Учебное издание

МАРЧЕНКОВ Сергей Серафимович

РЕКУРСИВНЫЕ ФУНКЦИИ

Редактор *В.С. Аролович*
Оригинал-макет: *Е.Н. Водоватова*
Оформление переплета: *Н.В. Гришина*

Подписано в печать 23.03.07. Формат 60×90/16. Бумага офсетная.
Печать офсетная. Усл. печ. л. 4. Уч.-изд. л. 4,4. Тираж экз.
Заказ №

Издательская фирма «Физико-математическая литература»
МАИК «Наука/Интерпериодика»
117997, Москва, ул. Профсоюзная, 90
E-mail: fizmat@maik.ru, fmlsale@maik.ru;
http://www.fml.ru

Отпечатано с готовых диапозитивов
в ППП «Типография «Наука»
121099, г. Москва, Шубинский пер., 6

ISBN 978-5-9221-0825-6

УДК 519.7
ББК 22.176
М 30

Марченков С. С. **Рекурсивные функции.** — М.: ФИЗМАТЛИТ, 2007. — 64 с. — ISBN 978-5-9221-0825-6.

Брошюра знакомит читателя с алгоритмически вычислимыми функциями натурального аргумента — рекурсивными функциями. Вначале изучается простейший тип рекурсивных функций — примитивно рекурсивные функции. Затем происходит расширение круга вычислимых функций: рассматриваются частично определенные вычислимые функции, а также всюду определенные вычислимые функции, не являющиеся примитивно рекурсивными. В заключение определяются абстрактные вычислительные устройства — машины Тьюринга, и класс функций, вычислимых на машинах Тьюринга, связывается с классом частично рекурсивных функций. Для школьников старших классов и студентов вузов, знакомящихся с основами теории алгоритмов.

ISBN 978-5-9221-0825-6

ОГЛАВЛЕНИЕ

Предисловие 4

Глава 1. **Примитивно рекурсивные функции** 6
§ 1.1. Определение функций по индукции 6
§ 1.2. Операции примитивной рекурсии и суперпозиции 8
§ 1.3. Класс примитивно рекурсивных функций 10
§ 1.4. Некоторые свойства примитивно рекурсивных функций 14
§ 1.5. Элементарные рекурсивные функции 18

Глава 2. **Частично рекурсивные функции** 22
§ 2.1. Непримитивные рекурсии 22
§ 2.2. Частичные функции и операция минимизации 24
§ 2.3. Класс частично рекурсивных функций 26
§ 2.4. Рекурсивно перечислимые множества. Нормальная форма Клини 30

Глава 3. **Функции, вычислимые на машинах Тьюринга** 35
§ 3.1. Машина Тьюринга 35
§ 3.2. Композиция и итерация машин Тьюринга 39
§ 3.3. Моделирование машин Тьюринга 43
§ 3.4. Вычисление частично рекурсивных функций на машинах Тьюринга 46
§ 3.5. Частичная рекурсивность функций, вычислимых на машинах Тьюринга 49
§ 3.6. Универсальная машина Тьюринга 52

Ответы, решения, указания 57

Предисловие

Каждый взрослый человек, даже далекий от математики, несомненно знаком с термином *алгоритм*. Что скрывается за этим понятием? Исчерпывающий ответ на этот вопрос мы находим в разделе математики, который называется *теория алгоритмов*.

Теория алгоритмов как самостоятельное направление в математике возникла в середине 30-х годов прошлого столетия, когда сразу нескольким математикам удалось дать точное математическое определение интуитивного понятия алгоритма. Бóльшая часть теории алгоритмов имеет дело с алгоритмически вычислимыми функциями, или, как их называют в теории алгоритмов, *рекурсивными функциями*. Термин *рекурсия* происходит от латинского слова *recurro* (бежать назад, возвращаться), и первоначально рекурсивными функциями назвали лишь функции, вычисление которых происходит на основе обращения к «предыдущим» значениям функции. Однако впоследствии рекурсивными функциями стали называть все алгоритмически вычислимые функции натурального аргумента.

В настоящей брошюре мы хотим познакомить читателя с основными понятиями теории рекурсивных функций, дать представление о задачах, решаемых в теории рекурсивных функций, и по возможности научить выражать простейшие числовые алгоритмы с помощью рекурсивных функций.

С некоторыми рекурсивными функциями знаком каждый школьник. Это сложение, умножение, возведение в степень и близкие к ним функции. Если проанализировать строгие математические определения этих функций, то можно увидеть, что в основе определений лежит обычная математическая индукция. Сразу возникает вопрос: а какие вообще функции можно определить по индукции? Оказывается, что эти функции давно и хорошо известны в теории алгоритмов. Они носят название *примитивно рекурсивные функции*. В главе I брошюры мы подробно рассказываем об этом простом виде рекурсивных функций.

Дальнейшее изучение рекурсивных функций невозможно без двух радикальных шагов. Во-первых, наряду с всюду определенными функциями необходимо допустить к рассмотрению частично определенные функции. А во-вторых, следует расширить арсенал средств, используемых для задания рекурсивных функций. В результате получаются как частично определенные вычислимые функции, так и всюду определенные вычислимые функции, которые не являются примитивно рекурсивными. Применение операции минимизации приводит к наиболее широкому классу алгоритмически вычислимых функций — классу частично рекурсивных функций. В главе II брошюры описываются эти этапы в построении класса частично рекурсивных функций.

Одно из самых первых уточнений алгоритмически вычислимой функции принадлежит английскому математику А. Тьюрингу и основано на предложенном им абстрактном вычислительном устройстве — машине Тьюринга. Ни один современный курс теории алгоритмов не может обойтись без изложения тех или иных вариантов машины Тьюринга. В главе III брошюры также вводятся и изучаются машины Тьюринга. Основная задача этой главы состоит в том, чтобы доказать совпадение класса частично рекурсивных функций с классом функций, вычислимых на машинах Тьюринга. В конце главы III строится универсальная машина Тьюринга и на ее основе доказываются утверждения о невычислимости некоторых функций.

Книга ориентирована в первую очередь на учеников старших классов средней школы. Автор надеется, что она будет также полезна студентам младших курсов университетов и технических вузов, изучающим дискретную математику.

Глава 1

ПРИМИТИВНО РЕКУРСИВНЫЕ ФУНКЦИИ

§ 1.1. Определение функций по индукции

Одним из самых распространеных способов рассуждения в математике является рассуждение по индукции. По индукции можно как доказывать утверждения, так и определять некоторые объекты.

