $ew \in sw$  и, по лемме 1, класс эквивалентности является подмножеством L тогда и только тогда, когда в нем есть хотя бы одно слово, принадлежащее L. Вместе с тем  $sw \subseteq L$ 

 $<sup>^{9}</sup>$ Пусть в клетках сидят зайцы. Если один и тот же заяц не может одновременно сидеть в двух разных клетках и нет пустых клеток, то число клеток не больше, чем число зайцев.

 $<sup>^{10}</sup>$ Здесь мы рассматриваем a как однобуквенное слово, то есть как элемент  $\Sigma^*$ .

$$L \leftrightarrow sw \in F$$
, a  $sw = p_M(s, w)$ .  $\square$ 

Теорема 2 имеет множество полезных следствий. В качестве одного из них укажем алгоритм построения по данному детерминированному конечному автомату эквивалентного ему (то есть распознающего тот же самый язык) детерминированного конечного автомата с минимально возможным числом состояний.

Пусть детерминированный конечный автомат  $M = \langle Q, \Sigma, \delta, s, F \rangle$  распознает язык L. Чтобы построить автомат для языка L с минимально возможным числом состояний, нужно в качестве состояний этого автомата взять классы эквивалентности отношения  $\sim_L^{11}$  Эти классы эквивалентности можно задавать по разному. Можно, например, задать класс эквивалентности  $l \in {}^{\Sigma^*}/_{\sim_L}$  в виде  $l = [w]_{\sim_L}$ , приведя слово w как один из элементов класса l в качестве обозначения для этого класса, но на практике это обычно бывает не очень удобно. К счастью, есть другой способ.

При доказательстве теоремы 2 мы установили, что каждый класс эквивалентности отношения  $\sim_L$  является объединением некоторого множества классов эквивалентности отношения  $\sim_M$ . Классы же эквивалентности отношения  $\sim_M$  находятся во взаимно однозначном соответствии с некоторым подмножеством множества состояний автомата M, а именно с множеством таких состояний, в которые автомат может перейти из начального состояния под действием хоть какого-нибудь слова. Обычно при взгляде на диаграмму автомата сразу становится ясно, какие состояния достижимы из начального, а какие нет. Таким образом, первым нашим действием при построении автомата с минимальным числом состояний будет выделение тех состояний, которые достижимы из начального.

После того, как мы выделили из множества состояний автомата M множество состояний Q', достижимых из начального, мы знаем все классы эквивалентности отношения  $\sim_M$ . Каждому состоянию  $q \in Q'$  соответствует класс эквивалентности, состоящий из всех таких слов w, что  $p_M(s,w)=q$ . Остается сгруппировать классы эквивалентности отношения  $\sim_M$  согласно их включению в классы эквивалентности отношения  $\sim_L$ . А именно, мы должны разбить множество Q' на непустые непересекающиеся подмножества так, чтобы два состояния из Q' лежали в од-

 $<sup>^{11}</sup>$ Либо какое-нибудь другое конечное множество мощности o(L), элементы которого находятся во взаимно-однозначном соответствии с элементами множества  $^{\sum^*}/_{\sim_L}.$  Что мы, собственно, и сделаем.

ном элементе разбиения тогда и только тогда, когда соответствующие им классы эквивалентности отношения  $\sim_M$  являлись подмножествами одного и того же класса эквивалентности отношения  $\sim_L$ . Другими словами, мы должны найти отношение эквивалентности  $\sim$  на Q', для которого  $q \sim q'$  тогда и только тогда, когда соответствующие состояниям q и q' классы эквивалентности отношения  $\sim_M$  являются подмножествами одного и того же класса эквивалентности отношения  $\sim_L$ .

Заметим, что  $q \sim q'$  тогда и только тогда, когда  $(\forall u \in \Sigma^*)(p_M(q,u) \in F \leftrightarrow p_M(q',u) \in F)$ . Действительно, пусть m и m' — классы эквивалентности отношения  $\sim_M$ , соответствующие состояниям q и q' и  $v \in m$ ,  $v' \in m'$  — представители этих классов. Тогда  $q = p_M(s,v)$ ,  $q' = p_M(s,v')$  и  $q \sim q'$  равносильно  $v \sim_L v'$ . Далее,  $v \sim_L v'$  тогда и только тогда, когда  $(\forall u \in \Sigma^*)(vu \in L \leftrightarrow v'u \in L)$ , а это, в свою очередь, равносильно  $(\forall u \in \Sigma^*)(p_M(s,vu) \in F \leftrightarrow p_M(s,v'u) \in F)$ . Теперь  $p_M(s,vu) = p_M(p_M(s,v),u) = p_M(q,u)$  и  $p_M(s,v'u) = p_M(p_M(s,v'),u) = p_M(q',u)$ , из чего следует справедливость нашего замечания.

Введем последовательность  $\sim_0, \sim_1, \ldots$  отношений эквивалентности на Q' следующим образом: для  $n \in \mathbb{N}$  полагаем  $\sim_n = \{\langle q,q' \rangle :$  для любого  $u \in \Sigma^*$ , такого что  $|u| \leqslant n$ , справедливо  $p_M(q,u) \in F \leftrightarrow p_M(q',u) \in F\}$ . Тогда  $^{12} \sim_0 \supseteq \sim_1 \supseteq \ldots$  и  $\sim = \bigcap_{n \in \mathbb{N}} \sim_n$ . Поскольку множество  $Q' \times Q'$  конечно, каждая из введенных выше эквивалентностей является конечным множеством и для всех достаточно больших n мы имеем  $\sim_n = \sim_{n+1}$  и  $\sim = \sim_n$ . Для введенной последовательности справедливо реккурентное соотношение:  $\sim_{n+1} = \sim_n \cap \{\langle q,q' \rangle : (\forall a \in \Sigma)(\delta(q,a) \sim_n \delta(q',a))\}$ . Действительно,  $q \sim_{n+1} q'$  тогда и только тогда, когда, во-первых, для всех слов u длины  $\leqslant n$  имеет место  $p_M(q,u) \in F \leftrightarrow p_M(q',u) \in F$ , а, во-вторых, для всех слов u, таких что  $1 \leqslant |u| \leqslant n+1$ , справедливо  $p_M(q,u) \in F \leftrightarrow p_M(q',u) \in F$ . Первое равносильно тому, что  $q \sim_n q'$ , а второе выполняется тогда и только тогда, когда для всех  $a \in \Sigma \delta(q,a) \sim_n \delta(q',a)$ , так как каждое слово длины  $\geqslant 1$  можно представить в виде au, где  $a \in \Sigma$  и  $|u| \leqslant n$ .

Одна из важных особенностей введенного нами реккурентного соотношения состоит в том, что  $\sim_{n+1}$  не зависит явно от n, а зависит только от  $\sim_n$ . Значит, если  $\sim_n=\sim_m$ , то  $\sim_{n+1}=\sim_{m+1}$ . В частности,  $\sim_n=\sim_{n+1}$ 

 $<sup>^{12}</sup>$ Введенные эквивалентности являются, прежде всего, подмножествами декартова квадрата множества Q'. Значит, мы можем рассматривать их как множества, то есть говорить о пересечении семейства эквивалентностей и о том, что одна эквивалентность является подмножеством другой.

влечет  $\sim_{n+1}=\sim_{n+2}$  и в цепочке  $\sim_0\supseteq\sim_1\supseteq\ldots$  сначала идут только строгие включения, а потом, начиная с некоторого момента, одни равенства. Значит, мы можем, стартуя с отношения  $\sim_0$  и пользуясь доказанным выше реккурентным соотношением, последовательно вычислять  $\sim_0, \sim_1, \ldots$ , пока не обнаружим, что для некоторого  $n\in\mathbb{N}$  отношения  $\sim_n$  и  $\sim_{n+1}$  совпадают. Когда такое n будет найдено, мы можем прекратить процесс порождения последовательности и сказать, что  $\sim=\sim_n$ .

Теперь, зная отношение  $\sim$ , мы имеем взаимно однозначное соответствие между фактор-множествами  $\Sigma^*/_{\sim_L}$  и  $Q'/_{\sim}$  и можем, используя построенный в теореме 2 автомат с множеством состояний  $\Sigma^*/_{\sim_L}$ , построить эквивалентный ему автомат для распознавания языка L с множеством состояний  $Q'/_{\sim}$ . Для  $l \in \Sigma^*/_{\sim_L}$  соответствующий этому l элемент фактор-множества  $Q'/_{\sim}$  (обозначим его через f(l)) равен множеству  $\{p_M(s,w): w \in l\}$ . Легко понять, что  $f([e]_{\sim_L}) = [s]_{\sim}, l \subseteq L \leftrightarrow f(l) \subseteq F$  и для произвольного  $a \in \Sigma$  l' = la тогда и только тогда, когда хотя бы для одного  $Q'/_{\sim}$  (обозначим втомат, множество состояний которого равно  $Q'/_{\sim}$ , начальным состоянием является  $Q'/_{\sim}$ , множество конечных состояний равно  $Q'/_{\sim}$  и символу  $Q'/_{\sim}$  я функция перехода сопоставляет элементу  $Q'/_{\sim}$  и символу  $Q'/_{\sim}$  я функция перехода сопоставляет элементу  $Q'/_{\sim}$  и символу  $Q'/_{\sim}$  по этот автомат будет распознавать язык  $Q'/_{\sim}$  для некоторого  $Q'/_{\sim}$  по этот автомат будет распознавать язык  $Q'/_{\sim}$  по состояний.

Описание алгоритма построения автомата с минимальным числом состояний закончено. Поясним сказанное на примере. Пусть диаграмма автомата  $M = \langle Q, \Sigma, \delta, s, F \rangle$  алфавита  $\Sigma = \{a, b\}$  изображена на рисунке 2.

Ясно, что состояния  $q_3$  и  $q_7$  не достижимы из начального состояния  $q_0$ . Все остальные достижимы. Множество Q' будет равно  $\{q_0, q_1, q_2, q_4, q_5, q_6\}$ . Мы должны вычислить, чему равно отношение эквивалентности  $\sim$  на Q', для чего мы должны найти n, такое что  $\sim_0\supset\ldots\supset\sim_n=\sim_{n+1}$ .

Каждую из эквивалентностей  $\sim_i$ ,  $i\leqslant n$ , мы будем задавать через перечисление элементов ее фактор-множества (то есть классов эквивалентности). Для  $q,q'\in Q'$  имеем  $q\sim_0 q'$  тогда и только тогда, когда для любого слова w длины 0 справедливо  $p_M(q,w)\in F\leftrightarrow p_M(q',w)\in F$ . Так как имеется только одно слово длины 0, равное e, то  $q\sim_0 q'$  равносильно

 $<sup>^{13}</sup>$ А, значит, и для всех остальных  $q \in f(l)$ . Это следует из того, что  $q \sim q'$  влечет  $\delta(q,a) \sim \delta(q',a)$  для любого  $a \in \Sigma$ .

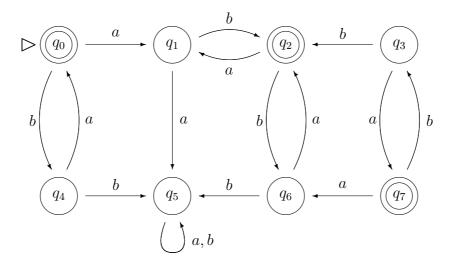


Рис. 2:

 $q = p_M(q,e) \in F \leftrightarrow F \ni p_M(q',e) = q'$  и отношение  $\sim_0$  имеет 2 класса эквивалентности, равные  $\{q_0,q_2\}$  и  $\{q_1,q_4,q_5,q_6\}^{14}$ .

Далее, для  $q, q' \in Q'$   $q \sim_1 q'$  тогда и только тогда, когда  $q \sim_0 q'$  и  $(\forall a \in \Sigma)(\delta(q,a) \sim_0 \delta(q',a))$ . В связи с первым условием имеет смысл проверять, что  $q \sim_1 q'$ , только если  $q \sim_0 q'$ . Имеем  $\delta(q_0,a) = q_1 \sim_0 q_1 = \delta(q_2,a)$  и  $\delta(q_0,b) = q_4 \sim_0 q_6 = \delta(q_2,b)$ , так что  $q_0 \sim_1 q_2$ . Для состояния  $q_5$  имеем  $\delta(q_5,a) = q_5 \not\sim_0 q_0 = \delta(q_4,a)$ ,  $\delta(q_5,a) = q_5 \not\sim_0 q_2 = \delta(q_6,a)$  и  $\delta(q_5,b) = q_5 \not\sim_0 q_2 = \delta(q_1,b)$  и, значит,  $q_5 \not\sim_1 q_4, q_1, q_6$ . Для состояния  $q_1 \delta(q_1,b) = q_2 \not\sim_0 q_5 = \delta(q_4,b) = \delta(q_6,b)$  и  $q_1 \not\sim_1 q_4, q_6$ . Наконец,  $\delta(q_4,a) = q_0 \sim_0 q_2 = \delta(q_6,a)$ ,  $\delta(q_4,b) = q_5 \sim_0 q_5 = \delta(q_6,b)$  и  $q_4 \sim_1 q_6$ . Таким образом, отношение  $\sim_1$  имеет 4 класса эквивалентности:  $\{q_0,q_2\}$ ,  $\{q_1\}$ ,  $\{q_4,q_6\}$  и  $\{q_5\}$ .

Для вычисления  $\sim_2$  надо проверить всего 2 пары состояний. Для  $q_0,q_2$  имеем  $\delta(q_0,a)=q_1\sim_1 q_1=\delta(q_2,a)$  и  $\delta(q_0,b)=q_4\sim_1 q_6=\delta(q_2,b)$ , из чего следует  $q_0\sim_2 q_2$ . Для  $q_4,q_6$   $\delta(q_4,a)=q_0\sim_1 q_2=\delta(q_6,a)$ ,  $\delta(q_4,b)=q_5\sim_1 q_5=\delta(q_6,b)$  и  $q_4\sim_2 q_6$ . Значит, эквивалентности  $\sim_2$  и  $\sim_1$  совпадают друг

 $<sup>^{14}</sup>$ Какой бы автомат мы ни минимизировали, фактор-множество  $Q'\!/_{\!\sim_0}$  всегда равно  $\{\{q\in Q':q\in F\},\{q\in Q':q\not\in F\}\}\setminus\{\varnothing\}.$ 

с другом и с эквивалентностью  $\sim$ .

Состояниями автомата  $M' = \langle Q'', \Sigma, \delta', s', F' \rangle$  (то есть элементами множества Q''), эквивалентного автомату M и содержащему минимальное число состояний, будут классы эквивалентности  $\sim$ , равные  $p_0 = \{q_0, q_2\}$ ,  $p_1 = \{q_1\}$ ,  $p_2 = \{q_4, q_6\}$  и  $p_3 = \{q_5\}$  (см. рисунок 3). Начальным состоянием M' будет  $s' = p_0$ , так как  $p_0 \ni q_0$  и  $q_0$  является начальным состоянием автомата M. Множество конечных состояний F' будет равно  $\{p_0\}$ , так как  $p_0$  является единственным элементом Q'', состоящим из конечных состояний автомата M. Осталось задать  $\delta'$ . Так как  $q_0 \in p_0$  и  $\delta(q_0, a) = q_1 \in p_1$ ,  $\delta(q_0, b) = q_4 \in p_2$ , то  $\delta'(p_0, a) = p_1$  и  $\delta'(p_0, b) = p_2$ . Аналогично  $\delta'(p_1, a) = p_3$ ,  $\delta'(p_1, b) = p_0$  ( $q_1 \in p_0$  и  $\delta(q_1, a) \in p_3$ ,  $\delta(q_1, b) \in p_0$ );  $\delta'(p_2, a) = p_0$ ,  $\delta'(p_2, b) = p_3$  ( $q_4 \in p_2$  и  $\delta(q_4, a) \in p_0$ ,  $\delta(q_4, b) \in p_3$ );  $\delta'(p_3, a) = \delta'(p_3, b) = p_3$  ( $q_5 \in p_3$  и  $\delta(q_5, a)$ ,  $\delta(q_5, b) \in p_3$ ). Построение автомата M' закончено. Упомянутое при описании алгоритма минимизации взаимнооднозначное соответствие между  $Q'/_{\sim}$  и  $\sum^*/_{\sim_L}$  сопоставляет элементу  $p_0$  класс  $[e]_{\sim_L}$ , элементу  $p_1$  класс  $[a]_{\sim_L}$ , элементу  $p_2$  класс  $[b]_{\sim_L}$  и элементу  $p_3$  класс  $[aa]_{\sim_L} = [bb]_{\sim_L}$ .

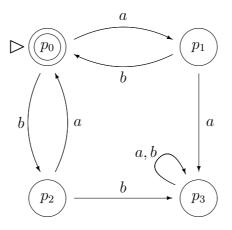


Рис. 3:

Ниже (на странице 38) мы увидим, как при помощи теоремы 2 можно доказать, что некоторые языки не распознаются никаким конечным автоматом.

Чтобы продолжить изучение детерминированных конечных автоматов, введем одно полезное обобщение этого понятия.

Определение 8 Недетерминированным конечным автоматом называется произвольная пятерка  $\langle Q, \Sigma, \Delta, s, F \rangle$ , такая что Q и  $\Sigma$  — конечные множества,  $s \in Q$ ,  $F \subseteq Q$ , а  $\Delta$  является подмножеством  $Q \times (\Sigma \cup \{e\}) \times Q$ .

Терминология, связанная с недетерминированными конечными автоматами, почти такая же, как и для детерминированных. Так,  $\Sigma$  в данном выше определении — алфавит автомата, Q — множество состояний и т. д. Отличие состоит в том, что  $\Delta$  — это теперь не функция, а *отношение переходов*. Если для  $q_1, q_2 \in Q$ ,  $a \in \Sigma$  тройка  $\langle q_1, a, q_2 \rangle$  принадлежит  $\Delta$ , то мы говорим, что автомат может перейти из состояния  $q_1$  в состояние  $q_2$  под действием символа a. Если тройка  $\langle q_1, e, q_2 \rangle$  принадлежит  $\Delta$ , то мы говорим, что автомат может совершить скачок из состояния  $q_1$  в состояния  $q_2$ .

Также как и детерминированные конечные автоматы, недетерминированные автоматы принято изображать в виде диаграмм. Аналогично случаю детерминированных автоматов, рисуются кружочки, соответствующие состояниям автомата, конечные состояния обводятся еще одним кружочком, напротив начального состояния ставится знак в виде маленького треугольника. Потом рисуются стрелки, соответствующие всем возможным переходам и скачкам. То есть, для каждой пары состояний  $q_1, q_2$  и для каждого символа  $a \in \Sigma$  если автомат может перейти из состояния  $q_1$  в состояние  $q_2$  под действием символа a, то на диаграмме рисуется стрелка из кружочка, обозначающего состояние  $q_1$  в кружочек, обозначающий состояние  $q_2$ , помеченная символом a. Если же автомат может совершить скачок из состояния  $q_1$  в состояние  $q_2$ , то из  $q_1$  в  $q_2$  рисуется стрелка, помеченная символом e.

При рассмотрении диаграмм становится очевидным, что детерминированные конечные автоматы — это, по сути, частный случай недетерминированных. На диаграмме детерминированного конечного автомата из каждого состояния выходит ровно столько стрелок, сколько символов в алфавите, по одной на каждый символ. Если же автомат недетерминированный, то из одного состояния может выходить несколько стрелок, помеченных одним и тем же символом, а может и не выходить ни одной. Кроме того, появляются стрелки, помеченные символом пустого слова. Можно считать, что недетерминированный автомат  $\langle Q, \Sigma, \Delta, s, F \rangle$  является детерминированным, если для каждой пары  $q_1 \in Q, x \in \Sigma$  существует единственное состояние  $q_2 \in Q$ , такое что  $\langle q_1, x, q_2 \rangle \in \Delta$  (в этом случае  $q_2$  равно значению функции переходов от аргументов  $q_1$  и x) и не

существует пары состояний  $\langle q_1, q_2 \rangle$ , такой что тройка  $\langle q_1, e, q_2 \rangle$  принадлежит  $\Delta$ .

Недетерминированные автоматы также, как и детерминированные, в ходе своей работы последовательно, буква за буквой, читают подаваемые им на вход слова и в зависимости от последней прочитанной буквы меняют свое состояние. Предположим, что в какой-то момент времени автомат находится в состоянии q и читает букву x. Если существуют какие-то состояния, в которые автомат может перейти из состояния q под действием буквы x, то он выбирает одно из них и переходит в него. Если таких состояний нет, то он "зависает", то есть переходит в неопределенное состояние и перестает читать какие-либо дальнейшие символы. Кроме того, он может в любой момент, не читая никакого символа, совершить скачок из одного состояния в другое, если такой скачок возможен (т. е. на диаграмме есть стрелка, помеченная символом e, которая ведет из первого состояния во второе).

Недетерминированные автоматы, аналогично детерминированным, служат для распознавания слов. Мы говорим, что автомат распознает слово, если он, стартуя из начального состояния и прочитав данное слово может, следуя описанным выше правилам, остановится в одном из конечных состояний и при этом не "зависнуть". Другими словами, можно считать, что недетерминированный конечный автомат обладает определенной "интуицией"; читая слово, он на некотором шаге в процессе чтения может встать перед выбором из нескольких альтернатив: некоторые из вариантов выбора могут привести его в одно из конечных состояний, а некоторые — нет. Автомат, как бы предвидя исход своей работы, всегда выбирает одну из тех альтернатив, которые могут привести его в одно из конечных состояний. Если же все варианты одинаково плохи, то он выбирает какой-то один из них, неважно какой.

Дадим формальное определение. Мы говорим, что недетерминированный конечный автомат  $\langle Q, \Sigma, \Delta, s, F \rangle$  может перейти из состояния  $q \in Q$  в состояние  $p \in Q$  под действием слова  $w \in \Sigma^*$ , если существует конечная последовательность  $q_0, q_1, \ldots, q_n$  элементов из Q и функция  $t: \{0, \ldots, n\} \to \mathbb{N}$ , такие что  $q_0 = q, q_n = p, t(0) = 0, t(n) = |w|$  и для каждого i < n либо t(i+1) = t(i) и  $\langle q_i, e, q_{i+1} \rangle \in \Delta$ , либо t(i+1) = t(i) + 1 и  $\langle q_i, (w)_{t(i+1)}, q_{i+1} \rangle \in \Delta$ .

Соотнося это формальное определение с данным выше неформальным, можно сказать, что  $q_0, \ldots, q_n$  — это последовательность состояний, которые автомат проходит, читая слово w, а t(i) — это количество букв

слова w, которые автомат успел прочитать к моменту достижения состояния  $q_i$ .

Множество всех состояний, в которые недетерминированный конечный автомат  $M = \langle Q, \Sigma, \Delta, s, F \rangle$  может перейти из состояния  $q \in Q$  под действием слова  $w \in \Sigma^*$ , мы обозначаем через  $P_M(q,w)$ . Мы говорим, что этот автомат распознает слово w, если множество  $P_M(s,w)$  содержит хотя бы один элемент из F, то есть если  $P_M(s,w) \cap F \neq \emptyset$ . Также, как и раньше, для недетерминированного конечного автомата M через L(M) обозначим множество всех слов, которые этот автомат распознает.

Недетерминированные конечные автоматы являются мощным и удобным инструментом в исследовании вопроса о распознаваемости языков. Пусть, например, требуется построить автомат, который распознает язык алфавита  $\Sigma = \{a, b, c\}$ , состоящий из слов, заканчивающихся на три одинаковые буквы. Построить с ходу детерминированный автомат для этого языка достаточно трудно (хотя, безусловно, возможно, как следует из доказанной ниже теоремы 3). Конструкция же недетерминированного автомата очевидна, достаточно обратиться к приведенной ниже диаграмме.

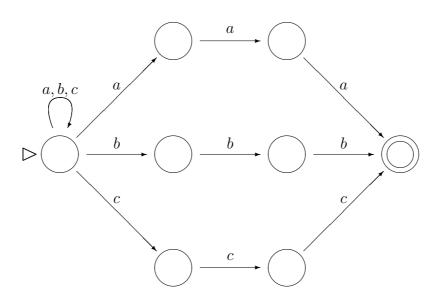


Рис. 4:

Распознавая слово, изображенный на этой диаграмме автомат оста-

ется в начальном состоянии до тех пор, пока он не начал читать третью с конца букву, после чего он вступает на один из трех путей и идет до конечного состояния. Для этого он должен "угадать", какая из прочитанных их букв является третьей с конца, но "угадывание" — это как раз то, для чего недетерминированные автоматы и предназначены. Ясно, что каждое слово, которое заканчивается на три одинаковые буквы, способно привести этот автомат в конечное состояние, после чего буквы в слове иссякнут. Также ясно, что ни одно слово, которое на три одинаковые буквы не заканчивается, либо вообще не способно привести этот автомат в конечное состояние, либо оно все же может привести его туда, но тогда слово не будет прочитано до конца и, поскольку из конечного состояния не выходит никаких стрелок, то автомат, пытаясь прочитать очередную букву слова, зависнет и все равно это слово не распознает.

Конечно, недетерминированные автоматы не могут рассматриваться как формализация понятия алгоритма. Согласно определению 1, алгоритм — это набор четких, недвусмысленных инструкций, а как раз этой четкости и недвусмысленности недетерминированным автоматам и не хватает. Они являются не более чем удобным инструментом исследования.

На первый взгляд может показаться, что языков, которые распознаются недетерминированными конечными автоматами, должно быть больше, чем языков, которые распознаются детерминированными. Однако это не так. Как показывает доказанная ниже теорема 3, недетерминированность в плане распознавания языков ничего не добавляет.

**Лемма 2** Если  $M = \langle Q, \Sigma, \Delta, s, F \rangle$  — недетерминированный конечный автомат,  $w \in \Sigma^*$  и  $a \in \Sigma$ , то справедливо равенство  $P_M(s, wa) = \bigcup_{q \in P_M(s,w)} P_M(q,a)$ .

Доказательство. Покажем, что если состояние p принадлежит левой части равенства, то оно принадлежит и правой. Пусть  $p \in P_M(s,wa)$ . Тогда существует последовательность  $q_0,\ldots,q_n$  элементов из Q и функция  $t:\{0,\ldots,n\}\to\mathbb{N}$ , такие что  $q_0=s,\ q_n=p,\ t(0)=0,\ t(n)=|wa|$  и для каждого i< n либо t(i+1)=t(i) и  $\langle q_i,e,q_{i+1}\rangle\in\Delta$ , либо t(i+1)=t(i)+1 и  $\langle q_i,(wa)_{t(i+1)},q_{i+1}\rangle\in\Delta$ . Из определения ясно, что функция t принимает на множестве  $\{0,\ldots,n\}$  все значения из множества  $\{0,\ldots,|wa|\}^{15}$ . Значит, существует k< n, такое что t(k)=|w|.

 $<sup>^{15}</sup>$ Пусть это не так. Рассмотрим минимальное  $m\leqslant |wa|$ , которое не попадает в

Пусть  $q=q_k$ . Имеем  $q\in P_M(s,w)$ , так как последовательность  $q_0,\ldots,q_k$  и функция t, рассматриваемая как функция из  $\{0,\ldots,k\}$  в  $\mathbb{N}$ , удовлетворяют всем условиям определения. Теперь рассмотрим последовательность  $q'_0,\ldots,q'_{n-k}$  и функцию  $t':\{0,\ldots,n-k\}\to\mathbb{N}$ , такие что для каждого  $i\leqslant n-k$   $q'_i=q_{k+i}$  и t'(i)=t(k+i)-|w|. Эта последовательность и эта функция дают нам, согласно определению, что  $p\in P_M(q,a)$ . Раз  $p\in P_M(q,a)$  для некоторого  $q\in P_M(s,w)$ , то, значит, p принадлежит правой части равенства.

Теперь докажем обратное включение. Если  $p \in \bigcup_{q \in P_M(s,w)} P_M(q,a)$ , то для некоторого  $q \in P_M(s,w)$  имеем  $p \in P_M(q,a)$ . Зафиксируем одно из таких q. Так как  $q \in P_M(s,w)$ , то существует последовательность состояний  $q'_0, \ldots, q'_k$  и функция  $t' : \{0, \ldots, k\} \to \mathbb{N}$ , такие что  $q'_0 = s$ ,  $q'_k = q$ , t'(0) = 0, t'(k) = |w| и для каждого i < k либо t'(i+1) = t'(i) и  $\langle q'_i, e, q'_{i+1} \rangle \in \Delta$ , либо t'(i+1) = t'(i) + 1 и  $\langle q'_i, (w)_{t'(i+1)}, q'_{i+1} \rangle \in \Delta$ . Раз  $p \in P_M(q,a)$ , то существует последовательность состояний  $q''_0, \ldots, q''_m$  и функция  $t'' : \{0, \ldots, m\} \to \mathbb{N}$ , такие что  $q''_0 = q$ ,  $q''_m = p$ , t''(0) = 0, t''(m) = 1 и для каждого i < m либо t''(i+1) = t''(i) и  $\langle q''_i, e, q''_{i+1} \rangle \in \Delta$ , либо t''(i+1) = t''(i) + 1 и  $\langle q''_i, (a)_{t''(i+1)}, q''_{i+1} \rangle \in \Delta$ . Рассмотрим последовательность  $q_0, \ldots, q_{k+m}$  и функцию  $t : \{0, \ldots, k+m\} \to \mathbb{N}$ , такие что для  $i \leqslant k$   $q_i = q'_i$  и t(i) = t'(i), а для  $k < i \leqslant k + m$   $q_i = q''_{i-k}$  и t(i) = t''(i-k) + |w|. Проверив все свойства определения, убеждаемся, что эта последовательность дает нам включение  $p \in P_M(s, wa)$ .  $\square$ 

**Следствие 1** Для недетерминированного конечного автомата  $M = \langle Q, \Sigma, \Delta, s, F \rangle$ , слов  $w, v \in \Sigma^*$  и символа  $a \in \Sigma$  если  $P_M(s, w) = P_M(s, v)$ , то  $P_M(s, wa) = P_M(s, va)$ .

Доказательство. Очевидно.  $\square$ 

**Теорема 3** Язык распознается некоторым детерминированным конечным автоматом тогда и только тогда, когда он распознается некоторым недетерминированным конечным автоматом.

область значений функции t. Поскольку t(0)=0, то m>0. Так как m — минимальное число с указанным свойством, то существует  $i\leqslant n$ , для которого t(i)=m-1. Пусть  $i_0$  — максимальное среди всех таких i. Так как  $t(n)=|wa|\geqslant m>m-1=t(i_0)$ , то  $i_0\neq n$ . Если  $t(i_0+1)=t(i_0)$ , то  $i_0$  не может быть максимальным. Если же  $t(i_0+1)=t(i_0)+1$ , то, поскольку  $t(i_0)=m-1$ ,  $t(i_0+1)=m$  и m попадает в область значений функции t. Противоречие.

Доказательство. Необходимость очевидна, поскольку, как мы уже отмечали, детерминированные конечные автоматы можно рассматривать как частный случай недетерминированных. Докажем достаточность.

Пусть  $M = \langle Q, \Sigma, \Delta, s, F \rangle$  — недетерминированный конечный автомат. Построим детерминированный конечный автомат M', такой что L(M') = L(M).

В качестве множества состояний автомата M' возьмем множество  $Q' = \{P_M(s,w) : w \in \Sigma^*\}$ . Несмотря на то, что w в этом определении пробегает бесконечное семейство  $\Sigma^*$ , множество Q' конечно, так как все его элементы являются подмножествами Q, а таких подмножеств конечное число. Начальным состоянием автомата M' будет являться элемент Q', равный  $P_M(s,e)$ , который мы обозначим через S. Множеством конечных состояний будет  $F' \subseteq Q'$ , равное  $\{P_M(s,w) : w \in \Sigma^* \text{ и } P_M(s,w) \cap F \neq \varnothing\}$ . Алфавитом, естественно, будет  $\Sigma$ . Осталось определить функцию переходов  $\delta$ .

Пусть  $P = P_M(s, w) \in Q'$  и  $a \in \Sigma$ . Полагаем  $\delta(P, a)$  равным  $P_M(s, wa)$ . Это определение корректно, так как если  $P = P_M(s, v)$  для некоторого слова v, отличного от w, то, по следствию 1,  $P_M(s, wa) = P_M(s, va)$ .

Определение автомата M' закончено. Чтобы доказать, что он обладает требуемым свойством, сначала покажем, что для любого  $w \in \Sigma^*$   $p_{M'}(S,w) = P_M(s,w)$ . Для пустого слова этот факт непосредственно следует из определений. Пусть для слова w это верно и  $a \in \Sigma$ . Тогда  $p_{M'}(S,wa) = \delta(p_{M'}(S,w),a) = \delta(P_M(s,w),a) = P_M(s,wa)$ . Первое равенство в этой цепочке следует из определения функции  $p_{M'}$ , второе — из индукционного предположения, третье — из определения функции  $\delta$ .

Осталось показать, что для любого  $w \in \Sigma^*$   $p_{M'}(S,w) \in F'$  тогда и только тогда, когда  $P_M(s,w) \cap F \neq \varnothing^{16}$ . Эта эквивалентность почти очевидна: действительно,  $p_{M'}(S,w) = P_M(s,w)$  и  $P_M(s,w)$  принадлежит F' тогда и только тогда, когда пересечение этого множества с множеством F непусто.  $\square$ 

Теперь, после доказательство теоремы 3, мы можем говорить просто об языках, распознаваемых автоматами, не уточняя, какие именно автоматы, детерминированные или недетерминированные, имеются в виду $^{17}$ .

<sup>&</sup>lt;sup>16</sup>Напомним, что по определению автомат M' распознает слово w тогда и только тогда, когда  $p_{M'}(S,w) \in F'$ . Опять же по определению автомат M распознает слово w тогда и только тогда, когда  $P_M(s,w) \cap F \neq \varnothing$ .

 $<sup>^{17}</sup>$ Доказательство теоремы 3 дает нам также алгоритм построения по недетермини-

Попытаемся дать альтернативное описание этих языков, независящее от понятия конечного автомата. Для этого нам потребуется ввести ряд операций над языками.

Языки алфавита  $\Sigma$  — это прежде всего подмножества  $\Sigma^*$ , то есть множества, и к ним применимы те же операции, что и к множествам. Так, например, пересечением языков  $L_1$  и  $L_2$  будет язык, состоящий из слов, которые входят как в  $L_1$ , так и в  $L_2$ . Объединением  $L_1$  и  $L_2$  будет язык, состоящий из слов, которые входят либо в  $L_1$ , либо в  $L_2$ , либо в оба языка сразу. Кроме того, если L — язык алфавита  $\Sigma$ , то язык  $\Sigma^* \setminus L = \{w \in \Sigma^* : w \not\in L\}$  мы называем дополнением к языку L (относительно алфавита  $\Sigma^{18}$ ) и обозначаем  $\overline{L}$ .

Кроме теоретико-множественных операций есть еще две операции, которые применимы только к языкам, а не к произвольным множествам. Это операции конкатенации и навешивания звездочки Клини.

Пусть  $L_1, L_2 \subseteq \Sigma^*$  — языки алфавита  $\Sigma$ . Конкатенацией языков  $L_1$  и  $L_2$  называется язык  $\{wv: w \in L_1, v \in L_2\}$ , то есть язык, словами которого будут конкатенации слов языка  $L_1$  со словами языка  $L_2$ .

Для определения звездочки Клини нам понадобится ввести ряд промежуточных обозначений. Пусть L — язык алфавита  $\Sigma$ . Для каждого натурального числа n определим по индукции язык  $L^n$ :

1. 
$$L^0 = \{e\};$$

2. 
$$L^{n+1} = L^n L$$
.

Пусть теперь  $L^* = \bigcup_{n \in \mathbb{N}} L^n$ . Мы говорим, что язык  $L^*$  получается из языка L навешиванием звездочки Клини. Словами языка  $L^*$  будут всевозможные конкатенации слов языка L. Отметим, что каким бы не был язык L, даже равным пустому множеству, язык  $L^*$  обязательно содержит пустое слово<sup>19</sup>.

рованному конечному автомату эквивалентного ему детерминированного. Этот алгоритм будет подробно разбираться на семинарских занятиях.

 $<sup>^{18}</sup>$ Алфавит, относительно которого рассматривается дополнение к языку, здесь важен. Действительно, один и тот же язык можно рассматривать как язык разных алфавитов: например, некоторого алфавита  $\Sigma$  и более широкого алфавита  $\Sigma' \supset \Sigma$ . Значения операции дополнения в этих случаях будут различны. Однако у нас, как правило, алфавит будет фиксирован и ясен из контекста. В связи с этим мы будем говорить просто о дополнениях, не упоминая алфавит.

 $<sup>^{19}</sup>$ Начиная разговор об языках, мы не случайно обозначили множество всех слов

Теперь определим множество регулярных языков. Для этого нам опять придется ввести ряд промежуточных подмножеств. Пусть  $\Sigma$  — некоторый фиксированный конечный алфавит. Определим для каждого натурального числа n множество языков  $\mathcal{R}_n$ :

1. 
$$\mathcal{R}_0 = \{\emptyset\} \cup \{\{a\} : a \in \Sigma\}^{20};$$

2. 
$$\mathcal{R}_{n+1} = \mathcal{R}_n \cup \{L_1 \cup L_2 : L_1, L_2 \in \mathcal{R}_n\} \cup \{L_1 L_2 : L_1, L_2 \in \mathcal{R}_n\}$$
  
  $\cup \{L^* : L \in \mathcal{R}_n\}.$ 

Пусть теперь  $\mathcal{R} = \bigcup_{n \in \mathbb{N}} \mathcal{R}_n$ . Множество  $\mathcal{R}$  называется множеством регулярных языков, а его элементы — регулярными языками.

Чуть менее формально можно определить множество регулярных языков следующим образом. Имеется набор базовых языков: это те языки, которые являются элементами  $\mathcal{R}_0$ . Так, например, если  $\Sigma = \{a_1, \ldots, a_n\}$ , то базовыми языками будут  $\emptyset$ ,  $\{a_1\}, \ldots, \{a_n\}$ . Регулярные же языки — это те, которые можно получить из базовых при помощи операций объединения, конкатенации и навешивания звездочки Клини.

Поскольку  $\mathcal{R}$  равно объединению  $\mathcal{R}_n$ -ых, то для каждого регулярного языка существует  $\mathcal{R}_n$ , которому он принадлежит. Пусть L — регулярный язык и n — наименьшее натуральное число, такое что  $L \in \mathcal{R}_n$ . Тогда язык L можно получить из базовых при помощи n упомянутых выше операций<sup>21</sup>. Пусть, например,  $\Sigma = \{a, b\}$ . Тогда элементами  $\mathcal{R}_1$  будут 3 базовых языка и языки, которые можно получить при помощи одной операции из

алфавита  $\Sigma$  как  $\Sigma^*$ . Действительно, если отождествлять символы алфавита с соответствующими однобуквенными словами, то можно воспринимать сам алфавит  $\Sigma$  как язык, состоящий из всех однобуквенных слов. Тогда навешивание звездочки Клини на этой язык дает нам множество всех слов алфавита  $\Sigma$ .