В простейшем случае доказательство по индукции выглядит так. Предположим, что у нас имеется утверждение $P(x)$, зависящее от натурального параметра x, и мы хотим доказать справедливость утверждения $P(x)$ для всех значений x. Это можно сделать в два этапа. Сначала устанавливаем базис индукции — утверждение $P(1)$. Затем проделываем индуктивный переход: предполагая, что утверждение $P(x)$ уже доказано, доказываем утверждение $P(x+1)$. Справедливость утверждения $P(x)$ для всех натуральных значений x теперь следует из *принципа математической индукции*.

В более сложных случаях утверждение P может зависеть от параметров $y_1, \ldots, y_m$, которые в процессе индукции не должны меняться. Кроме того, вместо перехода от x к $x+1$ иногда рассматривается «спуск» от x к $d(x)$, где d — некоторая функция, дающая для любого $x > 1$ значение $d(x) < x$.

Анализируя доказательство утверждения «для всех x справедливо $P(x)$», можно заметить, что в нем неявным образом присутствует определение натуральных чисел по индукции. Более того, доказательство в известном смысле следует за определением. В самом деле, натуральные числа определяются по индукции следующим образом: 1 есть натуральное число; если x — натуральное число, то $x+1$ — также натуральное число. (Мы не будем здесь обсуждать вопросы о способах задания натуральных чисел и о существовании натурального числа $x+1$, непосредственно следующего за числом x. Надеемся, что имеющийся у читателя математический опыт позволяет ему успешно справиться с этими задачами.) Поэтому базис индукции и индуктивный

переход в доказательстве утверждения «для всех x справедливо $P(x)$» лишь повторяет структуру определения натуральных чисел.

Определение натуральных чисел по индукции служит исходной основой для многочисленных определений функций по индукции. Рассмотрим, например, определение по индукции функции $f(x)$, которое состоит из двух равенств

$$\begin{cases} f(1) = 3, \\ f(x+1) = f(x) + 1. \end{cases} \tag{1.1}$$

Первое из равенств (1.1) есть базис индукции, второе — индуктивный переход: значение $f(x+1)$ определяется через значение $f(x)$. Нетрудно сообразить, что равенства (1.1) определяют единственную функцию $f(x)$, равную $x+2$.

Прежде чем переходить к определениям других функций по индукции, сделаем два замечания. Во-первых, мы расширим наше исходное числовое множество — множество натуральных чисел, добавив к нему число 0 (впрочем, некоторые математики изначально причисляют число 0 к натуральным числам). Таким образом, начиная с этого места, мы будем рассматривать функции $f(x_1, \ldots, x_n)$, заданные на множестве $N = \{0, 1, 2, \ldots\}$. Это означает, что как аргументы $x_1, \ldots, x_n$ функций $f(x_1, \ldots, x_n)$, так и сами функции f принимают значения из N.

Второе замечание касается самого определение по индукции. Как уже отмечалось, помимо определений с помощью обычной индукции (например, как это сделано в равенствах (1.1)), существуют определения по индукции с параметрами. При этом индукция проводится по одной переменной, а все остальные переменные представляют собой параметры индукции, которые в процессе индуктивного определения остаются неизменными.

Обратимся к дальнейшим примерам. Пусть функция $\mathrm{sum}\,(x_1, x_2)$ определяется по индукции равенствами

$$\begin{cases} \mathrm{sum}\,(x_1, 0) = x_1, \\ \mathrm{sum}\,(x_1, x_2 + 1) = \mathrm{sum}\,(x_1, x_2) + 1. \end{cases} \tag{1.2}$$

Как видно, индукция в равенствах (1.2) ведется по переменной x_2, а переменная x_1 является (неизменяемым) параметром индукции. Легко понять, что равенства (1.2) определяют единственную функцию $\mathrm{sum}\,(x_1, x_2)$, равную $x_1 + x_2$. Это утверждение станет еще более понятным, если равенства (1.2) переписать в более привычной форме:

$$\begin{cases} x_1 + 0 = x_1, \\ x_1 + (x_2 + 1) = (x_1 + x_2) + 1. \end{cases} \tag{1.3}$$

Во втором из равенств (1.3) мы специально расставили скобки, чтобы подчеркнуть связь с определением (1.2).

Так же, как для функции $x_1 + x_2$, хорошо известны индуктивные определения функций $x_1 \cdot x_2$ и $x_1^{x_2}$:

$$\begin{cases} \operatorname{prod}(x_1, 0) = 0, \\ \operatorname{prod}(x_1, x_2+1) = \operatorname{prod}(x_1, x_2)+x_1, \end{cases} \qquad \begin{cases} x_1 \cdot 0 = 0, \\ x_1 \cdot (x_2 + 1) = (x_1 \cdot x_2)+x_1, \end{cases} \tag{1.4}$$

$$\begin{cases} \operatorname{pow}(x_1, 0) = 1, \\ \operatorname{pow}(x_1, x_2 + 1) = \operatorname{pow}(x_1, x_2) \cdot x_1, \end{cases} \qquad \begin{cases} x_1^0 = 1, \\ x_1^{x_2+1} = x_1^{x_2} \cdot x_1 \end{cases} \tag{1.5}$$

(чтобы не делать исключений в определении функции $x_1^{x_2}$, мы приняли $0^0 = 1$).

Упражнения

1. Пусть $x!$ (читается: икс факториал) есть произведение $1 \cdot 2 \cdot \ldots \cdot x$ (при $x = 0$ это произведение считается равным 1) и $[x/2]$ есть целая часть от деления x на 2. Покажите, что функции $x!$ и $[x/2]$ можно определить по индукции.

§ 1.2. Операции примитивной рекурсии и суперпозиции

В теории алгоритмов наряду с индукцией как средством доказательства утверждений и определения различных объектов имеется и вполне формализованная схема определения функций по индукции. Эта схема (а точнее сказать — операция) носит название *примитивная рекурсия* (английский термин *primitive recursion*).