 $^{20}$ Не следует путать  $\varnothing$  и  $\{\varnothing\}$ . Первое — это пустое множество, у него вообще нет элементов. Второе — это множество, состоящее из одного элемента; этим элементом является пустое множество. Аналогично, a и  $\{a\}$  — не одно и то же: a — это символ или однобуквенное слово, а  $\{a\}$  — это множество слов, то есть язык, состоящий из одного однобуквенного слова.  $\mathcal{R}_0$  — это не множество слов, а множество языков; его элементами являются языки. В частности, элементами  $\mathcal{R}_0$  являются: язык, не содержащий ни одного слова (то есть  $\varnothing$ ) и языки, состоящие из одного однобуквенного слова.

 $^{21}$ На самом деле, это не совсем верно. Для определения языка из  $\mathcal{R}_n$  может потребоваться больше чем n операций. Здесь правильно говорить не об n операциях, а об n вложенных уровнях операций. Тем, кого заинтересовал этот вопрос, предлагается самому разобраться в нем и ввести соответствующие понятия.

базовых, а именно:  $\varnothing^* = \{e\}, \{a\}^* = \{a^i : i \in \mathbb{N}\}$  (здесь  $a^i$  обозначает слово, которое состоит из i идущих подряд символов a),  $\{b\}^* = \{b^i : i \in \mathbb{N}\}$ ,  $\varnothing\{a\} = \{a\}\varnothing = \varnothing$ ,  $\varnothing\{b\} = \{b\}\varnothing = \varnothing$ ,  $\{a\}\{b\} = \{ab\}, \{b\}\{a\} = \{ba\}, \varnothing = \{a\}\{a\} = \{aa\}, \{b\}\{b\} = \{bb\}, \varnothing \cup \{a\} = \{a\} \cup \varnothing = \{a\}, \varnothing \cup \{b\} = \{b\} \cup \varnothing = \{b\}, \{a\} \cup \{b\} = \{b\} \cup \{a\} = \{a,b\}, \{a\} \cup \{a\} = \{a\}, \{b\} \cup \{b\} = \{b\}$  и  $\varnothing \cup \varnothing = \varnothing$ . Некоторые из языков, получающихся применением одной операции к базовым языкам сами являются базовыми, но появляются и новые. Далее, чтобы образовать  $\mathcal{R}_2$ , мы должны применять операции к языкам из  $\mathcal{R}_1$  и т. д.

Для обозначения регулярных языков применяют регулярные выражения. Регулярное выражение — это то, что остается, если записать в строчку в виде выражения, как язык получается из базовых при помощи операций, а затем убрать все фигурные скобки<sup>22</sup>. Например, рассмотрим регулярный язык  $(\{a\} \cup \{b\})^{**} \cup (\{a\}\{b\})$  из  $\mathcal{R}_4$ . Регулярным выражением для него будет  $(a \cup b)^{**} \cup (ab)$ . Для каждого регулярного языка можно придумать много регулярных выражений, которые его обозначают: например, упомянутый выше язык, который задается выражением  $(a \cup b)^{**} \cup (ab)$ , можно задать и как  $\varnothing \cup (ab) \cup (a \cup b)^{**}$ , и даже как  $(a \cup b)^{*}$  (на самом деле это язык из  $\mathcal{R}_2$  и он состоит из множества всех слов алфавита  $\Sigma$ )<sup>23</sup>.

При записи регулярных выражений обычно стараются не писать лишних скобок (хотя, в принципе, они никогда не помещают). Так, например, операция конкатенации языков ассоциативна (то есть для любых трех языков  $L_1, L_2, L_3$  справедливо равенство  $L_1(L_2L_3) = (L_1L_2)L_3$ ), поэтому вместо выражений a(bc) и (ab)c обычно пишут просто abc. Также чтобы не писать лишних скобок, операциям назначают приоритеты: самый высокий приоритет у звездочки Клини, более низкий у конкатенации и самый низкий у объединения. Имея в виду эти приоритеты, мы, к примеру, пишем  $ab^* \cup a^*b$  вместо  $(a(b^*)) \cup ((a^*)b)$ , подразумевая, что мы в любой момент можем расставить недостающие скобки согласно приоритетам операций<sup>24</sup>.

Теперь мы готовы доказать теорему, связывающую введенное только что понятие регулярного языка с введенным ранее понятием конечного

<sup>&</sup>lt;sup>22</sup>Можно дать и строгое формальное определение того, что такое регулярное выражение, но нам это определение не нужно.

 $<sup>^{23}</sup>$ То же самое можно наблюдать в арифметике. Например число 4 можно задать и как 2+2, и как  $2\cdot 2$ , и как  $\sqrt{16}$ .

 $<sup>^{24}</sup>$  То же самое в алгебре. Мы обычно пишем  $xy^n + x^ny$  вместо  $(x(y^n)) + ((x^n)y)$ .

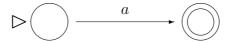
автомата.

**Теорема 4** Пусть  $L \subseteq \Sigma^*$  — произвольный язык конечного алфавита  $\Sigma$ . Тогда следующие условия эквивалентны.

- 1. Язык L является регулярным.
- 2. Язык L распознается некоторым конечным автоматом.

Доказательство. (1  $\Rightarrow$  2) Так как множество регулярных языков равно объединению  $\mathcal{R}_n$ -ых, то достаточно доказать, что для каждого  $n \in N$  все языки из  $\mathcal{R}_n$  распознаются конечными автоматами. Докажем это утверждение индукцией по n.

Для языков из  $\mathcal{R}_0$  это верно. Действительно, пустой язык распознается любым автоматом с пустым множеством конечных состояний, а язык  $\{a\}$  для  $a \in \Sigma$  распознается недетерминированным конечным автоматом, изображенным на следующей диаграмме:



Пусть все языки из  $\mathcal{R}_n$  распознаются конечными автоматами. Покажем, что то же самое верно для всех языков из  $\mathcal{R}_{n+1}$ .

Пусть  $L=L_1\cup L_2$  для  $L_1,L_2\in\mathcal{R}_n$ . Пусть  $M_1=\langle Q_1,\Sigma,\Delta_1,s_1,F_1\rangle$  и  $M_2=\langle Q_2,\Sigma,\Delta_2,s_2,F_2\rangle$  — недетерминированные конечные автоматы, распознающие языки  $L_1$  и  $L_2$  соответственно. Покажем, как построить недетерминированный конечный автомат для распознавания L.

Без ограничения общности можно считать, что  $Q_1 \cap Q_2 = \emptyset$ , то есть что автоматы  $M_1$  и  $M_2$  не имеют общих состояний  $C^{25}$ . Возьмем произвольное s, не принадлежащее  $Q_1 \cup Q_2$ . Пусть  $Q = Q_1 \cup Q_2 \cup \{s\}$ ,  $F = F_1 \cup F_2$  и  $\Delta = \Delta_1 \cup \Delta_2 \cup \{\langle s, e, s_1 \rangle, \langle s, e, s_2 \rangle\}$ . Тогда автомат  $\langle Q, \Sigma, \Delta, s, F \rangle$  распознает язык L

 $<sup>^{25}</sup>$ Если это не так, то пусть  $Q_2'$  — произвольное множество, содержащее столько же элементов, сколько  $Q_2$  и не имеющее общих элементов с  $Q_1$ . Пусть  $h:Q_2\to Q_2'$  — взаимно однозначное отображение из  $Q_2$  на  $Q_2'$ . Пусть  $s_2'=h(s_2),\,F_2'=\{h(q):q\in F_2\}$  и  $\Delta_2'=\{\langle h(q_1),x,h(q_2)\rangle:q_1,q_2\in Q_2,x\in\Sigma\cup\{e\},\langle q_1,x,q_2\rangle\in\Delta_2\}.$  Тогда автомат  $M_2'=\langle Q_2',\Sigma,\Delta_2',s_2',F_2'\rangle$  распознает тот же самый язык, что и автомат  $M_2$ .

В общих чертах мы делаем следующее. Мы "ставим рядом" автоматы  $M_1$  и  $M_2$ , вводим новое начальное состояние s и добавляем e-переходы из s в  $s_1$  и  $s_2$ . Ясно, что получившийся автомат должен распознавать объединение языков, распознаваемых автоматами  $M_1$  и  $M_2$ . Действительно, чтобы распознать слово, этот автомат должен сначала совершить скачок из s в  $s_1$  или в  $s_2$ , а затем распознать его как слово из  $L_1$  или  $L_2$  в зависимости от того, в какое из состояний он перешел.

Теперь пусть  $L=L_1L_2$  для  $L_1,L_2\in\mathcal{R}_n$ . Пусть  $M_1=\langle Q_1,\Sigma,\Delta_1,s_1,F_1\rangle$  и  $M_2=\langle Q_2,\Sigma,\Delta_2,s_2,F_2\rangle$  — недетерминированные конечные автоматы, распознающие языки  $L_1$  и  $L_2$  соответственно. Покажем, как построить недетерминированный конечный автомат для распознавания L.

Опять можно считать, что  $Q_1\cap Q_2=\varnothing$ . Пусть  $Q=Q_1\cup Q_2$  и  $\Delta=\Delta_1\cup \Delta_2\cup \{\langle q,e,s_2\rangle: q\in F_1\}$ . Тогда автомат  $\langle Q,\Sigma,\Delta,s_1,F_2\rangle$  будет распознавать язык L.

На неформальном уровне эта конструкция выглядит следующим образом. Мы "ставим друг за другом" автоматы  $M_1$  и  $M_2$ , конечные состояния первого автомата перестаем считать таковыми, добавляем e-переходы из бывших конечных состояний автомата  $M_1$  в начальное состояние автомата  $M_2$  и начальным состоянием того, что получилось, объявляем  $s_1$ . Эта конструкция дает требуемый результат: распознавая слово, автомат должен сначала дойти до одного из конечных состояний автомата  $M_1$ , затем совершить скачок в  $s_2$  и затем прийти в конечное состояние автомата  $M_2$ . Ясно, что каждое слово, распознаваемое таким образом, должно являться конкатенацией слов из языков  $L_1$  и  $L_2$ .

Пусть, наконец,  $L \in \mathcal{R}_n$  и недетерминированный конечный автомат  $M' = \langle Q', \Sigma, \Delta', s', F' \rangle$  распознает язык L. Покажем, как построить автомат для распознавания языка  $L^*$ .

Возьмем произвольное  $s \notin Q'$ . Пусть  $Q = Q' \cup \{s\}$  и  $\Delta = \Delta' \cup \{\langle q, e, s \rangle : q \in F'\} \cup \{\langle s, e, s' \rangle\}$ . Тогда автомат  $\langle Q, \Sigma, \Delta, s, \{s\} \rangle$  будет распознавать язык  $L^*$ . На неформальном уровне мы сначала добавляем к автомату новое состояние, которое объявляем начальным и конечным, все старые конечные состояния перестаем считать таковыми и добавляем из них епереход в новое начальное состояние и, наконец, добавляем e-переход из нового начального состояния в старое.

Чтобы распознать слово, получившийся автомат должен прийти под действием этого слово в состояние s. Для этого он должен либо вообще не уходить из s, тогда слово будет пустым, либо, выйдя из него, пройти несколько циклов и вернуться назад в s. Но тогда каждый цикл будет

соответствовать некоторому слову языка L, а все исходное слово будет конкатенацией нескольких слов языка L, то есть словом языка  $L^*$ .

Поскольку мы исчерпали все возможности для образования языков, принадлежащих  $\mathcal{R}_{n+1}$ , то мы тем самым доказали то, что нам требуется. Конечно, наше доказательство нельзя считать завершенным. Мы описали правила построения соответствующих автоматов и сделали несколько замечаний по поводу того, что эти автоматы будут распознавать нужные нам языки, однако эти замечания ни в коем случае нельзя рассматривать в качестве доказательств. Строгие доказательства в данном случае подразумевают работу с определениями, то есть анализ различных последовательностей вида  $q_0, \ldots, q_n$  и функций  $t: \{0, \ldots, n\} \to \mathbb{N}$ . Однако, поскольку идея этих доказательств достаточно ясна, а сами они являются слишком громоздкими, автор курса предпочел не загромождать изложения обилием технических подробностей, а ограничиться рядом неформальных замечаний, оставляя формальные доказательстве наиболее дотошным читателям в качестве упражнения.

 $(2 \Rightarrow 1)$  Пусть  $M = \langle Q, \Sigma, \delta, s, F \rangle$  — детерминированный конечный автомат, распознающий язык L. Покажем, что этот язык регулярен. Для этого введем несколько промежуточных обозначений.

Пусть  $w \in \Sigma^*$  — произвольное слово. Через  $\pi(w)$  обозначим множество слов  $\{v \in \Sigma^* : v \leq w, v \neq e, v \neq w\}$ . Другими словами,  $\pi(w)$  — это множество всех непустых слов, с которых начинается слово w и которые не равны самому w. Если  $|w| \leq 1$ , то  $\pi(w) = \emptyset$ , если же |w| > 1, то  $\pi(w) \neq \emptyset$ .

Пусть  $Q=\{q_1,\ldots,q_n\}$ . Для  $0\leqslant k\leqslant n$  через  $Q_k$  обозначим множество  $\{q_1,\ldots,q_k\}$   $(Q_0=\varnothing)$ . Для  $1\leqslant i,j\leqslant n$  пусть  $R_{i,j}=\{w\in \Sigma^*:p_M(q_i,w)=q_j\}$ , то есть множеству таких слов, которые переводят автомат M из состояния  $q_i$  в состояние  $q_j$ . Для  $1\leqslant i,j\leqslant n$  и  $0\leqslant k\leqslant n$  пусть  $R_{i,j}^k=\{w\in R_{i,j}:$  для любого  $v\in\pi(w)$   $p_M(q_i,v)\in Q_k\}$ . Словами это можно выразить так:  $R_{i,j}^k=$  это множество таких слов, под действием которых автомат переходит из состояния  $q_i$  в состояние  $q_j$ , не заходя в промежутке в состояния с номерами, большими чем k. Ясно, что  $R_{i,j}^n=R_{i,j}$ .

Мы утверждаем, что для каждого  $0 \le k \le n$  языки  $R_{i,j}^k$  регулярны для всех i и j. Докажем это утверждение индукцией по k.

Для всех i, j язык  $R^0_{i,j}$  регулярен, так как он может содержать только однобуквенные слова или пустое слово (то есть только такие слова w, для которых  $\pi(w) = \varnothing$ ). Пусть k < n и утверждение верно для k; докажем его для k+1.

Достаточно доказать равенство  $R_{i,j}^{k+1}=R_{i,j}^k\cup R_{i,k+1}^k(R_{k+1,k+1}^k)^*R_{k+1,j}^k,$  поскольку в этом случае для всех i,j язык  $R_{i,j}^{k+1}$  получается при помощи объединения, двух конкатенаций и звездочки Клини из языков, которые по индукционному предположению являются регулярными. Равенство же является достаточно очевидным. Действительно, пусть  $w \in R_{i,j}^{k+1}$ . Автомат M, стартуя из состояния  $q_i$  и читая слово w, проходит последовательность каких-то состояний и, прочитав все слово, останавливается в  $q_i$ . Все промежуточные состояния в этой последовательности имеют номера  $\leq k+1$ . При этом автомат может вообще не заходить в состояние  $q_{k+1}$ , тогда  $w \in R_{i,j}^k$ . Если же автомат заходит в состояние  $q_{k+1}$ , то это происходит так: сначала он приходит из  $q_i$  в  $q_{k+1}$ , потом может сделать ноль или более циклов, выходя на время из состояния  $q_{k+1}$  и снова в него возвращаясь и, наконец, окончательно покинуть состояние  $q_{k+1}$ , чтобы перейти в состояние  $q_i$  и там остановиться. В этом случае w можно представить в виде конкатенации некоторой последовательности слов, первое слово в которой принадлежит  $R_{i,k+1}^k$ , последнее —  $R_{k+1,j}^k$ , а все промежуточные слова, если они есть —  $R_{k+1,k+1}^k$ . Язык  $R_{i,k+1}^k(R_{k+1,k+1}^k)^*R_{k+1,j}^k$  как раз и состоит из всех таких конкатенаций. Что касается обратного включения  $R_{i,j}^k \cup R_{i,k+1}^k(R_{k+1,k+1}^k)^*R_{k+1,j}^k \subseteq R_{i,j}^{k+1}$ , то его обосновать еще проще $^{26}$ .

Таким образом, для всех i,j,k язык  $R^k_{i,j}$  регулярен. В частности, регулярными будут все языки  $R_{i,j}$  для  $1 \leqslant i,j \leqslant n$ . Пусть  $i_0$  — номер начального состояния. Для завершения доказательства нам осталось заметить, что  $L = \bigcup_{q_j \in F} R_{i_0,j}$ . Само равенство очевидно, а язык L будет регулярным, так как является объединением регулярных языков.  $\square$ 

После доказательсва теоремы 4 мы можем утверждать, что следующие понятия являются эквивалентными: язык, распознаваемый конечным автоматом; регулярный язык; язык, задаваемый регулярным выражением. Мы будем пользоваться термином "регулярный язык" как наиболее коротким.

Если дан регулярный язык L, то его можно задать двумя способами. Первый способ — это диаграмма конечного автомата, который распознает этот язык. Второй — это регулярное выражения. Первый способ можно рассматривать как способ задания языка через алгоритм, отвечающий на вопрос о принадлежности произвольного данного слова языку L. Второй способ можно интерпретировать как алгоритм порождения языка L, то есть такой алгоритм, который на входе не получает никаких данных, а на выходе последовательно, одно за другим, выдает все слова языка  $L^{27}$ . Пусть, например, язык L задается регулярным выражением  $aa(ab \cup ba)^*bb$ . Тогда произвольное слово языка L можно получить следующим образом: сначала записать два символа a, потом вслед за ними записать несколько слов, каждое из которых равно либо ab, либо ba и, наконец, дописать в конец два символа b. Получается, что  $L = \{aabb, aaabbb, aaababb, aaababbb, aaabbabb, aabaabbb, ...\}$ . Регулярное выражение  $aa(ab \cup ba)^*bb$  можно рассматривать как способ задания алгоритма, порождающего записанную в фигурных скобках последовательность.

Мы говорим, что некоторое множество языков *замкнуто* относительно какой-то операции, если применение этой операции к языкам из нашего множества дает нам язык из этого же множества.

Следствие 2 (из теоремы 4) Множество регулярных языков замкнуто относительно операций объединения, пересечения, дополнения, конкатенации и навешивания звездочки Клини.

Доказательство. Замкнутость относительно объединения, конкатенации и звездочки Клини следует из определения множества регулярных языков.

Докажем замкнутость относительно дополнения. Пусть L — регулярный язык алфавита  $\Sigma$  и  $M = \langle Q, \Sigma, \delta, s, F \rangle$  — детерминированный конечный автомат распознающий язык L. Тогда автомат  $M' = \langle Q, \Sigma, \delta, s, Q \rangle$ 

<sup>&</sup>lt;sup>27</sup>В следующем разделе курса мы рассмотрим понятия вычислимого и вычислимо перечислимого множеств. Вычислимое множество — это такое подмножество конструктивного пространства, для которого существует алгоритм для ответа на вопрос о принадлежности данного конструктивного объекта нашему множеству. Вычислимо перечислимое — это подмножество конструктивного пространства, для которого существует алгоритм, позволяющий перечислять все его элементы. Мы докажем, что в отличие от случая с регулярными языками и конечными автоматами вычислимость и вычислимая перечислимость — принципиально разные понятия.

 $F\rangle$ , то есть автомат, который получается из автомата M, если в нем поменять местами множества конечных и неконечных состояний, распознает язык  $\Sigma^* \setminus L$ . Действительно,  $p_{M'} = p_M$  и для любого  $w \in \Sigma^*$   $p_{M'}(s,w) \in Q \setminus F \leftrightarrow p_M(s,w) \notin F$ .

Замкнутость относительно пересечений следует из следующего теоретико-множественного равенства:  $L_1 \cap L_2 = \Sigma^* \setminus (\Sigma^* \setminus L_1 \cup \Sigma^* \setminus L_2)$ .  $\square$ 

Докажем одну теорему, которая позволит нам еще глубже разобраться в строении регулярных языков.

**Теорема 5 (о накачке)** Пусть L — регулярный язык и n = o(L). Тогда для любого слова  $w \in L$ , такого что  $|w| \geqslant n$ , существуют слова x, y, z, такие что:

- 1. w = xyz;
- 2.  $|xy| \leqslant n, y \neq e$ ;
- 3. для любого  $i \in \mathbb{N}$  слово  $xy^iz$  принадлежит  $L^{28}$ .

Доказательство. Пусть L — регулярный язык, n = o(L) и  $M = \langle Q, \Sigma, \delta, s, F \rangle$  — детерминированный конечный автомат с n состояниями, распознающий язык L. Существование автомата M следует из теорем 4, 3 и 2. Пусть  $w \in L$ , |w| = m и  $m \geqslant n$ . Тогда  $w = a_1 \dots a_m$ , где  $a_1, \dots, a_m \in \Sigma$ . Пусть для  $0 \leqslant t \leqslant m$   $v_t = a_1 \dots a_t$  (мы считаем, что  $v_0 = e$ ).

Элементы последовательности  $p_M(s, v_0), p_M(s, v_1), \ldots, p_M(s, v_n)$ , состоящей из n+1 элемента, принадлежат состоящему из n элементов множеству Q. Значит, существуют  $p < r \le n$ , такие что  $p_M(s, v_p) = p_M(s, v_r)$ .

Зафиксируем пару таких p и r. Пусть  $x=v_p, y$  таково, что  $v_r=v_p y$ , а z таково, что  $w=v_r z$ . Покажем, что x, y и z удовлетворяют требуемым свойствам.

Ясно, что xyz=w, |y|=r-p>0 и  $|xy|=r\leqslant n$ . Остается доказать, что для любого  $i\in\mathbb{N}$   $p_M(s,xy^iz)\in F$ . Для этого достаточно доказать, что для любого i  $p_M(s,xy^iz)=p_M(s,xyz)$ , поскольку  $w\in L$ . Докажем этот факт индукцией по i.

 $<sup>^{28}</sup>$  Под  $y^i$  мы понимаем слово  $yy\dots y$ , состоящее из (i-1)-ой конкатенации слова y с самим собой. Мы считаем, что  $y^0=e$ . Соответственно  $xy^0z=xz$ , а  $xy^2z=xyyz$ .

Сначала заметим, что  $p_M(s,xy)=p_M(s,x)$ . Действительно,  $x=v_p$  и  $xy=v_r$ . Теперь для i=0  $p_M(s,xy^iz)=p_M(s,xz)=p_M(p_M(s,x),z)=p_M(p_M(s,xy),z)=p_M(s,xyz)$ . Пусть теперь доказываемое утверждение верно для некоторого i. Тогда  $p_M(s,xy^{i+1}z)=p_M(s,(xy)y^iz)=p_M(p_M(s,xy),y^iz)=p_M(p_M(s,x),y^iz)=p_M(s,xy^iz)=p_M(s,xyz)$ . Последнее равенство справедливо по индукционному предположению.

В общих чертах идею приведенного выше доказательства можно выразить следующим образом. Автомат с n состояниями, читая слово длины  $\geqslant n$ , неизбежно побывает дважды в одном и том же состоянии. Цикл, который автомат проходит, выходя из этого состояния и затем опять попадая в него, можно либо "выбросить", удалив соответствующий ему y из слова, либо "продублировать" любое конечное число раз, дописывая к уже существующему y еще какое-то количество игреков.  $\square$ 

Теорема 5 показывает, что в строении каждого бесконечного регулярного языка действительно присутствует определенная "регулярность": в частности, наличие в таком языке некоторых слов вида xyz влечет за собой наличие в нем и всех слов  $xz, xyz, xy^2z, \ldots$  Языки, в которых эта закономерность не прослеживается, регулярными быть не могут. Теорема 5 дает нам удобный инструмент для формального доказательства нерегулярности таких языков.

Пусть, например,  $L=\{a^ib^i:i\in\mathbb{N}\}$  — язык алфавита  $\Sigma=\{a,b\}$ . Докажем, что язык L не является регулярным. Доказательство проведем методом "от противного".

Пусть L регулярен. Положим n=o(L); по теореме 2 это число меньше бесконечности. Рассмотрим слово  $a^nb^n$  (именно для этого n): оно принадлежит языку L. Тогда, по теореме 5,  $a^nb^n=xyz$ , где  $|xy|\leqslant n,\ y\neq e$  и  $xz\in L$ . Ясно, что  $y=a^k$  для  $0< k\leqslant n$ . Получаем, что  $xz=a^{n-k}b^n\in L$ . Однако  $n-k\neq n$ . Противоречие.

Нерегулярность языка L можно было получить и напрямую из теоремы 2. Действительно, для  $i \neq j$  справедливо  $a^i \not\sim_L a^j$ , поскольку  $a^i u \in L$  и  $a^j u \notin L$  для  $u = b^i$ . Получаем, что при разных i и j слова  $a^i$  и  $a^j$  лежат в разных классах эквивалентности отношения  $\sim_L$ . Таким образом,  $o(L) = \infty$  и язык L нерегулярен.

Приведем еще пару примеров нерегулярных языков с доказательствами их нерегулярности.

1. Пусть  $\Sigma$  — конечный алфавит и  $L = \{w \in \Sigma^* : |w|$  — простое число  $\}$ . Предположим, что L регулярен. Пусть n = o(L) и p —

простое число, большее n. Пусть  $a \in \Sigma$  — произвольный символ алфавита  $\Sigma$ . Тогда  $a^p \in L$ . По теореме 5  $a^p = xyz$ , где  $y = a^k$  для k > 0 и  $\{xy^iz : i \in \mathbb{N}\} \subseteq L$ . Так как  $|xy^iz| = p - k + ik$ , то получаем, что все числа p + ik для  $i \geqslant 0$  — простые. В частности, для i = p число p + pk также должно быть простым. Однако p + pk = p(k+1), p > 1, k + 1 > 1. Противоречие.

2. Пусть  $\Sigma = \{a,b\}$  и  $L = \{w \in \Sigma^* :$  количество букв a в слове  $w \neq$  количеству букв b в слове  $w\}$ . Чтобы доказать, что L нерегулярен, можно воспользоваться одним из двух методов, изложенных выше. Но мы поступим по другому. Пусть L' — это язык  $\{a^ib^i : i \in \mathbb{N}\}$ , нерегулярность которого мы уже доказали. Пусть L'' — это язык, задаваемый регулярным выражением  $a^*b^*$ ; он, естественно, регулярен. Тогда  $L' = L'' \cap (\Sigma^* \setminus L)$ . Если бы язык L был регулярным, то по следствию 2 регулярным был бы и язык L', однако это не так.

В принципе, для понимания сути этих примеров уже достаточно. Выше мы говорили о том, что конечные автоматы — это один из способов формализации понятия алгоритма для вычисления функции, отображающей конструктивное пространство  $\Sigma^*$  в конструктивное пространство возможных ответов {да, нет}. Как видно из примеров, формализация явно неудачная. Такой простой язык, как  $\{a^ib^i:i\in\mathbb{N}\}$  имеет четкую структуру и для него явно должен существовать алгоритм, отвечающий на вопрос о принадлежности слов этому языку. Однако мы доказали, что реализовать любой такой алгоритм при помощи конечного автомата невозможно.

Попытаемся разобраться, в чем тут дело. Для начала обсудим, по какому алгоритму мог бы действовать человек, если бы он вооружился бумагой и карандашом и поставил бы себе задачу распознать любое слово из этого языка.

Пусть, например, человек сел за стол, положил перед собой чистый лист бумаги, и ему кто-нибудь начал бы зачитывать слово  $w \in a^*b^*$ , символ за символом. Тогда он мог бы начать считать буквы a, а когда они закончатся — буквы b. После прочтения слова он сравнил бы два полученных числа и выдал ответ, в зависимости от того, равны эти числа или нет. Если слово не слишком длинное, то у него бы все получилось, однако если длина w очень велика, то он мог бы просто сбиться со счета. Чтобы этого не произошло, он может записывать числа на бумаге. Но опять же, если слово очень велико, то одного листа бумаги ему может не хватить.

Для человека это не проблема, он возьмет второй чистый лист и продолжит на нем. Когда вся бумага рядом с ним закончится, он попросит прервать чтение слова, побежит в магазин, купит новую пачку и продолжит. Конечно, в реальном мире до бесконечности этот процесс продолжаться не может: либо у этого человека и всех его спонсоров кончатся деньги, либо во всех магазинах города закончится бумага; может случится и так, что этот человек просто умрет от старости, не дождавшись конца слова, а его потомки не захотят продолжить начатое им дело. Но давайте будем оптимистами: предположим, что человечество будет существовать вечно и найдет способ неограниченного воспроизводства бумаги и карандашей, а бессмертные боги, сжалившись над нашим тружеником, согласятся помочь ему и будут неограниченно продлевать срок его жизни. Тогда предпринятое нашим героем начинание когда-нибудь неизбежно завершится, поскольку какой бы большой не была длина слова w, она все же конечна. Предложенный алгоритм работает всегда; единственным препятствием для его реализации может является ограниченность ресурсов. Если же мы предполагаем, что ресурсы потенциально бесконечны<sup>29</sup>, то больше никаких препятствий не существует.

Можно предположить, что слово, принадлежность которого к языку человек должен распознать, не диктуется ему буква за буквой, а уже написано перед ним на бумаге. Тогда человек может поступить следующим

Если мы рассматриваем пример с нашим героем, то для него достаточно, чтобы множество листов бумаги, предоставленных для его работы, было потенциально бесконечным. Действительно, в каждый момент своей работы ему нужно лишь конечное число этих листов. Можно представить себе и актуально бесконечную пачку бумаги: это когда бесконечное число листов, пронумерованных числами  $0,1,2,\ldots$ , лежат на столе одновременно.

Приведем еще один пример, иллюстрирующий разницу между понятиями актуальной и потенциальной бесконечности. Пусть есть некий человек, который способен думать о натуральных числах, причем он не может думать одновременно о двух разных числах и для того, чтобы подумать о каком-то числе, ему требуется не менее секунды. Тогда множество чисел, о которых он может подумать в течении своей жизни, является потенциально бесконечным, поскольку он может подумать, в принципе, о любом числе. Однако оно не является актуально бесконечным из-за того, что время жизни этого человека ограничено.

<sup>&</sup>lt;sup>29</sup>Слово "потенциально" здесь имеет вполне четкий смысл. Различают потенциальную и актуальную бесконечности. Актуальная бесконечность — это завершенная бесконечность; актуально бесконечные объекты существуют как единое целое, во всех своих бесконечных подробностях. Потенциальная же бесконечность не предполагает этой завершенности, а лишь возможность неограниченного увеличения.

образом: последовательно зачеркивать в этом слове буквы a и b, по очереди, а когда либо все буквы a, либо все буквы b иссякнут, посмотреть на то, что осталось. На первый взгляд, этот алгоритм свободен от описанных выше ограничений, однако по сути это не так. Несмотря на то, что человек в ходе реализации этого алгоритма никакой дополнительной бумаги не тратит, все же непонятно, сколько этой бумаги потребуется для того, чтобы записать исходное слово, поскольку мы допускаем слова произвольной конечной длины. Получается, что множество листов бумаги, необходимых для реализации этого алгоритма, опять должно быть потенциально бесконечным.

Итак, человек может реализовать алгоритм для распознавания слов языка  $\{a^ib^i:i\in\mathbb{N}\}$ , однако для этого ему потребуется потенциально бесконечное количество таких ресурсов, как время и память. Вместо человека это может сделать и какое-нибудь идеальное механическое устройство с бесконечной памятью; например, машина Тьюринга, которую мы будем изучать в следующем разделе курса. Однако конечный автомат этого сделать не может. Несмотря на то, что мы ничем не ограничиваем время работы конечных автоматов, память каждого автомата конечна. Конечный автомат — это, по определению, устройство с конечным числом состояний. Распознавая слово, автомат меняет свои состояния; в ходе своей работы, прочитав некоторое количество букв слова, автомат приходит в одно из состояний, однако он не может помнить, как он туда попал. Если два разных слова приводят автомат в одно и то же состояние, то он не может отличить одно от другого. Именно в этом и кроется причина того, что никакой конечный автомат не может распознать язык  $\{a^ib^i: i \in \mathbb{N}\}$ : для распознавания этого языка необходимо отличать друг от друга все слова вида  $a^i$  при различных i.

Конечно, для каждого конечного фрагмента языка  $\{a^ib^i:i\in\mathbb{N}\}$  можно придумать автомат, который будет правильно работать на этом фрагменте. Например, можно придумать автомат, который будет правильно отвечать на вопрос о принадлежности слова этому языку, если длина слова не превосходит 1000000, однако для слов больший длины он будет давать ошибочные ответы. Можно добавить к автомату какое-то количество новых состояний и он станет давать правильные ответы для всех слов длины меньше 1000000000, но все равно для слов еще большей длины неизбежны ошибки и, что самое важное, это будет уже другой автомат. Единственный выход в данном случае — это автомат, который в ходе своей работы мог бы сам добавлять себе новые состояния по мере

необходимости. То есть, устройство с потенциально бесконечной памятью.

Любое устройство с конечной памятью, какой бы не была его конструкция, можно представить в виде конечного автомата. Состояниями этого автомата будут состояния памяти данного устройства. Если мы хотим, чтобы устройство использовалось для представления алгоритмов, то, в силу определения 1, состояние памяти этого устройства на каждом шаге его работы должно быть полностью обусловлено состоянием памяти на предыдущем шаге и входными данными (мы понимаем слова о "четкости" и "недвусмысленности" инструкций в определении 1 именно таким образом). Добавляя к состояниям памяти устройства правила перехода между ними, зависящие от входных данных, получаем конечный автомат. Со всеми вытекающими отсюда последствиями. В частности, с возможностью распознавать только регулярные языки.

Однако, несмотря на все ограничения того мира, в котором нам выпала честь родиться, и, в частности, несмотря на невозможность сконструировать устройство с бесконечной памятью, состоящее из микросхем и проводов, нам все же хотелось бы считать, что для таких простых языков, как  $\{a^ib^i:i\in\mathbb{N}\}$ , существует алгоритм распознавания. Хотя бы потому, что конечные автоматы слишком просты и как объект исследования мало интересны. Значительно интереснее посмотреть, что можно вычислить при помощи алгоритмов, если допустить, что устройства для их реализации не имеют заранее установленных ограничений на количество ячеек памяти. К этому мы и переходим.

## 3 Рекурсивные функции и общее понятие вычислимости

Во введении мы дали определения понятий конструктивного объекта, конструктивного пространства и вычислимой функции. Конструктивными объектами мы назвали объекты, которые могут быть полностью описаны при помощи конечной последовательности символов, а конструктивным пространством — множество всех конструктивных объектов одного вида. Попытаемся немного уточнить это довольно расплывчатое определение.

Поскольку различные объекты одного и того же конструктивного

пространства схожи по своей природе, можно считать, что и их описания в целом похожи друг на друга, отличаясь лишь в деталях. В частности, естественно ожидать, что в этих описаниях используются символы одного и того же алфавита. Если принять этот тезис, то, не ограничивая общности рассмотрения, можно представить ситуацию следующим образом: для каждого конструктивного пространства  $\mathcal{K}$  существует конечный алфавит  $\Sigma_{\mathcal{K}}$ , а описаниями объектов пространства являются слова этого алфавита.

Идеальной была бы ситуация, когда соответствие между описаниями объектов и самими объектами взаимно-однозначно. Однако на практике этот идеал редко достижим<sup>30</sup>. Рассмотрим, например, конструктивное пространство  $\mathbb{Q}$ , состоящее из всех рациональных чисел. Для каждого объекта этого пространства существует его естественное описание, являющееся словом алфавита  $\Sigma_{\mathbb{Q}} = \{0, \dots, 9, /, -\}$ . Однако функция, сопоставляющая описанию каждого рационального числа само это число, не является разнозначной и даже не всюду определена. Например, -3/7 и -9/21 являются описаниями одного и того же числа, а последовательность символов //-/11 не соответствует никакому рациональному числу.

Тем не менее, в одном, самом важном для нас случае, ситуация полностью соответствует идеалу. Это происходит, когда конструктивным пространством является множество всех слов некоторого конечного алфавита  $\Sigma$ . В этом случае алфавитом описания является сам алфавит  $\Sigma$ , а описанием любого слова является само это слово.

Самым важным этот случай является потому, что все остальные к нему сводятся. Напомним, что основным объектом изучения для нас являются вычислимые функции, то есть такие функции из одного конструктивного пространства в другое, для которых существует алгоритм, позволяющий по описанию объекта первого конструктивного пространства получить описание значения функции на этом объекте как объекта второго конструктивного пространства. Для алгоритмов объекты доступны только через их описания.

Пусть есть конструктивное пространство  $\mathcal{K}_1$ , описаниями объектов которого являются слова алфавита  $\Sigma_1$ , и конструктивное пространство  $\mathcal{K}_2$ , для описания которого служит алфавит  $\Sigma_2$ . Пусть для i=1,2  $d_i$ :

 $<sup>^{30}</sup>$ На самом деле, конечно, этого идеала всегда можно достигнуть, однако при этом теряется "естественность" описаний.

 $\Sigma_i^* \to \mathcal{K}_i$  — частичные<sup>31</sup> функции, сопоставляющие описаниям объектов сами описываемые объекты. Тогда функция  $f: \mathcal{K}_1 \to \mathcal{K}_2$  является вычислимой, если существует алгоритм A, который любое слово  $w \in \Sigma_1^*$  преобразует в слово  $v \in \Sigma_2^*$ , такое что если  $d_1(w)$  определено, то  $d_2(v)$  определено и  $f(d_1(w)) = d_2(v)$ . Для каждого слова  $w \in \Sigma_1^*$  обозначим через  $\varphi_A(w)$  слово алфавита  $\Sigma_2$ , в которое наш алгоритм перерабатывает исходное слово w. Получаем функцию<sup>32</sup>  $\varphi_A: \Sigma_1^* \to \Sigma_2^*$ , характеризующую результат работы алгоритма. От этой функции помимо того, что она определяется при помощи алгоритма, требуются две вещи: во-первых, чтобы она была определена на всех w из области определения  $d_1$ , а вовторых, чтобы для любого такого w  $f(d_1(w))$  было равно  $d_2(\varphi_A(w))^{33}$ . При соблюдении этих двух условий функция f полностью восстанавливается по функции  $\varphi_A$ ; действительно, для любого  $k \in \mathcal{K}_1$  зная функцию  $\varphi_A$  можно узнать, чему равно  $f(k)^{34}$ . Таким образом, вместо того, чтобы изучать функцию  $\varphi_A$ .