Чтобы привести точные определения, предположим, что заданы функции $g(x_1, \ldots, x_{n-1})$ и $h(x_1, \ldots, x_{n+1})$, зависящие соответственно от $(n-1)$ и $(n+1)$ переменных. Будем говорить, что функция $f(x_1, \ldots, x_n)$, зависящая от n переменных, получается из функций g, h с помощью *операции примитивной рекурсии* (по переменной x_n), если выполняются соотношения

$$\begin{cases} f(x_1, \ldots, x_{n-1}, 0) = g(x_1, \ldots, x_{n-1}), \\ f(x_1, \ldots, x_{n-1}, x_n + 1) = h(x_1, \ldots, x_{n-1}, x_n, f(x_1, \ldots, x_n)). \end{cases} \tag{1.6}$$

При $n = 1$ схема (1.6) превращается в схему

$$\begin{cases} f(0) = a, \\ f(x + 1) = h(x, f(x)), \end{cases} \tag{1.7}$$

где a — число из N. В схеме (1.7) функция h часто не зависит от первой переменной, т. е. по существу является функцией от одной (второй) переменной. Этот случай называется *итерацией функции h в точке a* и представляется схемой

$$\begin{cases} f(0) = a, \\ f(x + 1) = h(f(x)), \end{cases} \tag{1.8}$$

Понять механизм действия операции примитивной рекурсии проще всего на примере схемы (1.8). В самом деле, из первого и второго равенств системы (1.8) последовательно выводим:

$$f(0) = a, \quad f(1) = h(f(0)) = h(a), \quad f(2) = h(f(1)) = h(h(a)), \dots .$$

Таким образом, при $x > 0$ справедливо соотношение

$$f(x) = \underbrace{h(\dots h}_{x \text{ раз}}(a)\dots). \tag{1.9}$$

Иначе говоря, для получения значения $f(x)$ при $x > 0$ необходимо x раз применить функцию h к числу a.

Вновь обратимся к примеру (1.1), начав в нем итерацию со значения $x = 0$:

$$\begin{cases} f(0) = 2, \\ f(x+1) = f(x) + 1. \end{cases}$$

Как видим, здесь $a = 2$, $h(x) = x + 1$ и $f(x) = x + 2$.

Рассмотрим два более сложных примера итераций:

$$\begin{cases} f_1(0) = 1, \\ f_1(x+1) = 2f_1(x); \end{cases} \qquad \begin{cases} f_2(0) = 2, \\ f_2(x+1) = (f_2(x))^2. \end{cases}$$

Здесь в первом случае $h_1(x) = 2x$, во втором — $h_2(x) = x^2$. Рассуждая по индукции, легко установить, что

$$f_1(x) = 2^x, \quad f_2(x) = 2^{2^x}.$$

Посмотрим теперь на схему (1.7). Действуя так же, как в случае схемы (1.8), последовательно находим

$$f(0) = a, \ f(1) = h(0, f(0)) = h(0, a), \ f(2) = h(1, f(1)) = h(1, h(0, a)), \dots .$$

Так что при $x > 0$ будет выполняться соотношение

$$f(x) = h(x-1, h(x-2, \dots, h(1, h(0, a))\dots)).$$

Как видим, оно представляет собой обобщение соотношения (1.9), обусловленное тем, что у функции h есть еще одна переменная.

В том же духе может быть рассмотрена и общая схема (1.6). Впрочем, при фиксированных значениях переменных $x_1, \dots, x_{n-1}$ она сводится к схеме (1.7).

Еще одной важной операцией над функциями, без которой нам не обойтись в дальнейшем, является операция *суперпозиции* (подстановки). Формально она опеределяется следующим образом. Пусть заданы функции

$$g_0(y_1, \dots, y_m), \quad g_1(x_1, \dots, x_n), \dots, g_m(x_1, \dots, x_n). \tag{1.10}$$

Образуем из них функцию f:

$$f(x_1, \dots, x_n) = g_0(g_1(x_1, \dots, x_n), \dots, g_m(x_1, \dots, x_n)). \tag{1.11}$$

О функции f говорят, что она получена из функций (1.10) с помощью *операции суперпозиции* (или короче: *суперпозицией функций* (1.10)).

Исходя из структуры формулы (1.11), нетрудно понять, как, имея функции (1.10), найти значения функции f.

Операция суперпозиции, вообще говоря, не связана с определением функций по индукции. Однако без операции суперпозиции трудно в полной мере охарактеризовать те возможности, которые открываются при использовании операции примитивной рекурсии в определении функции.

Упражнения

2. Подобрав в схеме (1.7) подходящие a и h, получите функции $f_1(x) = x^2$ и $f_2(x) = x^3$.

3. Пусть функция h в схеме (1.8) принимает ровно r значений, где $r > 1$. Сколько значений может принимать функция f?

4. Пусть заданы функции

$$g_1(x_1, \ldots, x_{n-1}), \quad g_2(x_1, \ldots, x_{n-1}), \quad h(x_1, \ldots, x_{n+2}),$$

а функция $f(x_1, \ldots, x_n)$ определяется через функции g_1, g_2, h следующей схемой рекурсии:

$$\begin{cases} f(x_1, \ldots, x_{n-1}, 0) = g_1(x_1, \ldots, x_{n-1}), \\ f(x_1, \ldots, x_{n-1}, 1) = g_2(x_1, \ldots, x_{n-1}), \\ f(x_1, \ldots, x_{n-1}, x_n + 2) = h(x_1, \ldots, x_{n-1}, x_n, f(x_1, \ldots, x_{n-1}, x_n), \\ \qquad\qquad f(x_1, \ldots, x_{n-1}, x_n + 1)). \end{cases}$$

Попытайтесь свести эту не примитивно рекурсивную схему к стандартной схеме (1.6).

§ 1.3. Класс примитивно рекурсивных функций

В предыдущих двух параграфах мы выяснили, как можно определять функции по индукции. Сразу возникает вопрос: а какие вообще функции можно определить по индукции? Немного поразмыслив над этим вопросом, нетрудно прийти к выводу, что ответ на него зависит от того, какие именно функции мы будем использовать в индуктивных определениях в качестве «данных» функций. Понятно, что такие функции должны широко применяться в математической практике и вместе с тем быть настолько простыми, чтобы не допускать индуктивных определений через еще более простые функции. В теории рекурсивных функций давно сформировался список таких функций:

0 (константа 0, которую можно считать функцией, зависящей от одной переменной), $x + 1$, $I_i^n(x_1, \ldots, x_i, \ldots, x_n) = x_i$ (функция выбора i-го аргумента). (1.12)

При этом для функций выбора параметр n принимает все натуральные значения, а параметр i изменяется в пределах от 1 до n. Обозначим совокупность всех функций выбора через I.

Мы подошли к одному из центральных понятий теории рекурсивных функций — понятию *примитивно рекурсивной функции* (английский термин *primitive recursive function*). Это понятие вводится также по индукции.

Итак, базис индукции: функции (1.12) по определению считаем *примитивно рекурсивными функциями*.

Индуктивный переход. Пусть функции $g_0, g_1, \ldots, g_m$ примитивно рекурсивны, а функция f определяется через функции $g_0, g_1, \ldots, g_m$ с помощью операции суперпозиции (1.11). Тогда функция f является *примитивно рекурсивной функцией*. Далее, пусть функции g, h примитивно рекурсивны, а функция f получается из функций g, h с помощью операции примитивной рекурсии (1.6). Тогда функция f также является *примитивно рекурсивной функцией*.

Из приведенного определения следует, что примитивно рекурсивная функция — это функция, которую можно получить из исходных функций (1.12) путем (многократного) применения операций суперпозиции и примитивной рекурсии. Множество всех примитивно рекурсивных функций обычно называют *классом примитивно рекурсивных функций*.

Рассмотрим примеры примитивно рекурсивных функций. Начнем с констант. Функция-константа 0 является исходной примитивно рекурсивной функцией. Подставляя ее в функцию $x + 1$ (т. е. осуществляя суперпозицию функций $x + 1$ и 0), получаем примитивно рекурсивную функцию-константу 1. Аналогичным путем получаем остальные функции-константы $2, 3, \ldots$. Отметим, что любую функцию-констнту c можно считать зависящей от произвольного числа переменных — достаточно рассмотреть суперпозицию вида $c(I_i^n(x_1, \ldots, x_n))$.

Проанализируем теперь схему (1.2). Мы знаем, что она задает функцию $\mathrm{sum}\,(x_1, x_2) = x_1 + x_2$. Чтобы убедиться, что функция $x_1 + x_2$ примитивно рекурсивна, схему (1.2) необходимо привести к виду (1.6), где функции g, h уже примитивно рекурсивны. Для этого в схеме (1.2) переменную x_1, стоящую в правой части первого равенства, можно заменить примитивно рекурсивной функцией $I_1^1(x_1)$, а выражение $\mathrm{sum}\,(x_1, x_2) + 1$ следует рассмотреть как результат замены переменной x_3 в примитивно рекурсивной функции $I_3^3(x_1, x_2, x_3) + 1$ функцией $\mathrm{sum}\,(x_1, x_2)$.

Имея теперь примитивно рекурсивную функцию $\mathrm{sum}\,(x_1, x_2)$, установим примитивную рекурсивность функции $x_1 \cdot x_2$. С этой целью вернемся к первой из схем (1.4). Она, как известно, определяет функцию $\mathrm{prod}\,(x_1, x_2) = x_1 \cdot x_2$. Чтобы согласовать эту схему со схемой при-

митивной рекурсии (1.6), достаточно заметить, что выражение $\operatorname{prod}(x_1, x_2) + x_1$ можно представить в виде

$$\operatorname{sum}(I_3^3(x_1, x_2, \operatorname{prod}(x_1, x_2)), I_1^3(x_1, x_2, x_3)). \tag{1.13}$$

(В дальнейшем мы не будем выписывать громоздкие выражения, подобные выражению (1.13). Поскольку $I_1^3(x_1, x_2, x_3) = x_1$ и $I_3^3(x_1, x_3, x_3) = x_3$, мы будем вместо (1.13) записывать более короткое выражение $\operatorname{sum}(x_1, \operatorname{prod}(x_1, x_2))$.)

Точно так же можно использовать примитивно рекурсивную функцию $x_1 \cdot x_2$ и с помощью первой из схем (1.5) убедиться в примитивной рекурсивности функции $\operatorname{pow}(x_1, x_2) = x_1^{x_2}$.

Нам хотелось бы доказать примитивную рекурсивность функции $x - y$. Однако это невозможно сделать, поскольку функция $x - y$ принимает отрицательные значения. Вместо функции $x - y$ мы рассмотрим близкую к ней функцию $x \div y$, которую определим соотношениями

$$x \div y = \begin{cases} x - y, & \text{если } x \geqslant y, \\ 0 & \text{в противном случае.} \end{cases}$$

Другое определение функции $x \div y$ таково:

$$x \div y = \max(x - y, 0).$$

Доказательство примитивной рекурсивности функции $x \div y$ проведем в два этапа. Сначала установим примитивную рекурсивность более простой функции $x \div 1$:

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

Затем, используя функцию $x \div 1$, получим функцию $x \div y$:

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

Суперпозициями функций $x + y$, $x \div y$ можно определить известные арифметические функции:

$$\min(x, y) = x \div (x \div y),$$
$$\max(x, y) = (x + y) \div \min(x, y),$$
$$|x - y| = (x \div y) + (y \div x).$$

Введем еще две важные функции $\operatorname{sg} x$ (читается: сигнум икс) и $\overline{\operatorname{sg}}\, x$ (читается: не сигнум икс), которые часто встречаются не только в теории рекурсивных функций, но и в других разделах математики. Положим

$$\operatorname{sg} x = \begin{cases} 0, & \text{если } x = 0, \\ 1, & \text{если } x \neq 0; \end{cases} \qquad \overline{\operatorname{sg}}\, x = \begin{cases} 1, & \text{если } x = 0, \\ 0, & \text{если } x \neq 0. \end{cases}$$

Примитивная рекурсивность функций $\operatorname{sg} x$, $\overline{\operatorname{sg}}\, x$ следует из равенств

$$\overline{\operatorname{sg}}\, x = 1 \div x,$$
$$\operatorname{sg} x = \overline{\operatorname{sg}}\,\overline{\operatorname{sg}}\, x.$$

На основе имеющихся примитивно рекурсивных функций можно определить много других интересных примитивно рекурсивных функций. Так, суперпозициями функций (1.12) и функций $x + y$, $x \cdot y$ можно получить любой полином с целыми неотрицательными коэффициентами. Еще более широкие возможности открывает применение функций $x \div y$, $\operatorname{sg} x$, $\overline{\operatorname{sg}}\, x$. Продемонстрируем одну из этих возможностей.