Функция  $\varphi_A$  является функцией из конструктивного пространства  $\Sigma_1^*$  в конструктивное пространство  $\Sigma_2^*$ . Эта функция вычислима при помощи того же самого алгоритма A; действительно, по описанию любого объекта  $w \in \Sigma_1^*$ , которым является само слово w, алгоритм A получает слово  $\varphi_A(w)$ , являющееся описанием слова  $\varphi_A(w)$  как объекта конструктивного пространства  $\Sigma_2^*$ . Подводя итоги, можно сказать, что вместо вычислимых функций из  $\mathcal{K}_1$  в  $\mathcal{K}_2$  можно изучать вычислимые функции из  $\Sigma_1^*$  в  $\Sigma_2^*$ , не теряя при этом ничего существенного.

Можно ввести еще одно допущение, не ограничивающее общность рассмотрений. Можно всегда считать, что  $\Sigma_1 = \Sigma_2$ , взяв при необходимости объединение этих алфавитов. Сузив таким образом нашу задачу, дадим следующие определения<sup>35</sup>.

 $<sup>^{31}</sup>$ То есть, возможно не всюду определенные.

<sup>&</sup>lt;sup>32</sup>Вообще говоря, частичную, поскольку алгоритмы при некоторых входных данных могут "зацикливаться" и не выдавать никаких результатов.

 $<sup>^{33}</sup>$ В частности,  $\varphi_A(w)$  должно попадать в область определения функции  $d_2$ . Отметим еще следующее. Если  $d_1(w_1)=d_1(w_2)$ , то слова  $\varphi_A(w_1)$  и  $\varphi_A(w_2)$  могут быть как равными, так и нет; главное, чтобы значение функции  $d_2$  на этих словах было одинаковым. Если w не попадает в область определения  $d_1$ , то  $\varphi_A(w)$  может быть определено, а может быть и не определено; если оно определено, то оно может быть равно чему угодно.

 $<sup>^{34}</sup>f(k)=d_2(\varphi_A(d_1^{-1}(k)))$  (точнее даже так:  $\{f(k)\}=d_2(\varphi_A(d_1^{-1}(k)))$ ).

 $<sup>^{35}</sup>$ Более точные, чем определение 4, но все же недостаточно точные для того, чтобы с ними можно было работать. Эти определения, как и определение 4, в большей

Определение 9 Пусть  $\Sigma$  — конечный алфавит. Частичная функция  $f: \Sigma^* \to \Sigma^*$  называется *частично вычислимой*, если существует алгоритм, перерабатывающий любое слово w из области определения f в слово f(w) и не заканчивающий свою работу, если он получает на вход слово, не принадлежащее области определения  $f^{36}$ .

**Определение 10** Функция  $f: \Sigma^* \to \Sigma^*$  называется *вычислимой*, если она частично вычислима и всюду определена.

Далее, поскольку нам придется много работать с частичными функциями, примем следующие обозначения. Пусть  $f: X \to Y$  — частичная функция из множества X в множество Y. Область определения функции f обозначим  $\delta f$ , а область значений —  $\rho f$ .

Переходим к более точным рассмотрениям. Для начала введем одно из самых известных идеальных устройств для вычисления функций, а именно — машину Тьюринга. Как и ранее, у нас будет два сорта определений: формальные (то есть те, на основании которых можно что-то доказывать) и неформальные, поясняющие суть формальных. На неформальном уровне мы введем саму машину и программы для нее, а на формальном — только программы и конфигурации.

Машина Тьюринга состоит из бесконечной и ограниченной с одного (левого) конца ленты, разбитой на ячейки, в каждой из которых может быть записан символ некоторого конечного алфавита. Кроме ленты машина также имеет головку для считывания и записи значений ячеек. Головка на каждом шаге работы машины указывает на одну из ячеек. Кроме этого, имеется конечное множество состояний головки, причем на каждом шаге работы машины головка находится в одном из состояний. В переходе между шагами работы головка машины может менять свое состояние и при этом либо записывать в ячейку, на которую она указывает, символ алфавита, либо сдвигаться влево или вправо по ленте на одну ячейку. Самая левая ячейка содержит символ m, который не может быть изменен; из этой ячейки головка может только сдвигаться вправо. На остальные ячейки это ограничение не распространяется. Мы всегда считаем, что в каждой момент работы машины почти все $^{37}$  ячейки ленты содержат символ n и что символ m может содержать только самая левая ячейка.

степени "философские", чем "математические".

 $<sup>^{36}</sup>$ Другими словами, если  $f = \varphi_A$  для некоторого алгоритма A.

 $<sup>^{37}</sup>$ То есть все за исключением конечного числа.

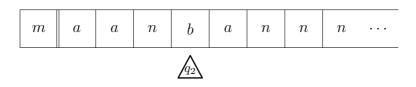


Рис. 5:

На рисунке 5 изображено одно из возможных состояний машины. В нулевой ячейке ленты записан символ m, в первой и второй -a, в третьей -n, в четвертой -b, в пятой -a, а во всех остальных -n. Головка машины находится в состоянии  $q_2$  и указывает на четвертую ячейку ленты.

Прежде чем переходить к описанию программ для машины Тьюринга, введем несколько соглашений. Пусть  $\Sigma$  — некоторый конечный алфавит, который не содержит символов n, m, L, R. В ячейки ленты машины могут быть записаны все символы алфавита  $\Sigma$ , а также n и m (m только в самую левую ячейку). Множество состояний машины будем обозначать через  $Q = \{q_0, q_1, \ldots, q_k\}$ ; при этом мы считаем, что  $k \geqslant 1$ , то есть что имеется не менее двух состояний. Состояние  $q_1$  мы называем hayanbhum; это то состояние, из которого машина обычно начинает работу. Состояние  $q_0$  мы называем koneyhum и считаем, что попав в это состояние, машина останавливается.

Определение 11 Конфигурацией машины Тьюринга называется тройка  $\langle w, q, l \rangle$ , где  $w \in (\Sigma \cup \{n\})^*$ ,  $q \in Q$ ,  $l \in \mathbb{N}$  и если слово w не равно пустому, то его последний символ не равен n.

Конфигурации описывают состояния машины. Слово w отражает содержимое ячеек ленты: если  $w=a_1\ldots a_s$ , для  $a_1,\ldots,a_s\in\Sigma\cup\{n\}$ , то мы считаем, что для  $0< i\leqslant s$  в i-ой ячейке ленты записан символ  $a_i$ , а в остальных — символ n. Вторая компонента тройки — это состояние головки, а третья — номер ячейки, на которую она указывает. Например, состояние машины, изображенное на рисунке 5, описывается конфигурацией  $\langle aanba, q_2, 4 \rangle$ . Для каждого состояния машины существует ровно одна конфигурация, которая его описываетс.

Определение 12 Программой для машины Тьюринга называется пара  $\langle \varphi, \psi \rangle$ , где  $\varphi$  — функция из  $(Q \setminus \{q_0\}) \times (\Sigma \cup \{n, m\})$  в  $Q, \psi$  — функция из  $(Q \setminus \{q_0\}) \times (\Sigma \cup \{n, m\})$  в  $\Sigma \cup \{n, L, R\}$  и для любого  $q \in Q \setminus \{q_0\}$   $\psi(q, m) = R$ .

 ${
m He}$ формально программа — это список команд одного из следующих трех видов:

$$\begin{array}{ll} q_i a \to q_j b, & \text{где } q_i \in Q \setminus \{q_0\}, q_j \in Q, a,b \in \Sigma \cup \{n\}; \\ q_i a \to q_j L, & \text{где } q_i \in Q \setminus \{q_0\}, q_j \in Q, a \in \Sigma \cup \{n\}; \\ q_i a \to q_j R & \text{где } q_i \in Q \setminus \{q_0\}, q_j \in Q, a \in \Sigma \cup \{n,m\}. \end{array}$$

Для команды  $q_i a \to q_j x \; q_j = \varphi(q_i, a),$  а  $x = \psi(q_i, a).$  Для каждого  $q_i \neq q_0$  и  $a \in \Sigma \cup \{n, m\}$  в списке команд программы найдется ровно одна команда вида  $q_i a \to \dots$ , причем если a = m (головка машины указывает на самую левую ячейку ленты), то это команда третьего вида (заставляющая головку сдвинуться вправо). Вспомним, что в каждый момент работы машины ее головка находится в некотором состоянии  $q_i$  и указывает на ячейку, в которой записан некоторый символ a. Тогда, после исполнения команды первого вида  $q_i a \rightarrow q_i b$  состояние головки станет равным  $q_i$ , ее позиция не изменится, а в ячейку ленты, на которую указывает головка, запишется символ b. При исполнении команды второго вида  $q_i a o q_i L$ головка машины перейдет в состояние  $q_i$  и сдвинется на одну позицию влево, а состояние ячеек ленты не изменится. Это всегда возможно, поскольку  $a \neq m$  и головка указывает не на самую левую ячейку. Наконец, действие команды третьего вида аналогично действию команды второго вида, только сдвиг происходит не влево, а вправо. Из-за того, что для каждого неконечного состояния  $q_i$  и для каждого символа a, который может быть записан в ячейке, в программе существует ровно одна команда вида  $q_i a \to \dots$ , машина в каждом своем состоянии "знает что делать" и действие, предписанное ей программой, определяется однозначно. Если есть программа, то можно задать состояние машины и запустить ее работать по этой программе. Тогда она будет по шагам исполнять команды и менять свое состояние. Причем либо после некоторого числа шагов головка машины придет в состояние  $q_0$  и процесс остановится, либо машина будет работать бесконечно.

Прежде чем давать формальные определения, отражающие работу машины Тьюринга, введем два обозначения. Пусть w — слово алфавита

 $\Sigma \cup \{n\}$  и  $k \in \mathbb{N}$ . Тогда пусть lg(w,k) — это слово того же алфавита, равное w при  $|w| \geqslant k$  и  $wn^{k-|w|}$ , если |w| < k. Другими словами, lg(w,k) — это слово длины  $\geqslant k$ , которое получается из слова w, если к нему приписать справа недостающее число символов n. Наоборот, через sh(w) обозначим слово, которое получается из w, если в нем стереть все символы n справа вплоть до первого символа, не равного n. Более точно, sh(w) равно единственному слову  $v \in (\Sigma \cup \{n\})^*$ , такому что последняя буква слова v не равна n и  $w = vn^k$  для некоторого  $k \in \mathbb{N}$ .

Дадим формальные определения. Пусть  $P = \langle \varphi, \psi \rangle$  — программа и  $C = \langle w, q, l \rangle$  — конфигурация машины. Определим значение на этих аргументах функции Step, сопоставляющей каждой конфигурации и каждой программе некоторую конфигурацию.

- 1. Если  $q = q_0$ , то Step(C, P) = C (если состояние головки равно  $q_0$ , то машина стоит и ничего не делает).
- 2. Если  $q \neq q_0$  и l = 0, то Step(C, P) полагаем равным  $\langle w, \varphi(q, m), 1 \rangle$ .
- 3. Если  $q \neq q_0$  и l>0, то пусть v=lg(w,l). Тогда  $v=v_1av_2$ , где  $v_1$  и  $v_2$  слова алфавита  $\Sigma \cup \{n\}$  и a символ того же алфавита, причем  $|v_1|=l-1$ . Рассмотрим три случая:
  - (a)  $\psi(q, a) = L$ . Полагаем  $Step(C, P) = \langle w, \varphi(q, a), l 1 \rangle$ ;
  - (b)  $\psi(q,a) = R$ . Полагаем  $Step(C,P) = \langle w, \varphi(q,a), l+1 \rangle$ ;
  - (c)  $\psi(q, a) \neq L, R$ . Полагаем  $Step(C, P) = \langle sh(v_1\psi(q, a)v_2), \varphi(q, a), l \rangle$ .

Определение Step закончено. На неформальном уровне если конфигурация C отражает состояние машины Тьюринга на каком-то шаге работы, то конфигурация  $\operatorname{Step}(C,P)$  отражает состояние машины на следующем шаге, после выполнения подходящей команды программы P. Определим теперь трехместную функцию Com, первый аргумент которой — это конфигурация, второй — программа, а третий — натуральное число.

- 1. Com(C, P, 0) = C;
- 2.  $\operatorname{Com}(C, P, t + 1) = \operatorname{Step}(\operatorname{Com}(C, P, t), P)$ .

На неформальном уровне конфигурация Com(C, P, t) отражает состояние машины, которая стартовала из состояния, соответствующего конфигурации C и проделала t шагов работы. Поясним все сказанное на примере. Пусть  $\Sigma = \{a,b\},\ Q = \{q_0,q_1,q_2,q_3\}$  и программа P задается следующим списком команд:

$$\begin{array}{lll} q_1a \rightarrow q_2b, & q_1b \rightarrow q_2a, & q_1m \rightarrow q_1R, & q_1n \rightarrow q_3L, \\ q_2a \rightarrow q_1R, & q_2b \rightarrow q_1R, & q_2m \rightarrow q_2R, & q_2n \rightarrow q_2R, \\ q_3a \rightarrow q_3L, & q_3b \rightarrow q_3L, & q_3m \rightarrow q_0R, & q_3n \rightarrow q_3L. \end{array}$$

Пусть  $C = \langle aba, q_1, 1 \rangle$ . Тогда значения функции Com(C, P, t) задаются следующей таблицей:

```
\mathrm{Com}(C,P,0) = \langle aba,q_1,1 \rangle, \qquad \mathrm{Com}(C,P,1) = \langle bba,q_2,1 \rangle, \ \mathrm{Com}(C,P,2) = \langle bba,q_1,2 \rangle, \qquad \mathrm{Com}(C,P,3) = \langle baa,q_2,2 \rangle, \ \mathrm{Com}(C,P,4) = \langle baa,q_1,3 \rangle, \qquad \mathrm{Com}(C,P,5) = \langle bab,q_2,3 \rangle, \ \mathrm{Com}(C,P,6) = \langle bab,q_1,4 \rangle, \qquad \mathrm{Com}(C,P,7) = \langle bab,q_3,3 \rangle, \ \mathrm{Com}(C,P,8) = \langle bab,q_3,2 \rangle, \qquad \mathrm{Com}(C,P,9) = \langle bab,q_3,1 \rangle, \ \mathrm{Com}(C,P,10) = \langle bab,q_3,0 \rangle, \qquad \mathrm{Com}(C,P,11) = \langle bab,q_0,1 \rangle и \mathrm{Com}(C,P,t) = \langle bab,q_0,1 \rangle, \ \mathrm{для} \ \mathrm{Bcex} \ t > 11.
```

Неформально головка машины, работающей по программе P, идет вправо, меняя символ a на символ b и наоборот, пока не доходит до символа n, после чего принимает состояние  $q_3$ , идет влево до конца, сдвигается на одну ячейку вправо и останавливается. Стартовав из состояния, в котором на ленте записано слово aba, а головка находится в состоянии  $q_1$  и указывает на первую ячейку, машина работала 11 шагов и остановилась. Не все команды программы выполнились хотя бы один раз; например, команда  $q_2n \to q_2R$  не была выполнена не разу. Однако если бы машина стартовала из состояния, описываемого конфигурацией  $\langle e, q_2, 1 \rangle$ , то наоборот выполнялась бы только эта команда, и машина бы никогда не закончила работу (головка уходила бы все дальше и дальше вправо по ленте).

Прежде чем дать следующее определение, напомним, что мы продолжаем придерживаться наших соглашений относительно  $\Sigma$  и Q. В частности,  $\Sigma$  — конечный алфавит, не содержащий символы n,m,L,R.

Определение 13 Частичная функция  $f: \Sigma^* \to \Sigma^*$  называется *вычис*лимой на машине Тьюринга, если существует программа P, такая что для каждого слова  $w \in \Sigma^*$ :

1. если  $w \in \delta f$ , то для некоторого  $t \in \mathbb{N} \operatorname{Com}(\langle w, q_1, 1 \rangle, P, t) = \langle f(w), q_0, 1 \rangle;$ 

2. если  $w \notin \delta f$ , то для всех  $t \in \mathbb{N}$  если  $\mathrm{Com}(\langle w,q_1,1\rangle,P,t) = \langle v,q,l\rangle,$  то  $q \neq q_0$ .

Другими словами, частичная функция f вычислима на машине Тьюринга (с подходящим множеством состояний Q, работающей по программе P), если для любого  $w \in \Sigma^*$  эта машина, стартуя из состояния, в котором на ленте последовательно записаны символы слова w, а головка находится в состоянии  $q_1$  и указывает на первую ячейку

- 1. остановится в состоянии, в котором на ленте последовательно записаны символы слова f(w), а головка находится в состоянии  $q_0$  и указывает на первую ячейку, в случае если f(w) определено;
- 2. никогда не остановится, если f(w) не определено.

В качестве примера можно рассмотреть функцию  $f: \Sigma^* \to \Sigma^*$  для  $\Sigma = \{a,b\}$ , такую что f(w) равно слову, которое получается из слова w, если в нем все буквы a заменить на буквы b и наоборот. Эта функция всюду определена и, как показывает предыдущий пример, вычислима на машине Тьюринга (с множеством  $Q = \{q_0, q_1, q_2, q_3\}$  по приведенной выше программе).

В отличии от определений 4 и 9, которые мы давали ранее, определение 13 является формальным, строгим математическим определением, на основании которого возможно что-то доказывать. Вместе с тем определение 9 выглядит более общим, чем определение 13. Ясно, что каждая частичная функция из  $\Sigma^*$  в  $\Sigma^*$ , вычислимая на машине Тьюринга, вычислима и по определению 9, поскольку работающая по программе машина Тьюринга — это детерминированное механическое устройство, которое несомненно можно рассматривать как реализацию алгоритма. Насчет обратного возможны сомнения. Ранее мы уже ввели одну формализацию алгоритма — конечные автоматы, но потом доказали, что существуют функции, алгоритмически вычислимые но не вычислимые на автоматах. Мы выяснили и причину — конечность числа состояний автомата. Теперь мы ввели новое устройство — машину Тьюринга, которое свободно от этого ограничения (количество ячеек ленты у машины Тьюринга бесконечно). Можно было бы ожидать, что мы опять чего-то не учли и что опять найдутся функции, существование алгоритма для которых не подлежит сомнению, но которые на машине Тьюринга не вычислимы. Однако с тех пор, как известны машины Тьюринга, прошло уже много времени, а такие функции так никто и не обнаружил. Более того, в настоящий момент не представляется возможным придумать хоть какой-нибудь алгоритм, инструкции которого нельзя было бы промоделировать в виде команд машины Тьюринга. В связи с этим все математики в настоящее время принимают

**Тезис Тьюринга**: Всякая вычислимая (согласно определению 9) частичная функция из  $\Sigma^*$  в  $\Sigma^*$  вычислима на машине Тьюринга.

Тезис Тьюринга нельзя доказать, потому что определение 9 не является математическим определением. Однако им можно пользоваться: например, если есть описание алгоритма для вычисления функции из  $\Sigma^*$  в  $\Sigma^*$ , данное на обычном<sup>38</sup> языке, можно, пользуясь этим тезисом, утверждать, что существует программа для машины Тьюринга, реализующая этот алгоритм, не выписывая саму программу. Со временем и мы станем так поступать. А пока продолжим изложение на формальном уровне.

Переходим к рассмотрению частичных функций на натуральных числах со значениями в  $\mathbb{N}$ . Мы будем рассматривать частичные функции от нескольких аргументов. Количество аргументов функции мы называем ее местностью или арностью. Для каждого натурального числа k k-местная (или k-арная) функция — это функция от k аргументов. Ясно, что такое k-местная частичная функция для k > 0; для k = 0 0-местная частичная функция из  $\mathbb{N}$  в  $\mathbb{N}$  — это просто константа, равная либо некоторому натуральному числу, либо неопределенному значению  $^{39}$ .

Пусть  $k, n \in \mathbb{N}$  и имеется одна n-местная частичная функция  $g(x_1, \ldots, x_n)$  и n k-местных частичных функций  $f_1(x_1, \ldots, x_k), \ldots, f_n(x_1, \ldots, x_k)$ .

<sup>&</sup>lt;sup>38</sup>Например, русском.

<sup>&</sup>lt;sup>39</sup>Можно представлять себе ситуацию следующим образом. Часто встречаются случаи, когда значение функции от нескольких аргументов реально зависит от меньшего числа аргументов. Например, есть функция от двух аргументов f(x,y)=x+y; значение f(x,y) зависит как от x, так и от y. Есть другая функция от двух аргументов: g(x,y)=x+(y-y). Значение g(x,y) зависит только от того, чему равен x. Формально g— это 2-местная функция, но ее можно рассматривать и как одноместную функцию g'(x)=x. Мы в нашем курсе считаем, что функции g и g'— разные, хотя бы потому что это функции разной местности, однако нельзя отрицать, что функция g' в некотором смысле соответствует функции g. Теперь рассмотрим функцию h(x,y)=2, равную константе g'0 при всех значениях g'0. В том же смысле, в каком функции g'1 соответствовала функция g'2, функции g'3 по соответствуют две 1-местные функции g'4 при рассмотрим 2-местную частичную функцию g'5, которая не определена при всех значениях g'6 при g'7 по ей соответствует 0-местная функция g'8 с неопределенным значениях g'9 по ей соответствует 0-местная функция g'9 при всех значениях g'9 при g'9

Мы говорим, что k-местная частичная функция h является cynepnosu-uueŭ функций  $g, f_1, \ldots, f_n$ , если  $h(x_1, \ldots, x_k) = g(f_1(x_1, \ldots, x_k), \ldots, f_n(x_1, \ldots, x_k))$ . Для набора натуральных чисел  $m_1, \ldots, m_k$  значение  $h(m_1, \ldots, m_k)$  определено тогда и только тогда, когда во-первых определены числа  $l_1 = f_1(m_1, \ldots, m_k), \ldots, l_n = f_n(m_1, \ldots, m_k)$ , а во-вторых, определено  $g(l_1, \ldots, l_n)$ . Заметим, что если  $g, f_1, \ldots, f_n$  — всюду определеные функции, то h также всюду определена.

Пусть теперь  $k \in \mathbb{N}$  и есть одна k-местная частичная функция  $g(x_1, \ldots, x_k)$  и одна (k+2)-местная частичная функция  $h(z, y, x_1, \ldots, x_k)$ . Мы говорим, что (k+1)-местная частичная функция  $f(t, x_1, \ldots, x_k)$  получается из функций g и h по схеме npumumuehoù pexypcuu, если для всех  $t, x_1, \ldots, x_k \in \mathbb{N}$  выполнены следующие равенства:

$$\begin{cases}
f(0, x_1, \dots, x_k) &= g(x_1, \dots, x_k) \\
f(t+1, x_1, \dots, x_k) &= h(f(t, x_1, \dots, x_k), t, x_1, \dots, x_k)
\end{cases}$$

Ясно, что эти два равенства однозначно определяют значение функции f на всех аргументах. Область определения f следует из способа ее задания. Пусть, например,  $m_1, \ldots, m_k \in \mathbb{N}$ . Тогда  $f(0, m_1, \ldots, m_k)$  определено тогда и только тогда, когда определено  $g(m_1, \ldots, m_k)$ . Значение  $f(1, m_1, \ldots, m_k)$  определено тогда и только тогда, когда определены  $f(0, m_1, \ldots, m_k)$  и  $h(f(0, m_1, \ldots, m_k), 0, m_1, \ldots, m_k)$ . Значение  $f(2, m_1, \ldots, m_k)$  определено тогда и только тогда, когда определены  $f(1, m_1, \ldots, m_k)$  и  $h(f(1, m_1, \ldots, m_k), 1, m_1, \ldots, m_k)$ . И так далее. Получаем, что при фиксированных  $m_1, \ldots, m_k$  множество чисел t, таких что функция t0 определена на наборе  $t, m_1, \ldots, m_k$ 0 равно либо всему натуральному ряду, либо конечному множеству t1, t2, t3, t4, t5, t6. Отметим еще, что если t7 и t8 всюду определены, то t7 также всюду определена.

Мы будем называть базисными (или базовыми) следующие функции:

- 1. 0-местную функцию, равную константе 0;
- 2. 1-местную функцию s(x) = x + 1;
- 3. для каждого  $n \geqslant 1$  и каждого  $1 \leqslant k \leqslant n$  *n*-местную функцию  $I_n^k(x_1,\ldots,x_n) = x_k$ .

**Определение 14** Функция f называется npumumueho pekypcuehoŭ, если существует конечная последовательность функций  $f_1, \ldots, f_n$ , такая

что  $f_n = f$  и для каждого  $1 \le i \le n$  функция  $f_i$  либо базисная, либо получается из предыдущих членов последовательности суперпозицией или по схеме примитивной рекурсии.

Другими словами, примитивно рекурсивные функции — это функции, которые можно получить из базисных при помощи конечного числа суперпозиций и примитивных рекурсий. Так как базисные функции всюду определены, то каждая примитивно рекурсивная функция является всюду определенной.

**Предложение 1** Следующие функции являются примитивно рекурсивными:

- 1. 0-местная функция, равная константе k для произвольного  $k \in \mathbb{N}$ ;
- 2. 1-местная функция o(x), равная нулю для всех x;
- 3. 1-местная функция, равная константе k при всех x для произвольного  $k \in \mathbb{N}$ ;
- 4. 2-местная функция x + y;
- 5. 2-местная функция  $x \cdot y$ ;
- 6. 1-местная функция x 1, равная 0 при x = 0 и x 1 при x > 0;
- 7. 2-местная функция x y, равная 0 при x < y и x y при  $x \geqslant y$ .

Доказательство. (1) Для  $k \in \mathbb{N}$  0-местная функция, равная k, получается при при помощи k суперпозиций из базисных функций s и 0 ( $k = ss \dots s(0)$ , символ s встречается в записи k раз).

(2) и (3) Ясно, что (2) — это частный случай (3). Пусть для  $k \in \mathbb{N}$   $f_k(x) = k$  — одноместная функция, равная константе k. Тогда  $f_k$  получается по схеме примитивной рекурсии из 0-местной функции k и двухместной функции  $I_2^1$  следующим образом:

$$\begin{bmatrix}
f_k(0) &= k \\
f_k(t+1) &= I_2^1(f_k(t),t)
\end{bmatrix}$$

(4) Эта функция получается по схеме примитивной рекурсии из одноместной функции  $I_1^1(x)$  и трехместной функции  $s(I_3^1(z,y,x))$ :

$$\begin{bmatrix}
x+0 & = I_1^1(x) \\
x+(t+1) & = s(I_3^1(x+t,t,x))
\end{bmatrix}$$

(5) Эта функция получается по схеме примитивной рекурсии из одноместной функции o(x) и трехместной функции  $I_3^1(z,y,x) + I_3^3(z,y,x)$ :

$$\begin{bmatrix} x \cdot 0 & = & o(x) \\ x \cdot (t+1) & = & I_3^1(x \cdot t, t, x) + I_3^3(x \cdot t, t, x) \end{aligned}$$

(6) Эта функция получается по схеме примитивной рекурсии из 0-местной функции 0 и 2-местной функции  $I_2^2(z,y)$ :

$$\begin{bmatrix}
0 - 1 & = 0 \\
(t + 1) - 1 & = I_2^2(x - t, t)
\end{bmatrix}$$

(7) Эта функция получается по схеме примитивной рекурсии из одноместной функции  $I^1_1(x)$  и трехместной функции  $I^1_3(z,y,x) \doteq 1$ :

Сделаем одно замечание по поводу доказательства. Рассмотрим, например, пункт 5. В этом пункте приведена схема примитивной рекурсии, которая задает двухместную функцию  $f(x,y) = x \cdot y$ . Согласно определению примитивной рекурсии, в этой схеме должны участвовать две функции q и h, такие что число аргументов у функции q на единицу меньше, чем у f, а у h, наоборот, на единицу больше. Соответственно, в доказательстве  $g(x) = I_1^1(x)$  и  $h(z, y, x) = I_3^1(z, y, x) + I_3^3(z, y, x)$ . Функция g базисная. Функция h получается суперпозицией из двухместной функции сложения, примитивную рекурсивность которой мы доказали в предыдущем пункте, и двух трехместных базисных функций  $I_3^1$  и  $I_3^3$ . Таким образом, все формальности соблюдены. Можно было бы написать просто h(z, y, x) = z + x и это выглядело бы более понятно, однако такая запись формально является некорректной. Дело в том, что если мы хотим получить трехместную функцию при помощи суперпозиции из двухместной функции, то у нас в этой суперпозиции должны участвовать еще две трехместные функции, причем каждая из них должна быть функцией от тех же самых аргументов, что и функция, которую мы хотим получить. А в записи z+x вместе с двухместной функцией сложения участвуют две одноместные функции  $f_1(z) = z$  и  $f_2(x) = x$ , причем от разных аргументов, и получается запись, формально не имеющая никакого смысла. Тем не менее, запись z+x выглядит более понятной и лучше отражает смысл функции h, а восстановить по ней формально корректную запись  $I_3^1(z,y,x)+I_3^3(z,y,x)$  не составляет труда. В связи с этим мы больше не будем загромождать изложение обилием лишних формальностей и если нам потребуется, например, записать схему примитивной рекурсии из пункта 5, мы вместо строгой с формальной точки зрения записи, приведенной в доказательстве предложения 1, будем писать просто

$$\begin{bmatrix} x \cdot 0 & = 0 \\ x \cdot (t+1) & = x \cdot t + x, \end{bmatrix}$$

считая, что восстановить по этой записи правильную запись не составляет труда.

**Предложение 2** Пусть  $f(y, x_1, ..., x_k)$  — примитивно рекурсивная функция. Тогда следующие функции являются примитивно рекурсивными:

- 1.  $\operatorname{Sum}_f(y, x_1, \dots, x_k) = \sum_{i=0}^y f(i, x_1, \dots, x_k);$
- 2.  $\operatorname{Prod}_f(y, x_1, \dots, x_k) = \prod_{i=0}^y f(i, x_1, \dots, x_k).$

 $\mathcal{A}$ оказательство. (1) Функция  $\operatorname{Sum}_f$  задается следующей схемой примитивной рекурсии:

$$\begin{cases}
Sum_f(0, x_1, ..., x_k) &= f(0, x_1, ..., x_k) \\
Sum_f(t+1, x_1, ..., x_k) &= Sum_f(t, x_1, ..., x_k) + f(s(t), x_1, ..., x_k)
\end{cases}$$

(2) Функция  $\operatorname{Prod}_f$  задается схемой примитивной рекурсии:

$$\begin{bmatrix}
\operatorname{Prod}_f(0, x_1, \dots, x_k) &= f(0, x_1, \dots, x_k) \\
\operatorname{Prod}_f(t+1, x_1, \dots, x_k) &= \operatorname{Prod}_f(t, x_1, \dots, x_k) \cdot f(s(t), x_1, \dots, x_k)
\end{bmatrix}$$

Для  $k \in \mathbb{N}$  под k-местным предикатом мы понимаем функцию из  $\mathbb{N}^k$  в двухэлементное множество {истина, ложь}. Мы говорим, что k-местный предикат  $P(x_1,\ldots,x_k)$  истинен на наборе  $m_1,\ldots,m_k \in \mathbb{N}$ , если значение  $P(m_1,\ldots,m_k)$  равно истине. В противном случае мы говорим, что предикат ложен на этом наборе. В качестве примера приведем двух-местный предикат  $P(x,y) = x \leq y$ . Этот предикат истинен на наборе 2, 3 (то есть P(2,3) истинно) и ложен на наборе 5, 4 (P(5,4) ложно). Вообще, для  $m,n \in \mathbb{N}$  P(m,n) истинно тогда и только тогда, когда  $m \leq n$ .

Под конъюнкцией предикатов  $P(x_1,\ldots,x_k)$  и  $Q(x_1,\ldots,x_k)$  (обозначается  $P(x_1,\ldots,x_k)$  &  $Q(x_1,\ldots,x_k)$ ) мы понимаем предикат, который истинен на наборе  $x_1,\ldots,x_k$  тогда и только тогда, когда на этом наборе истинны сразу оба предиката: P и Q. Под дизъюнкцией предикатов  $P(x_1,\ldots,x_k)$  и  $Q(x_1,\ldots,x_k)$  (обозначается  $P(x_1,\ldots,x_k) \vee Q(x_1,\ldots,x_k)$ ) понимается предикат, который истинен на наборе  $x_1,\ldots,x_k$  тогда и только тогда, когда на этом наборе истиннен хотя бы один из этих двух предикатов. Отрицанием предиката  $P(x_1,\ldots,x_k)$  (обозначается  $\neg P(x_1,\ldots,x_k)$ ) называется предикат, который истинен на наборе  $x_1,\ldots,x_k$  тогда и только тогда, когда предикат P ложен на этом наборе. Если  $P(x_1,\ldots,x_k) - k$ -местный предикат, то под его характеристической функцией (обозначается  $\chi_P$ ) понимается всюду определенная функция от k аргументов, такая что

$$\chi_P(x_1, \dots, x_k) = \begin{cases} 1, & \text{если } P(x_1, \dots, x_k) \text{ истинно} \\ 0, & \text{если } P(x_1, \dots, x_k) \text{ ложно,} \end{cases}$$

**Определение 15** Предикат  $P(x_1, ..., x_k)$  называется *примитивно рекурсивным*, если его характеристическая функция является примитивно рекурсивной.

**Предложение 3** Следующие функции и предикаты являются примитивно рекурсивными:

1. Одноместная функция

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

2. Одноместная функция

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

- 3. Двухместный предикат " $x \leq y$ ";
- 4. Двухместный предикат "x = y".

Доказательство. (1)  $\overline{sg}(x) = 1 - x$ .

- (2)  $\operatorname{sg}(x) = 1 \overline{\operatorname{sg}}(x)$ .
- (3) Для P(x,y)= " $x\leqslant y$ "  $\chi_P(x,y)=\mathrm{sg}(s(y)-x)$ . (4) Для P(x,y)= "x=y"  $\chi_P(x,y)=\overline{\mathrm{sg}}((y-x)+(x-y))$ .  $\square$

## Предложение 4 Справедливы следующие утверждения:

- 1. конъюнкция и дизъюнкция примитивно рекурсивных предикатов являются примитивно рекурсивными предикатами;
- 2. отрицание примитивно рекурсивного предиката есть примитивно рекурсивный предикат;
- 3. если  $P(x_1,\ldots,x_k)$  k-местный примитивно рекурсивный предикат и  $f_1(x_1,...,x_n),...,f_k(x_1,...,x_n)$  — примитивно рекурсивные функции, то n-местный предикат  $P(f_1(x_1,\ldots,x_n),\ldots,f_k(x_1,\ldots,x_n))$ является примитивно рекурсивным.

Доказательство. (1)  $\chi_{P\&Q}(x_1,\ldots,x_k)=\chi_P(x_1,\ldots,x_k)\cdot\chi_Q(x_1,\ldots,x_k)$  и  $\chi_{P\lorQ}(x_1,\ldots,x_k)=\mathrm{sg}(\chi_P(x_1,\ldots,x_k)+\chi_Q(x_1,\ldots,x_k)).$  (2)  $\chi_{\neg P}(x_1,\ldots,x_k)=\overline{\mathrm{sg}}(\chi_P(x_1,\ldots,x_k)).$ 

$$(2) \chi_{\neg P}(x_1, \dots, x_k) = \operatorname{sg}(\chi_P(x_1, \dots, x_k)).$$

$$(3) \text{ Пусть } Q(x_1, \dots, x_n) = P(f_1(x_1, \dots, x_n), \dots, f_k(x_1, \dots, x_n)). \text{ Тогда}$$

$$\chi_Q(x_1, \dots, x_n) = \chi_P(f_1(x_1, \dots, x_n), \dots, f_k(x_1, \dots, x_n)). \square$$

**Следствие 3** Двухместные предикаты "x < y" и " $x \neq y$ " являются примитивно рекурсивными.

Доказательство. "
$$x \neq y$$
" = ¬" $x = y$ " и " $x < y$ "=" $x \leqslant y$ " & " $x \neq y$ ".  $\square$ 

Пусть  $P(y, x_1, \dots, x_k) - (k+1)$ -местный предикат. Определим функцию от k+1 аргумента  $f(y, x_1, ..., x_k)$  следующим образом:

$$f(y,x_1,\dots,x_k) = \begin{cases} i_0, & \text{если } i_0 \leqslant y, P(i_0,x_1,\dots,x_k) \text{ истинно и для} \\ & \text{всех } i < i_0 \ P(i,x_1,\dots,x_k) \text{ ложно} \\ y+1, & \text{если не существует } i \leqslant y, \text{ такого что} \\ & P(i,x_1,\dots,x_k) \text{ истинно.} \end{cases}$$

Другими словами, значение  $f(y, x_1, ..., x_k)$  равно минимальному числу  $i \leq y$ , для которого предикат  $P(i, x_1, \dots, x_k)$  истинен или числу y + 1, если такого i не существует. Определенную таким образом функцию f мы будем обозначать  $(\mu i \leq y) P(i, x_1, \dots, x_k)$  и говорить, что она получается из предиката P при помощи ограниченной минимизации.

**Предложение 5** Если  $P(y, x_1, \ldots, x_k)$  — примитивно рекурсивный предикат, то функция  $(\mu i \leq y)P(i, x_1, \ldots, x_k)$  примитивно рекурсивна.

Доказательство. Введем вспомогательную функцию  $g(j,x_1,\ldots,x_k)=\prod_{i=0}^j\chi_{\neg P}(i,x_1,\ldots,x_k)$ . По предложениям 2 и 4 она примитивно рекурсивна. Остается заметить, что  $(\mu i\leqslant y)P(i,x_1,\ldots,x_k)=\sum_{j=0}^yg(j,x_1,\ldots,x_k)$ .  $\square$ 

**Предложение 6** Следующие функции и предикаты являются примитивно рекурсивными:

- 1. одноместная функция x!;
- 2. двухместная функция  $x^{y}$  (считаем, что  $0^{0} = 1$ );
- 3. двухместная функция

$$\operatorname{div}(x,y) = \begin{cases} [x/y], & \text{если } y > 0 \\ x, & \text{если } y = 0; \end{cases}$$

4. двухместная функция

$$\operatorname{rest}(x,y) = \begin{cases} \operatorname{остаток} \ \operatorname{ot} \ \operatorname{деления} \ x \ \operatorname{нa} \ y, & \operatorname{если} \ y > 0 \\ x, & \operatorname{если} \ y = 0; \end{cases}$$

5. двухместная функция

$$\log(x,y) = \begin{cases} \text{наибольшему } k, \text{ такому} \\ \text{что } y \text{ делится на } x^k, & \text{если } y > 0 \text{ и } x > 1 \\ y, & \text{если } x = 1 \\ 0, & \text{если } x = 0 \text{ или } y = 0; \end{cases}$$

- 6. одноместный предикат P(x), истинный тогда и только тогда, когда x простое число;
- 7. одноместная функция p, такая что для  $i \in \mathbb{N}$  p(i) i-ое простое число  $(p(0)=2,\,p(1)=3,\,p(2)=5$  и т. д.).

 $\mathcal{A}$ оказательство. (1) Функция x! задается следующей схемой примитивной рекурсии:

$$\begin{bmatrix}
0! & = 1 \\
(t+1)! & = t! \cdot s(t)
\end{bmatrix}$$

(2) Функция  $x^y$  задается следующей схемой примитивной рекурсии:

$$\begin{bmatrix} x^0 & = 1 \\ x^{t+1} & = x^t \cdot x \end{bmatrix}$$

- (3)  $\operatorname{div}(x, y) = (\mu i \le x)(x < i \cdot y) 1.$
- (4)  $\operatorname{rest}(x, y) = x y \cdot \operatorname{div}(x, y)$ .
- (5)  $\log(x, y) = (\mu i \leq y)(\text{rest}(y, x^i) \neq 0) 1.$
- (6)  $P(x) = "(\mu i \le x)(\operatorname{rest}(x, i+2) = 0) + 2 = x".$
- (7) Функция p(x) задается следующей схемой примитивной рекурсии:

$$\begin{bmatrix}
p(0) &= 2 \\
p(t+1) &= (\mu i \leq p(t)! + 1)("i > p(t)" \& P(i)),
\end{bmatrix}$$

ибо чему бы ни было равно x, в промежутке [x+1,x!+1] обязательно найдется хотя бы одно простое число.  $\square$ 

Для дальнейших рассмотрений нам понадобится следующая двухместная функция:

$$c(x,y) = \frac{(x+y)^2 + 3x + y}{2}.$$

Легко заметить, что число, стоящее в числителе дроби, будет четным для любых натуральных x и y, так что функция c принимает только целые значения.

## Предложение 7 Справедливы следующие утверждения:

- 1. функция c(x, y) примитивно рекурсивна;
- 2. функция c взаимно однозначно отображает  $\mathbb{N}^2$  на  $\mathbb{N}$ ;
- 3. если  $x_1 \leqslant x_2$  и  $y_1 \leqslant y_2$ , то  $c(x_1, y_1) \leqslant c(x_2, y_2)$ ;
- 4. для любых  $x, y \in \mathbb{N}$   $x \leq c(x, y)$  и  $y \leq c(x, y)$ .

Доказательство. Пункты 1, 3 и 4 очевидны. Докажем пункт 2. Сначала покажем, что если  $x_1 \neq x_2$  или  $y_1 \neq y_2$ , то  $c(x_1, y_1) \neq c(x_2, y_2)$ . Пусть  $x_1 \neq x_2$  или  $y_1 \neq y_2$ . Возможны два случая.

- 1.  $x_1+y_1=x_2+y_2$ . Тогда  $x_1\neq x_2$ . Имеем  $c(x_1,y_1)-c(x_2,y_2)=x_1-x_2\neq 0$ .
- 2.  $x_1+y_1 \neq x_2+y_2$ . Без ограничения общности можно считать, что  $x_1+y_1 < x_2+y_2$ . Заметим, что для любых  $x,y \in \mathbb{N}$   $(x+y)(x+y+1) \leqslant 2c(x,y) < (x+y+1)(x+y+2)$ . Имеем  $2c(x_1,y_1) < (x_1+y_1+1)(x_1+y_1+2) \leqslant (x_2+y_2)(x_2+y_2+1) \leqslant 2c(x_2,y_2)$ .

Осталось доказать, что для любого  $z \in \mathbb{N}$  существуют натуральные числа x и y, такие что z = c(x,y). Установим справедливость этого утверждения индукцией по z. Для z = 0 имеем 0 = c(0,0). Пусть теперь для некоторых x и y z = c(x,y). Тогда если  $y \neq 0$ , то z + 1 = c(x + 1, y - 1), а если y = 0, то z + 1 = c(0, x + 1).  $\square$ 

В связи с пунктом 2 предложения 7 для любого натурального числа z существуют единственные x и y, такие что z=c(x,y). Обозначим x, который однозначно находится по z, через l(z), а y — через r(z).

**Предложение 8** Одноместные функции l(z) и r(z) примитивно рекурсивны.

Доказательство. Введем вспомогательную двухместную функцию  $\alpha(z,y)=(\mu i\leqslant z)(z\leqslant c(i,y)).$  Тогда, по пунктам 3 и 4 предложения 7, имеем  $r(z)=(\mu i\leqslant z)(c(\alpha(z,i),i)=z)$  и  $l(z)=(\mu i\leqslant z)(c(i,r(z))=z).$  Действительно, при каждом фиксированном y  $c(0,y)< c(1,y)<\dots$  и, следовательно, почти для всех i  $z\leqslant c(i,y).$  Значение  $\alpha(z,y)$  равно минимальному такому i, поскольку оно не превосходит z Для этого i неравенство  $z\leqslant c(i,y)$  превратится в равенство в том и только в том случае, если y=r(z). Таким образом, чтобы вычислить r(z), достаточно найти (очевидно, единственный) y, для которого  $z=c(\alpha(z,y),y).$   $\square$ 

Настала пора отойти на время от формальных рассмотрений и снова поговорить об основных понятиях нашего курса: о конструктивных пространствах и вычислимых функциях. В определении 4 мы обозначили общее понятие вычислимой функции из одного конструктивного

пространства в другое. В определениях 9 и 10 мы задали ограничение этого понятия для частного случая функций, отображающих  $\Sigma^*$  в  $\Sigma^*$ . Мы также увидели, что общий случай сводится к частному и что для того, чтобы изучать вычислимые функции, отбражающие одно конструктивное пространство в другое, достаточно изучать вычислимые функции из  $\Sigma^*$  в  $\Sigma^*$ . Однако редукция к функциям из  $\Sigma^*$  в  $\Sigma^*$  — это не единственный способ изучать функции из конструктивного пространства  $\mathcal{K}_1$  в конструктивное пространство  $\mathcal{K}_2$ . Для некоторых  $\mathcal{K}_1$  и  $\mathcal{K}_2$  возможны и другие подходы.

Остановимся на случае, когда  $\mathcal{K}_2 = \mathbb{N}$ , а  $\mathcal{K}_1$  равно одному из следующих множеств:  $\mathbb{N}^0, \mathbb{N}^1, \mathbb{N}^2, \dots$  Для  $\mathcal{K}_1 = \mathbb{N}^k$  функция из  $\mathcal{K}_1$  в  $\mathcal{K}_2$  — это функция от k натуральных аргументов, которая принимает натуральные значения. По определению 4 такая функция будет вычислимой, если существует алгоритм, который по записи набора аргументов дает запись значения функции.

Речь опять зашла об описаниях. Существует множество традиционных способов задания натуральных чисел и последовательностей натуральных чисел в виде слов конечного алфавита. Например, можно зафиксировать алфавит  $\{0,\ldots,9,'',''\}$  и задавать последовательность  $n_1,\ldots,n_k$ , записывая через запятую числа из этой последовательности в десятичной системе счисления. Можно вместо десятичной системы счисления взять, например, двоичную или любую другую, а вместо запятой использовать какой-либо иной разделитель. Все это малосущественно, поскольку любой "естественный" способ задания конечных последовательностей натуральных чисел алгоритмически сводится к описанному выше  $^{40}$ . Примем один из способов задания натуральных чисел и последовательностей натуральных чисел и зафиксируем его.

При выборе любого "естественного" способа все базисные функции оказываются вычислимыми. То же самое можно сказать и про функции, которые получаются из вычислимых при помощи суперпозиции или примитивной рекурсии. Действительно, рассмотрим суперпозицию  $h(x_1,\ldots,x_k)=f(g_1(x_1,\ldots,x_k),\ldots,g_n(x_1,\ldots,x_k))$ . Если есть алгоритмы для вычисления  $g_i$ -ых и функции f, то для любого набора натуральных чисел  $m_1,\ldots,m_k$  можно последовательно вычислить значения  $g_i$ -ых от этого

 $<sup>^{40}</sup>$ При желании можно рассматривать алгоритмическую сводимость какого-либо способа задания натурального числа к десятичной записи (и наоборот) как определение "естественности" этого способа.

набора, а потом подставить полученные значения и вычислить от них функцию f. То, что получится, будет значением функции h на исходном наборе. Эту процедуру можно рассматривать как описание алгоритма для вычисления функции h. Если же функция f задается по схеме примитивной рекурсии

$$\begin{bmatrix}
f(0, x_1, \dots, x_k) & = & g(x_1, \dots, x_k) \\
f(t+1, x_1, \dots, x_k) & = & h(f(t, x_1, \dots, x_k), t, x_1, \dots, x_k),
\end{bmatrix}$$

а алгоритмы для вычисления функций g и h известны, то алгоритм вычисления функции f от набора аргументов  $n, m_1, \ldots, m_k$  будет следующим: надо, пользуясь схемой примитивной рекурсии и алгоритмами для вычисления g и h, построить конечную последовательность чисел  $f(0, m_1, \ldots, m_k), f(1, m_1, \ldots, m_k), \ldots, f(n, m_1, \ldots, m_k)$  длины n+1, а затем взять последний элемент этой последовательности.

Таким образом, можно утверждать, что каждая примитивно рекурсивная функция будет удовлетворять определению 4, то есть будет вычислимой. Обратное, к сожалению, не верно. Одна из причин заключается в том, алгоритмы, как мы уже отмечали, могут "зацикливаться" и не давать никакого результата при некоторых входных данных, а каждая примитивно рекурсивная функция является всюду определенной. Но есть и другие, более существенные причины<sup>41</sup>. В связи с этим в дополнение к операторам суперпозиции и примитивной рекурсии нам понадобится еще один оператор.

Пусть  $k \in \mathbb{N}$  и  $g(y, x_1, \dots, x_k) - (k+1)$ -местная частичная функция. Мы говорим, что k-местная частичная функция получается из функции g при помощи muhumusauuu, если она задается следующим образом:

$$f(x_1,\dots,x_k) = \begin{cases} i_0, & \text{если } g(i_0,x_1,\dots,x_k) = 0 \text{ и для всех} \\ & i < i_0 \ g(i,x_1,\dots,x_k) \text{ определено и} \\ & \text{ не равно нулю;} \\ & \text{ не определено,} & \text{ если не существует } i_0 \in \mathbb{N}, \text{ удовлет-} \\ & \text{ воряющего первому условию.} \end{cases}$$

Если  $f(x_1,\ldots,x_k)$  получается из  $g(y,x_1,\ldots,x_k)$  при помощи минимизации, то мы пишем  $f(x_1,\ldots,x_k)=\mu y(g(y,x_1,\ldots,x_k)=0)$ . Минимизация,

 $<sup>^{41}</sup>$ Можно доказать, что существует рекурсивная функция (определение см. ниже), которая не является примитивно рекурсивной.

в отличии от суперпозиции и примитивной рекурсии, может давать из всюду определенной функции функцию, которая не является таковой. Например, функция обычной разности x-y, неопределенная при x < y, задается следующей формулой:  $x-y=\mu z(((y+z)\dot{-}x)+(x\dot{-}(y+z))=0).$  Отметим еще такой момент: наличие y, для которого  $g(y,x_1,\ldots,x_k)=0$  еще не достаточно для того, чтобы  $f(x_1,\ldots,x_k)$  была определена; требуется, чтобы для такого y при всех  $i\leqslant y$  значение  $g(i,x_1,\ldots,x_k)$  тоже являлось определенным. Например, если  $g(0)=1,\ g(1)$  не определено и g(2)=0, то 0-местная функция  $\mu y(g(y)=0)$  принимает неопределенное значение.

Можно утверждать, что если существует алгоритм для вычисления функции  $g(y,x_1,\ldots,x_k)$ , то функция  $\mu y(g(y,x_1,\ldots,x_k)=0)$  тоже вычислима при помощи алгоритма. Этот алгоритм следующий: мы последовательно вычисляем значения  $g(0,x_1,\ldots,x_k),g(1,x_1,\ldots,x_k),\ldots$  и ждем, когда в этой последовательности появится ноль; как только он появился, мы объявляем номер первого нулевого члена последовательности значением функции  $f(x_1,\ldots,x_k)$ . Если нуля так и не появится, то алгоритм никогда не закончит свою работу и значение  $f(x_1,\ldots,x_k)$  окажется неопределенным. Это может произойти в двух случаях:

- 1. Для любого i значение  $g(i, x_1, \ldots, x_k)$  определено и не равно нулю. Тогда алгоритм будет вычислять все новые и новые члены последовательности, но так никогда и не остановится;
- 2. для некоторого  $i_0$  значение  $g(i_0, x_1, \ldots, x_k)$  не определено, а для всех  $i < i_0$  это значение не равно нулю. Тогда алгоритм "зависнет" на первом таком  $i_0$ , вычисляя  $g(i_0, x_1, \ldots, x_k)$ , и никогда не закончит свою работу, так и не вычислив, чему равны остальные члены последовательности.

Определение оператора минимизации построено так, чтобы были учтены обе эти возможности.

Определение 16 Функция f называется *частично рекурсивной*, если существует конечная последовательность частичных функций  $f_1, \ldots, f_n$ , такая что  $f_n = f$  и для каждого  $1 \le i \le n$  функция  $f_i$  либо базисная, либо получается из предыдущих членов последовательности при помощи суперпозиции, примитивной рекурсии или минимизации.

**Определение 17** Функция называется  $peкурсивной^{42}$ , если она частично рекурсивна и всюду определена.

Из приведенных выше замечаний следует, что каждая частично рекурсивная функция является вычислимой в самом общем смысле (согласно определению 4). Обратное можно поставить под сомнение. Однако до сих пор никому так и не удалось придумать пример функции, для которой существует алгоритм вычисления, но которая не является частично рекурсивной. Более того, все известные на данный момент методы построения алгоритмов сводятся к трем перечисленным выше приемам: суперпозиции, примитивной рекурсии и минимизации. В связи с этим все математики в настоящее время принимают

**Тезис Черча**: Всякая вычислимая (в общем смысле, согласно определению 4) частичная функция из  $\mathbb{N}^k$  в  $\mathbb{N}$  является частично рекурсивной.

Как и в случае с тезисом Тьюринга, тезис Черча нельзя доказать, но им можно пользоваться. Например, если известен алгоритм вычисления функции, описанный на одном из естественных языков<sup>43</sup> или в виде программы для компьютера, то можно считать, что эта функция частично рекурсивна. В будущем мы тоже иногда будем так поступать. А пока попробуем выяснить, в каких отношениях тезис Черча находится с введенным ранее тезисом Тьюринга.

Нам понадобится одно вспомогательное устройство, известное как машина Шенфилда. Машина Шенфилда состоит из бесконечного списка пронумерованных регистров: R0, R1,..., каждый из которых может содержать произвольное натуральное число. Машина работает по программе, в ходе выполнения которой содержимое регистров может меняться. Программа представляет собой пронумерованный список команд, каждая из которых имеет один из следующих двух видов:

 $<sup>^{42}</sup>$ В литературе также встречается термин "общерекурсивная функция", который обозначает то же самое, что и термин "рекурсивная функция". Первоначально общерекурсивной функцией называлась такая частично рекурсивная функция, для которой все члены определяющей последовательности  $f_1, \ldots, f_n = f$  являются всюду определенными. Впоследствии было доказано, что функция является общерекурсивной тогда и только тогда, когда она рекурсивна, после чего эти термины стали употреблятся как синонимы.

<sup>&</sup>lt;sup>43</sup>Например, на русском.

- 1. INC i для  $i \in \mathbb{N}$ . При исполнении этой команды машина увеличивает содержимое i-го регистра на 1, после чего управление переходит к следующей команде.
- 2. DEC i, n для  $i, n \in \mathbb{N}$ . Характер действий, совершаемых машиной при исполнении этой команды, зависит от содержимого регистра Ri. Если это содержимое равно нулю, то никаких изменений в регистрах не происходит, а управление передается на следующую по списку команду. Если оно не равно нулю, то содержимое i-го регистра уменьшается на единицу, после чего управление передается на команду с номером n.

В ходе своей работы машина выполняет последовательно команду за командой, меняя при этом состояние регистров. На каждом шаге исполняется та команда, на которую в данный момент передано управление. Перед началом работы управление передается на команду с номером 1, то есть на первую команду программы. Если в ходе работы управление передается на несуществующую команду, то выполнение программы прекращается и машина останавливается.

Например, рассмотрим следующую короткую программу:

1: INC 1 2: DEC 0,1 3: DEC 1,4

Работая по этой программе, машина будет исполнять по очереди первую и вторую команды, увеличивая содержимое первого регистра и уменьшая содержимое нулевого до тех пор, пока содержимое нулевого регистра не окажется равным нулю. После этого управление будет передано на команду с номером 3, исполнив которую, машина уменьшит содержимое первого регистра на единицу и перейдет к команде с номером 4. Поскольку такой команды не существует, то машина остановится. После того, как программа закончит работу, в регистрах по сравнению с их первоначальным состоянием произойдут следующие изменения: к содержимому первого регистра прибавится содержимое нулевого, а содержимое нулевого регистра станет равным нулю.

Заметим, что хотя машина содержит бесконечно много регистров, в каждой программе упомянуто лишь конечное их число. Регистры, не упомянутые в программе, не оказывают никакого влияния на ход работы машины по этой программе и на результат работы.

Пусть P — программа для машины Шенфилда и  $k \in \mathbb{N}$ . Определим частичную k-местную функцию  $f_P^k(x_1,\ldots,x_k)$ . Пусть состояние регистров машины Шенфилда следующее: в регистрах  $R1,R2,\ldots,Rk$  записаны числа  $x_1,\ldots,x_k$  соответственно, а во всех остальных регистрах содержатся нули. Пусть машина Шенфилда начинает свою работу по программме P из этого состояния регистров. Тогда полагаем  $f_P^k(x_1,\ldots,x_k)$  равным содержимому регистра R0 после окончания работы программы, если машина заканчивает работу; в противном случае полагаем, что  $f_P^k(x_1,\ldots,x_k)$  не определено.

Пусть  $k \in \mathbb{N}$  и P — программа<sup>44</sup>. Расширим список команд для машины Шенфилда, введя так называемые макрокоманды, то есть команды вида  $P(i_1,\ldots,i_k)\to j$ . При выполнении такой макрокоманды машина ведет себя следующим образом: она берет набор из k натуральных чисел, соответсвующих значениям регистров  $Ri_1,\ldots,Ri_k$  и пытается вычислить значение функции  $f_P^k$  на этом наборе. Если это значение определено, то она заносит его в регистр Rj, не меняя при этом содержимое других регистров, и переходит к исполнению следующей команды. Если же значение функции не определено, то машина "зависает", не выполняя при этом никаких команд, но и не останавливаясь. Можно сказать, что исполнение команды  $P(i_1,\ldots,i_k)\to j$  аналогично вызову подпрограммы, вычисляющей функцию  $f_P^k$ .

Мы говорим, что программы  $Q_1$  и  $Q_2$  эквивалентны, если для любого  $k\in\mathbb{N}$  частичные функции  $f_{Q_1}^k$  и  $f_{Q_2}^k$  равны.

**Теорема 6** Для каждой программы, содержащей макрокоманды, существует эквивалентная ей программа, состоящая только из стандартных команд.

Доказательство. Покажем, как по программе, содержащей макрокоманды, получить эквивалентную ей программу, число макрокоманд в которой на единицу меньше. Применяя эту процедуру к любой программе с макрокомандами необходимое число раз, можно получить эквивалентную ей программу, не содержащую макрокоманд.

Пусть программа Q состоит из команд  $C_1, C_2, \ldots, C_m$ , причем для некоторого  $m_0 \leqslant m$   $C_{m_0}$  — это команда  $P(i_1, \ldots, i_k) \to j$ . Пусть программа P состоит из n стандартных команд  $D_1, \ldots, D_n$  и N — номер наибольшего регистра, который упоминается в программе P. Напишем

 $<sup>^{44}\</sup>mathrm{Cocтoящая}$ из стандартных команд, то есть команд вида INC i и DEC i,n.

программу Q', которая эквивалентна Q и состоит из N+8k+n+m+4 команд  $F_1,\ldots,F_{N+8k+n+m+4}$ . Выберем число M, которое больше чем номер любого регистра, упоминаемого в программе Q. Необходимо для каждого  $1 \le s \le N+8k+n+m+4$  задать команду  $F_s$ . Возможны следующие случаи:

- 1.  $s < m_0$ . Если  $C_s$  это либо команда вида INC i, либо команда DEC i, l для  $l \le m_0$ , либо макрокоманда, то полагаем  $F_s = C_s$ . В противном случае  $C_s$  это команда DEC i, l для  $l > m_0$ . Полагаем  $F_s$  равным DEC i, l + N + 8k + n + 4.
- 2.  $s = m_0 + t$  для  $t \leq N$ . Полагаем  $F_s$  равным команде DEC M + t, s.
- 3.  $m_0 + N + 8t < s \le m_0 + N + 8t + 8$  для  $0 \le t < k$ . Соответствующие 8 команд задаются следующим списком:

```
\begin{array}{lll} m_0 + N + 8t + 1 : & \text{INC} & 0 \\ m_0 + N + 8t + 2 : & \text{DEC} & 0, m_0 + N + 8t + 5 \\ m_0 + N + 8t + 3 : & \text{INC} & M + t + 1 \\ m_0 + N + 8t + 4 : & \text{INC} & M \\ m_0 + N + 8t + 5 : & \text{DEC} & i_{t+1}, m_0 + N + 8t + 3 \\ m_0 + N + 8t + 6 : & \text{INC} & i_{t+1} \\ m_0 + N + 8t + 7 : & \text{DEC} & M, m_0 + N + 8t + 6 \\ m_0 + N + 8t + 8 : & \text{DEC} & i_{t+1}, m_0 + N + 8t + 9 \end{array}
```

- 4.  $s=m_0+N+8k+t$  для  $1\leqslant t\leqslant n$ . Если  $D_t$  это команда INC i для некоторого i, то полагаем  $F_s$  равным INC M+i. Если  $D_t$  это команда DEC i,l для некоторого i и некоторого  $1\leqslant l\leqslant n$ , то полагаем  $F_s$  равным DEC  $M+i,m_0+N+8k+l$ . Наконец, если  $D_t$  это команда DEC i,l для  $l\notin\{1,\ldots,n\}$ , то полагаем  $F_s$  равным команде DEC  $M+i,m_0+N+8k+n+1$ .
- 5.  $m_0 + N + 8k + n < s \le m_0 + N + 8k + n + 4$ . Соответствующие 4 команды задаются следующим списком:

```
m_0 + N + 8k + n + 1: DEC j, m_0 + N + 8k + n + 1

m_0 + N + 8k + n + 2: INC j

m_0 + N + 8k + n + 3: DEC M, m_0 + N + 8k + n + 2

m_0 + N + 8k + n + 4: DEC j, m_0 + N + 8k + n + 5
```

6.  $m_0 + N + 8k + n + 4 < s \le m + N + 8k + n + 4$ . Пусть  $t = s - m_0 - N - 8k - n - 4$ . Действуем почти так же, как в первом случае. Если  $C_t$  — это либо команда вида INC i, либо команда DEC i, l для  $l \le m_0$ , либо макрокоманда, то полагаем  $F_s = C_t$ . В противном случае  $C_t$  — это команда DEC i, l для  $l > m_0$ . Полагаем  $F_s$  равным DEC i, l + N + 8k + n + 4.

Описание программы Q' закончено. Из этого описания видно, что программа Q' содержит на одну макрокоманду меньше, чем программа Q. Действительно, команда с номером  $m_0$  заменена списком стандартных команд с номерами  $m_0, \ldots, m_0 + N + 8k + n + 4$ . В приведенном выше описании в пунктах 1 и 6 описана модификация команд программы Q с номерами, не равными  $m_0$ . Те команды, номера которых больше  $m_0$ , из-за вставки меняют свои номера и поэтому в некоторых командах DEC i, n требуется изменить n на новый номер. Команды, которые вставляются в программу вместо команды с номером  $m_0$ , описаны в пунктах 2 – 5 и располагаются следующим образом. Сначала идут команды, описанные в пункте 2, которые обнуляют регистры с M-го по M+N-ый. Затем идут команды, описанные в пункте 3, которые присвают регистрам  $M+1,\ldots,M+k$  значения, содержащиеся в регистрах  $i_1,\ldots,i_k$ . Группа из восьми команд, приведенная в пункте 3, присваивает регистру с номером M + t + 1 значение регистра  $i_{t+1}$ , не меняя значения других регистров. При этом используется тот факт, что перед выполнением этих восьми команд в регистрах с номерами M и M+t+1 содержатся нули. Команды, описанные в пункте 4 — это команды программы P со сдвинутыми номерами регистров и переходов. Исполнив эти команды, машина вычислит значение функции  $f_P^k$  от чисел, содержащихся в регистрах  $M+1, \ldots, M+k$  и поместит результат в регистр M. Наконец, 4 команды, описанные в пункте 5, помещают в регистр  $R_j$  число, содержащееся в регистре с номером M. Собирая все вместе, видим, что команды, которые мы вставили в программу Q вместо команды с номером  $m_0$ , выполняют в совокупности те же действия, что и макрокоманда  $P(i_1,\ldots,i_k) \to j^{45}$ и, следовательно, программа Q' эквивалентна Q.  $\square$ 

Скажем, что k-местная частичная функция  $f(x_1, \ldots, x_k)$  вычислима на машине Шенфилда, если  $f = f_P^k$  для некоторой программы P.

 $<sup>^{45}</sup>$ Конечно, эти команды, в отличие от макрокоманды с номером  $m_0$ , меняют значения некоторых регистров с номерами  $\geqslant M$ , однако эти регистры не упоминаются в программе Q и не оказывают никакого влияния на работу машины по этой программе.

**Теорема 7** Каждая частично рекурсивная функция вычислима на машине Шенфилда.

Доказательство. Частично рекурсивные функции получаются из базисных при помощи операторов суперпозиции, примитивной рекурсии и минимизации. Значит, достаточно доказать, что класс функций, вычислимых на машине Шенфилда, содержит все базисные функции и замкнут относительно этих трех операторов.

Базисная функция, равная константе 0, вычисляется при помощи следующей программы:

состоящей из одной команды (или даже при помощи пустой программы, не содержащей ни одной команды). Базисная функция s(x) вычисляется при помощи программы

1: INC 0 2: DEC 1,1

Наконец, базисная функция  $I_n^k(x_1,\ldots,x_n)$  вычисляется программой

1: INC 0 2: DEC k, 13: DEC 0, 4

Пусть n-местная функция  $f(x_1, \ldots, x_n)$  вычисляется программой P, а для  $1 \le i \le n$  k-местная функция  $g_i(x_1, \ldots, x_k)$  вычисляется программой  $P_i$ . Тогда суперпозиция  $h(x_1, \ldots, x_k) = f(g_1(x_1, \ldots, x_k), \ldots, g_n(x_1, \ldots, x_k))$  вычисляется при помощи следующей программы (с макрокомандами):

1: 
$$P_1(1,...,k) \to k+1$$
  
2:  $P_2(1,...,k) \to k+2$   
:  $n: P_n(1,...,k) \to k+n$   
 $n+1: P(k+1,...,k+n) \to 0$ 

Пусть k-местная функция  $g(x_1,\ldots,x_k)$  вычисляется программой P, (k+2)-местная функция  $h(z,y,x_1,\ldots,x_k)$  вычисляется программой Q и

(k+1)-местная функция  $f(t,x_1,\ldots,x_k)$  задается по схеме примитивной рекурсии:

$$\begin{bmatrix}
f(0, x_1, \dots, x_k) & = & g(x_1, \dots, x_k) \\
f(t+1, x_1, \dots, x_k) & = & h(f(t, x_1, \dots, x_k), t, x_1, \dots, x_k)
\end{bmatrix}$$

Тогда функция f вычисляется на машине Шенфилда по следующей программе:

1:  $P(2,...,k+1) \to 0$ 2: INC 0 3: DEC 0,6 4:  $Q(0,k+2,2,...,k+1) \to 0$ 5: INC k+26: DEC 1,4

Наконец, пусть  $f(x_1,\ldots,x_k)=\mu y(g(y,x_1,\ldots,x_k)=0)$  и функция g вычисляется при помощи программы P. Тогда программа

1:  $P(0,1,...,k) \rightarrow k+1$ 2: INC 0 3: DEC k+1,14: DEC 0,5

вычисляет функцию f.  $\square$ 

Вернемся к машинам Тьюринга. Напомним, что машины Тьюринга работают со словами конечного алфавита, в который не входят символы m, n, L, R. Для дальнейших рассмотрений зафиксируем алфавит  $\Sigma = \{0, 1\}$ . Для набора натуральных чисел  $n_1, \ldots, n_k$  через  $w(n_1, \ldots, n_k)$  обозначим слово этого алфавита, равное  $01^{n_1}01^{n_2}0\ldots 01^{n_k}$ . Дадим определение вычислимости на машине Тьюринга для числовой функции.

Определение 18 Частичная числовая k-местная функция  $f(x_1, ..., x_k)$  называется вычислимой на машине Тьюринга, если существует программа P (для машины Тьюринга), такая что для любых  $n_1, ..., n_k \in \mathbb{N}$ :

- 1. если  $f(n_1, ..., n_k)$  определено, то для некоторых  $t, l \in \mathbb{N}$  и  $v \in (\Sigma \cup \{n\})^*$  Сом $(\langle w(n_1, ..., n_k), q_1, 1 \rangle, P, t) = \langle v, q_0, l \rangle$  и число единиц в слове v равно  $f(n_1, ..., n_k)$ ;
- 2. если  $f(n_1, ..., n_k)$  не определено, то для всех  $t \in \mathbb{N}$  если  $\text{Com}(\langle w(n_1, ..., n_k), q_1, 1 \rangle, P, t) = \langle v, q, l \rangle$ , то  $q \neq q_0$ .

Другими словами, вычислимость функции f на машине Тьюринга при помощи программы P означает следующее. При данных  $n_1, \ldots, n_k \in \mathbb{N}$  мы записываем на ленту слово  $01^{n_1} \ldots 01^{n_k}$ , устанавливаем головку на первую позицию и запускаем машину работать по программе P из состояния  $q_1$ . Тогда если  $f(n_1, \ldots, n_k)$  определено, то машина через некоторое время остановится и число единиц на ленте будет равно  $f(n_1, \ldots, n_k)$ , а если это значение не определено, то машина будет работать бесконечно.

**Теорема 8** Каждая функция, вычислимая на машине Шенфилда, вычислима на машине Тьюринга.

Доказательство. Пусть  $f(x_1, ..., x_k)$  — частичная k-местная функция, вычислимая на машине Шенфилда по программе  $P^{46}$  и M — некоторое число, которое больше чем k и больше чем номер любого регистра, упоминаемого в программе P.

Ниже нам предстоит написать программу для машины Тьюринга Q, моделирующую работу машины Шенфилда по программе P. Программа Q, согласно определению программы для машины Тьюринга, для любого состояния q (из списка упомянутых в ней и отличных от  $q_0$  состояний) и любого символа  $a \in \Sigma \cup \{n,m\}$  должна содержать команду  $qa \to \ldots$  Однако некоторые команды программы, которую мы напишем, никогда не смогут исполниться, если машина начнет свою работу из состояния, задаваемого конфигурацией  $\langle w(n_1,\ldots,n_k),q_1,1\rangle$  при произвольных натуральных  $n_1,\ldots,n_k$  (то есть в случае, когда мы используем эту программу для вычисления k-местной функции). В связи с этим мы, чтобы не писать лишнего, примем следующее соглашение: если команда  $qa \to \ldots$  отсутствует в приведенном ниже списке команд, то эта команда имеет вид  $qa \to qR$ .

Пусть программа P состоит из n строк. Программа Q будет содержать команды  $qa \to \ldots$  для  $a \in \Sigma \cup \{n,m\}$  и  $q \in \{q_1,\ldots,q_{M-k+1}\} \cup \{q'_1,q'_2,q'_3,q'_4\} \cup \bigcup_{m=1}^n \{q_1^m,\ldots,q_{s(m)}^m\}$ . Мы считаем, что все упомянутые здесь состояния различны. Кроме того, примем следующее соглашение: для m>n или m=0 запись  $q_1^m$  обозначает состояние  $q'_1$ .

Способ, которым программа Q моделирует работу машины Шенфилда по программе P, заключается в следующем. Предположим, что мы используем программу P для вычисления  $f(n_1, \ldots, n_k)$ . В процессе вычисления машина Шенфилда, работая по шагам, последовательно меняет

 $<sup>^{46}</sup>P$  — программа для машины Шенфилда.

содержимое своих регистров, не затрагивая регистры с номерами, большими чем M. Пусть перед выполнением шага  $a \in \{1, 2, 3, ...\}$  содержимое регистром с номерами от 0 до M равно  $r_0^a, ..., r_M^a$  соответственно и на шаге a исполняется команда с номером m(a).

Тогда для машины Тьюринга, начинающей работу по программе Q из состояния, задаваемого конфигурацией  $\langle w(n_1,\ldots,n_k),q_1,1\rangle$ , существует последовательность шагов  $t_1 < t_2 < \ldots$  ее работы, такая что:

- 1. перед выполнением шага  $t_a$  на ленте записано слово  $1^{r_0^a}01^{r_1^a}0\dots01^{r_M^a}$ ;
- 2. перед выполнением шага  $t_a$  головка находится в состоянии  $q_1^{m(a)}$  и указывает на первую ячейку ленты;
- 3. на шагах  $t_a \leqslant t < t_{a+1}$  все состояния, которые принимает головка, принадлежат множеству  $\{q_1^{m(a)}, \dots, q_{s(m(a))}^{m(a)}\}.$

Таким образом, команды программы Q, имеющие вид  $qa \to \dots$  для  $q \in \{q_1^m, \dots, q_{s(m)}^m\}$ , моделируют исполнение команды исходной программы P под номером m, меняя содержимое ленты машины Тьюринга в соответствии с тем, как команда под номером m меняет содержимое регистров машины Шенфилда. Если при каких-то начальных данных машина Шенфилда останавливается на шаге a (то есть  $m(a) \notin \{1, \ldots, n\}$ ), то в начале шага  $t_a$  головка машины Тьюринга, работающей по программе Q, оказывается в состоянии  $q'_1$  и машина Тьюринга переходит к исполнению заключительных команд, после выполнения которых на ленте остается только содержимое нулевого регистра (значение функции, вычисленной на машине Шенфилда) и машина останавливается. Кроме заключительных команд и команд, моделирующих исполнение конкретной команды в исходной программе P, программа Q содержит так называемые подготовительные команды. Это команды, исполняемые на шагах  $t < t_1$  и приводящие содержимое ленты машины Тьюринга в соответствие с начальным состоянием регистров машины Шенфилда.

Переходим к детальному описанию команд каждой группы.

Подготовительные команды. Эти команды подготавливают ленту машины, приводя записанные на ней данные в соответствие с содержимым

регистров машины Шенфилда перед началом работы по программе P.

$$\begin{array}{lll} q_1 1 \to q_1 R & q_3 n \to q_4 0 \\ q_1 0 \to q_1 R & \dots \\ q_1 n \to q_2 0 & q_{M-k} n \to q_{M-k+1} 0 \\ q_2 0 \to q_2 R & q_{M-k+1} 0 \to q_{M-k+1} L \\ q_2 n \to q_3 0 & q_{M-k+1} 1 \to q_{M-k+1} L \\ q_3 0 \to q_3 R & q_{M-k+1} m \to q_1^1 R \end{array}$$

Исполняя эти команды, головка машины движется вправо по ленте, находясь все время в начальном состоянии  $q_1$ , пока не встанет напротив ячейки с символом n. После этого она дописывает справа M-k символов 0, перейдя в состояние  $q_{M-k+1}$ . Затем она в этом состоянии движется по ленте влево "до упора" и из самой левой ячейки передвигается вправо, переходя в состояние  $q_1^1$ . Содержимое ленты при этом преобразуется из  $w(n_1,\ldots,n_k)$  в  $1^{r_0^1}01^{r_1^1}0\ldots01^{r_M^1}$ , где  $r_i^1$  равно  $n_i$  при  $1\leqslant i\leqslant k$  и нулю в противном случае.

Команды, моделирующие выполнение команды машины Шенфилда с номером m. Возможны два случая.

1. Команда с номером m имеет вид INC i. Тогда список команд для этой группы следующий:

Исполняя эту группу команд, головка машины сначала сдвигается вправо, отсчитывая i нулей и переходя в состояние  $q_{i+1}^m$ . Затем она вставляет единицу в ту ячейку ленты, на которую указывает, сдвигая содержимое следующих ячеек на одну позицию вправо. Закончив сдвиг, она переходит в состояние  $q_{i+5}^m$ , из которого движется влево "до упора", а потом переходит на первую позицию, меняя состояние на  $q_1^{m+1}$ .

2. Команда с номером m имеет вид DEC i, l. Список команд для этой группы таков:

$$\begin{array}{lll} q_1^m 0 \to q_2^m R & q_{i+2}^m 1 \to q_{i+2}^m L & q_{i+6}^m 1 \to q_{i+3}^m R \\ q_1^m 1 \to q_1^m R & q_{i+2}^m m \to q_1^{m+1} R & q_{i+6}^m 0 \to q_{i+6}^m 1 \\ q_2^m 0 \to q_3^m R & q_{i+1}^m 1 \to q_{i+3}^m 1 & q_{i+4}^m n \to q_{i+7}^m L \\ q_2^m 1 \to q_2^m R & q_{i+3}^m 0 \to q_{i+4}^m R & q_{i+7}^m 0 \to q_{i+8}^m n \\ & \dots & q_{i+3}^m 1 \to q_{i+4}^m R & q_{i+7}^m 1 \to q_{i+8}^m n \\ q_i^m 0 \to q_{i+1}^m R & q_{i+4}^m 0 \to q_{i+5}^m L & q_{i+8}^m n \to q_{i+8}^m L \\ q_i^m 1 \to q_i^m R & q_{i+4}^m 1 \to q_{i+6}^m L & q_{i+8}^m 0 \to q_{i+8}^m L \\ q_{i+1}^m 0 \to q_{i+2}^m L & q_{i+5}^m 0 \to q_{i+3}^m R & q_{i+8}^m 1 \to q_{i+8}^m L \\ q_{i+2}^m 0 \to q_{i+2}^m L & q_{i+5}^m 1 \to q_{i+5}^m 0 & q_{i+8}^m m \to q_1^l R \end{array}$$

Исполняя эту группу команд, машина выполняет следующие действия. Сначала, как и в предыдущем случае, головка машины сдвигается вправо, отсчитывая i нулей и переходя в состояние  $q_{i+1}^m$ . Дальнейшие действия зависят от содержимого ячейки, на которую указывает головка. Если в этой ячейке находится ноль (то есть содержимое i-го регистра машины Шенфилда равно нулю), то головка машины Тьюринга переходит в состояние  $q_{i+2}^m$ , идет влево "до упора", а затем сдвигается на одну позицию вправо, переходя в состояние  $q_{i+3}^{m+1}$ . Если же головка в состоянии  $q_{i+1}^m$  указывает на ячейку, в которой записана единица, то она переходит в состояние  $q_{i+3}^m$  и затем стирает содержимое этой ячейки, сдвигая все ячейки справа от этой на одну позицию влево и постепенно перемещаясь вправо. После того, как сдвиг осуществлен, головка переходит в состояние  $q_{i+8}^m$ , находясь в котором, движется "до упора" влево, а затем сдвигается вправо на одну позицию и переходит в состояние  $q_i^l$ .