Пусть a — число из N. Примитивно рекурсивная функция $\overline{\operatorname{sg}}\,|x - a|$ принимает значение 1 при $x = a$ и значение 0 в остальных случаях. Если $a_1, \ldots, a_m, b_1, \ldots, b_m$ — числа из N, причем числа $a_1, \ldots, a_m$ попарно различны, то функция

$$b_1 \cdot \overline{\operatorname{sg}}\,|x - a_1| + b_2 \cdot \overline{\operatorname{sg}}\,|x - a_2| + \ldots + b_m \cdot \overline{\operatorname{sg}}\,|x - a_m| \qquad (1.14)$$

при $x = a_1, \ldots, a_m$ принимает соответственно значения $b_1, \ldots, b_m$ и равна 0 в остальных случаях.

Идея построения функции (1.14) позволяет «исправлять» значения примитивно рекурсивной функции в конечном числе точек. Именно, рассмотрим, например, примитивно рекурсивную функцию $f(x)$ одной переменной и предположим, что нам необходимо заменить значения функции $f(x)$ в точках $a_1, \ldots, a_m$ значениями $b_1, \ldots, b_m$. Обозначим «исправленную» так функцию $f(x)$ через $f'(x)$. Тогда

$$f'(x) = b_1 \cdot \overline{\operatorname{sg}}\,|x - a_1| + \ldots + b_m \cdot \overline{\operatorname{sg}}\,|x - a_m| + \\ + f(x) \cdot \operatorname{sg}|x - a_1| \cdot \ldots \cdot \operatorname{sg}|x - a_m|.$$

Разумеется, подобные построения можно выполнить и в случае функции многих переменных.

Упражнения

5. Не заглядывая в § 1.4, докажите примитивную рекурсивность функций $[x/y]$ и $[\log_2 x]$ (здесь $[z]$ означает целую часть числа z; функцию $[x/y]$ при $y = 0$ и функцию $[\log_2 x]$ при $x = 0$ можно определить произвольным образом).

6. Постарайтесь с использованием примитивной рекурсии как можно дальше продолжить ряд функций $x + y$, $x \cdot y$, x^y.

7. Докажите, что примитивно рекурсивной является любая периодическая функция $f(x)$ (функция $f(x)$ называется периодической, если существует такое натуральное число d, что равенство $f(x + d) = f(x)$ выполняется для любого числа x из N).

8. Можно ли в определении примитивно рекурсивной функции отказаться от некоторых функций I_i^n?

§ 1.4. Некоторые свойства примитивно рекурсивных функций

В предыдущем параграфе при задании некоторых функций мы пользовались отношениями вида

$$x = 0, \quad x \neq 0, \quad x < y, \quad x \leqslant y.$$

В принципе можно использовать и более сложные отношения. Мы хотим далее рассматривать произвольные отношения на множестве N. Более точно, мы хотим рассматривать специальные функции $\rho(x_1, \ldots, x_n)$, переменные $x_1, \ldots, x_n$ которых принимают значения из N, а сами функции принимают истинностные значения «истина» и «ложь». Поскольку эти значения не являются числовыми, для применения отношений при задании функций удобно ввести функции, которые принимали бы числовые значения и полностью описывали бы исследуемые отношения. Такие функции называются *характеристическими функциями отношений*.

Итак, пусть $\rho(x_1, \ldots, x_n)$ — отношение на N, а $f(x_1, \ldots, x_n)$ — функция, принимающая лишь значения 0 и 1. Функция $f(x_1, \ldots, x_n)$ называется *характеристической функцией отношения* $\rho(x_1, \ldots, x_n)$, если для любых значений переменных $x_1, \ldots, x_n$ значение $f(x_1, \ldots, x_n)$ равно 1 в том и только том случае, когда значение $\rho(x_1, \ldots, x_n)$ есть «истина».

С помощью характеристических функций понятие примитивной рекурсивности можно перенести с функций на отношения. Именно, отношение ρ считаем примитивно рекурсивным, если примитивно рекурсивной является его характеристическая функция.

Для числовых отношений

$$x = y, \quad x \neq y, \quad x < y, \quad x \leqslant y \tag{1.15}$$

характеристическими функциями служат соответственно функции

$$\overline{\operatorname{sg}}\,|x - y|, \quad \operatorname{sg}|x - y|, \quad \operatorname{sg}(y \div x), \quad \overline{\operatorname{sg}}\,(x \div y).$$

Поэтому отношения (1.15) примитивно рекурсивны.

Как мы могли убедиться, к отношениям часто прибегают, когда необходимо определить функцию с помощью так называемого разбора случаев (см., например, определение функций $x \div y$, $\operatorname{sg} x$, $\overline{\operatorname{sg}}\, x$). Сейчас мы рассмотрим подобную ситуацию в общем виде.

Предположим, что заданы отношения

$$\rho_1(x_1, \ldots, x_n), \ldots, \rho_m(x_1, \ldots, x_n) \tag{1.16}$$

и функции

$$f_1(x_1, \ldots, x_n), \ldots, f_{m+1}(x_1, \ldots, x_n), \tag{1.17}$$

причем отношения (1.16) попарно не «перекрываются», т. е. что при любых значениях переменных $x_1, \ldots, x_n$ истинным может быть не более чем одно из отношений (1.16). На основе отношений (1.16) и функций (1.17) определим функцию f:

$$f(x_1, \ldots, x_n) = \begin{cases} f_1(x_1, \ldots, x_n), & \text{если } \rho_1(x_1, \ldots, x_n) \text{ истинно}, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & \\ f_m(x_1, \ldots, x_n), & \text{если } \rho_m(x_1, \ldots, x_n) \text{ истинно}, \\ f_{m+1}(x_1, \ldots, x_n) & \text{в остальных случаях.} \end{cases} \tag{1.18}$$

Покажем, что в случае примитивной рекурсивности отношений (1.16) и функций (1.17) функция (1.18) также будет примитивно рекурсивной. В самом деле, пусть $g_1, \ldots, g_m$ — характеристические функции отношений $\rho_1, \ldots, \rho_m$. Тогда

$$f(x_1, \ldots, x_n) = \\ = f_1(x_1, \ldots, x_n) \cdot g_1(x_1, \ldots, x_n) + \ldots + f_m(x_1, \ldots, x_n) \cdot g_m(x_1, \ldots, x_m) + \\ + f_{m+1}(x_1, \ldots, x_n) \cdot \overline{\mathrm{sg}}\,(g_1(x_1, \ldots, x_n) + \ldots + g_m(x_1, \ldots, x_n)).$$