Заключительные команды. После того, как работа машины Шенфилда закончена, на ленте машины Тьюринга сначала записана группа единиц, число которых равно вычисленному значению функции, затем ноль и далее — еще нули и единицы, отображающие состояние регистров машины Шенфилда, отличных от R0. Эти лишние символы надо стереть. Для этого предназначена следующая группа команд:

$$\begin{array}{lll} q_1'1 \rightarrow q_1'R & q_2'1 \rightarrow q_3'n & q_4'n \rightarrow q_4'L \\ q_1'0 \rightarrow q_2'0 & q_3'n \rightarrow q_2'R & q_4'1 \rightarrow q_4'L \\ q_2'0 \rightarrow q_3'n & q_2'n \rightarrow q_4'L & q_4'm \rightarrow q_0R \end{array}$$

Действия, производимые этими командами, очевидны.

Из описаний следует, что все группы команд выполняют те действия, которые от них требуются. Собирая вместе все команды из различных групп, получаем программу Q, вычисляющую на машине Тьюринга функцию f.  $\square$ 

**Следствие 4** Каждая частично рекурсивная функция вычислима на машине Тьюринга.

Доказательство. Это прямое следствие теорем 7 и 8.  $\square$ 

На самом деле мы доказали чуть больше, чем требовалось. Пусть  $f(x_1,...,x_k)$  — частично рекурсивная функция и P — программа для вычисления функции f, которую мы построили в теореме 8. Если для  $n_1,\ldots,n_k\in\mathbb{N}$   $f(n_1,\ldots,n_k)$  определено и мы запустим машину Тьюринга работать по этой программе, записав на ленте слово  $w(n_1, \ldots, n_k)$ , установив головку на первую позицию и приведя ее в состояние  $q_1$ , то через некоторое время машина остановится, придя в состояние  $q_0$ , и на ленте окажется записанным некоторое слово v. Нам было бы достаточно, чтобы число единиц в слове v было равно  $f(n_1, \ldots, n_k)$ , причем эти единицы могут быть раскиданы по ленте как угодно, перемежаясь символами 0 и n. Однако у нас эти единицы идут подряд, одна за другой, начиная с первой ячейки; нулей на ленте нет, а головка находится в первой позиции. Если вспомнить определение 13, то этот результат можно интерпретировать следующим образом: мы доказали, что каждая частично рекурсивная функция является вычислимой в самом общем смысле, то есть в смысле определения 4. У нас есть два конструктивных пространства:  $\mathbb{N}^k$  и  $\mathbb{N}$ , а также функция f из первого пространства во второе. Эта функция будет вычислимой, если существует алгоритм, который по описанию набора аргументов дает описание значения функции на этих аргументах. Можно зафиксировать следующие описания: набор чисел  $n_1,\ldots,n_k$  описывается словом  $w(n_1, \ldots, n_k)$ , а натуральное число n — словом  $1^n$ . Тогда реализацией требуемого алгоритма будет программа для машины Тьюринга, для построения которой можно использовать конструкции теорем 7 и 8.

В настоящий момент мы собираемся определить кодировки конфигураций и программ машин Тьюринга. Позже мы введем общее понятие

нумерации и увидим, что кодировка — это, по сути, частный случай нумерации. А пока скажем, что кодировка объекта — это способ его задания при помощи натуральных чисел.

Кодирование конфигураций. Мы продолжаем работать с алфавитом  $\Sigma = \{0,1\}$ . Вспомним, что конфигурация — это тройка  $\langle w, q_i, l \rangle$ , где w — слово алфавита  $\Sigma \cup \{n\}$ , не заканчивающееся на  $n, q_i$  — состояние головки, а l — натуральное число (номер ячейки ленты, на которую указывает головка). Конфигурации мы будем кодировать тройками натуральных чисел  $\langle m_1, m_2, m_3 \rangle$ , где  $m_1$  — код слова  $w, m_2$  — код состояния  $q_i$ , а  $m_3$  — код числа l. Кодом числа l будет само число l (то есть  $m_3 = l$ ). Код состояния  $q_i$  — это номер этого состояния, то есть число i. Наконец, кодом слова  $w = a_1 \dots a_k$  назовем число  $p(1)^{\varepsilon_1} p(2)^{\varepsilon_2} \dots p(k)^{\varepsilon_k}$ , где для  $1 \leqslant i \leqslant k \ \varepsilon_i$  определяется следующим образом:

$$\varepsilon_i = \begin{cases} 0, & a_i = n; \\ 1, & a_i = 0; \\ 2, & a_i = 1. \end{cases}$$

Из единственности разложения каждого натурального числа на простые множители следует, что разным словам сопоставляются разные коды, то есть по коду слова можно однозначно восстановить само слово.

**Кодирование программ**. Задавая кодировку программ, мы ставим перед собой более сложную задачу, чем в предыдущем случае. Мы хотим, чтобы наша кодировка удовлетворяла следующим трем требованиям:

- 1. Код каждой программы это некоторое натуральное число (а не тройка чисел, как в предыдущем случае).
- 2. Каждая программа имеет хотя бы один код (мы допускаем, что разные числа могут кодировать одну и ту же программу).
- 3. Каждое натуральное число является кодом некоторой программы.

Для задания кодировки поступим следующим образом: опишем, как по натуральному числу восстановить программу, которую оно кодирует. Из описания будет следовать, что каждая программа получит какой-то код.

Пусть  $k \in \mathbb{N}$ . Программа, которую это число кодирует, содержит команды вида  $q_i a \to \dots$ , где a — символ алфавита  $\{0,1,n,m\}$ , а  $q_i$  — элемент множества  $\{q_0,q_1,\dots,q_s\}$  для некоторого  $s\geqslant 1$ . Значит, по коду k

мы должны уметь получать число s. Положим s=l(k)+1, рассматривая "левую часть" числа k как код для множества состояний головки. "Правую часть" r(k) мы интерпретируем как код для списка команд программы.

Сопоставим каждой паре  $\langle q_i, a \rangle$ , где  $i \leqslant s$  и  $a \in \{0, 1, n, m\}$  натуральное число  $\beta(q_i, a)$  следующим образом:  $\beta(q_i, a) = c(i, \varepsilon(a))$ , где  $\varepsilon(a)$  определяется так:

$$\varepsilon(a) = \begin{cases} 0, & a = n \\ 1, & a = 0 \\ 2, & a = 1 \\ 3, & a = m \end{cases}$$

Ясно, что разным парам будут сопоставлены разные натуральные числа. Число r(k)+1 единственным способом раскладывается на простые множители. Поступим следующим образом: показатель степени при простом числе  $p(\beta(q_i,a))$  в разложении r(k)+1 будем рассматривать как число, кодирующее правую часть команды  $q_ia \to \dots$ 

Итак, пусть t — это максимальное натуральное число, такое что  $p(\beta(q_i,a))^t$  делит r(k)+1. Соответствующая числу t команда программы, кодом которой является число k, имеет вид  $q_i a \to q_j b$ , где  $j \leqslant s$  и  $b \in \{0,1,n,L,R\}$ . "Левую часть" l(t) числа t будем интерпретировать как код состояния  $q_j$ , а "правую часть" r(t) — как код символа b. Примем следующие правила кодирования: j равно остатку от деления числа l(t) на s+1 (то есть  $j=\mathrm{rest}(l(t),s+1)$ ), а

$$b = \begin{cases} n, & \operatorname{rest}(r(t), 5) = 0 \text{ и } a \neq m \\ 0, & \operatorname{rest}(r(t), 5) = 1 \text{ и } a \neq m \\ 1, & \operatorname{rest}(r(t), 5) = 2 \text{ и } a \neq m \\ L, & \operatorname{rest}(r(t), 5) = 3 \text{ и } a \neq m \\ R, & \operatorname{rest}(r(t), 5) = 4 \text{ или } a = m \end{cases}$$

Описание кодировки закончено. Легко видеть, что каждому натуральному числу действительно сопоставляется некоторая единственная программа, кодом которой оно является. Разным натуральным числам может сопоставляться одна и та же программа, однако этот факт для нас

 $<sup>^{47}</sup>$ Имеем k=c(x,y) для некоторых единственных x и y. Число x мы называем "левой частью" числа k, а число y — "правой частью".

не является помехой. Также легко понять, что для каждой программы найдется натуральное число, которое является ее кодом. Действительно, если есть программа, написанная для множества состояний  $\{q_0,q_1\dots q_s\}$ , то ее кодом будет число c(s-1,x-1), где x вычисляется так: надо для каждой команды  $q_ia\to \dots$ , взять число  $p(\beta(q_i,a))$ , возвести его в степень t, где t— какой-либо из кодов правой части команды, а затем взять произведение этих степеней по всем командам программы.

Определим три четырехместные функции  $Step_1$ ,  $Step_2$  и  $Step_3$ . Пусть  $\langle m_1, m_2, m_3 \rangle$  — произвольная тройка натуральных чисел и n — натуральное число. Пусть P — программа, кодом которой является число n. Возможны два случая.

- 1. Тройка  $\langle m_1, m_2, m_3 \rangle$  не является кодом никакой конфигурации либо  $\langle m_1, m_2, m_3 \rangle$  код конфигурации  $\langle w, q_i, l \rangle$ , но состояние  $q_i$  не входит в число состояний, с которыми работает программа P (то есть i > l(n)+1). Полагаем  $\operatorname{Step}_1(n, m_1, m_2, m_3) = m_1$ ,  $\operatorname{Step}_2(n, m_1, m_2, m_3) = m_2$  и  $\operatorname{Step}_3(n, m_1, m_2, m_3) = m_3$ .
- 2. Тройка  $\langle m_1, m_2, m_3 \rangle$  является кодом конфигурации  $\langle w, q_i, l \rangle$  и  $i \leqslant l(n) + 1$ . Тогда определена конфигурация  $\langle v, q_j, k \rangle$ , равная  $\operatorname{Step}(\langle w, q_i, l \rangle, P)$ , которая отражает состояние машины Тьюринга, получающееся после одного шага работы машины по программе P из состояния, соответсвующего конфигурации  $\langle w, q_i, l \rangle$ . Конфигурация  $\langle v, q_j, k \rangle$  имеет код, равный  $\langle m'_1, m'_2, m'_3 \rangle$ . Полагаем  $\operatorname{Step}_1(n, m_1, m_2, m_3) = m'_1$ ,  $\operatorname{Step}_2(n, m_1, m_2, m_3) = m'_2$  и  $\operatorname{Step}_3(n, m_1, m_2, m_3) = m'_3$ .

Лемма 3 (о кусочном определении функции) Пусть  $P_1(x_1, \ldots, x_k)$ , ...,  $P_n(x_1, \ldots, x_k)$  — примитивно рекурсивные предикаты, такие что для любых  $n_1, \ldots, n_k \in \mathbb{N}$  существует единственное  $1 \leq i \leq n$ , для которого предикат  $P_i(n_1, \ldots, n_k)$  истинен. Пусть  $f_1(x_1, \ldots, x_k), \ldots, f_n(x_1, \ldots, x_k)$  — примитивно рекурсивные функции. Тогда функция

$$f(x_1, \dots, x_k) = \begin{cases} f_1(x_1, \dots, x_k), & P_1(x_1, \dots, x_k) \\ f_2(x_1, \dots, x_k), & P_2(x_1, \dots, x_k) \\ & \dots & \dots \\ f_n(x_1, \dots, x_k), & P_n(x_1, \dots, x_k) \end{cases}$$

примитивно рекурсивна.

Доказательство. 
$$f(x_1,...,x_k) = \sum_{i=1}^n \chi_{P_i}(x_1,...,x_k) f_i(x_1,...,x_k)$$
.  $\square$ 

**Лемма 4** Функции Step<sub>1</sub>, Step<sub>2</sub> и Step<sub>3</sub> примитивно рекурсивны.

Доказательство. Ясно, что тройка  $\langle m_1, m_2, m_3 \rangle$  является кодом некоторой конфигурации тогда и только тогда, когда для любого  $i \log(p(i), m_1) \leqslant 2$ ,  $\log(p(0), m_1) = 0$  и  $m_1 \neq 0$ . Легко заметить, что для любого i p(i) > i и что если  $\log(p(i), m_1) > 2$ , то  $p(i) < m_1$ . В связи с этим условие, при выполнении которого функции  $\operatorname{Step}_i (1 \leqslant i \leqslant 3)$  задаются в пункте 2 приведенного выше определения, выполняется тогда и только тогда, когда на наборе  $n, m_1, m_2$  истинен примитивно рекурсивный предикат  $S(y, x_1, x_2) = (x_2 \leqslant l(y) + 1) \& (x_1 \neq 0) \& ((\mu i \leqslant x_1)(\log(p(i), x_1) > 2) = x_1 + 1) \& (\log(p(0), x_1) = 0)$ .

Введем несколько примитивно рекурсивных функций:

$$i(x_1, x_2, x_3) = x_2$$

$$a(x_1, x_2, x_3) = \begin{cases} 3, & x_3 = 0 \\ \log(p(x_3), x_1), & x_3 > 0 \end{cases}$$

$$t(y, x_1, x_2, x_3) = \log(p(c(i(x_1, x_2, x_3), a(x_1, x_2, x_3))), r(y) + 1)$$

$$b(y, x_1, x_2, x_3) = \begin{cases} rest(r(t(y, x_1, x_2, x_3)), 5), & x_3 > 0\\ 4, & x_3 = 0 \end{cases}$$

 $j(y, x_1, x_2, x_3) = \text{rest}(l(t(y, x_1, x_2, x_3)), l(y) + 2)$ 

Смысл этих функций таков. Пусть n — номер программы P и  $\langle m_1, m_2, m_3 \rangle$  — код конфигурации  $\langle w, q_i, k \rangle$ , причем  $i \leqslant l(n)+1$ . Рассмотрим состояние машины, соответствующее этой конфигурации. В этом состоянии головка находится в состоянии  $q_i$  и указывает на k-ую ячейку ленты, в которой записан некоторый символ  $a \in \{0, 1, m, n\}$ . В программе P есть команда  $q_i a \to q_j b$ , где  $b \in \{n, 0, 1, L, R\}$ . Тогда  $i = i(m_1, m_2, m_3)$ ,

 $a(m_1,m_2,m_3)$  — код символа  $a,\ t(n,m_1,m_2,m_3)$  — код правой части команды  $q_i a \to q_j b,\ j=j(n,m_1,m_2,m_3)$  и  $b(n,m_1,m_2,m_3)$  — код символа b. Пусть  $T(y,x_1,x_2)$  — примитивно рекурсивный предикат, равный  $S(y,x_1,x_2)$  &  $(i(x_1,x_2,x_3)\neq 0).$  Окончательно получаем

$$\begin{aligned} \operatorname{Step}_3(y,x_1,x_2,x_3) &= \begin{cases} x_3, & (b(y,x_1,x_2,x_3) \leqslant 2) \vee \neg T(y,x_1,x_2) \\ x_3 &\vdash 1, & (b(y,x_1,x_2,x_3) = 3) \& T(y,x_1,x_2) \\ x_3 &+ 1, & (b(y,x_1,x_2,x_3) \geqslant 4) \& T(y,x_1,x_2); \end{cases} \\ \operatorname{Step}_2(y,x_1,x_2,x_3) &= \begin{cases} x_2, & \neg T(y,x_1,x_2) \\ j(y,x_1,x_2,x_3), & T(y,x_1,x_2); \end{cases} \\ \operatorname{Step}_1(y,x_1,x_2,x_3) &= \begin{cases} x_1, & (b(y,x_1,x_2,x_3) \geqslant 3) \vee \neg T(y,x_1,x_2) \\ \gamma(y,x_1,x_2,x_3), & (b(y,x_1,x_2,x_3) \leqslant 2) \& T(y,x_1,x_2), \end{cases} \\ \operatorname{The} \gamma(y,x_1,x_2,x_3) &= \operatorname{div}(x_1,p(x_3)^{a(x_1,x_2,x_3)}) \cdot p(x_3)^{b(y,x_1,x_2,x_3)}. \ \Box \end{aligned}$$

Введем теперь три пятиместные функции  $\mathrm{Com}_1,\ \mathrm{Com}_2,\ \mathrm{Com}_3.$  Для  $1\leqslant i\leqslant 3$  полагаем

$$\begin{bmatrix} \operatorname{Com}_{i}(y, x_{1}, x_{2}, x_{3}, 0) & = x_{i} \\ \operatorname{Com}_{i}(y, x_{1}, x_{2}, x_{3}, t + 1) & = \operatorname{Step}_{i}(y, \operatorname{Com}_{1}(y, x_{1}, x_{2}, x_{3}, t), \\ \operatorname{Com}_{2}(y, x_{1}, x_{2}, x_{3}, t), \operatorname{Com}_{3}(y, x_{1}, x_{2}, x_{3}, t)) \end{bmatrix}$$

Смысл этих функций ясен. Если  $\langle m_1, m_2, m_3 \rangle$  — код конфигурации C, n — натуральное число, кодирующее программу P, и программа P применима к конфигурации C (то есть  $m_2 \leq l(n)+1$ ), то для любого t  $\langle \operatorname{Com}_1(n, m_1, m_2, m_3, t), \operatorname{Com}_2(n, m_1, m_2, m_3, t), \operatorname{Com}_3(n, m_1, m_2, m_3, t) \rangle$  — код конфигурации  $\operatorname{Com}(C, P, t)$ , отражающей состояние машины, которое получается из состояния, задаваемого конфигурацией C, после t шагов работы по программе P.

**Лемма 5** Функции Com<sub>1</sub>, Com<sub>2</sub> и Com<sub>3</sub> примитивно рекурсивны.

Доказательство. Рассмотрим трехместную функцию  $\alpha(y,x,t)$ , которая задается следующей схемой примитивной рекурсии:

$$\begin{bmatrix} \alpha(y,x,0) &= x \\ \alpha(y,x,t+1) &= 2^{\beta_1(y,\alpha(y,x,t))} \cdot 3^{\beta_2(y,\alpha(y,x,t))} \cdot 5^{\beta_3(y,\alpha(y,x,t))}, \end{cases}$$

где для  $1 \leqslant i \leqslant 3$ 

$$\beta_i(y, x) = \text{Step}_i(y, \log(2, x), \log(3, x), \log(5, x)).$$

Ясно, что функция  $\alpha$  примитивно рекурсивна. Легко видеть, что если  $x=2^{x_1}\cdot 3^{x_2}\cdot 5^{x_3}$ , то  $\alpha(y,x,t)=2^{\mathrm{Com}_1(y,x_1,x_2,x_3,t)}\cdot 3^{\mathrm{Com}_2(y,x_1,x_2,x_3,t)}\cdot 5^{\mathrm{Com}_3(y,x_1,x_2,x_3,t)}$ . Окончательно получаем, что для  $1\leqslant i\leqslant 3$ 

$$Com_i(y, x_1, x_2, x_3, t) = log(p(i-1), \alpha(y, 2^{x_1}3^{x_2}5^{x_3}, t)) \square$$

**Теорема 9** Каждая функция, вычислимая на машине Тьюринга, частично рекурсивна.

Доказательство. Пусть  $f(x_1,\ldots,x_k)$  — частичная k-местная функция, вычислимая на машине Тьюринга при помощи программы с кодом n. Тогда для каждых  $x_1,\ldots,x_k\in\mathbb{N}$  машина Тьюринга, стартуя из состояния, описываемого конфигурацией  $\langle w(x_1,\ldots,x_k),q_1,1\rangle$ , и работая по программе с кодом n никогда не закончит работу, если  $f(x_1,\ldots,x_k)$  не определено, а если  $f(x_1,\ldots,x_k)$  определено, то она закончит работу и число единиц на ленте в момент окончания будет равно  $f(x_1,\ldots,x_k)$ . Введем (k+2)-местную функцию  $\varphi$ , полагая для всех  $y,x_1,\ldots,x_k,t$  значение  $\varphi(y,x_1,\ldots,x_k,t)$  равным:

- 1. 0, если машина, работая по программе с кодом y из состояния, задаваемого конфигурацией  $\langle w(x_1,\ldots,x_k),q_1,1\rangle$ , за t шагов работы не достигла состояния  $q_0$ , то есть не остановилась;
- 2. s+1, если машина, работая по программе с кодом y из состояния, задаваемого конфигурацией  $\langle w(x_1,\ldots,x_k),q_1,1\rangle$ , достигла конечного состояния  $q_0$  не более чем за t шагов работы и число единиц на ленте в момент остановки машины равно s.

Покажем, что функция  $\varphi$  примитивно рекурсивна. Пусть для  $1\leqslant i\leqslant k+1$ 

$$\alpha_i(x_1, \dots, x_k) = i + \sum_{j=1}^{i-1} x_j,$$

для  $1 \leqslant i \leqslant k$ 

$$\beta_i(s, x_1, \dots, x_k) = \begin{cases} 0, & (s < \alpha_i(x_1, \dots, x_k)) \lor (s \geqslant \alpha_{i+1}(x_1, \dots, x_k)) \\ 1, & s = \alpha_i(x_1, \dots, x_k) \\ 2, & \alpha_i(x_1, \dots, x_k) < s < \alpha_{i+1}(x_1, \dots, x_k), \end{cases}$$

$$\beta(s, x_1, \dots, x_k) = \sum_{i=1}^k \beta_i(s, x_1, \dots, x_k),$$

$$\gamma(x_1, \dots, x_k) = \prod_{s=0}^{\alpha_{k+1}(x_1, \dots, x_k)} p(s)^{\beta(s, x_1, \dots, x_k)}.$$

Все  $\alpha_i$ -ые — примитивно рекурсивные функции. Отсюда все  $\beta_i$ -ые примитивно рекурсивны. Значит, примитивно рекурсивными являются функции  $\beta$  и  $\gamma$ . Легко видеть, что для произвольных  $x_1, \ldots, x_k \in \mathbb{N}$  значение  $\gamma(x_1, \ldots, x_k)$  — это код слова  $w(x_1, \ldots, x_k)$ . Пусть

$$\delta(x) = \sum_{i=0}^{x} \left( \log(p(i), x) \stackrel{\cdot}{-} 1 \right).$$

Тогда функция  $\delta$  примитивно рекурсивна и если x — код слова w, то  $\delta(x)$  равно числу единиц в слове w. Теперь для функции  $\varphi$  имеем

$$\varphi(y, x_1, \dots, x_k, t) = \begin{cases} 0, & \operatorname{Com}_2(y, \gamma(x_1, \dots, x_k), 1, 1, t) \neq 0 \\ \delta(\operatorname{Com}_1(y, \gamma(x_1, \dots, x_k), 1, 1, t)) + 1, \operatorname{Com}_2(y, \gamma(x_1, \dots, x_k), 1, 1, t) = 0. \end{cases}$$

Таким образом,  $\varphi$  действительно примитивно рекурсивна. Пусть  $\tau(y,x_1,\ldots,x_k)=\mu t(\overline{\operatorname{sg}}(\varphi(y,x_1,\ldots,x_k,t))=0)$  и  $\psi(y,x_1,\ldots,x_k)=\varphi(y,x_1,\ldots,x_k,\tau(y,x_1,\ldots,x_k))\div 1$ . Функции  $\tau$  и  $\psi$  частично рекурсивны. Остается заметить, что  $f(x_1,\ldots,x_k)=\psi(n,x_1,\ldots,x_k)$ .  $\square$ 

Таким образом, числовая функция частично рекурсивна тогда и только тогда, когда она вычислима на машине Тьюринга. Этот результат можно было предсказать, принимая во внимание тезисы Тьюринга и Черча. Действительно, для каждой частично рекурсивной функции существует алгоритм вычисления и, значит, по тезису Тьюринга для каждой частично рекурсивной функции должна существовать программа, преобразующая запись набора аргументов в запись значения функции на этих аргументах (которая должна работать бесконечно, если значение функции не определено). Обратно, если функция вычислима на машине Тьюринга, то есть некоторая программа преобразует запись значений аргументов в запись значения функции, то для этой функции существует

алгоритм вычисления (реализацией которого является данная программа) и, следовательно, по тезису Черча эта функция должна быть частично рекурсивна. Что мы и наблюдаем. Конечно, мы не можем доказать тезисы Тьюринга и Черча, поскольку они не являются математическими утверждениями. Однако мы доказали, что класс частично рекурсивных функций и класс числовых функций, вычислимых на машинах Тьюринга, совпадают. То есть мы доказали эквивалентность этих двух тезисов, по крайней мере в отношении числовых функций.

Настала пора окончательно разобраться с общим понятием вычислимой функции. Напомним, что вычислимая функция — это функция из конструктивного пространства  $\mathcal{K}_1$  в конструктивное пространство  $\mathcal{K}_2$ , для которой существует алгоритм, позволяющий по описанию значения аргумента получит описание значения функции. В настоящий момент у нас есть формальные определения для двух случаев:  $\mathcal{K}_1 = \mathcal{K}_2 = \Sigma^*$  (вычислимость на машине Тьюринга) и  $\mathcal{K}_1 = \mathbb{N}^k$ ,  $\mathcal{K}_2 = \mathbb{N}$  (частично рекурсивные функции). Мы уже говорили о том, что общий случай можно свести к первому. Покажем, как общий случай сводится ко второму.

Пусть S — произвольное непустое не более чем счетное множество<sup>48</sup>.

**Определение 19** *Нумерацией* множества S называется произвольное сюрьективное отображение  $\mathbb N$  на  $S^{49}$ .

Если  $\nu: \mathbb{N} \to S$  — нумерация, то для произвольного  $s \in S$  найдется хотя бы одно  $n \in \mathbb{N}$ , для которого  $\nu(n) = s$ . Мы будем называть каждое такое n  $\nu$ -номером элемента s (или просто номером, если ясно, о какой нумерации идет речь).

Определение 20 Нумерации  $\nu$  и  $\mu$  множества S называются эквивалентными, если существуют рекурсивные 1-местные функции<sup>50</sup> f и g, такие что для любого  $n \in \mathbb{N}$   $\mu(n) = \nu(f(n))$  и  $\nu(n) = \mu(g(n))$ .

Если нумерации  $\nu$  и  $\mu$  эквивалентны, то мы пишем  $\nu \equiv \mu$ . Легко показать, что отношение  $\equiv$  является отношением эквивалентности на

 $<sup>^{48}</sup>$ То есть множество конечной или счетной мощности. Понятие мощности множества вводится в курсе математической логики. В настоящий момент можно принять следующее определение: множество S имеет не более чем счетную мощность, если существует отображение из  $\mathbb{N}$  в S, область значений которого равна S, либо  $S=\varnothing$ .

 $<sup>^{49}</sup>$ То есть такое отображение, область значений которого равна S.

 $<sup>^{50}</sup>$ Напомним, что функция называется рекурсивной, если она частично рекурсивна и всюду определена.

множество нумераций любого не более чем счетного множества S. Принимая тезис Черча, можно сказать, что нумерации эквивалентны, если существует алгоритм, вычисляющий по  $\nu$ -номеру любого объекта его  $\mu$ -номер и наоборот. С точки зрения теории алгоритмов эквивалентные нумерации "совпадают", то есть наиболее важные свойства у них одинаковы $^{51}$ .

**Определение 21** Нумерация  $\nu$  называется *разрешимой*, если функция

$$\eta_{\nu}(x,y) = \begin{cases} 1, & \nu(x) = \nu(y) \\ 0, & \nu(x) \neq \nu(y) \end{cases}$$

рекурсивна.

Имея в виду тезис Черча, можно интерпретировать это определение следующим образом: нумерация разрешима, если существует алгоритм, который по двум номерам позволяет определить, являются они номерами одного и того же объекта или нет.

**Определение 22** Нумерация  $\nu$  называется *разнозначной*, если для любых  $x \neq y$  из  $\mathbb{N}$   $\nu(x) \neq \nu(y)$ .

Всякая разнозначная нумерация разрешима, так как если  $\nu$  разнозначна, то  $\eta_{\nu}(x,y)=\overline{\mathrm{sg}}((x\dot{-}y)+(y\dot{-}x))$ — примитивно рекурсивная функция. В некотором смысле разнозначность — это "идеал" для всякой нумерации, поскольку разнозначная нумерация  $\nu:\mathbb{N}\to S$  является взаимнооднозначным соответствием между  $\mathbb{N}$  и S и позволяет отождествлять элементы множества S и их номера.

**Предложение 9** Для всякой разрешимой нумерации бесконечного множества существует эквивалентная ей разнозначная нумерация.

Доказательство. Пусть  $\nu: \mathbb{N} \to S$  — разрешимая нумерация и множество S бесконечно. Определим рекурсивную функцию f:

$$\begin{cases}
f(0) &= 0 \\
f(t+1) &= f(t) + \mu y(\sum_{i=0}^{t} \eta_{\nu}(f(i), f(t) + y) = 0)
\end{cases}$$

 $<sup>^{51}</sup>$ Точно так же в геометрии две фигуры считаются равными, если переводятся друг в друга преобразованием движения. Аналогично в алгебре две структуры "совпадают", если они изоморфны.

Так как f определяется через рекурсивную функцию  $\eta_{\nu}$  при помощи минимизации и примитивной рекурсии, то она частично рекурсивна. Из бесконечности S следует, что f всюду определена.

Полагаем  $\nu'(n) = \nu(f(n))$  для любого  $n \in \mathbb{N}$ . Из определения f видно, что  $\nu'$  — разнозначная нумерация множества S. Пусть теперь  $g(x) = \mu y(\overline{\operatorname{sg}}(\eta_{\nu}(f(y),x)) = 0)$ . Ясно, что g всюду определена и что  $\nu(n) = \nu'(g(n))$ . Так как  $\nu' = \nu \circ f$  и  $\nu = \nu' \circ g$ , то  $\nu' \equiv \nu$ .

На неформальном уровне  $\nu'$  устроена следующим образом. Пусть  $X = \{\min(s) : s \in S, \min(s) -$ минимальный номер элемента  $s\}$ . Пусть  $x_0 < x_1 < \ldots -$  перечисление всех элементов множества X в порядке возрастания. Тогда для любого i  $f(i) = x_i$  и  $\nu'(i) = \nu(x_i)$ .  $\square$ 

Рассмотрим несколько примеров нумераций.

- 1.  $S=\mathbb{N}$  и нумерация  $\nu_1:\mathbb{N}\to S$  задается следующим образом:  $\nu_1(n)=n$  для любого  $n\in\mathbb{N}.$  Это разнозначная нумерация.
- 2. Пусть f произвольная перестановка натурального ряда,  $S = \mathbb{N}$  и нумерация  $\nu_2 : \mathbb{N} \to S$  такова, что  $\nu_2(n) = f(n)$  для любого  $n \in \mathbb{N}$ . Это другой пример разнозначной нумерации.
- 3.  $S=\mathbb{N}$  и нумерация  $\nu_3:\mathbb{N}\to S$  задается следующим образом:  $\nu_3(n)=l(n)$  для любого  $n\in\mathbb{N}$ . Эта нумерация не разнозначна, однако является разрешимой.
- 4.  $S=\mathbb{N}^2$  и нумерация  $\nu_4:\mathbb{N}\to S$  задается следующим образом:  $\nu_4(n)=\langle l(n),r(n)\rangle$  для любого  $n\in\mathbb{N}$ . Эта нумерация тоже разнозначна.
- 5. Пусть S множество программ для машин Тьюринга, написанных для алфавита  $\Sigma = \{0,1\}$  и для  $n \in \mathbb{N}$   $\nu_5(n)$  программа с кодом n. По тезису Черча нумерация  $\nu_5$  разрешима. Алгоритм вычисления функции  $\eta_{\nu_5}$  следующий: для данных x и y надо выписать программы с номерами x и y (что, очевидно, можно сделать при помощи алгоритма), а затем сравнить эти программы и решить, чему равно  $\eta_{\nu_5}(x,y)$ . При желании можно дать формальное доказательство рекурсивности функции  $\eta_{\nu_5}^{52}$ .

 $<sup>^{52}</sup>$ Всякий раз, когда мы привлекаем для доказательства тезис Черча, мы можем дать формальное доказательство, которое его не использует. На этом факте, в общемто, и основана уверенность в справедливости этого тезиса.

- 6. Пусть S множество k местных частично рекурсивных функций и для  $n \in \mathbb{N}$   $\nu_6(n)$  k-местная функция, вычислимая на машине Тьюринга при помощи программы с кодом n. По теореме 9  $\nu_6$  это нумерация множества S. Нумерация  $\nu_6$  уже не будет разрешимой. Хотя по номерам программ мы и можем выписать сами программы, однако нет алгоритма, который позволял бы определить, вычисляют две разные программы одну и ту же функцию или нет. Позже мы приведем формальное доказательство того, что функция  $\eta_{\nu_6}(x,y)$  не рекурсивна.
- 7. Пусть  $\Sigma = \{a_1, \dots, a_k\}$  конечный алфавит и  $S = \Sigma^*$ . Сопоставим каждому  $n \in \mathbb{N}$  слово  $\nu_7(n) \in \Sigma^*$ , равное  $a_{i_1} \dots a_{i_s}$ , следующим образом: s = l(n) и для  $1 \leqslant j \leqslant s$   $i_j = \operatorname{rest}(\log(p(j), r(n) + 1), k) + 1$ . Тогда  $\nu_7$  будет нумерацией S. Действительно, каждое слово  $a_{i_1} \dots a_{i_s}$  имеет  $\nu_7$ -номер, равный  $c(s, p(1)^{i_1-1} \cdot \dots \cdot p(s)^{i_s-1} 1)$ . Эта нумерация не будет разнозначной, однако будет разрешимой. Действительно, нумерация построена так, что по номеру слова легко выписать само слово: алгоритм для этой процедуры очевиден. Тогда функция  $\eta_{\nu_7}(x,y)$  вычислима при помощи следующего алгоритма: надо для данных x и y выписать слова  $\nu_7(x)$  и  $\nu_7(y)$ , сравнить их и в зависимости от результата сравнения решить, чему равно  $\eta_{\nu_7}(x,y)$ . Формальное доказательство в данном случае тоже является достаточно коротким. Действительно, предикат  $P(x,y) = (l(x) = l(y)) \& ((\mu i \leqslant l(x)))(\operatorname{rest}(\log(p(i+1), r(x)+1), k) \neq \operatorname{rest}(\log(p(i+1), r(y)+1), k)) \geqslant l(x))$  примитивно рекурсивен и  $\eta_{\nu_7}(x,y) = \chi_P(x,y)$ .

Сделаем еще несколько замечаний относительно этих примеров. Можно показать, что нумерации  $\nu_1$  и  $\nu_3$  эквивалентны. Действительно,  $\nu_3(x) = \nu_1(l(x))$  и  $\nu_1(x) = \nu_3(c(x,0))$  для любого  $x \in \mathbb{N}$ . Нумерации  $\nu_1$  и  $\nu_2$  эквивалентны не всегда, а лишь тогда, когда функция f, участвующая в определении  $\nu_2$ , рекурсивна. Про эквивалентность нумераций  $\nu_1$  и  $\nu_4$  говорить бессмысленно, поскольку эти нумерации нумеруют разные множества.

Пусть теперь  $\mathcal{K}$  — конструктивное пространство. Легко показать, что  $\mathcal{K}$  не более чем счетно (элементам  $\mathcal{K}$  соответствуют их описания, то есть слова некоторого конечного алфавита, а множество слов конечного алфавита счетно).

**Определение 23** Нумерация  $\nu : \mathbb{N} \to \mathcal{K}$  называется *геделевской* <sup>53</sup>, если существует алгоритм, который позволяет по записи любого натурального числа получить описание объекта, номером которого это число является.

Определение 23 носит более формальный характер, чем может показаться на первый взгляд. Под алгоритмом в этом определении можно понимать программу для машины Тьюринга, преобразующую слово  $1^n$  в описание объекта  $\nu(n)$ , являющееся словом некоторого конечного алфавита  $\Sigma_{\mathcal{K}}$ . Однако это определение нельзя считать полностью формальным, поскольку понятия конструктивного пространства и описания элемента этого пространства у нас все же довольно расплывчаты. Тем не менее, для каждого конкретного примера конструктивного пространства можно дать формальное определение геделевской нумерации этого пространства, опираясь на определение  $23^{54}$ .

В приведенных выше примерах нумерации  $\nu_1$ ,  $\nu_3$ ,  $\nu_4$ ,  $\nu_5$  и  $\nu_7$  — геделевские. Нумерация  $\nu_2$  может являться геделевской в случае, если f — рекурсивная функция, однако в общем случае она не является таковой. Нумерация  $\nu_6$  — не геделевская, так как нумеруемое множество вообще не является конструктивным пространством.

## **Предложение 10** Пусть $\mathcal{K}$ — конструктивное пространство (не равное $\varnothing$ ). Тогда

- 1. существует геделевская нумерация пространства  $\mathcal{K}$ ;
- 2. всякая геделевская нумерация K разрешима;
- 3. любые две геделевские нумерации K эквивалентны;
- 4. всякая нумерация пространства  $\mathcal{K}$ , эквивалентная геделевской, сама является геделевской.

<sup>&</sup>lt;sup>53</sup>Термин "геделевская нумерация" возник в связи с работами Курта Геделя, который, введя подобную нумерацию для множества формул языка арифметики первого порядка, доказал свою знаменитую теорему о неполноте арифметики.

<sup>&</sup>lt;sup>54</sup>Отметим еще, что в литературе термин "геделевская нумерация" иногда понимают более широко, причисляя к геделевским частичные нумерации с рекурсивной областью определения. Такова, например, классическая геделевская нумерация формул и термов, используемая при доказательстве теоремы Геделя о неполноте. Легко показать, что этот общий случай всегда сводится к нашему частному через перечисление области определения частичной нумерации. В нашем курсе мы не будем останавливаться на этом вопросе и вводить соответствующие понятия.

 $\mathcal{A}$ оказательство. Конструктивное пространство — это множество всех конструктивных объектов одинакового вида. Конструктивный объект — это объект, который может быть полностью описан при помощи конечной последовательности символов. Как мы уже отмечали в начале этой части курса, мы принимаем следующий тезис: для каждого конструктивного пространства  $\mathcal{K}$  существует конечный алфавит  $\Sigma_{\mathcal{K}}$  и описаниями элементов  $\mathcal{K}$  являются слова этого алфавита. Примем еще два тезиса:

- 1. Существует алгоритм, который позволяет по слову алфавита  $\Sigma_{\mathcal{K}}$  определить, является это слово описанием некоторого элемента  $\mathcal{K}$  или нет.
- 2. Существует алгоритм, который по двум словам алфавита  $\Sigma_{\mathcal{K}}$ , являющихся описаниями элементов  $\mathcal{K}$ , позволяет определить, являются они описаниями одного и того же объекта или нет.

Оба этих тезиса достаточно естественны. Раз описание конструктивного объекта содержит всю информацию об этом объекте (см. определение 2), то имея слово алфавита  $\Sigma_{\mathcal{K}}$ , мы можем понять, содержит ли оно в себе информацию об объекте или просто является бессмысленным набором символов (как ранее в примере с  $\mathcal{K}=\mathbb{Q}$  мы признали, что слова -3/7 и -9/21 являются описаниями рациональных чисел, а слово //-/11 — нет). Точно так же если есть два описания, то по ним можно получить всю информацию об объектах, которые они описывают; а обладая полной информацией о двух объектах, мы можем решить, совпадают они или нет. Ни один из этих тезисов нельзя доказать, поскольку такие понятия, как "конструктивный объект" и "описание конструктивного объекта" у нас довольно расплывчаты и для них нет четких определений. Можно рассматривать эти тезисы не как утверждения, истинность которых надо доказывать, а как уточнения определений 2 и 3.

Итак, признаем, что эти тезисы справедливы. Имеется много алгоритмов, позволяющих перечислять все слова алфавита  $\Sigma_{\mathcal{K}}$ . Можно взять, например, такой алгоритм: сначала перечислим пустое слово, потом все слова длины 1, потом все слова длины 2 и так далее. Воспользуемся одним из этих алгоритмов и запишем  $\Sigma_{\mathcal{K}}^*$  в виде последовательности слов  $w_0, w_1, w_2, \ldots$ , такой что существует алгоритм, позволяющий по каждому  $i \in \mathbb{N}$  вычислить слово  $w_i$ . Пусть  $X = \{i :$  слово  $w_i$  является описанием некоторого элемента пространства  $\mathcal{K}\}$ . По первому тезису существует

алгоритм, позволяющий вычислять числовую функцию

$$\chi_{\scriptscriptstyle X}(i) = \begin{cases} 1, & i \in X \\ 0, & i \not \in X. \end{cases}$$

По тезису Черча функция  $\chi_{\scriptscriptstyle X}$  рекурсивна.

Так как  $\mathcal{K} \neq \emptyset$ , то X непусто. Зафиксируем  $i_0 \in X$ . Теперь пусть  $f(x) = x \cdot \chi_X(x) + i_0 \cdot \overline{\operatorname{sg}}(\chi_X(x))$ . Функция f получается при помощи суперпозиций из рекурсивных функций и, следовательно, рекурсивна. Ясно, что  $\rho f = X$ .

Теперь введем нумерацию  $\nu: \mathbb{N} \to \mathcal{K}$ . Для  $i \in \mathbb{N}$  полагаем  $\nu(i)$  равным объекту, описанием которого является слово  $w_{f(i)}$ . Нумерация  $\nu$  является геделевской. Действительно, алгоритм, вычисляющий по числу i описание объекта  $\nu(i)$  таков: надо вычислить j = f(i), по j вычислить слово  $w_i$  и взять это слово в качестве искомого описания.

Мы доказали первый пункт предложения. Докажем третий. Пусть  $\mu_1, \, \mu_2$  — две геделевских нумерации пространства  $\mathcal{K}$ . Для i=1,2 существует алгоритм, позволяющий по каждому n получать описание объекта с  $\mu_i$ -номером n. Другими словами, существуют вычислимые функции  $v_1, v_2 : \mathbb{N} \to \Sigma_{\mathcal{K}}^*$ , такие что для i = 1, 2  $v_i(n)$  — описание объекта  $\mu_i(n)$ . Опишем алгоритм вычисления функции  $h_1: \mathbb{N} \to \mathbb{N}$ . Этот алгоритм таков: при данном  $x \in \mathbb{N}$  последовательно выписываем слова  $v_2(0), v_2(1), \dots$ и для каждого y смотрим, является ли слово  $v_2(y)$  описанием того же самого объекта, что и слово  $v_1(x)$ . Согласно второму тезису мы можем это сделать при помощи алгоритма. Рано или поздно найдется такой y, что слова  $v_1(x)$  и  $v_2(y)$  описывают один и тот же объект. Как только нам встретился такой y, останавливаем процедуру перечисления слов  $v_2(0), v_2(1), \dots$  и полагаем  $h_1(x)$  равным этому y. Функция  $h_2$  определяется аналогично  $h_1$ , с заменой  $v_2$  на  $v_1$  и наоборот. Легко видеть, что  $\mu_2 = \mu_1 \circ h_2$  и  $\mu_1 = \mu_2 \circ h_1$ . Так как у нас имеются алгоритмы для вычисления функций  $h_1$  и  $h_2$ , то по тезису Черча эти функции рекурсивны и, следовательно,  $\mu_1 \equiv \mu_2$ .

Докажем второй пункт. Пусть  $\mu$  — геделевская нумерация и v — вычислимая функция из  $\mathbb{N}$  в  $\Sigma_{\mathcal{K}}^*$ , такая что для любого  $n \in \mathbb{N}$  v(n) — описание объекта  $\mu(n)$ . Тогда для  $\eta_{\mu}(x,y)$  существует следующий алгоритм вычисления: надо взять x и y, выписать слова v(x) и v(y), а затем положить  $\eta_{\mu}(x,y)$  равным единице, если эти слова описывают один и тот же объект, и нулю в противном случае. Так как алгоритм для вычисления

 $\eta_{\mu}$  описан, то по тезису Черча функция  $\eta_{\mu}$  рекурсивна и нумерация  $\mu$  разрешима.

Осталось доказать четвертый пункт. Пусть  $\mu_1: \mathbb{N} \to \mathcal{K}$  — геделевская нумерация и  $\mu_2 \equiv \mu_1$ . Тогда существует рекурсивная функция h, такая что  $\mu_2 = \mu_1 \circ h$ . По любому  $\mu_2$  -номеру i можно вычислить описание объекта  $\mu_2(i)$  следующим образом: сначала вычисляем j = h(i), а затем по j вычисляем описание объекта  $\mu_1(j)$ . Так как  $\mu_1$  — геделевская нумерация, то мы можем это сделать. Описанная процедура является алгоритмом и, значит, нумерация  $\mu_2$  — геделевская.  $\square$ 

Итак, для каждого конструктивного пространства существует геделевская нумерация, единственная с точностью до эквивалентности.

**Определение 24** Конструктивное пространство  $\mathcal{K}_1$  *изоморфно* конструктивному пространству  $\mathcal{K}_2$ , если существует взаимно-однозначное отображение пространства  $\mathcal{K}_1$  на пространство  $\mathcal{K}_2$ , являющееся вычислимой функцией.

**Предложение 11** Для конструктивных пространств справедливы следующие утверждения:

- 1. Если  $\mathcal{K}_1$  изоморфно  $\mathcal{K}_2$ , то  $\mathcal{K}_2$  изоморфно  $\mathcal{K}_1$ ;
- 2. Если  $\mathcal{K}_1$  изоморфно  $\mathcal{K}'_1$  и  $\mathcal{K}_2$  изоморфно  $\mathcal{K}'_2$ , то существует взаимнооднозначное соответствие между множеством вычислимых функций из  $\mathcal{K}_1$  в  $\mathcal{K}_2$  и множеством вычислимых функций из  $\mathcal{K}'_1$  в  $\mathcal{K}'_2$ .

Доказательство. (1) Пусть  $f: \mathcal{K}_1 \to \mathcal{K}_2$  — изоморфизм, то есть вычислимая функция, взаимно-однозначно отображающая  $\mathcal{K}_1$  на  $\mathcal{K}_2$ . Так как f взаимно-однозначно, то определено обратное отображение  $f^{-1}: \mathcal{K}_2 \to \mathcal{K}_1$ , которое тоже является взаимно-однозначным. Покажем, что функция  $f^{-1}$  вычислима.

Пусть  $\nu$  — геделевская нумерация  $\mathcal{K}_1$ . Требуется задать алгоритм, который позволяет по слову  $w \in \Sigma_{\mathcal{K}_2}^*$ , являющемуся описанием объекта  $k \in \mathcal{K}_2$ , получить описание объекта  $f^{-1}(k)$ . Этот алгоритм следующий. Для произвольного  $i \in \mathbb{N}$  мы можем получить слово  $v_i \in \Sigma_{\mathcal{K}_1}^*$ , являющеся описанием объекта  $\nu(i)$ . Далее, пользуясь вычислимостью функции f, мы по  $v_i$  можем получить слово  $w_i \in \Sigma_{\mathcal{K}_2}^*$ , являющеся описанием объекта  $f(\nu(i))$ . Начнем последовательно выписывать слова  $w_0, w_1, \ldots$  и для

каждого  $w_i$  проверять, описывает ли оно тот же самый объект, что и слово w. По второму тезису из доказательства предложения 10 мы можем делать это эффективно, то есть при помощи некоторого алгоритма. Как только мы обнаружим, что для некоторого s слово  $w_s$  описывает тот же самый объект, что и слово w, мы останавливаем процедуру перечисления  $w_i$ -ых и объявляем слово  $v_s$  результатом работы нашего алгоритма. Из того, что  $\rho f = \mathcal{K}_2$  следует, что для каждого  $w \in \Sigma_{\mathcal{K}_2}^*$ , которое является описанием некоторого объекта пространства  $k \in \mathcal{K}_2$ , алгоритм через некоторое время завершит свою работу. Ясно, что полученное слово  $v_s$  будет описанием объекта  $f^{-1}(k)$ .

(2) Пусть  $g: \mathcal{K}_1 \to \mathcal{K}_1'$  и  $h: \mathcal{K}_2 \to \mathcal{K}_2'$  — изоморфизмы. По предыдущему пункту  $g^{-1}$  и  $h^{-1}$  — тоже изоморфизмы. Ясно, что композиция вычислимых функций всегда будет вычислимой функцией. Сопоставим каждой вычислимой функции  $f: \mathcal{K}_1 \to \mathcal{K}_2$  вычислимую функцию  $h \circ f \circ g^{-1}: \mathcal{K}_1' \to \mathcal{K}_2'$ . Ясно, что соответствие  $f \mapsto h \circ f \circ g^{-1}$  является взаимно-однозначным отображением множества вычислимых функций из  $\mathcal{K}_1$  в  $\mathcal{K}_2$  на множество вычислимых функций из  $\mathcal{K}_1'$  в  $\mathcal{K}_2'$ .  $\square$ 

Мы видим, что с точки зрения теории алгоритмов изоморфные конструктивные пространства не отличимы друг от друга. Если  $\mathcal{K}_1 \cong \mathcal{K}_1'$  и  $\mathcal{K}_2 \cong \mathcal{K}_2'$ , то ввиду наличия взаимно-однозначного соответствия, о котором говорится в предложении 11, можно вместо вычислимых функций из  $\mathcal{K}_1$  в  $\mathcal{K}_2$  изучать вычислимые функции из  $\mathcal{K}_1'$  в  $\mathcal{K}_2'$ .

**Предложение 12** Любые два бесконечных конструктивных пространства изоморфны.

Доказательство. Достаточно доказать, что любое бесконечное конструктивное пространство изоморфно пространству  $\mathbb{N}$ . Пусть  $\mathcal{K}$  — бесконечное конструктивное пространство. По предложению 10(1) существует геделевская нумерация  $\nu: \mathbb{N} \to \mathcal{K}$ . По пункту 2 того же предложения  $\nu$  разрешима. По предложению 9 существует разнозначная нумерация  $\mu$ , эквивалентная  $\nu$ . По предложению 10(4) нумерация  $\mu$  является геделевской. Остается заметить, что геделевость и разнозначность нумерации  $\mu$  равносильны тому, что  $\mu$  — изоморфизм из  $\mathbb{N}$  на  $\mathcal{K}$ .  $\square$ 

Таким образом, изучая вычислимые функции из  $\mathcal{K}_1$  в  $\mathcal{K}_2$  для бесконечных  $\mathcal{K}_1$  и  $\mathcal{K}_2$ , можно заменить  $\mathcal{K}_1$  и  $\mathcal{K}_2$  на любую другую пару бесконечных конструктивных пространств. В частности, можно считать, что

 $\mathcal{K}_1 = \mathcal{K}_2 = \mathbb{N}$  и изучать рекурсивные и частично рекурсивные функции вместо вычислимых функций из  $\mathcal{K}_1$  в  $\mathcal{K}_2$ .

Случай, когда  $\mathcal{K}_1$  конечно, малоинтересен, поскольку легко показать, что при конечном  $\mathcal{K}_1$  любая функция из  $\mathcal{K}_1$  в  $\mathcal{K}_2$  будет вычислимой. Остается случай, когда  $\mathcal{K}_1$  бесконечно, а  $\mathcal{K}_2$  конечно. Этот случай тоже легко сводится к функциям из  $\mathbb{N}$  в  $\mathbb{N}$ . Заменим бесконечное конструктивное пространство  $\mathcal{K}_1$  на изоморфное ему пространство  $\mathbb{N}$ . Пусть  $\mathcal{K}_2$  состоит из n элементов и  $X = \{1, \dots, n\} \subseteq \mathbb{N}$ . Рассмотрим взаимно однозначное отображение  $\alpha : \mathcal{K}_2 \to X$ . Каждой вычислимой функции  $f : \mathbb{N} \to \mathcal{K}_2$  сопоставим функцию  $\alpha \circ f : \mathbb{N} \to \mathbb{N}$ . Легко доказать, что функция f вычислима тогда и только тогда, когда вычислима функция  $\alpha \circ f$ . Соответствие  $f \mapsto \alpha \circ f$  сопоставляет вычислимым функциям из  $\mathbb{N}$  в  $\mathcal{K}_2$  вычислимые функции из  $\mathbb{N}$  в  $\mathbb{N}$ .

Предложение 12 имеет множество далеко идущих следствий. Можно, например, для произвольного конечного алфавита  $\Sigma$  ввести разнозначную геделевскую нумерацию, которая будет ни чем иным, как изоморфизмом из  $\mathbb{N}$  на  $\Sigma^*$ . Пользуясь этой нумерацией и отождествляя функции из  $\Sigma^*$  в  $\Sigma^*$  с функциями из  $\mathbb{N}$  в  $\mathbb{N}$ , можно доказать, что тезис Тьюринга равносилен тезису Черча.

На практике весьма распространена следующая ситуация. Исследователи, работая с функциями из конструктивного пространства  $\mathcal{K}_1$  в конструктивное пространство  $\mathcal{K}_2$ , фиксируют какие-либо разнозначные геделевские нумерации этих пространств (то есть изоморфизмы из N на эти пространства) и отождествляют элементы пространств  $\mathcal{K}_1$  и  $\mathcal{K}_2$  с их номерами. Соответственно, вычислимым функциям из  $\mathcal{K}_1$  в  $\mathcal{K}_2$  соответствуют вычислимые функции из N в N. Вычислимость же функции из № в № эквивалентна, по тезису Черча, ее частичной рекурсивности. В связи с этим можно, например, говорить, о функциях из  $\mathcal{K}_1$  в  $\mathcal{K}_2$  как о частично рекурсивных или рекурсивных, несмотря на то, что эти понятия являются достаточно специфическими и определены только для числовых функций. В современной литературе, правда, прослеживается прямо противоположная тенденция. Термины "частично рекурсивная функция" и "рекурсивная функция" сейчас употребляются все реже и реже; вместо них ученые стараются использовать термины "вычислимая функция" и "вычислимая всюду определенная функция". Причина этого тоже достаточно понятна. "Рекурсивность" и "частичная рекурсивность" — очень узкие термины, имеющие смысл только для числовых функций. А когда мы говорим о вычислимой функции из N в N, то мы можем мыслить о ней как о функции из  $\mathcal{K}_1$  в  $\mathcal{K}_2$ , подразумевая под аргументами и значениями этой функции не сами натуральные числа, а объекты конструктивных пространств, номерами которых эти числа являются. Когда исследователь формулирует свои результаты, используя прилагательное "рекурсивный", то он тем самым как бы подразумевает, что этот результат носит очень узкий характер и справедлив только для числовых функций. А если он использует прилагательное "вычислимый", то он, наоборот, показывает тем самым, что хотя его рассуждения и имели дело только с множеством  $\mathbb{N}$ , однако их можно обобщить и на случай произвольного конструктивного пространства  $\mathcal{K}$ , введя подходящую геделевскую нумерацию.

В принципе, нам тоже было бы неплохо поменять терминологию. Однако большая часть курса уже позади. Поэтому ничего менять не будем, а, напротив, продолжим называть частично рекурсивные функции "частично рекурсивными". Но при этом постараемся не забыть, что дальнейшие результаты, которые мы сформулируем для числовых функций, можно обобщить для случая, когда функции отображают произвольное конструктивное пространство.

Прежде чем вернуться к числовым функциям, сделаем еще одно замечание. Во введении к курсу мы сформулировали два результата.

- 1. Доказано, что не существует алгоритма, который по многочлену от нескольких переменных с целыми коэффициентами определял бы, есть ли у него целочисленные корни.
- 2. Доказано, что не существует алгоритма, который бы по утверждению о натуральных числах выдавал ответ на вопрос, истинно оно или ложно.

В настоящий момент мы обладаем достаточными знаниями для того, чтобы дать точные формулировки этих результатов. На самом деле было доказано следующее:

1. Зафиксируем геделевскую нумерацию  $\nu$  пространства многочленов от нескольких переменных с целыми коэффициентами (то есть такую нумерацию  $\nu$ , для которой существует машина Тьюринга, перерабатывающая слово  $1^n$  в запись многочлена  $\nu(n)$ ). Пусть  $X = \{n \in \mathbb{N} :$  многочлен  $\nu(n)$  имеет целочисленные корни $\}$ . Тогда характеристическая функция  $\chi_X(x)$  не является рекурсивной.

2. Введем формальный язык для записи утверждений о натуральных числах (например, язык арифметики первого порядка, который изучается в курсе математической логики). Зафиксируем геделевскую нумерацию  $\nu$  множества утверждений, записанных на формальном языке (то есть такую нумерацию  $\nu$ , для которой существует машина Тьюринга, перерабатывающая слово  $1^n$  в запись утверждения  $\nu(n)$ ). Пусть  $Y = \{n \in \mathbb{N} : \text{утверждение } \nu(n) \text{ истинно}\}$ . Тогда характеристическая функция  $\chi_Y(x)$  не является рекурсивной.

Вернемся, наконец, назад к числовым функциям.

Пусть  $\nu$  — какая-нибудь геделевская нумерация множества программ для машин Тьюринга. Это может быть и нумерация  $\nu_5$ , описанная выше, которая сопоставляет каждому натуральному числу n программу с кодом n. Но может быть и какая-нибудь другая. Можно, пользуясь предложениями 9 и 10, считать, что  $\nu$  — разнозначная нумерация. Однако это не принципиально.

Для  $n, k \in \mathbb{N}$  через  $\varphi_n(x_1, \ldots, x_k)$  обозначим частично рекурсивную функцию, вычислимую на машине Тьюринга по программе  $\nu(n)$ . Определение вычислимости числовой функции на машине Тьюринга (определение 18) устроено так, что при любом k каждая программа для машины Тьюринга вычисляет некоторую k-местную функцию<sup>55</sup>. По следствию 4 в последовательности  $\varphi_0(x_1,\ldots,x_k), \varphi_1(x_1,\ldots,x_k),\ldots$  содержатся все k-местные частично рекурсивные функции. По теореме 9 при каждом  $n \in \mathbb{N}$  функция  $\varphi_n(x_1,\ldots,x_k)$  частично рекурсивна. На самом деле, справедливо гораздо более сильное утверждение.

**Предложение 13** Функция  $\varphi_y(x_1, ..., x_k)$  является частично рекурсивной как функция от (k+1)-го аргумента<sup>56</sup>.

 $\mathcal{A}$ оказательство. Если  $\nu = \nu_5$ , то этот факт уже доказан в ходе доказательства теоремы 9. Действительно, при доказательстве этой теоремы

 $<sup>^{55}</sup>$ Вот почему мы положили в определении 18, что значение функции равно числу единиц на ленте в момент остановки машины. Можно было бы поступить более естественно: считать, что машина вычисляет функцию  $f(x_1,\ldots,x_k)$ , если она перерабатывает слово  $w(x_1,\ldots,x_k)$  в слово  $1^{f(x_1,\ldots,x_k)}$ , и это было бы ближе к определению 4. Однако тогда было бы непонятно, что считать значением функции, если, например, программа останавливает машину, когда головка находится не в первой позиции и единицы на ленте в момент остановки перемежаются нулями и символами n.

<sup>&</sup>lt;sup>56</sup>То есть как функция от  $y, x_1, ..., x_k$ .

мы ввели частично рекурсивную функцию  $\psi(y, x_1, \ldots, x_k)$ , которая равна  $\varphi_y(x_1, \ldots, x_k)$  в случае, если мы при определении  $\varphi_y$  берем  $\nu = \nu_5$ . Пусть теперь  $\nu$  — произвольная геделевская нумерация и  $\varphi_y$  вводится через  $\nu$ . Тогда, по предложению 10,  $\nu$  эквивалентна  $\nu_5$  и существует рекурсивная функция f, для которой  $\nu = \nu_5 \circ f$ . Но тогда для любых  $y, x_1, \ldots, x_k$   $\varphi_y(x_1, \ldots, x_k) = \psi(f(y), x_1, \ldots, x_k)$ , где  $\psi$  — частично рекурсивная функция из доказательства теоремы 9.  $\square$ 

Значение только что доказанного предложения трудно переоценить. Большая часть современной теории алгоритмов так или иначе связана с ним. Суть доказанного можно интерпретировать следующим образом. Все аргументы функции  $\varphi_u(x_1,\ldots,x_k)$  являются натуральными числами, однако их можно разделить на 2 категории:  $x_1, \ldots, x_k$  — это натуральные числа сами по себе, которые являются входными данными для алгоритма, реализуемого программой с номером y. А аргумент y — это номер программы, который можно отождествлять с самой программой. Меняя у при фиксированных иксах, мы меняем программы, при помощи которых мы вычисляем функции от  $x_1, \ldots, x_k$ . Далее, существует программа для вычисления функции  $\varphi_{v}(x_{1},...,x_{k})$  как функции от (k+1)-го аргумента. В некотором смысле это универсальная программа. Работает она так: получив на входе набор данных  $y, x_1, \dots, x_k$ , она берет программу с номером y и запускает ее считать значение функции  $\varphi_y$  на аргументах  $x_1, \ldots, x_k$ . Работу этой программы можно уподобить работе "операционной системы", которая может вызвать любую из возможных программ, указанных "пользователем". Входными данными этой программы являются не только натуральные числа, но и другие программы.

Определим понятие универсальной функции. Пусть X, Y и Z — произвольные множества,  $k \in \mathbb{N}$  и  $\mathcal{F}$  — класс<sup>57</sup> k-местных частичных функций из Y в Z.

Определение 25 Функция  $f: X \times Y \to Z$  называется универсальной функцией для класса  $\mathcal{F}$ , если  $\mathcal{F} = \{f(x, y_1, \dots, y_k) : x \in X\}$  (здесь под  $f(x, y_1, \dots, y_k)$  понимается k-местная функция от аргументов  $y_1, \dots, y_k$  при фиксированном x).

**Теорема 10** Пусть k — натуральное число. Тогда

<sup>&</sup>lt;sup>57</sup>То есть совокупность, семейство. В данном случае можно сказать множество.

- 1. Существует (k+1)-местная частично рекурсивная функция, универсальная для класса всех k-местных частично рекурсивных функций;
- 2. Не существует 2-местной рекурсивной функции, универсальной для класса всех 1-местных рекурсивных функций;
- 3. Не существует 2-местной примитивно рекурсивной функции, универсальной для класса всех 1-местных примитивно рекурсивных функций;
- 4. Существует (k+1)-местная рекурсивная функция, универсальная для класса всех k-местных примитивно рекурсивных функций.

Доказательство. (1) Функция  $\varphi_y(x_1, \dots, x_k)$  будет универсальной для класса всех k-местных частично рекурсивных функций.

- (2) Пусть f(y,x) рекурсивная функция от двух аргументов, универсальная для класса всех рекурсивных 1-местных функций. Рассмотрим функцию g(x) = f(x,x) + 1. Функция g рекурсивна и, значит, в силу универсальности f существует  $n_0 \in \mathbb{N}$ , такое что  $g(x) = f(n_0,x)$  для любого  $x \in \mathbb{N}$ . Имеем  $f(n_0,n_0) = g(n_0) = f(n_0,n_0) + 1$ . Противоречие.
- (3) Аналогично (2), с заменой термина "рекурсивный" на "примитивно рекурсивный".
- (4) Не останавливась на подробностях, приведем лишь общую идею доказательства. Примитивно рекурсивные функции это функции, которые можно получить из базисных при помощи суперпозиции и примитивной рекурсии. Введем систему обозначений для примитивно рекурсивных функций.

Обозначениями будут являться слова алфавита  $\Sigma = \{c_0, s, I, 1, ", ", (,), [,], \mathcal{S}, \mathcal{R}\}$ . Базисная функция, равная константе 0, обозначается символом  $c_0$ . Базисная функция s — символом s. Базисная функция  $I_n^m$  обозначается словом  $I[1^m, 1^n]$ . Если w — обозначение для k-местной функции f и  $w_1, \ldots, w_k$  — обозначения для n-местных функций  $g_1, \ldots, g_k$ , то функция  $f(g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n))$  обозначается словом  $\mathcal{S}(w, w_1, \ldots, w_k)$ . Если w — обозначение для k-местной функции g и v — обозначение для (k+2)-местной функции k, то (k+1)-местная функция, получающаяся при помощи примитивной рекурсии из g и k, обозначается словом  $\mathcal{R}(w,v)$ .

Ясно, что существует алгоритм, позволяющий по любому слову алфавита  $\Sigma$  узнать, является ли это слово обозначением некоторой примитивно рекурсивной функции или просто набором символов. Существует также алгоритм, позволяющий по любому обозначению примитивно рекурсивной функции вычислить количество ее аргументов. Существует и третий алгоритм, позволяющий обозначение любой примитивно рекурсивной функции преобразовать в программу для машины Тьюринга, которая вычисляет эту функцию.

Пусть  $\nu$  — геделевская нумерация пространства  $\Sigma^*$ . Пусть  $n_0$  — номер программы, вычисляющей какую-нибудь базисную k-местную функцию. Определим функцию  $f: \mathbb{N} \to \mathbb{N}$  так.

Если слово  $\nu(i)$  не является обозначением k-местной примитивно рекурсивной функции, то полагаем  $f(i)=n_0$ . Если является, то выпишем по слову  $\nu(i)$  программу для вычисления функции, обозначаемой этим словом. Перебирая программы с номерами  $0,1,\ldots$  и сравнивая их с полученной, найдем y, который является номером нашей программы. Положим этот y равным значению f(i).

Так как имеется алгоритм для вычисления функции f, то она, по тезису Черча, является рекурсивной. Пусть теперь  $g(i,x_1,\ldots,x_k)=\varphi_{f(i)}(x_1,\ldots,x_k)$ . Анализируя все сказанное выше, нетрудно понять, что g — требуемая универсальная функция. Ясно, что она будет частично рекурсивной и всюду определенной.  $\square$ 

Основываясь на изложенной идее, можно дать формальное доказательство четвертого пункта теоремы, выписав в явном виде нумерацию  $\nu$  и функцию f. При формальном доказательстве можно добиться того, что функция f будет примитивно рекурсивной. Однако при k=1 (можно показать, что и при всех  $k\geqslant 1$ ) функция g не будет таковой никогда; иначе мы пришли бы к противоречию с третьим пунктом теоремы. Построенная при доказательстве (для k=1) функция g(y,x) — это пример рекурсивной, но не примитивно рекурсивной функции. Это функция от двух аргументов. Функция g(x,x)+1 дает пример одноместной рекурсивной, но не примитивно рекурсивной функции.

На первый взгляд может показаться, что слегка видоизменив доказательство четвертого пункта теоремы 10 и введя обозначение для оператора минимизации, можно построить рекурсивную (k+1)-местную функцию, универсальную для класса k-местных рекурсивных функций. Однако это не так (если бы мы сумели это сделать, то пришли бы к

противоречию со вторым пунктом теоремы). Дело здесь в следующем. Если у нас есть выражение, показывающее, как функция получается из базисных при помощи суперпозиции, примитивной рекурсии и минимизации, то мы не можем по этому выражению определить, будет ли эта функция всюду определенной (то есть рекурсивной) или просто частично рекурсивной. Другими словами, не существует алгоритма, который бы по записи частично рекурсивной функции давал ответ на вопрос, является ли она всюду определенной. И применив к функциям, в определении которых участвует оператор минимизации, конструкцию из четвертого пункта теоремы 10, мы можем получить только универсальную частично рекурсивную функцию, лишний раз подтвердив справедливость первого пункта этой теоремы. А с примитивно рекурсивными функциями этой проблемы не возникает, поскольку каждая примитивно рекурсивная функция заведомо является всюду определенной.

**Теорема 11** (s-m-n теорема) Пусть 0 < m < n. Тогда существует рекурсивная<sup>58</sup> (m+1)-местная функция  $s_n^m$ , такая что для всех  $y, x_1, \ldots, x_n$   $\varphi_y(x_1, \ldots, x_n) = \varphi_{s_n^m(y, x_1, \ldots, x_m)}(x_{m+1}, \ldots, x_n)$ .

Доказательство. Для доказательства воспользуемся тезисом Черча. Опишем алгоритм вычисления функции  $s_n^m$ . Пусть даны числа  $y, x_1, \ldots, x_m$ . Тогда значение  $s_n^m(y, x_1, \ldots, x_m)$  равно номеру программы, вычисляющей функцию  $\varphi_y(x_1, \ldots, x_m, x_{m+1}, \ldots, x_n)$  как функцию от переменных  $x_{m+1}, \ldots, x_n$  при фиксированных  $y, x_1, \ldots, x_m$ . Алгоритм таков. Выпишем программу с номером y. Добавим в нее команды, вставляющие в начало ленты запись  $01^{x_1} \ldots 01^{x_m}$ , изменив подходящим образом номера состояний головки. Для измененной программы найдем ее номер, перебирая программы с номерами  $0, 1, 2, \ldots$  и сравнивая их с нашей программой. Найденный номер программы объявим значением функции  $s_n^m$  от аргументов  $y, x_1, \ldots, x_m$ . Описанная процедура является алгоритмом и, значит, по тезису Черча функция  $s_n^m$  частично рекурсивна. Ясно, что она всюду определена. Теперь если мы при данных  $y, x_1, \ldots, x_m$  запишем на ленте слово  $01^{x_{m+1}} \ldots 01^{x_n}$  и запустим программу с номером  $s_n^m(y, x_1, \ldots, x_m)$ , то эта программа сначала запишет на ленте слово

 $<sup>^{58}</sup>$ Напомним, что функция  $\varphi_y(x_1,\ldots,x_n)$  определяется через геделевскую нумерацию множества программ. При подходящем выборе этой нумерации можно добиться, чтобы функция  $s_n^m$  была примитивно рекурсивной.

 $01^{x_1}\dots 01^{x_m}01^{x_{m+1}}\dots 01^{x_n}$ , а затем запустит программу с номером y работать с этими данными. Программа с номером y вычислит значение  $\varphi_y(x_1,\dots,x_n)$ . Таким образом, мы видим, что при каждых  $y,x_1,\dots,x_m$  число  $s_n^m(y,x_1,\dots,x_m)$  обладает требуемыми свойствами (то есть является номером программы, вычисляющей функцию  $\varphi_y(x_1,\dots,x_m,x_{m+1},\dots,x_n)$  как функцию от переменных  $x_{m+1},\dots,x_n$  при фиксированных  $y,x_1,\dots,x_m$ ).  $\square$ 

Следствие 5 (теорема о неподвижной точке) Пусть f(y) — рекурсивная функция и  $k \in \mathbb{N}$ . Тогда существует натуральное число n, такое что  $\varphi_n(x_1,\ldots,x_k) = \varphi_{f(n)}(x_1,\ldots,x_k)$ .

Доказательство. Чтобы не загромождать выкладки лишними индексами, обозначим через s функцию  $s_{k+1}^1$ . Функция  $\varphi_{\varphi_y(y)}(x_1,\ldots,x_k)$  является частично рекурсивной функцией от (k+1)-го аргумента и, значит,  $\varphi_{\varphi_y(y)}(x_1,\ldots,x_k)=\varphi_a(y,x_1,\ldots,x_k)=\varphi_{s(a,y)}(x_1,\ldots,x_k)$  для некоторого  $a\in\mathbb{N}$ . Пусть g(y)=f(s(a,y)). Функция g рекурсивна как суперпозиция рекурсивных функций и, значит, для некоторого числа b  $g(y)=\varphi_b(y)$ . Пусть n=s(a,b). Тогда  $\varphi_n(x_1,\ldots,x_k)=\varphi_{s(a,b)}(x_1,\ldots,x_k)=\varphi_a(b,x_1,\ldots,x_k)=\varphi_{g(b)}(x_1,\ldots,x_k)=\varphi_{g(b)}(x_1,\ldots,x_k)$ .  $\square$ 

Термин "неподвижная точка" имеет естественное происхождение. Обычно теорему о неподвижной точке применяют для функций f, которые задают алгоритмические преобразования программ. Пусть есть алгоритм, преобразующий программы в программы. Обозначим для  $y \in \mathbb{N}$  через f(y) номер программы, которая получается из программы с номером y, если ее преобразовать согласно имеющемуся алгоритму. По тезису Черча можно считать, что функция f рекурсивна. Тогда теорема о неподвижной точке, примененная к функции f, дает следующее следствие: какой бы алгоритм преобразования программ мы не взяли, всегда найдется программа, такая что она сама и полученная из нее преобразованием программы вычисляют одну и ту же функцию. Данная функция и будет "неподвижной точкой". Мы же, следуя традиции, будем допускать некоторую неточность в использовании терминов и называть неподвижной точкой не функцию, вычисляемую программой, а сам номер программы (то есть число n из формулировки следствия 5).

Теорема о неподвижной точке имеем множество интересных применений. Приведем три примера.

- 1. Различные интересные свойства функции  $\varphi_y(x_1,\dots,x_k)$ . Покажем, например, что существует a, для которого  $\varphi_a(x)=a+x$ . Двухместная функция y+x рекурсивна и, значит, для некоторого b  $y+x=\varphi_b(y,x)=\varphi_{s^1_2(b,y)}(x)$ . Пусть a— неподвижная точка для рекурсивной функции  $s^1_2(b,y)$ . Тогда для любого x  $\varphi_a(x)=\varphi_{s^1_2(b,a)}(x)=\varphi_b(a,x)=a+x$ .
- 2. Неразрешимость нумерации  $\nu_6$ , определенной на странице 86. Нумерация  $\nu_6$ , определенная на этой странице, является нумерацией множества k-местных частично рекурсивных функций и сопоставляет числу y функцию, вычислимую программой с номером y. Предположим, что эта нумерация разрешима. Тогда функция

$$\eta(y,z) = \begin{cases} 1, & \varphi_y(x_1,\ldots,x_k) = \varphi_z(x_1,\ldots,x_k) \\ 0, & \varphi_y(x_1,\ldots,x_k) \neq \varphi_z(x_1,\ldots,x_k) \end{cases}$$

является рекурсивной. Пусть  $f(y)=\mu z(\eta(y,z)=0)$ . Функция f рекурсивна и для любого y  $\varphi_y(x_1,\ldots,x_k)\neq \varphi_{f(y)}(x_1,\ldots,x_k)$ . Взяв неподвижную точку для функции f, получаем очевидное противоречие.

3. "Самовоспроизводящиеся машины". Прикрепим к машине Тьюринга устройство, которое, получив на входе записанное на ленте число n, выводит на экране монитора программу с номером n. Тогда существует программа, которая, начав работать с произвольными входными данными, заканчивает свою работу, после чего на мониторе выводится текст самой этой программы  $^{59}$ . Более точно, докажем следующее: существует  $a \in \mathbb{N}$ , такое что для любого  $x \varphi_a(x) = a$ . Базисная функция  $I_2^1(y,x)$  рекурсивна и, значит, для некоторого  $b \in \mathbb{N}$   $I_2^1(y,x) = \varphi_b(y,x) = \varphi_{s_2^1(b,y)}(x)$ . Возьмем в качестве a неподвижную точку для рекурсивной функции  $s_2^1(b,y)$ .

<sup>&</sup>lt;sup>59</sup>Есть классическая задача для программистов, которая заключается в том, чтобы написать программу, выводящую на экране свой собственный текст. Часто студенты, решающие эту задачу, хитрят и пытаются вывести на экране файл с хранящейся в нем программой. Между тем теорема о неподвижной точке утверждает, что можно написать такую программу, которая не использует команд обращения к файлам, а лишь команду вывода на экран. Такая программа может быть написана на любом из языков программирования, на которых можно промоделировать работу машины Тьюринга. В частности, на Си или Паскале.

Мы собираемся определить рекурсивные и рекурсивно перечислимые множества. При работе с этими понятиями под множеством, если не оговорено противное, будем понимать произвольное подмножество натурального ряда.

**Определение 26** Множество X называется perypcuehum, если его характеристическая функция

$$\chi_X(x) = \begin{cases} 1, & x \in X \\ 0, & x \notin X \end{cases}$$

рекурсивна.

Рекурсивные множества — это множества, для которых существует алгоритм распознавания, то есть алгоритм, позволяющий по произвольному натуральному числу отвечать на вопрос о принадлежности этого числа данному множеству <sup>60</sup>. Рекурсивными будут такие множества, как множество четных чисел, множество простых чисел и другие. Если  $\Sigma$  — конечный алфавит,  $\nu$  — геделевская нумерация пространства  $\Sigma^*$ ,  $L \subseteq \Sigma^*$  — регулярный язык и  $X = \{n : \nu(n) \in L\}$ , то множество X рекурсивно <sup>61</sup>: алгоритм распознавания для X реализуется при помощи конечного автомата.

Прежде чем давать следующее определение, докажем теорему (в формулировке этой теоремы все функции — одноместные).

**Теорема 12** Пусть  $X \subseteq \mathbb{N}$ . Тогда следующие условия равносильны:

- 1.  $X=\varnothing$  либо существует примитивно рекурсивная функция f, такая что  $X=\rho f$ ;
- 2.  $X = \emptyset$  либо существует рекурсивная функция f, такая что  $X = \rho f$ ;
- 3. существует частично рекурсивная функция  $\varphi$ , такая что  $X = \rho \varphi$ ;

 $<sup>^{60}</sup>$ В более общем случае, когда X — подмножество конструктивного пространства  $\mathcal{K}$ , принято вместо термина "рекурсивное множество" использовать термин "вычислимое множество". Про подмножество множества  $\mathbb N$  тоже можно говорить "вычислимое множество" вместо "рекурсивное множество", так как  $\mathbb N$  — частный случай конструктивного пространства.

 $<sup>^{61}</sup>$ Соответственно, множество L как подмножество конструктивного пространства  $\Sigma^*$  вычислимо.