Интересные рекурсивные построения можно выполнить на основе функций $x + y$ и $x \cdot y$. Пусть $g(x_1, \ldots, x_n)$ — произвольная функция. По определению

$$\sum_{i \leqslant x_n} g(x_1, \ldots, x_{n-1}, i) \tag{1.19}$$

есть обозначение для суммы

$$g(x_1, \ldots, x_{n-1}, 0) + g(x_1, \ldots, x_{n-1}, 1) + \ldots + g(x_1, \ldots, x_{n-1}, x_n),$$

а

$$\prod_{i \leqslant x_n} g(x_1, \ldots, x_{n-1}, i) \tag{1.20}$$

— для произведения

$$g(x_1, \ldots, x_{n-1}, 0) \cdot g(x_1, \ldots, x_{n-1}, 1) \cdot \ldots \cdot g(x_1, \ldots, x_{n-1}, x_n).$$

Обозначим функцию (1.19) через $f(x_1, \ldots, x_n)$. Если функция g, входящая в (1.19), примитивно рекурсивна, то примитивно рекурсивной будет и функция f. Это сразу следует из соотношений примитивной рекурсии для функции f:

$$\begin{cases} f(x_1, \ldots, x_{n-1}, 0) = g(x_1, \ldots, x_{n-1}, 0), \\ f(x_1, \ldots, x_{n-1}, x_n + 1) = f(x_1, \ldots, x_{n-1}, x_n) + g(x_1, \ldots, x_{n-1}, x_n + 1). \end{cases}$$

Аналогичные рассуждения применимы и к функции (1.20).

В дальнейшем $\sum\limits_{i \leqslant x_n}$ и $\prod\limits_{i \leqslant x_n}$ будем рассматривать как операции, действующие на функции g. Операцию $\sum\limits_{i \leqslant x_n}$ будем называть *ограниченным суммированием*, а $\prod\limits_{i \leqslant x_n}$ — *ограниченным мультиплицированием*.

Применение операций ограниченного суммирования и ограниченного мультиплицирования в ряде случаев позволяет значительно упро-

стить доказательство примитивной рекурсивности некоторых функций. Продемонстрируем это на примерах.

Пусть $[x/y]$ есть целая часть от деления x на y, если $y \neq 0$, и есть 0, если $y = 0$ (выбор здесь числа 0 в качестве значения функции $[x/0]$ существенной роли не играет, с равным успехом можно было бы взять любое другое число). Мы хотим установить примитивную рекурсивность функции $[x/y]$. Заметим, что при $y \neq 0$ величина $[x/y]$ есть такое (единственное!) число i, что $iy \leqslant x$ и $(i+1)y > x$. Обозначим через $g_1(x,y,i)$ и $g_2(x,y,i)$ характеристические функции отношений $iy \leqslant x$ и $(i+1)y > x$ (они, очевидно, примитивно рекурсивны). Поскольку $[x/y] \leqslant x$, получаем теперь

$$[x/y] = \sum_{i \leqslant x} i \cdot g_1(x,y,i) \cdot g_2(x,y,i).$$

С помощью функции $[x/y]$ легко определить функцию $\mathrm{rm}\,(x,y)$, равную остатку от деления x на y, если $y \neq 0$, и равную x, если $y = 0$ (вновь значение x при $y = 0$ выбираем лишь для того, чтобы получить более простую формулу для выражения функции $\mathrm{rm}\,(x,y)$). В самом деле, имеем

$$\mathrm{rm}(x,y) = x \div [x/y] \cdot y.$$

Найденный выше прием применим для доказательства примитивной рекурсивности еще двух функций $[\sqrt{x}\,]$ и $[\log_2 x]$ (здесь вновь для простоты положим $\log_2 0 = 0$). Если $g_1(x,i), g_2(x,i)$ — характеристические функции примитивно рекурсивных отношений $i^2 \leqslant x$ и $(i+1)^2 > x$, то

$$[\sqrt{x}\,] = \sum_{i \leqslant x} i \cdot g_1(x,i) \cdot g_2(x,i).$$

Аналогично, если $h_1(x,i), h_2(x,i)$ — характеристические функции примитивно рекурсивных отношений $2^i \leqslant x$ и $2^{i+1} > x$, то

$$[\log_2 x] = \sum_{i \leqslant x} i \cdot h_1(x,i) \cdot h_2(x,i).$$

Понятно, что этим приемом можно доказать примитивную рекурсивность функции $[\log_a x]$ при любом натуральном $a \geqslant 3$, а также функции $[\log_y x]$.

Обозначим через $x|y$ отношение «x делит нацело y» (отношение $0|y$ считаем истинным только при $y = 0$). Нетрудно видеть, что отношение $x|y$ эквивалентно отношению

$$[y/x] \cdot x = y,$$

откуда вытекает его примитивная рекурсивность.

Используя отношение $x|y$, попробуем подсчитать количество делителей числа x. Соответствующую функцию обозначим через $d(x)$. Если $g(x,i)$ — характеристическая функция отношения $i|x$, то, очевидно,

$$d(x) = \sum_{i \leqslant x} g(x,i).$$

Как известно, простым числом называется натуральное число $p > 1$, которое имеет только два делителя, 1 и p. Обозначим через $\Pr(x)$ отношение «x есть простое число». Тогда $\Pr(x)$ истинно в том и только том случае, когда $x > 1$ и $d(x) = 2$. Следовательно, характеристическая функция отношения $\Pr(x)$ равна произведению характеристических функций отношений $x > 1$ и $d(x) = 2$. Отсюда вытекает примитивная рекурсивность отношения $\Pr(x)$.

В теории чисел иногда бывает необходимо найти произведение всех простых чисел, не превосходящих заданного числа x. Эту задачу решает примитивно рекурсивная функция

$$\prod_{i \leqslant x} (i \cdot g(i) + \overline{\operatorname{sg}}\, g(i)),$$

где $g(x)$ — характеристическая функция отношения $\Pr(x)$.

Упражнения

9. Пусть $P(x_1, \ldots, x_n), Q(x_1, \ldots, x_n)$ — многочлены с натуральными коэффициентами, а $f(x)$ равно числу решений уравнения

$$P(x_1, \ldots, x_n) = Q(x_1, \ldots, x_n)$$

при ограничениях $0 \leqslant x_1 \leqslant x, \ldots, 0 \leqslant x_n \leqslant x$. Докажите примитивную рекурсивность функции $f(x)$.

10. Пусть $\exp_2(x)$ равно показателю числа 2 в разложении x на простые множители (при $x = 0, 1$ значение $\exp_2(x)$ можно выбрать произвольно). Докажите примитивную рекурсивность функции $\exp_2(x)$.