4. существует частично рекурсивная функция  $\varphi$ , такая что  $X=\delta \varphi$ .

Доказательство.  $(1 \Rightarrow 2 \Rightarrow 3)$  очевидно.

 $(4\Rightarrow 1)$  Пусть  $X=\delta \varphi$  для частично рекурсивной функции  $\varphi$ . Если  $X=\varnothing$  (то есть функция  $\varphi$  нигде не определена), то условие 1 для X очевидно. Пусть  $X\neq\varnothing$ . Зафиксируем  $a\in X$ . Функция  $\varphi$  вычислима на машине Тьюринга. Пусть n — код программы, вычисляющей функцию  $\varphi$ . Из доказательства теоремы 9 следует, что функция

$$\varphi^t(x) = \begin{cases} \varphi(x) + 1, & \text{программа с кодом } n \text{ вычисляет } \varphi(x) \text{ не более чем} \\ & \text{за } t \text{ шагов работы} \\ 0, & \text{программа с кодом } n, \text{ вычисляя } \varphi(x), \text{ не останав-} \\ & \text{ливается через } t \text{ шагов работы} \end{cases}$$

примитивно рекурсивна (как функция от двух аргументов t и x). Пусть  $f(x) = r(x) \cdot \mathrm{sg}(\varphi^{l(x)}(r(x))) + a \cdot \overline{\mathrm{sg}}(\varphi^{l(x)}(r(x)))$ . Функция f примитивно рекурсивна. Теперь если y = f(x), то либо y = a, либо  $y = r(x) \in \delta \varphi$  и  $y \in X$ . Наоборот, если  $y \in \delta \varphi$ , то для некоторого  $t \varphi^t(y) > 0$  и y = f(c(t,y)).

 $(3\Rightarrow 4)$  Пусть  $X=\rho \varphi$  для частично рекурсивной функции  $\varphi$ . Если  $X=\varnothing$ , то  $\varphi$  — нигде не определенная функция и  $X=\delta \varphi$ . Пусть  $\delta \varphi \neq \varnothing$ . По доказанному выше существует примитивно рекурсивная функция f, такая что  $\delta \varphi = \rho f$ . Функция  $g(x) = \varphi(f(x))$  рекурсивна и  $\rho g = \rho \varphi$ . Пусть  $\psi(y) = \mu x((g(x) \dot{-} y) + (y \dot{-} g(x)) = 0)$ . Тогда функция  $\psi$  частично рекурсивна и  $X=\rho g = \delta \psi$ .  $\square$ 

**Определение 27** Множество называется *рекурсивно перечислимым*, если оно удовлетворяет одному из эквивалентных условий теоремы 12.

Термин "рекурсивно перечислимый" отсылает нас ко второму условию теоремы 12. Множество рекурсивно перечислимо, если оно перечисляется рекурсивной функцией, то есть если для некоторой рекурсивной функции f это множество равно  $\{f(0), f(1), \ldots\}$  (или пусто). Более общо, множество рекурсивно перечислимо, если существует алгоритм, позволяющий перечислять элементы этого множества  $^{62}$ .

 $<sup>^{62}</sup>$ В наиболее общей ситуации, когда речь идет о подмножествах конструктивного пространства, вместо рекурсивно перечислимых множеств говорят "вычислимо перечислимые множества". Например, если  $L\subseteq \Sigma^*$  — язык, задаваемый регулярным выражением, то L вычислимо перечислимо. Алгоритм, перечисляющий L, задается регулярным выражением для L. См. замечание и сноску на странице 36.

**Теорема 13 (Поста)** Множество X рекурсивно тогда и только тогда, когда множества X и  $\mathbb{N} \setminus X$  рекурсивно перечислимы.

Доказательство. Необходимость. Пусть X рекурсивно. Если  $X=\varnothing$  или  $X=\mathbb{N}$ , то множества X и  $\mathbb{N}\setminus X$ , очевидно, рекурсивно перечислимы. Пусть  $X\neq\varnothing$  и  $X\neq\mathbb{N}$ . Зафиксируем  $a\in X$  и  $b\in\mathbb{N}\setminus X$ . Рассмотрим рекурсивные функции  $f_1(x)=x\cdot\chi_X(x)+a\cdot\overline{\mathrm{sg}}(\chi_X(x))$  и  $f_2(x)=b\cdot\chi_X(x)+x\cdot\overline{\mathrm{sg}}(\chi_X(x))$ . Тогда  $X=\rho f_1$  и  $\mathbb{N}\setminus X=\rho f_2$ .

Достаточность. Пусть множества X и  $\mathbb{N}\setminus X$  рекурсивно перечислимы. Если  $X=\varnothing,\mathbb{N}$ , то рекурсивность X очевидна. Пусть  $X\neq\varnothing,\mathbb{N}$ . Существуют рекурсивные функции  $f_1$  и  $f_2$ , такие что  $X=\rho f_1$  и  $\mathbb{N}\setminus X=\rho f_2$ . Пусть  $f(x)=f_1(\operatorname{div}(x,2))\cdot\overline{\operatorname{sg}}(\operatorname{rest}(x,2))+f_2(\operatorname{div}(x,2))\cdot\operatorname{sg}(\operatorname{rest}(x,2))$ . Другими словами, для любого натурального k  $f(2k)=f_1(k)$  и  $f(2k+1)=f_2(k)$ . Имеем  $\rho f=\{f_1(0),f_2(0),f_1(1),\ldots\}=\mathbb{N}$ . Пусть  $g(y)=\mu x((f(x)-y)+(y-f(x))=0)$ , то есть g(y) равно минимальному x, такому что f(x)=y. Функция g рекурсивна и  $\chi_X(y)=\overline{\operatorname{sg}}(\operatorname{rest}(g(y),2))$ .

Действуя менее формально, можно было доказать достаточность, используя тезис Черча. Для этого требуется описать алгоритм, вычисляющий функцию  $\chi_X$ . Алгоритм таков. При данном  $y \in \mathbb{N}$  запускаем параллельно процедуры перечисления множеств X и  $\mathbb{N} \setminus X$ , после чего ждем, в каком из этих множеств появится y. Если он появился в первом множестве, то полагаем  $\chi_X(y) = 1$ , если во втором —  $\chi_X(y) = 0$ .  $\square$ 

В качестве одного из следствий теоремы Поста можно отметить следующий факт: каждое рекурсивное множество является рекурсивно перечислимым. Это естественно, поскольку если есть алгоритм для распознавания принадлежности натуральных чисел множеству X, то алгоритм перечисления X построить очень просто: нужно последовательно проверять числа  $0, 1, \ldots$  на принадлежность множеству X и перечислять те натуральные числа, которые множеству X принадлежат.

Через  $\mathcal{E}$  обозначим семейство всех рекурсивно перечислимых множеств. Пусть для  $n \in \mathbb{N}$   $W_n$  — это область определения функции  $\varphi_n(x)$ . Отображение  $n \mapsto W_n$  является нумерацией семейства  $\mathcal{E}$ . Эта нумерация известна как *клиниевская нумерация* семейства всех рекурсивно перечислимых множеств. Клиниевская нумерация — один из центральных объектов теории алгоритмов. В нашем курсе мы не будем останавливаться на вопросах, связанных с этим понятием. Отметим лишь, что клиниевская нумерация обладает следующим свойством: по номеру n можно эффек-

тивно (то есть при помощи алгоритма) выписать процедуру, которая перечисляет элементы множества  $W_n$ . Эта процедура такова. При данном n запускаем программу с номером n считать значения  $\varphi_n(0), \varphi_n(1), \ldots$  (требуется запустить бесконечное число параллельных процессов вычисления, однако легко добиться того, что в каждый конкретный момент времени будет работать лишь конечное число этих процессов) и, по мере того как натуральные числа попадают в область определения функции  $\varphi_n$ , перечисляем их в множество  $W_n$ .

Ряд интересных свойств клиниевской нумерации связан с теоремой о неподвижной точке. Непосредственное следствие этой теоремы гласит, что для каждой рекурсивной функции f(x) существует число n, такое что  $W_n = W_{f(n)}$ . В качестве примера покажем, что существует  $m \in \mathbb{N}$ , такое что  $W_m = \{m\}$ . Пусть  $\psi(y, x)$  — частично рекурсивная функция, равная 1 при x = y и не определенная в остальных случаях. Для некоторого  $a \in \mathbb{N}$  имеем  $\psi(y, x) = \varphi_a(y, x) = \varphi_{s_2^1(a, y)}(x)$ . Теперь в качестве m можно взять неподвижную точку для рекурсивной функции  $s_2^1(a, y)$ .

Через K обозначим множество  $\{x:\varphi_x(x) \text{ определено}\}$ . Это же множество можно записать и так:  $K=\{x:x\in W_x\}$ . Множество K рекурсивно перечислимо, так как является областью определения частично рекурсивной функции  $\varphi_x(x)$ .

## Предложение 14 Справедливы следующие утверждения.

- 1. Семейство рекурсивных множеств замкнуто относительно операций объединения, пересечения и дополнения.
- 2. Семейство рекурсивно перечислимых множеств замкнуто относительно операций объединения и пересечения.
- 3. Существуют рекурсивно перечислимые, но не рекурсивные множества.

 $\mathcal{A}$ оказательство. (1) Пусть X и Y — рекурсивные множества. Тогда  $\chi_{X \cup Y}(x) = \mathrm{sg}(\chi_X(x) + \chi_Y(x)), \; \chi_{X \cap Y}(x) = \chi_X(x) \cdot \chi_Y(x)$  и  $\chi_{\mathbb{N} \setminus X}(x) = \overline{\mathrm{sg}}(\chi_X(x))$  — рекурсивные функции.

(2) Пусть X и Y — рекурсивно перечислимые множества. Для некоторых частично рекурсивных функций  $\psi_1$  и  $\psi_2$  имеем  $X = \delta \psi_1$  и  $Y = \delta \psi_2$ . Пусть  $\psi(x) = \psi_1(x) + \psi_2(x)$ . Тогда  $\psi$  — частично рекурсивная функция и  $\delta \psi = X \cap Y$ .

Докажем замкнутость относительно объединения. Если  $X=\varnothing$  или  $Y=\varnothing$ , то рекурсивная перечислимость множества  $X\cup Y$  очевидна. В противном случае существуют рекурсивные функции  $f_1(x)$  и  $f_2(x)$ , такие что  $X=\rho f_1$  и  $Y=\rho f_2$ . Пусть  $f(x)=f_1({\rm div}(x,2))\cdot \overline{\rm sg}({\rm rest}(x,2))+f_2({\rm div}(x,2))\cdot {\rm sg}({\rm rest}(x,2))$ , то есть  $f(2k)=f_1(k)$  и  $f(2k+1)=f_2(k)$ . Тогда f — рекурсивная функция и  $X\cup Y=\rho f$ .

(3) Пусть множество K рекурсивно. Полагаем

$$\psi(x) = \begin{cases} \varphi_x(x) + 1, & x \in K \\ 0, & x \notin K. \end{cases}$$

По тезису Черча функция  $\psi$  рекурсивна<sup>63</sup>. Алгоритм для вычисления  $\psi(x)$  таков: при данном x надо сначала, пользуясь рекурсивностью функции  $\chi_K$ , определить, верно ли, что  $x \in K$ . Если  $x \notin K$ , то нужно сразу положить  $\psi(x) = 0$ , не производя никаких дальнейших вычислений. Если же  $x \in K$ , то надо вычислить  $\varphi_x(x)$ , прибавить к этому значению 1 и положить  $\psi(x)$  равным полученному числу.

Так как  $\psi$  — рекурсивная функция, то  $\psi(x) = \varphi_a(x)$  для некоторого  $a \in \mathbb{N}$ . Теперь если  $a \in K$ , то  $\varphi_a(a)$  определено и  $\varphi_a(a) = \psi(a) = \varphi_a(a) + 1$ , чего не может быть. Если же  $a \notin K$ , то с одной стороны  $\varphi_a(a)$  не определено, а с другой стороны  $\varphi_a(a) = \psi(a) = 0$ . Противоречие.  $\square$ 

На примере множества K мы видим, что в общем случае задача распознавания элементов множества и задача перечисления множества не равносильны (см. стр. 36). Существует алгоритм, позволяющий перечислять элементы K, но алгоритма, который позволял бы по любому натуральному числу определить, является ли оно элементом K, не существует. Если дано  $x \in \mathbb{N}$ , то мы можем запустить алгоритм перечисления K и ждать до тех пор, пока число x не будет перечислено. Если это произойдет, то можно остановить процедуру перечисления и дать утвердительный ответ на вопрос о принадлежности x множеству K. Однако если перечисление продолжается, а число x так и не появляется в списке элементов множества K, то мы на каждом шаге перечисления стоим перед дилеммой: либо прервать процесс перечисления и признать, что

<sup>&</sup>lt;sup>63</sup>Можно дать формальное доказательство рекурсивности  $\psi$ , не использующее тезис Черча. Ясно, что  $K \neq \varnothing$  (например, если x — номер всюду определенной функции, то  $x \in K$ ). Зафиксируем  $x_0 \in K$ . Пусть  $f(x) = x \cdot \chi_K(x) + x_0 \cdot \overline{\operatorname{sg}}(\chi_K(x))$ . Тогда  $\psi(x) = \chi_K(x) \cdot (\varphi_{f(x)}(f(x)) + 1)$ .

 $x \notin K$ , либо подождать еще немного и тогда x появится в упомянутом списке. Алгоритма, который разрешал бы эту дилемму, не существует, так как не существует алгоритма для ответа на вопрос о принадлежности числа x множеству K.

Итак, существуют множества, для которых нет алгоритма распознавания. Можно остановиться на этом, констатируя факт наличия рекурсивных и нерекурсивных множеств. Но можно пойти дальше и задаться странным на первый взгляд вопросом: какие из нерекурсивных множеств являются более нерекурсивными, а какие менее? Однако этот вопрос не такой уж и странный. Похожая ситуация имеет место и в связи с понятием мощности множества. Когда-то множества просто делились на конечные и бесконечные, и все бесконечные множества считались бесконечными в одинаковой степени. Потом ввели понятие мощности, и оказалось, что бесконечные множества делятся на счетные и несчетные и что, например, множество действительных чисел содержит больше элементов, чем множество натуральных чисел.

Существует множество подходов, позволяющих различать различные множества по степени их рекурсивности. Один из наиболее ранних связан с понятием m-сводимости<sup>64</sup>.

Определение 28 Пусть  $X, Y \subseteq \mathbb{N}$ . Тогда X m-сводится к Y, если существует рекурсивная функция  $f : \mathbb{N} \to \mathbb{N}$ , такая что для любого  $n \in \mathbb{N}$   $n \in X$  тогда и только тогда, когда  $f(n) \in Y$ .

Если X m-сводится к Y, то мы пишем  $X\leqslant_m Y$ . Отношение m-сводимости рефлексивно и транзитивно. Действительно,  $X\leqslant_m X$  при помощи функции f(x)=x для любого  $X\subseteq\mathbb{N}$ . Если же X m-сводится к Y при помощи функции f, а Y m-сводится к Z при помощи функции g, то X m-сводится к Z при помощи функции  $g \circ f$ .

Пусть  $X \leq_m Y$ . Несмотря на то, что оба множества X и Y могут быть нерекурсивными, можно считать, что X более рекурсивно, чем Y. Хотя Y может быть нерекурсивным, предположим, что мы каким-то чудом научились вычислять функцию  $\chi_Y(x)$ . Допустим, в наше распоряжение попал некий черный ящик, некое непонятное устройство, принципов работы которого мы не знаем, но которое, получив на входе число x, дает на выходе значение функции  $\chi_Y(x)$  (в теории алгоритмов такие устройства

 $<sup>^{-64}</sup>$ Термин "m-сводимость", или "много-одно-сводимость" — это перевод английского термина "many-one-reducibility". Собственно, из этого "many" и взялась буква m.

называются оракулами). Тогда при помощи этого устройства мы можем вычислять не только функцию  $\chi_Y$ , но и функцию  $\chi_X$ . Действительно, если X m-сводится к Y при помощи функции f, то при данном  $x \in \mathbb{N}$  можно сначала, не пользуясь никакими оракулами, вычислить f(x), а затем, обратившись к оракулу, узнать, чему равно  $\chi_Y(f(x))$  и положить  $\chi_X(x)$  равным этому значению.

Если  $X \leqslant_m Y$  и  $Y \leqslant_m X$ , то мы скажем, что X m-эквивалентно Y и будем писать  $X \equiv_m Y$ . Легко проверить, что отношение  $\equiv_m$  является отношением эквивалентности. Классы эквивалентности этого отношения называются m-степенями. Если у нас есть оракул для одного из множеств, принадлежащих некоторой m-степени, то характеристическую функцию любого из множеств, принадлежащих этой m-степени, можно вычислить, используя только этот оракул. Можно сказать, что все множества из одной m-степени "рекурсивны" (или "нерекурсивны") в одинаковой степени.

Легко понять, что к пустому множеству m-сводится только само пустое множество, а к множеству  $\mathbb{N}$  — только само множество  $\mathbb{N}$ . Также ясно, что оба этих множества m-сводятся ко всем остальным множествам. Исключим оба этих множества из рассмотрения при изучении вопросов, касающихся m-сводимости.

## Предложение 15 Справедливы следующие утверждения.

- 1. Всякое рекурсивное множество m-сводится к любому множеству.
- 2. Если множество m-сводится к рекурсивному множеству, то оно рекурсивно.
- 3. Если множество m-сводится к рекурсивно перечислимому множеству, то оно рекурсивно перечислимо.
- 4. Всякое рекурсивно перечислимое множество m-сводится к множеству K.

Доказательство. (1) Пусть X рекурсивно и  $Y \subseteq \mathbb{N}$ . Так как мы исключили из рассмотрения множества  $\emptyset$ ,  $\mathbb{N}$ , то можно считать, что существуют  $a \in Y$  и  $b \notin Y$ . Пусть  $f(x) = a \cdot \chi_X(x) + b \cdot \overline{\operatorname{sg}}(\chi_X(x))$ . Тогда  $x \in X \leftrightarrow f(x) \in Y$  и рекурсивная функция f сводит  $X \ltimes Y$ .

- (2) Пусть Y рекурсивно и  $X \leq_m Y$ . Существует рекурсивная функция f, такая что  $x \in X \leftrightarrow f(x) \in Y$  для всех  $x \in \mathbb{N}$ . Но тогда  $\chi_X = \chi_Y \circ f$  рекурсивная функция.
- (3) Пусть Y рекурсивно перечислимо и  $X \leqslant_m Y$ . Существует частично рекурсивная функция  $\psi$ , такая что  $Y = \delta \psi$ . Существует рекурсивная функция f, такая что  $x \in X \leftrightarrow f(x) \in Y$  для всех  $x \in \mathbb{N}$ . Пусть  $\varphi(x) = \psi(f(x))$ . Функция  $\varphi$  частично рекурсивна и  $X = \delta \varphi$ .
- (4) Пусть X рекурсивно перечислимое множество. Существует частично рекурсивная функция  $\psi$ , такая что  $X = \delta \psi$ . Пусть  $\varphi(x,y) = \psi(x) + y$ . Тогда  $\varphi$  частично рекурсивная 2-местная функция и для  $x,y \in \mathbb{N}$   $\varphi(x,y)$  определено тогда и только тогда, когда  $x \in X$ . Для некоторого  $a \in \mathbb{N}$  имеем  $\varphi(x,y) = \varphi_a(x,y) = \varphi_{s_2^1(a,x)}(y)$ . Пусть  $f(x) = s_2^1(a,x)$ . Если  $x \in X$ , то  $\varphi_{f(x)}(y)$  определена при всех y; в частности, и при y = f(x). Если же  $x \notin X$ , то значение  $\varphi_{f(x)}(y)$  не определено ни при каком y. Получаем, что  $x \in X \leftrightarrow f(x) \in K$  и функция f m-сводит множество X к множеству K.  $\square$

Как показывает предложение 15, все рекурсивные множества (за исключением  $\emptyset$ ,  $\mathbb{N}$ ) образуют одну единственную m-степень, которая является наименьшей среди всех m-степеней (любое множество из этой m-степени m-сводится к множеству из любой другой m-степени). Из пункта 3 этого предложения следует, что если m-степень содержит хотя бы одно рекурсивно перечислимое множество, то она вся состоит из рекурсивно перечислимых множеств. Степень, состоящую из рекурсивно перечислимых множеств, назовем pekypcusho nepevucnumoŭ. Четвертый пункт предложения 15 показывает, что m-степень множества K является наибольшей среди всех рекурсивно перечислимых m-степеней  $^{65}$ . В частности, если где-нибудь удастся добыть оракул для K, то тогда все рекурсивно перечислимые множества станут "рекурсивными"; для них появится алгоритм распознавания (использующий оракул для K).

Приведем еще два примера множеств, принадлежащих наибольшей рекурсивно перечислимой m-степени.

**Предложение 16** Следующие множества m-эквивалентны.

1. Множество  $K = \{x : \varphi_x(x) \text{ определено}\}.$ 

 $<sup>^{65}</sup>$ Возникает естественный вопрос: существуют ли какие-нибудь другие рекурсивно перечислимые степени, отличные от степени рекурсивных множеств и от степени множества K? Положительный ответ на этот вопрос получен Постом в 1944 году.

- 2. Множество  $K_1 = \{c(x, y) : \varphi_x(y) \text{ определено}\}.$
- 3. Множество  $K_2 = \{x : W_x \neq \emptyset\}.$

Доказательство. Достаточно показать наличие m-сводимостей  $K \leqslant_m K_1 \leqslant_m K_2 \leqslant_m K$ .

Множество K сводится к множеству  $K_1$  при помощи функции c(x,x). Покажем, что  $K_1$  сводится к  $K_2$ . Пусть  $\psi(x,y)=\varphi_{l(x)}(r(x))+y$ . Тогда  $\psi$  — частично рекурсивная функция и значение  $\psi(x,y)$  определено тогда и только тогда, когда  $x\in K_1$ . Для некоторого  $a\in\mathbb{N}$  имеем  $\psi(x,y)=\varphi_a(x,y)=\varphi_{s_2^1(a,x)}(y)$ . Пусть  $f(x)=s_2^1(a,x)$ . Если  $x\in K_1$ , то для некоторого (на самом деле для любого) y значение  $\psi(x,y)=\varphi_{f(x)}(y)$  определено и  $W_{f(x)}\neq\varnothing$ . Если же  $x\not\in K_1$ , то  $\psi(x,y)=\varphi_{f(x)}(y)$  не определено ни при каком y и  $W_{f(x)}=\varnothing$ . Получаем  $x\in K_1\leftrightarrow f(x)\in K_2$  и рекурсивная функция f сводит  $K_1$  к  $K_2$ .

Для доказательства сводимости  $K_2 \leqslant_m K$  достаточно показать, что  $K_2$  рекурсивно перечислимо и сослаться на предложение 15. Множество  $K_1$  рекурсивно перечислимо, так как является областью определения функции  $\varphi_{l(x)}(r(x))$ . Значит, существует рекурсивная функция f, такая что  $K_1 = \rho f$ . Нетрудно проверить, что  $K_2 = \rho g$ , где g(x) = l(f(x)) — рекурсивная функция.  $\square$ 

Множество  $K_1$  называют *множеством проблемы остановки* <sup>66</sup>. Сама проблема остановки заключается в следующем: по алгоритму и входным данным требуется определить, закончит ли алгоритм свою работу при этих входных данных или "зациклится" и будет работать бесконечно. Переходя к геделевским номерам алгоритмов и входных данных, проблеме остановки можно придать следующий вид: по данным x и y определить, закончит ли программа с номером y свою работу через конечное число шагов, если на ленте изначально записано слово  $01^x$ . Ясно, что это произойдет тогда и только тогда, когда значение  $\varphi_y(x)$  определено, то есть когда  $c(y,x) \in K_1$ . Если бы множество  $K_1$  было рекурсивным, то и проблема остановки была бы алгоритмически разрешимой. Однако это не так, поскольку  $K \leqslant_m K_1$ , а множество K не рекурсивно. Таким образом, проблема остановки неразрешима. Такое свойство алгоритмов, как

 $<sup>^{66}</sup>$ В связи с наличием эквивалентности  $K_1 \equiv_m K$  иногда множеством проблемы остановки называют множество K.

способность "зацикливаться" является принципиальной трудностью, которую не может обойти ни одна теория, рассматривающая алгоритмы в самом общем смысле.

Достаточно легко привети пример множества, которое не будет рекурсивно перечислимым. Таково, например, множество  $\mathbb{N} \setminus K$ . Если бы это множество было рекурсивно перечислимым, то по теореме 13 множество K было бы рекурсивным, что противоречит предложению 14. Вообще, ни одно дополнение к рекурсивно перечислимому, но не рекурсивному множеству не будет рекурсивно перечислимым.

Приведем более принципиальный пример. Пусть  $T = \{x : \text{функция} \varphi_x(y) \text{ рекурсивна}\}$ . Это же множество можно записать и по другому:  $T = \{x : W_x = \mathbb{N}\}$ .

**Предложение 17** Множества T и  $\mathbb{N} \setminus T$  не являются рекурсивно перечислимыми.

Доказательство. Пусть T рекурсивно перечислимо. Так как  $T \neq \emptyset$ , то существует рекурсивная функция f, такая что  $T = \rho f$ . Но тогда  $g(y,x) = \varphi_{f(y)}(x)$  — универсальная рекурсивная функция для класса всех одноместных рекурсивных функций, что противоречит теореме 10.

Пусть теперь  $\mathbb{N}\setminus T$  рекурсивно перечислимо. Тогда существует частично рекурсивная функция  $\psi$ , такая что  $\mathbb{N}\setminus T=\delta\psi$ . Пусть  $\varphi(x,y)=\psi(x)+y$ . Имеем  $\varphi(x,y)=\varphi_a(x,y)=\varphi_{s_2^1(a,x)}(y)$  для некоторого  $a\in\mathbb{N}$ . Пусть f(x) — рекурсивная функция, равная  $s_2^1(a,x)$ . Если  $x\in\delta\psi$ , то для любого y значение  $\varphi(x,y)$  определено и, значит,  $\varphi_{f(x)}(y)$  — рекурсивная функция и  $W_{f(x)}=\mathbb{N}$ . Если же  $x\not\in\delta\psi$ , то для любого y  $\varphi_{f(x)}(y)$  не определено и  $W_{f(x)}=\varnothing$ . Таким образом,  $W_x\neq\mathbb{N}\leftrightarrow x\not\in T\leftrightarrow x\in\delta\psi\leftrightarrow W_{f(x)}=\mathbb{N}$ . Пусть b — неподвижная точка для f. Получаем  $W_b\neq\mathbb{N}\leftrightarrow W_b=W_{f(b)}=\mathbb{N}$ . Противоречие.  $\square$ 

Заканчивая этот раздел курса, сделаем заключительное замечание. Развитие теории алгоритмов можно разделить на несколько этапов. Самый ранний этап (примерно  $1930-1950\ {\rm rr.})$  — это этап становления теории. На этом этапе были введены понятия частично рекурсивной функции и машины Тьюринга, рекурсивного и рекурсивно перечислимого множества, понятие m-сводимости, доказана теорема о неподвижной точке. Тогда же был сформулирован тезис Черча. Все, что мы сделали в нашем курсе, появилось именно на этом раннем этапе.

Следующий этап начался вместе с изобретением так называемого метода приоритета. С тех пор за полвека было получено множество глубоких и нетривиальных результатов. Было бы очень интересно изложить хотя бы часть этих достижений в нашем курсе. К сожалению, объем курса ограничен и мы вынуждены прервать изложение теории вычислимости там, где она, по сути, только начинается. Тех, кто заинтересовался проблемами теории вычислимости, автор отсылает к классической монографии Х. Роджерса<sup>67</sup> и к появившейся недавно в русском переводе книге Р. Соара<sup>68</sup>. Книга Роджерса также рекомендуется в качестве учебного пособия по этому разделу курса. Наряду с книгой Роджерса автор рекомендует также доступные в библиотеке НГУ книгу А. И. Мальцева<sup>69</sup> и учебник Ю. Л. Ершова и Е. А. Палютина по математической логике<sup>70</sup>.

### 4 Сложность вычислений 71

Пусть  $f: \mathcal{K}_1 \to \mathcal{K}_2$  — функция, отображающая конструктивное пространство  $\mathcal{K}_1$  в конструктивное пространство  $\mathcal{K}_2$ . На практике главный вопрос обычно заключается не в том, вычислима ли f, а в том, вычислима ли f практически. Другими словами, существует ли программа, вычисляющая f за приемлемое время и в том объеме памяти, которым мы располагаем? Ответ отчасти зависит от нашего мастерства в написании программ и возможностей наших компьютеров. Однако есть еще один дополнительный фактор, связанный с "внутренней сложностью" функции f. Обсуждать это понятие позволяет теория вычислительной сложности, к рассмотрению которой мы сейчас переходим. В связи с ограниченным объемом курса мы удовлетворимся беглым рассмотрением трех интересных вопросов, связанных с этой теорией.

 $<sup>^{67}{\</sup>rm X}.$  Роджерс, Теория рекурсивных функций и эффективная вычислимость, "Мир", Москва, 1972.

<sup>&</sup>lt;sup>68</sup>Р. Соар, Вычислимо перечислимые множества и степени, "Казанское математическое общество", Казань, 2000.

<sup>69</sup> А. И. Мальцев, Алгоритмы и рекурсивные функции, "Наука", Москва, 1965.

<sup>&</sup>lt;sup>70</sup>Ю. Л. Ершов, Е. А. Палютин, Математическая логика, "Наука", Москва, 1979. Есть также более поздние издания этой книги.

<sup>&</sup>lt;sup>71</sup>Второй и третий разделы этой части курса носит обзорный характер. Будут представлены только общие схемы определений и доказательств. Читателю, который хочет изучить теорию сложности более детально, автор советует обратиться к специальной литературе.

## 4.1 Примитивно рекурсивные функции

Интуитивно ясно, что вычисление примитивно рекурсивной функции должно занимать меньше ресурсов, чем вычисление произвольной рекурсивной функции. Действительно, примитивно рекурсивные функции получаются из базисных при помощи суперпозиции и примитивной рекурсии, без применения минимизации. Количество шагов, требующихся для вычисления таких функций, заранее предсказуемо, поскольку в алгоритме вычисления примитивно рекурсивной функции, который можно предложить исходя из ее определения, отсутствует момент ожидания наступления некоторого события<sup>72</sup>. Попытаемся немного уточнить эти довольно расплывчатые соображения.

Предположим, что у нас имеется универсальное вычислительное устройство для работы с числовыми функциями, которое может работать по заранее введенной программе и, шаг за шагом преобразовывая полученные на входе данные, через конечное число шагов давать результат вычисления. Типичным примером такого устройства является машина Тьюринга. Однако с тем же успехом это может быть и машина Шенфилда, и даже обычный персональный компьютер, модернизированный так, чтобы он мог работать с бесконечной памятью. Первое, что мы требуем от этого устройства — чтобы оно могло вычислять любую частично рекурсивную функцию (зацикливаясь, если входные данные не попадают в ее область определения)<sup>73</sup>.

Пусть P — произвольная программа для этого устройства. Через  $\varphi_P(x_1,\ldots,x_k)$  обозначим частичную k-местную функцию, вычисляемую программой P. Пусть  $t_P(x_1,\ldots,x_k)$  — количество шагов, которое требуется программе P для вычисления значения  $\varphi_P(x_1,\ldots,x_k)$  (если  $\varphi_P(x_1,\ldots,x_k)$  не определено, то  $t_P(x_1,\ldots,x_k)$  также не определено).

Мы считаем, что для каждого натурального числа t наше устрой-

 $<sup>^{72}</sup>$ Можно сказать, что вычисление примитивно рекурсивной функции можно запрограммировать, используя только цикл "for" и не привлекая циклы "while" и "repeat".

<sup>&</sup>lt;sup>73</sup>Ниже мы перечислим три других требования. Кроме того, есть еще одно требование к нашему устройству, которое мы в тексте явно не формулируем. Оно касается полиномиального времени (достаточно даже примитивно рекурсивного времени) "перемещения данных в памяти устройства". Несмотря на то, что это требование трудно сформулировать для абстрактного устройства, у которого даже не определено понятие памяти, для многих конкретных устройств (имеющих память) оно имеет вполне определенный смысл. В частности, для машин Тьюринга и машин Шенфилда это требование можно строго сформулировать и доказать.

ство может предоставить нам некий "промежуточный" результат вычисления, полученный после t шагов работы по программе P. Если  $t \geqslant t_P(x_1,\ldots,x_k)$ , то этот результат обязан быть равен  $\varphi_P(x_1,\ldots,x_k)$ ; в противном случае он может быть равен, в принципе, чему угодно. Через  $\varphi_P^t(x_1,\ldots,x_k)$  обозначим значение этого результата. Другие требования, предъявляемые нами к нашему устройству, таковы:

- 1. Для любой программы P(k+1)-местная функция  $\varphi_P^t(x_1,\ldots,x_k)$  примитивно рекурсивна;
- 2. Для любой программы P предикат  $R(t, x_1, \ldots, x_k) =$  " $t = t_P(x_1, \ldots, x_k)$ " примитивно рекурсивен;
- 3. Если  $f(x_1,\ldots,x_k)$  базисная k-местная функция, то существует программа P, такая что  $\varphi_P=f$  и функция  $t_P$  примитивно рекурсивна.

Из доказательства теоремы 9 следует, что машина Тьюринга обладает всеми нужными свойствами. Можно доказать, что многие другие вычислительные устройства (например, машина Шенфилда) также обладают ими.

**Предложение 18** Функция  $f(x_1, ..., x_k)$  примитивно рекурсивна тогда и только тогда, когда существует программа P, такая что  $f(x_1, ..., x_k) = \varphi_P(x_1, ..., x_k)$  и функция  $t_P(x_1, ..., x_k)$  примитивно рекурсивна.

Доказательство. Достаточность очевидна, так как  $f(x_1,\ldots,x_k)=\varphi_P^{t_P(x_1,\ldots,x_k)}(x_1,\ldots,x_k)$ . Докажем необходимость.

Все базисные функции обладают нужным свойством согласно требованиям, которые мы предъявляем к нашему устройству. Покажем, что если какие-то функции обладают требуемым свойством, то и функции, которые получаются из них суперпозицией или примитивной рекурсией, также им обладают.

Суперпозиция. Пусть  $f(x_1,\ldots,x_k)=g(h_1(x_1,\ldots,x_k),\ldots,h_n(x_1,\ldots,x_k))$ . Алгоритм, вычисляющий функцию f, таков: сначала вычислить все значения  $h_1(x_1,\ldots,x_k),\ldots,h_n(x_1,\ldots,x_k)$ , а затем для них вычислить функцию g. Пусть Q— программа, реализующая этот алгоритм. Количество шагов вычисления, которое должна сделать программа Q, не меньше чем  $s=t_{P_1}(x_1,\ldots,x_k)+\cdots+t_{P_n}(x_1,\ldots,x_k)+t_{P}(h_1(x_1,\ldots,x_k),\ldots,h_n(x_1,\ldots,x_n))$ 

 $(x_i, x_k)$ ), где  $P_i$  — программа, вычисляющая  $h_i$ , а P — программа для вычисления g. По предположению все  $t_P$ :-ые и  $t_P$  примитивно рекурсивны; значит, функция  $s = s(x_1, \dots, x_k)$  примитивно рекурсивна. В принципе, программа Q никаких других действий, кроме вычисления  $h_i$ -ых и g не производит; однако ей может потребоваться еще некоторое количество шагов для того, чтобы организовать распределение данных в памяти устройства (копирование данных из одних ячеек памяти в другие для того, чтобы было возможно вызвать  $P_i$ -ые и P в качестве подпрограмм и т.п.). Количество шагов, которое необходимо для этого, сравнимо с размером данных, которые машине приходится перемещать; можно считать, что оно не превосходит  $r(x_1, ..., x_k) = p(x_1, ..., x_k, h_1(x_1, ..., x_k), ...,$  $h_n(x_1,\ldots,x_k), f(x_1,\ldots,x_k)$ ), где p — некоторый полином, зависящий от того, какое устройство мы выбрали для вычислений $^{74}$ . Таким образом,  $t_{Q}(x_{1},...,x_{k}) \leqslant s(x_{1},...,x_{k}) + r(x_{1},...,x_{k})$ , то есть  $t_{Q}$  ограничено сверху примитивно рекурсивной функцией. Осталось заметить, что  $t_Q(x_1, \ldots, x_k)$  $= (\mu i \leqslant s(x_1, \dots, x_k) + r(x_1, \dots, x_k)) ("i = t_Q(x_1, \dots, x_k)").$ 

Примитивная рекурсия. Этот случай разбирается почти так же, как предыдущий. Пусть

$$\begin{cases}
f(0, x_1, \dots, x_k) &= g(x_1, \dots, x_k) \\
f(t+1, x_1, \dots, x_k) &= h(f(t, x_1, \dots, x_k), t, x_1, \dots, x_k)
\end{cases}$$

и P,Q — программы, вычисляющие функции g и h соответственно, для которых функции  $t_P$  и  $t_Q$  примитивно рекурсивны. Алгоритм для вычисления f таков: при данных  $t,x_1,\ldots,x_k$  последовательно вычисляем значения  $f(0,x_1,\ldots,x_k), f(1,x_1,\ldots,x_k),\ldots,f(t,x_1,\ldots,x_k),$  пользуясь программами P и Q; полагаем результат равным последнему члену этой последовательности. Пусть R — программа, реализующая этот алгоритм. Количество шагов вычисления, которое должна сделать эта программа, не меньше чем  $s=t_P(x_1,\ldots,x_k)+t_Q(f(0,x_1,\ldots,x_k),t,x_1,\ldots,x_k)+\cdots+t_Q(f(t-1,x_1,\ldots,x_k),t,x_1,\ldots,x_k)$ . Количество шагов, которые тратятся на перемещение данных в памяти, можно ограничить примитивно рекурсивной функцией  $r(t,x_1,\ldots,x_k)=p(t,x_1,\ldots,x_k,\sum_{i=0}^t f(i,x_1,\ldots,x_k))$ , где p — некоторый полином. Окончательно получаем  $t_R(x_1,\ldots,x_k)\leqslant s(x_1,\ldots,x_k)+r(x_1,\ldots,x_k)$  и  $t_R(x_1,\ldots,x_k)=(\mu i\leqslant s(x_1,\ldots,x_k)+r(x_1,\ldots,x_k))$  (" $i=t_R(x_1,\ldots,x_k)$ ").  $\square$ 

 $<sup>^{74}</sup>$ Для конкретных устройств, таких как машина Тьюринга, этот факт можно строго доказать.

Таким образом, примитивно рекурсивная функция — это такая рекурсивная функция, время вычисления которой можно ограничить . . . другой примитивно рекурсивной функцией. На первый взгляд, получается порочный круг. Однако это не так. Ведь время вычисления первой функции мы ограничиваем не временем вычисления, а значениями второй функции. Осталось разобраться с тем, насколько быстро могут расти примитивно рекурсивные функции<sup>75</sup>.