11. Пусть $g(x)$ — примитивно рекурсивная функция, а функция $f(x,y)$ определяется так:

$$f(x,y) = \begin{cases} \text{наименьшему решению } z \text{ уравнения } g(z) = y, \\ \quad \text{не превосходящему } x, \text{ если такое решение } z \text{ существует}, \\ 0 \text{ в противном случае.} \end{cases}$$

Докажите, что функция $f(x,y)$ примитивно рекурсивна.

12. Пусть $p(x)$ равно $(x+1)$-му простому числу, т. е. $p(0) = 2$, $p(1) = 3$, $p(2) = 5, \ldots$. Докажите, что функция $p(x)$ примитивно рекурсивна.

13. Пусть $f(x)$ равно $(x+1)$-му десятичному разряду после запятой в разложении числа $\sqrt{2}$. Докажите примитивную рекурсивность функции $f(x)$.

§ 1.5. Элементарные рекурсивные функции

Хотя класс примитивно рекурсивных функций в теории рекурсивных функций считается сравнительно простым, некоторые примитивно рекурсивные функции все же следует признать достаточно сложными. Так, примитивно рекурсивная функция

$$f(x) = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \quad \text{($x+1$) раз} \tag{1.21}$$

уже трудна для восприятия (не говоря уже о том, что, например, число $f(5)$ превосходит известные «астрономические» числа). Стоит еще отметить, что приведенная функция $f(x)$ далеко не самая «большая» в классе примитивно рекурсивных функций.

Эти и некоторые другие соображения заставили специалистов в области рекурсивных функций заняться поисками более узких классов примитивно рекурсивных функций — классов, которые состояли бы из «почти всех» функций, встречающихся в реальной математической практике. Такие классы были определены и получили название классов «элементарных» функций. Не стоит думать, что все функции, составляющие эти классы, являются совсем уж простыми. Нет, это далеко не так. Однако в классах элементарных функций отсутвуют «монстры», подобные функции (1.21).

В этом параграфе мы рассмотрим три класса элементарных функций: класс **S** функций, элементарных по Сколему, один из классов **E** иерархии Гжегорчика и класс **K** функций, элементарных по Кальмару. Начнем с класса **S**.

Так же, как класс примитивно рекурсивных функций, класс **S** определяется по индукции. Исходными функциями класса **S** являются функции $x+1, x \div y, I$, а порождающими операциями — операции суперпозиции и ограниченного суммирования. Понятно, что все функции класса **S** примитивно рекурсивны.

Несмотря на кажущуюся простоту класса **S**, он содержит многие примитивно рекурсивные функции, рассмотренные в предыдущих параграфах. Так, например, классу **S** принадлежат функции 0, $x+y$ и $x \cdot y$. Это сразу следует из равенств

$$0 = x \div x, \quad x(y+1) = \sum_{i \leqslant y} I_1^2(x,i), \quad xy = x(y+1) \div x,$$
$$x + y = (x+1)(y+1) \div (xy+1).$$

Суперпозициями функций $x+1$, $x+y$, $x \cdot y$, $x \div y$ можно, как и выше, получить в классе **S** функции

$$\operatorname{sg} x, \quad \overline{\operatorname{sg}}\, x, \quad |x-y|, \quad \min(x,y), \quad \max(x,y), \tag{1.22}$$

а также произвольные полиномы с натуральными коэффициентами. Кроме того, операция ограниченного суммирования позволяет определить в классе **S** функции $[x/y], [\sqrt{x}\,]$ и отношения $x|y$, $\Pr(x)$ (как

и для примитивно рекурсивных функций, считаем, что отношение ρ принадлежит классу **S**, если классу **S** принадлежит характеристическая функция отношения ρ).

Нетрудно видеть, что каждая функция из **S** ограничена сверху подходящим полиномом с натуральными коэффициентами. Этот факт устанавливается индукцией по построению функций в классе **S**. Для исходных функций $x+1$, $x \div y$, I класса **S** утверждение очевидно. Далее, если функции (1.10) класса **S** ограничены сверху соответственно полиномами

$$P_0(y_1, \ldots, y_m),\ P_1(x_1, \ldots, x_n), \ldots, P_m(x_1, \ldots, x_n),$$

а функция $f(x_1, \ldots, x_n)$ получается из функций (1.10) суперпозицией (1.11), то ввиду монотонности полиномов $P_0, P_1, \ldots, P_m$ по каждой переменной функция $f(x_1, \ldots, x_n)$ будет ограничена сверху функцией

$$P_0(P_1(x_1, \ldots, x_n), \ldots, P_m(x_1, \ldots, x_n)),$$

которая после очевидных преобразований превращается в полином с натуральными коэффициентами.

Пусть теперь функция $g(x_1, \ldots, x_n)$ из класса **S** ограничена сверху полиномом $P(x_1, \ldots, x_n)$ с натуральными коэффициентами, а функция $f(x_1, \ldots, x_n)$ получается из функции g с помощью операции ограниченного суммирования (1.19). Пользуясь монотонностью полинома $P(x_1, \ldots, x_n)$ по переменной x_n, убеждаемся, что функция $f(x_1, \ldots, x_n)$ будет ограничена сверху функцией $P(x_1, \ldots, x_n) \cdot (x_n + 1)$, которая приводится к полиному с натуральными коэффициентами.

Доказанное утверждение позволяет легко найти примитивно рекурсивную функцию, не входящую в класс **S**. Такой функцией будет, например, функция 2^x — она не может быть ограничена сверху никаким полиномом (от x) с натуральными коэффициентами.

При первом знакомстве с примитивно рекурсивными функциями может создаться впечатление, что классу **S** принадлежит любая примитивно рекурсивная функция, которая ограничена сверху некоторым полиномом с натуральными коэффициентами. Однако это не так. Можно даже построить примитивно рекурсивную функцию, принимающую лишь значения 0 и 1, которая тем не менее не входит в класс **S**. К сожалению, построение такой функции сопряжено со значительными техническими трудностями и выходит за рамки этой брошюры.

Класс **E** элементарных функций Гжегорчика определяется также по индукции. Исходными функциями класса **E** являются исходные функции (1.12) класса примитивно рекурсивных функций. Порождающими операциями служат операции суперпозиции и примитивной рекурсии. Однако в класс **E** зачисляются не все функции, полученные с помощью операции примитивной рекурсии, а только такие, которые ограничены сверху подходящими полиномами с натуральными коэффициентами. Легко понять, что при этом условии в класс **E** попадут лишь функции,

ограниченные сверху полиномами с натуральными коэффициентами (и вновь не все такие функции!).