Рассмотрим 3-местную функцию Аккермана  $^{76}$  A(z,x,y). Эта функция определяется следующей системой равенств:

$$\begin{bmatrix}
A(0, x, y) & = x + y \\
A(z + 1, x, 0) & = sg(z) \\
A(z + 1, x, y + 1) & = A(z, x, A(z + 1, x, y))
\end{bmatrix}$$

Принцип построения функции A(z,x,y) прост. Скажем, что при фиксированном z 2-местная функция A(z,x,y) — это операция уровня z над числами x и y. Тогда операция нулевого уровня — это сложение, а операция (z+1)-го уровня получается применением операции уровня z к числу x y раз. В частности, A(1,x,y)=xy (x, сложенный сам с собой y раз),  $A(2,x,y)=x^y$  (x, умноженный сам на себя y раз) и т. д.

# **Предложение 19** Функция A(z, x, y) рекурсивна.

Доказательство. Можно дать формальное доказательство  $^{77}$ . Однако мы воспользуемся тезисом Черча. Приведем текст программы (на "псевдопаскале", то есть на языке Паскаль, для которого переменная типа integer может принимать любые натуральные значения), вычисляющей функцию A.

```
function A(z,x,y): integer): integer; begin if z=0 then A:=x+y else if y=0 then A:=sg(z-1) else A:=A(z-1,x,A(z,x,y-1)) end;
```

 $<sup>^{75}</sup>$ Для функции из  $\mathbb{N}$  в  $\mathbb{N}$  можно определить так называемую "скорость роста". Например, скажем, что f растет быстрей чем g, если для всех  $x \in \mathbb{N}$  f(x) > g(x).

 $<sup>^{76}</sup>$ Известную также как обобщенная экспонента Аккермана.

 $<sup>^{77}</sup>$ Оно будет даже короче, чем то, что мы собираемся привести ниже. Оставляем это доказательство читателю в качестве упражнения.

Здесь используется рекурсивный вызов функции A. Чтобы доказать, что для любых x,y,z эта программа вычислит A(z,x,y) за конечное время, достаточно показать, что в ходе своей работы функция A вызывает сама себя конечное число раз. Предположим противное. Тогда существует бесконечная цепь рекурсивных вызовов функции A. Пусть  $(z_0,x,y_0),(z_1,x,y_1),\ldots$  — последовательность параметров, с которыми функция A вызывает себя вдоль этой бесконечной цепи. Тогда для любого  $i\in\mathbb{N}$  либо  $z_{i+1}< z_i$ , либо  $z_{i+1}=z_i$  и  $y_{i+1}< y_i$ . Так как  $z_i$  не увеличивается, то для некоторого  $i_0\in\mathbb{N}$   $z_i=z_{i_0}$  для всех  $i\geqslant i_0$ . Но тогда  $y_{i_0}>y_{i_0+1}>\ldots$  — бесконечная строго убывающая последовательность натуральных чисел. Противоречие.  $\square$ 

**Лемма 6** Для всех  $z, y, y_1, y_2 \in \mathbb{N}$  справедливы следующие соотношения:

```
1. A(z+2,2,0) = 1, A(z+2,2,1) = 2, A(z+2,2,2) = 4;
```

- 2.  $A(z+2,2,y) \ge 2^y$ :
- 3. A(z,2,y+1) > A(z,2,y);
- 4. A(z+2,2,y+3) > A(z+1,2,y+3);
- 5.  $A(z+2,2,y+3) \ge A(z+1,2,y+4)$ ;
- 6.  $A(z+2,2,y_1+y_2+3) \leq A(z+4,2,\max\{y_1,y_2\}+3)$ .

Доказательство. (1). Индукция по z. Для z=0 утверждение справедливо, так как  $A(2,2,y)=2^y$  для всех y. Теперь для произвольного z A(z+3,2,0)=1 по определению, A(z+3,2,1)=A(z+2,2,A(z+3,2,0))=A(z+2,2,1)=2 и A(z+3,2,2)=A(z+2,2,A(z+3,2,1))=A(z+2,2,2)=4.

- (2) Докажем индукцией по z, что для любого y  $A(z+2,2,y) \geqslant 2^y$ . При z=0 имеем равенство  $A(2,2,y)=2^y$ . Пусть теперь для z это верно; покажем, что это верно для z+1, используя индукцию по y. При y=0 имеем  $A(z+3,2,0)=1=2^0$ . Теперь  $A(z+3,2,y+1)=A(z+2,2,A(z+3,2,y)) \geqslant 2^{A(z+3,2,y)} \geqslant 2^{2^y} \geqslant 2^{y+1}$ . Здесь мы использовали неравенство  $2^y>y$ , которое справедливо для всех  $y\in\mathbb{N}$  и легко доказывается при помощи индукции.
- (3) При z=0,1,2 неравенство очевидно, так как A(0,2,y)=2+y, A(1,2,y)=2y и  $A(2,2,y)=2^y.$  Пусть  $z\geqslant 3.$  По предыдущему  $A(z,2,y+1)=A(z-1,2,A(z,2,y))\geqslant 2^{A(z,2,y)}>A(z,2,y).$

(4) и (5). Индукцией по z покажем, что оба утверждения справедливы. При z=0  $A(2,2,y+3)=2^{y+3}\geqslant 2(y+4)=A(1,2,y+4)>A(1,2,y+3)$  (неравенство  $2^{y+3}\geqslant 2(y+4)$  легко доказывается индукцией по y). Пусть для z оба неравенства имеют место для всех y; покажем индукцией по y, что они выполняются для z+1. Для y=0 A(z+3,2,3)=A(z+2,2,A(z+3,2,2))=A(z+2,2,4)>A(z+2,2,3). Наконец,  $A(z+3,2,y+4)=A(z+2,2,A(z+3,2,y+3))\geqslant A(z+2,2,A(z+2,2,y+4))>A(z+1,2,A(z+2,2,y+4))=A(z+2,2,y+4))=A(z+2,2,y+4)=A$ 

Пусть B(z,x) = A(z+2,2,x+3). Из определения A и леммы 6 имеем:

$$\begin{bmatrix}
B(0,x) & = 2^{x+3} \\
B(z+1,0) & = B(z,1) \\
B(z+1,x+1) & = B(z,B(z+1,x)).
\end{bmatrix}$$

Кроме того, из леммы 6 следует, что функция B(z,x) строго возрастает по обоим аргументам и что для нее справедливы неравенства:  $B(z+1,x)\geqslant B(z,x+1),\ B(z,x+y)\leqslant B(z+2,\max\{x,y\})$  и  $B(z,x)\geqslant x$ .

**Предложение 20** Пусть  $f(x_1, ..., x_k)$  — примитивно рекурсивная функция. Тогда существует  $n \in \mathbb{N}$ , такое что для всех  $x_1, ..., x_k$   $f(x_1, ..., x_k) \leq B(n, \max\{x_1, ..., x_k\})$ .

Доказательство. Если функция f базисная, то достаточно положить n=0 (если f — базисная 0-местная функция, равная константе 0, то k=0 и мы считаем, что  $\max\{x_1,\ldots,x_k\}=0$ ). Докажем, что если функция получается суперпозицией или примитивной рекурсией из функций, обладающих требуемым в предложении свойством, то она сама им обладает.

Суперпозиция. Пусть  $f(x_1,\ldots,x_k)=g(h_1(x_1,\ldots,x_k),\ldots,h_m(x_1,\ldots,x_k)), g(y_1,\ldots,y_m)\leqslant B(n_0,\max\{y_1,\ldots,y_m\})$  и  $h_i(x_1,\ldots,x_k)\leqslant B(n_i,\max\{x_1,\ldots,x_k\})$  для  $1\leqslant i\leqslant m$ . Пусть  $n\geqslant n_0,n_1,\ldots,n_m$ . Тогда  $f(x_1,\ldots,x_k)\leqslant B(n_0,\max\{h_1(x_1,\ldots,x_k),\ldots,h_m(x_1,\ldots,x_k)\})\leqslant B(n_0,\max\{B(n_1,\max\{x_1,\ldots,x_k\},\ldots,h_m(x_1,\ldots,x_k)\})$ 

 $(x_1, x_k), \dots, B(n_m, \max\{x_1, \dots, x_k\})) \le B(n, B(n+1, \max\{x_1, \dots, x_k\})) = B(n+1, \max\{x_1, \dots, x_k\} + 1) \le B(n+2, \max\{x_1, \dots, x_k\}).$ 

Примитивная рекурсия. Пусть

$$\begin{cases}
f(0, x_1, \dots, x_k) &= g(x_1, \dots, x_k) \\
f(t+1, x_1, \dots, x_k) &= h(f(t, x_1, \dots, x_k), t, x_1, \dots, x_k),
\end{cases}$$

 $g(x_1,\ldots,x_k)\leqslant B(n_1,\max\{x_1,\ldots,x_k\}),\ h(z,t,x_1,\ldots,x_k)\leqslant B(n_2,\max\{z,t,x_1,\ldots,x_k\})$  и  $n\geqslant n_1,n_2$ . Докажем индукцией по t, что  $f(t,x_1,\ldots,x_k)\leqslant B(n+1,\max\{x_1,\ldots,x_k\}+t).$ 

Для t=0  $f(0,x_1,\ldots,x_k)\leqslant B(n_1,\max\{x_1,\ldots,x_k\})\leqslant B(n+1,\max\{x_1,\ldots,x_k\}+0)$ . Пусть теперь для некоторого t это верно. Тогда  $f(t+1,x_1,\ldots,x_k)\leqslant B(n_2,\max\{f(t,x_1,\ldots,x_k),t,x_1,\ldots,x_k\})\leqslant B(n,B(n+1,\max\{x_1,\ldots,x_k\}+t))=B(n+1,\max\{x_1,\ldots,x_k\}+(t+1))$ , так как  $\max\{f(t,x_1,\ldots,x_k),t,x_1,\ldots,x_k\}\leqslant \max\{B(n+1,\max\{x_1,\ldots,x_k\}+t),t,x_1,\ldots,x_k\}=B(n+1,\max\{x_1,\ldots,x_k\}+t)$ .

Окончательно получаем  $f(t,x_1,\ldots,x_k) \leq B(n+1,\max\{x_1,\ldots,x_k\}+t) \leq B(n+3,\max\{\max\{x_1,\ldots,x_k\},t\}) = B(n+3,\max\{t,x_1,\ldots,x_k\}).$   $\square$ 

**Следствие 6** Функция  $f(x_1, ..., x_k)$  — примитивно рекурсивна тогда и только тогда, когда существует программа P, вычисляющая функцию f, такая что для некоторого  $n \in \mathbb{N}$   $t_P(x_1, ..., x_k) \leq B(n, \max\{x_1, ..., x_k\})$ .

Таким образом, как время вычисления, так и скорость роста любой примитивно рекурсивной функций ограничены некоторым уровнем функции B. У самой функции B счетное число уровней, и каждый следующий уровень растет значительно быстрее, чем предыдущий. Однако, как видно из следствия 7, некоторые функции растут быстрее, чем любой из уровней.

**Следствие 7** Функция C(x) = B(x,x) рекурсивна, но не примитивно рекурсивна.

Доказательство. Рекурсивность следует из предложения 19 и равества C(x) = A(x+2,2,x+3). Пусть C(x) примитивно рекурсивна. Тогда для некоторого n  $C(x) \leq B(n,x)$  при всех x. Однако C(n+1) = B(n+1,n+1) > B(n,n+1); противоречие.  $\square$ 

Можно сказать, что функция C растет слишком быстро для того, чтобы быть примитивно рекурсивной. То же самое верно и для функций B(z,x) и A(z,x,y). Каждый из уровней этих функций примитивно рекурсивен, однако сами функции — нет.

## 4.2 Меры сложности

Пусть, как и в предыдущем разделе, у нас имеется универсальное устройство для вычисление числовых функций. Также как и ранее, мы требуем от него, чтобы оно могло вычислять любую частично рекурсивную функцию ("зацикливаясь", если функция не определена). Мы считаем, что наше устройство работает согласно введенной в него программе. На этот раз мы не требуем, чтобы промежуточные результаты вычислений выражались примитивно рекурсивной функцией от времени и входных данных. Однако мы выдвигаем ряд других требований:

- 1. Каждая программа P вычисляет некоторую 1-местную частично рекурсивную функцию  $\varphi_P(x)$ ;
- 2. Множество программ является конструктивным пространством и, следовательно, обладает геделевской нумерацией. Зафиксируем некоторую геделевскую нумерацию, которая каждому натуральному числу e сопоставляет программу  $P_e$ ;
- 3. Функция  $\varphi_e(x) = \varphi_{P_e}(x)$  является частично рекурсивной функцией от двух аргументов.

Результаты предыдущей части курса показывают, что машины Тьюринга удовлетворяют всем этим требованиям. Можно показать, что этим требованиям удовлетворяют машины Шенфилда, а также большинство вычислительных устройств, которые известны в теории алгоритмов.

Теория сложности занимается следующим вопросом: "Как количество ресурсов, которые устройство привлекает для вычисления функции, зависит от входных данных". В зависимости от конструкции устройства можно уточнять понятие ресурса. Например, если устройство — это машина Тьюринга, то можно спросить, сколько времени потребуется для вычисления функции, а также какое количество ячеек ленты окажется задействованным при вычислении. Для машины Шенфилда можно спросить про время вычисления, а также про то, чему равно максимальное число, которое будет содержаться в регистрах машины в ходе вычислений. Для других устройств возможны и другие постановки вопроса о количестве задействованных ресурсов: например, количество переходов, которые программа выполняет в ходе вычисление, суммарное количество проходов для всех циклов и т. п. Несмотря на то, что все эти вопросы

весьма специфичны и их осмысленность зависит от того, какое конкретное устройство мы выбираем для вычислений, в них есть нечто общее. Ряд теорем может быть доказан для любого из перечисленных ресурсов (при условии, что для устройства, которое мы выбираем, понятие этого ресурса имеет смысл).

Определим общее понятие меры сложности. Пусть  $\varphi_e(x)$  — 2-местная частично рекурсивная функция, универсальная для класса всех 1-местных частично рекурсивных функций (мы будем мыслить о  $\varphi_e$  как об одноместной функции, вычислимой на нашем универсальном устройстве программой с номером e).

**Определение 29** *Мерой вычислительной сложности* называется семейство  $\{\Phi_e(x)\}_{e\in\mathbb{N}}$  одноместных частичных функций, такое что

- 1. Для всех  $e \in \mathbb{N}$   $\delta \Phi_e = \delta \varphi_e$ ;
- 2. Трехместный предикат  $R(e, x, y) = {}^{"}\Phi_e(x) = y"$  рекурсивен<sup>78</sup>.

Из определения легко следует, что  $\Phi_e(x)$  — частично рекурсивная функция от двух аргументов. Приведем три примера мер вычислительной сложности.

- 1. Универсальное вычислительное устройство это машина Тьюринга. Соответсвенно,  $\varphi_e(x)$  функция, вычисляемая программой для машины Тьюринга с номером e. Пусть  $\Phi_e(x) = t_e(x)$  количество шагов, за которое машина Тьюринга вычисляет  $\varphi_e(x)$ , работая по программе с номером e. Из результатов предыдущей части курса следует, что семейство  $\{t_e(x)\}_{e\in\mathbb{N}}$  удовлетворяет определению 29.
- 2. Устройство и функция  $\varphi_e(x)$  такие же, как в предыдущем примере. Пусть  $\Phi_e(x) = m_e(x)$  количество ячеек ленты, на которые головка хотя бы раз попадает в ходе работы программы с номером e. Читателю предлагается самому доказать, что семейство  $\{m_e(x)\}_{e\in\mathbb{N}}$  удовлетворяет определению 29.

<sup>&</sup>lt;sup>78</sup>То есть существует трехместная рекурсивная функция f, такая что f(e,x,y)=1 если  $\Phi_e(x)=y$  и f(e,x,y)=0 если  $\Phi_e(x)\neq y$ . Другими словами, существует алгоритм, который по данным e,x и y отвечает на вопрос: "Верно ли, что  $\Phi_e(x)=y$ ?"

3. Устройство — это машина Шенфилда,  $\varphi_e(x)$  — функция, вычисляемая программой для машины Шенфилда с номером<sup>79</sup> e. Пусть  $\Phi_e(x) = M_e(x)$  — максимальное число, которое будет содержаться в регистрах машины в ходе вычислений по программе с номером e. Можно опять доказать, что семейство  $\{M_e(x)\}_{e\in\mathbb{N}}$  удовлетворяет определению 29.

Можно привести множество других примеров.

Утверждения, которые можно доказать для произвольной меры вычислительной сложности, будут верны для любого из приведенных выше примеров. Такие утверждения принято называть машинно-независимыми. Приведем несколько примеров машинно-независимых результатов.

**Теорема 14 (о неограниченности сложности)** Пусть  $\{\Phi_e(x)\}_{e\in\mathbb{N}}$  — мера вычислительной сложности и b(x) — рекурсивная функция. Существует рекурсивная функция f(x), такая что для любого  $e\in\mathbb{N}$  если  $f=\varphi_e$ , то  $\Phi_e(x)>b(x)$  почти для всех<sup>80</sup> x.

Доказательство. Без доказательства 🗆

Теорема 14 показывает, что сложность вычислений невозможно ограничить сверху никакой рекурсивной функцией. Какой бы машинный ресурс мы не взяли (время, память и т. п.), для любой данной рекурсивной функции b(x) всегда найдется другая рекурсивная функция f(x), такая что любая программа, вычисляющая функцию f, должна почти для всех x затратить больше ресурсов, чем значение b на аргументе x. Для сравнения можно вспомнить, что мы доказали в предыдущем разделе этой части курса: если в качестве меры вычислительной сложности рассматривать время вычисления на машине Тьюринга (семейство  $\{t_e(x)\}_{e\in\mathbb{N}}$  из первого примера), то для любой примитивно рекурсивной функции f существует программа  $P_e$ , такая что  $f = \varphi_e$  и  $t_e(x) \leqslant C(x)$  почти для всех x (это легко следует из следствия 6 и определения функции C(x)). Таким образом, время вычисления каждой примитивно рекурсивной функции можно ограничить рекурсивной функцией C. Для каждой рекурсивной функции это сделать уже невозможно.

<sup>&</sup>lt;sup>79</sup>Необходимо предварительно определить геделевскую нумерацию программ для машин Шенфилда.

 $<sup>^{80}{\</sup>rm To}$ есть для всех, за исключением конечного числа. Смотри примечание на странице 45

В формулировке теоремы  $14 \Phi_e(x) > b(x)$  почти для всех x. Это ограничение существенно; усилить теорему 14, доказав, что  $\Phi_e(x) > b(x)$  для всех без исключения x не удастся. В принципе, понятно, почему это так, если мы рассмотрим какой-нибудь конкретный пример меры сложности; например, время вычисления. Пусть дана программа P, вычисляющая функцию f(x). Для любого конечного множества аргументов  $\{x_1, \ldots, x_k\}$ можно так видоизменить программу P, что модифицированная программа P' будет вычислять f(x) значительно быстрее, если x принадлежит нашему конечному множеству (и чуть-чуть медленнее, если x ему не принадлежит). Работа программы P' должна начинаться с проверки условия: верно ли, что аргумент x принадлежит  $\{x_1, \ldots, x_k\}$ ? Если ответ положительный, то программа P' дает результат немедленно, а если нет, то запускает программу Р. Поскольку значения любой функции на заранее заданном конечном числе аргументов можно задать явно, в виде таблицы, то сложность вычислений имеет смысл рассматривать только с точностью до конечного числа аргументов, то есть в "асимптотике".

Теорема 14, в принципе, довольно проста (ее доказательство можно даже рекомендовать слушателям курса в качестве упражнения). Приведем формулировку другой, более сложной (но и более интересной) теоремы.

**Теорема 15 (об ускорении)** Пусть  $\{\Phi_e(x)\}_{e\in\mathbb{N}}$  — мера вычислительной сложности и r(x) — произвольная рекурсивная функция. Существует рекурсивная функция f, такая что для каждого  $i\in\mathbb{N}$  если  $f=\varphi_i$ , то существует  $j\in\mathbb{N}$ , для которого  $f=\varphi_j$  и  $r(\Phi_j(x))<\Phi_i(x)$  почти для всех значений x.

*Доказательство*. Без доказательства □

Из теоремы об ускорении, в частности, следует, что для некоторых рекурсивных функций не существует оптимальной программы вычисления. Пусть, например, мера сложности — это время вычисления, а r(x) = 100x. Применим теорему 15 к этой функции r и получим некоторую функцию f. Тогда для любой программы, вычисляющей функцию f, найдется другая программа, которая вычисляет ту же самую функцию f, причем почти для всех аргументов вторая программа делает это в 100 раз быстрее, чем первая. Если мы, например, улучшим конструкцию компьютера, который мы обычно используем для вычислений, и сделаем так, что новая машина будет выполнять в 100 раз больше операций

в секунду, чем старая, то все равно вторая программа на старой машине будет работать быстрее, чем первая на новой (почти для всех входных данных).

Закончим этот раздел еще одной достаточно курьезной теоремой.

**Теорема 16 (о пробелах)** Пусть  $\{\Phi_e(x)\}_{e\in\mathbb{N}}$  — мера вычислительной сложности и r(x) — рекурсивная функция, для которой  $r(x)\geqslant x$  при всех x. Тогда существует рекурсивная функция b(x), такая что если для некоторого e  $\Phi_e(x)\geqslant b(x)$  почти для всех x, то  $\Phi_e(x)\geqslant r(b(x))$  почти для всех  $x^{81}$ .

Значение теоремы о пробелах лучше всего понять, если рассматривать ее вместе с теоремой 14. Пусть, для примера, опять  $\Phi_e(x) = t_e(x)$  — время вычисления значения  $\varphi_e(x)$  по программе  $P_e$ . Рассмотрим функцию r(x) = 100x и применим к ней теорему о пробелах. Получим рекурсивную функцию b(x). По теореме 14 существуют рекурсивная функция f, такая что время вычисления f(x) по любой программе больше, чем b(x) почти для всех x. А по теореме о пробелах это время должно быть даже больше, чем 100b(x). Таким образом, в упорядочивании функций по времени их вычисления существуют "пробелы"; есть функции, время вычисления которых меньше, чем b(x) и функции, время вычисления которых больше, чем 100b(x), но нет функций, время вычисления которых лежало бы в интервале [b(x), 100b(x)]. Вместо r(x) = 100x можно было взять и r(x) = 1000000x, и  $r(x) = x^{10000000}$ , и даже r(x) = C(x); чем быстрее растет функция r, тем большее впечатление производит результат.

#### 4.3 $\mathcal{P} = \mathcal{N}\mathcal{P}$ ?

В отличие от предыдущего раздела здесь мы конкретизируем ресурс — время (количество шагов) вычисления и устройство — машину Тьюринга. Однако наши рассмотрения носят значительно более широкий характер; все, о чем мы скажем, будет верно не только для машины Тьюринга,

 $<sup>^{81}</sup>$ Здесь допускается, что функция  $\varphi_e$  может быть не всюду определенной. Для этого следует положить  $\Phi_e(x)=\infty$  в тех случаях, когда  $\varphi_e(x)$  не определено.

но и для любой машины, которая "полиномиально" сводится к ней<sup>82</sup>. В частности, все известные детерминированные (см. ниже) модификации машины Тьюринга (машина Тьюринга с лентой, бесконечной в оба конца, с несколькими лентами, с квадратной "лентой", с прямым доступом к памяти) сводятся к известной нам машине с одной лентой, бесконечной с одного конца, "полиномиально".

Пусть  $\Sigma$  — конечный алфавит, не содержащий символы n и m. Под программой мы будем понимать программу для машины Тьюринга, предназначенную для работы со словами этого алфавита. Каждая частичная функция из  $\Sigma^*$  в  $\Sigma^*$ , вычислимая хоть по какому-нибудь алгоритму, согласно тезису Тьюринга вычислима при помощи некоторой программы (см. определение 13). Для программы P, вычисляющей функцию из  $\Sigma^*$  в  $\Sigma^*$ , через  $t_P(w)$  обозначим количество шагов, которое машина, работающая по программе P, сделает при вычислении значения этой функции на аргументе  $w \in \Sigma^*$ . Скажем, что функция  $f : \Sigma^* \to \Sigma^*$  вычислима за поли*номиальное время* (или что она принадлежит классу  $\mathcal{P}$ ), если существует программа P, вычисляющая эту функцию, такая что  $t_P(w) \leqslant p(|w|)$  для всех  $w \in \Sigma^*$ , где p — некоторый полином<sup>83</sup>. Будем говорить, что функция  $f: \Sigma^* \to \Sigma^*$  вычислима за экспоненциальное время (или что она принадлежит классу  $\mathcal{EXP}$ ), если существует программа P, вычисляющая эту функцию, такая что  $t_P(w) \leqslant 2^{p(|w|)}$  для всех  $w \in \Sigma^*$ , где p — некоторый полином.

Классы  $\mathcal{P}$  и  $\mathcal{EXP}$  определены для "символьных" функций, то есть для функций из  $\Sigma^*$  в  $\Sigma^*$ . Это принципиальный момент, поскольку нам важна зависимость времени вычисления функции от длины аргумента. Однако мы привыкли иметь дело с "числовыми" функциями, и здесь возникает одна сложность: как измерять длину входных данных? Существует множество способов задания натуральных чисел при помощи последовательности символов и, к сожалению, некоторые из них не сводятся друг к другу за полиномиальное время. Поэтому, если мы хотим определить классы  $\mathcal{P}$  и  $\mathcal{EXP}$  для числовых функций, нам надо зафиксировать способ записи чисел. В теории сложности принято следующее определение:

 $<sup>^{82}</sup>$ Скажем, что некая машина сводится к машине Тьюринга "полиномиально", если для каждой программы этой машины можно написать эквивалентную ей программу машины Тьюринга и задать полином p, такой что если t — время работы исходной машины по заданной программе, то время работы машины Тьюринга по соотвествующей программе не превосходит p(t).

<sup>&</sup>lt;sup>83</sup>Напомним, что через |w| обозначается длина слова w.

функция  $f: \mathbb{N} \to \mathbb{N}$  принадлежит классу  $\mathcal{P}$  (или  $\mathcal{EXP}$ ), если функция из  $g: \{0,1\}^* \to \{0,1\}^*$ , такая что для любого  $n \in \mathbb{N}$  g от двоичной записи числа n равно двоичной записи числа f(n), принадлежит классу  $\mathcal{P}$  (или  $\mathcal{EXP}$ ).

Классы  $\mathcal{P}$  и  $\mathcal{EXP}$  очень важны в практическом плане. Большинство реальных функций, которые приходится считать на компьютерах для решения прикладных задач, принадлежат классу  $\mathcal{EXP}^{85}$  и могут быть посчитаны лишь при малых значениях аргументов: все же экспонента растет слишком быстро. С полиномами дела обстоят лучше, поскольку их скорость роста не так велика; поэтому множество людей занято поиском полиномиальных алгоритмов для тех задач, экспоненциальный алгоритм решения которых уже известен. Имеется достаточно много оснований для того, чтобы рассматривать в качестве "практически вычислимых функций" функции, вычислимые за полиномиальное время  $^{86}$ .

Для дальнейших рассмотрений сузим круг задач, ограничившись так называемыми задачами распознавания. Пусть  $\Sigma$  — конечный алфавит и  $L\subseteq \Sigma^*$  — язык этого алфавита. Без ограничения общности можно считать, что  $0,1\in \Sigma$ . Связанная с языком L задача распознавания — это задача вычисления характеристической функции  $\chi_L$ , то есть такой

 $<sup>^{84}</sup>$ Вместо двоичной можно взять десятичную, шестнадцатиричную и т. п. запись числа, но не "одноичную", которую мы все время рассматривали ранее. Машина Тьюринга не может преобразовать двоичную запись числа n в слово  $1^n$  за полиномиальное время. Выбор позиционной системы счисления очень естественен: на практике мы привыкли работать с десятичной записью чисел, а компьютеры работают с двоичным представлением данных.

 $<sup>^{85}</sup>$ Конечно, есть вычислимые функции, не принадлежащие классу  $\mathcal{EXP}$ ; например, функция C(x). Однако все они представляют интерес исключительно в рамках абстрактной теории. Таковой же является функция, существование которой утверждается в теореме об ускорении: для нее не существует оптимального алгоритма, но его не существует за счет того, что любой алгоритм требует очень большого объема вычислений. Эти функции вычислимы, но вряд ли кто-нибудь захочет считать их на самом деле. Здесь можно провести параллель с матанализом. Известны примеры различных "паталогий" (коврик Серпинского, неизмеримое множество), однако все кривые, с которыми приходится сталкиваться на практике, все же имеют длину и нулевую площадь.

 $<sup>^{86}</sup>$ Хотя все эти соображения все же достаточно условны. Например, если есть два алгоритма для вычисления одной и той же функции f, причем первый считает f(x) за  $x^{100}$  шагов, а второй — за  $x^{[\ln \ln x]}$  шагов, то для тех аргументов, для которых значение функции представляет практический интерес, второй алгоритм может оказаться удобнее первого.

функции из  $\Sigma^*$  в  $\Sigma^*$ , что

$$\chi_{\scriptscriptstyle L}(w) = \begin{cases} 1, & w \in L \\ 0, & w \not\in L. \end{cases}$$

Скажем, что задача распознавания принадлежности слов языку L решается за полиномиальное (экспоненциальное) время (или принадлежит классу  $\mathcal{P}$  ( $\mathcal{EXP}$ )), если  $\chi_L \in \mathcal{P}$  ( $\chi_L \in \mathcal{EXP}$ ). Далее мы будет говорить о классах  $\mathcal{P}$  и  $\mathcal{EXP}$ , имея в виду совокупность задач распознавания, принадлежащих этим классам.

На практике задачам распознавания обычно придают более естественную формулировку. Приведем несколько примеров задач.

- 1. Эквивалентность детерминированных конечных автоматов. Пусть даны два конечных автомата одного и того же алфавита  $\Sigma$ : требуется определить, распознают ли они один и тот же язык. Более формально: необходимо ввести некоторый "естественный" способ задания всех пар детерминированных автоматов алфавита  $\Sigma$  при помощи слов некоторого другого алфавита  $\Sigma'$ . Требуется решить задачу распознавания языка, состоящего из всех слов алфавита  $\Sigma'$ , задающих пары эквивалентных автоматов.
- 2. Эквивалентность недетерминированных конечных автоматов. То же самое, но только автоматы могут быть недетерминированными.
- 3. Задача коммивояжера. Дан список городов и стоимость проезда из каждого в каждый. Кроме того, дано число *B*, которое является "бюджетом" поездки. Требуется определить, может ли коммивояжер посетить каждый из городов, перечисленных в списке, не выходя за рамки бюджета<sup>87</sup>.
- 4. Задача проверки истинности формулы. Дана формула исчисления

<sup>&</sup>lt;sup>87</sup>В более общей постановке задача коммивояжера заключается в нахождении оптимального по стоимости маршрута. Это уже не задача распознавания, а так называемая задача оптимизации. Формулировка с бюджетом поездки превращает задачу коммивояжера в задачу распознавания.

высказываний<sup>88</sup>: требуется определить, может ли эта формула быть истинной.

- 5. Существование эйлерова цикла в графе. Дан направленный конечный граф (множество вершин и стрелок): требуется определить, существует ли путь, проходящий через каждую стрелку ровно по одному разу.
- 6. Существование гамильтонова цикла в графе. Дан направленный конечный граф: требуется определить, существует ли путь, проходящий через каждую вершину ровно по одному разу.
- 7. Задача о разбиении. Дана последовательность натуральных чисел: требуется проверить, можно ли разбить эту последовательность на две части так, чтобы суммы чисел в каждой из частей совпадали.

Каждая из этих задач разрешима за экспоненциальное время, то есть принадлежит классу  $\mathcal{EXP}$ . Для первой и пятой задач найдены полиномиальные алгоритмы: они принадлежат классу  $\mathcal{P}$ . Про остальные задачи это не известно<sup>89</sup>. Правда, у большинства этих задач есть одна существенная особенность, которая позволяет нам отнести их к классу  $\mathcal{NP}^{90}$ . Прежде чем дать формальное определение этого класса, поясним эту особенность на неформальном уровне.

Рассмотрим, для примера, задачу коммивояжера. Пусть дано n городов. Простейший (так называемый "полнопереборный") алгоритм заключается в том, что мы по очереди рассмотриваем все n! возможных маршрутов и, может быть, лишь последний из рассмотренных окажется подходящим. Таким образом, в наихудшем случае время работы этого алгоритма будет сравнимо с n!; этот алгоритм экспоненциален  $(2^n \le n! \le n^n \le 2^{n^2}$  почти для всех n). Однако если некто, с помощью необъяснимой

<sup>&</sup>lt;sup>88</sup>Определение этого понятия можно найти практически в любом учебнике по математической логике. Отметим еще, что для этой задачи не надо вводить алфавит, слова которого описывают формулы, поскольку формулы уже являются словами некоторого алфавита. Другими словами, эта задача является задачей распознавания уже в исходной формулировке.

<sup>&</sup>lt;sup>89</sup>С одной оговоркой. Для задачи о разбиении не известен полиномиальный алгоритм для определения того, существует ли разбиение, при условии что числа в исходной последовательности даны в двоичной записи (как мы ранее и договорились). Если же задавать числа в "одноичной" записи, то такой алгоритм есть.

<sup>&</sup>lt;sup>90</sup>Определение смотри ниже.

интуиции, угадает оптимальный маршрут и скажет его, то нам останется только проверить, укладывается ли он в рамки бюджета. А простая проверка требует уже значительно меньше времени. В двух словах суть этой задачи можно сформулировать так: трудно найти ответ, но легко его проверить. Каждая из перечисленных выше задач, для которой не известен полиномиальный алгоритм решения, может быть охарактеризована подобным образом. И если бы мы смогли научить машину Тьюринга угадывать (кратчайший путь для коммивояжера, последовательность вершин в гамильтоновом цикле графа, разбиение последовательности чисел на две части с равной суммой), то для всех перечисленных выше задач появился бы полиномиальный алгоритм решения.

Для формализации "угадывания" введем так называемую недетерминированную машину Тьюринга. Недетерминированная машина Тьюринга — это та же самая машина (с лентой, головкой и т. д.), в программе для которой может присутствовать несколько команд вида  $qa \to \dots$  с разной правой частью. Если в ходе выполнения программы головка оказывается в состоянии q и указывает на ячейку ленты с записанным в ней символом a, то может быть выполнена любая из команд, и машина выбирает для исполнения какую-то одну из них, руководствуясь только ей известными соображениями  $^{91}$ . В процессе вычисления такая машина может сталкиваться с выбором из нескольких возможностей сколько угодно раз: соответственно, для нее существует множество путей, по которым она может вести вычисления. Если один из путей заканчивается переходом в конечное состояние, то можно подсчитать количество шагов, которые машина должна проделать, идя по этому пути, и результат, который на этом пути получен.

Пусть  $L\subseteq \Sigma^*$  — язык конечного алфавита  $\Sigma$ . Скажем, что задача распознавания принадлежности слов языку L недетерминированно решается за полиномиальное время (или принадлежит классу  $\mathcal{NP}$ ), если существует программа P для недетерминированной машины Тьюринга, работающая с алфавитом  $\Sigma$ , такая что для любого  $w \in \Sigma^*$ 

- 1. все возможные пути вычисления по программе P из состояния, задаваемого конфигурацией  $\langle w, q_1, 1 \rangle$ , завершаются не более чем за p(|w|) шагов вычисления, где p некоторый полином;
- 2. если  $w \in L$ , то хотя бы один из возможных путей вычисления дает

<sup>91</sup> Это очень похоже на недетерминированный конечный автомат.

результат 1; в противном случае все пути вычисления приводят к результату 0.

Задачи из класса  $\mathcal{NP}$  постоянно встречаются на практике. В частности, все перечисленные выше семь задач (кроме, возможно, второй), а также многие другие принадлежат классу  $\mathcal{NP}$ .

**Теорема 17** Справедливы включения:  $\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{EXP}$  и  $\mathcal{P} \subset \mathcal{EXP}$ .

Доказательство. Без доказательства □

Утверждение теоремы 17 — это практически все, что известно современной науке о взаимоотношении классов  $\mathcal{P}$ ,  $\mathcal{NP}$  и  $\mathcal{EXP}$ . Из-за того, что включение  $\mathcal{P} \subset \mathcal{EXP}$  — строгое, хотя бы одно из включений (а, может быть, и оба) в ряду  $\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{EXP}$  должно быть строгим. Какое из них какое — неизвестно, несмотря на то, что проблема нахождения полиномиальных алгоритмов для задач из класса  $\mathcal{NP}$  имеет большое практическое значение и в настоящее время признана одной из центральных проблем современной математики<sup>92</sup>.

Завершая последний раздел курса, покажем, как можно свести проблему " $\mathcal{P} = \mathcal{NP}$ " к выяснению вопроса о принадлежности классу  $\mathcal{P}$  одной единственной задачи. Пусть  $L_1, L_2$  — два языка конечного алфавита  $\Sigma$ . Скажем, что проблема распознавания принадлежности слов языку  $L_1$  полиномиально сводится к проблеме распознавания принадлежности слов языку  $L_2$ , если существует функция  $f: \Sigma^* \to \Sigma^*$ , вычислимая за полиномиальное время, такая что для любого  $w \in \Sigma^* w \in L_1 \leftrightarrow f(w) \in L_2^{93}$ . Ясно, что в случае наличия полиномиальной сводимости если проблема распознавания языка  $L_2$  решается за полиномиальное время, то аналогичная проблема для  $L_1$  также решается за полиномиальное время. Для ответа на вопрос " $w \in L_1$ ?" можно сначала за полиномиальное время вычислить f(w), а затем, затратив полиномиальное время, ответить на вопрос " $f(w) \in L_2$ ?".

**Определение 30** Задача распознавания называется  $\mathcal{NP}$ -полной, если любая другая задача распознавания, принадлежащая классу  $\mathcal{NP}$ , полиномиально сводится к ней.

 $<sup>^{92}</sup>$ За решение этого вопроса обещаны большие денежные премии. В частности, за решение проблемы " $\mathcal{P} = \mathcal{N}\mathcal{P}$ ?" один из американских институтов предлагает 1000000 долларов (см. http://www.claymath.org/Millennium\_Prize\_Problems).

 $<sup>^{93}</sup>$ Это напоминает определение m-сводимости.

Таким образом, если дана какая-нибудь  $\mathcal{NP}$ -полная задача, то решить проблему " $\mathcal{P} = \mathcal{NP}$ " можно, если установить (или опровергнуть) существование полиномиального алгоритма для нее.

В приведенном выше списке задачи под номерами 3,4,6 и 7 являются  $\mathcal{NP}$ -полными. Мы оставляем этот факт без доказательства.