Проанализировав построение примитивно рекурсивных функций в предыдущих параграфах, нетрудно прийти к выводу, что в классе **E** окажутся функции 1, $x+y$, $x\cdot y$ (а значит, и произвольные полиномы с натуральными коэффициентами) и функции (1.22). Более того, как мы знаем, операция ограниченного суммирования является частным случаем операции примитивной рекурсии. При этом свойство «быть ограниченным сверху полиномом с натуральными коэффициентами» сохраняется при применении операции ограниченного суммирования (см. соответствующее рассуждение при исследовании класса **S**). Отсюда следует, что все функции класса **S** входят в класс **E**.

Классы **S** и **E** близки друг к другу. Совпадают ли эти классы? Ответ на этот вопрос позволил бы существенно продвинуться в понимании природы рекурсивных функций.

В заключение параграфа рассмотрим класс **K** функций, элементарных по Кальмару. Класс **K** определяется почти так же, как класс **S**. Отличие состоит в том, что к порождающим операциям класса **S** добавляется операция ограниченного мультиплицирования. Из определения класса **K** сразу следует, что все функции, элементарные по Сколему, являются также элементарными по Кальмару.

Как мы знаем, в силу соображений о скорости роста функция x^y не может входить в классы **S** и **E**. Однако ее легко получить в классе **K**. В самом деле, имеем

$$x^{y+1}=\prod_{i\leqslant y} I_1^2(x,i),\quad x^y=\left[\frac{x^{y+1}}{x}\right]$$

(согласно этим равенствам получаем $0^0=0$, хотя в § 1.1 мы приняли $0^0=1$; при желании «исправить» значение функции x^y в точке (0,0) можно с помощью функций $\operatorname{sg}x$, $\overline{\operatorname{sg}}\,x$, $x+y$, $x\cdot y$). Таким образом, класс **K** оказывается шире классов **S** и **E**.

Насколько велик класс **K**? Можно показать (попытайтесь сделать это самостоятельно), что класс **K** не содержит функцию (1.21) и, следовательно, не совпадает с классом всех примитивно рекурсивных функций. С другой стороны, класс **K** является самым большим из известных классов элементарных функций — настолько большим, что всякая «естественная» функция (заданная, разумеется, на множестве N), которая встречается в комбинаторике, теории чисел или математическом анализе, оказывается в классе **K**. В качестве примера приведем лишь четыре функции (полное доказательство их принадлежности классу **K** может занять не одну страницу текста):

$$[e^x],\quad [\log x],\quad [\pi x],\quad [e^{\pi x}].$$

Класс **K** имеет несколько различных определений. Одно из них аналогично определению класса **E**. Отличие состоит в том, что вместо

полиномов с натуральными коэффициентами рассматриваются «экспоненциальные полиномы» — функции, получаемые суперпозициями функций $1,\ x+y,\ x\cdot y,\ x^y$.

Упражнения

14. Докажите, что в класс **S** входит функция $[\log_2 x]$.

15. Докажите, что классу **S** принадлежит функция $e(x)$, равная числу единиц в двоичном представлении числа x.

16. Пусть функция $g(x_1,\dots,x_n)$ принадлежит классу **S** и существует такое натуральное число b, что для любых значений $a_1,\dots,a_{n-1}$ функция $g(a_1,\dots,a_{n-1},x_n)$ принимает не более b значений, отличных от 0 и 1. Докажите, что классу **S** принадлежит функция (1.20).

17. Пусть функции g,h принадлежат классу **E**, а функция f получается из них с помощью операции примитивной рекурсии (1.6). Докажите, что классу **E** принадлежит такая функция $f'(x_1,\dots,x_n,y)$, что

$$f'(x_1,\dots,x_n,y)=\begin{cases} f(x_1,\dots,x_n), \text{ если для всех значений } i\leqslant x_n \\ \quad \text{выполняется неравенство } f(x_1,\dots,x_n,i)\leqslant y, \\ 0 \text{ в противном случае.}\end{cases}$$

18. Определим функцию $F(x)$ соотношениями

$$F(0)=F(1)=1,\quad F(x+2)=F(x)+F(x+1).$$

Числа вида $F(x)$ носят название *числа Фибоначчи*. Докажите, что функция $F(x)$ входит в класс **K**.

19. Докажите, что в определении класса **K** можно исключить операцию ограниченного мультиплицирования, если к исходным функциям класса **K** добавить функцию x^y.

Глава 2

ЧАСТИЧНО РЕКУРСИВНЫЕ ФУНКЦИИ

§ 2.1. Непримитивные рекурсии

Алгоритмически вычислимые функции можно определять не только с помощью обычной индукции (примитивной рекурсии). Существуют более сложные формы индукции и отвечающие им схемы рекурсии. Рассмотрим в качестве примера функцию $F(x, y)$, задаваемую следующими соотношениями:

$$\begin{aligned} &F(0, y) = y + 2, \quad F(x + 1, 0) = 1, \\ &F(x + 1, y + 1) = F(x, F(x + 1, y)). \end{aligned} \tag{2.1}$$

Убедимся, что равенства (2.1) действительно определяют единственную функцию $F(x, y)$.

В самом деле, если $x = 0$ или $y = 0$, то значения $F(x, y)$ однозначно находятся из первого или второго равенств системы (2.1). Рассмотрим далее значения $F(1, y + 1)$. Пользуясь третьим и первым равенствами, находим, что

$$F(1, y + 1) = F(0, F(1, y)) = F(1, y) + 2.$$

Эти равенства вместе с равенством $F(1, 0) = 1$ дают схему примитивной рекурсии для определения функции $f_1(y) = F(1, y)$:

$$\begin{cases} f_1(0) = 1, \\ f_1(y + 1) = f_1(y) + 2. \end{cases}$$

Понятно, что

$$f_1(y) = F(1, y) = 2y + 1.$$

Сделаем еще один шаг по пути вычисления значений функции F — рассмотрим значения $F(2, y + 1)$. Третье из равенств (2.1) вместе с установленным соотношением $F(1, y) = 2y + 1$ дают

$$F(2, y + 1) = F(1, F(2, y)) = 2F(2, y) + 1.$$

Если ввести обозначение $f_2(y) = F(2, y)$, то, учитывая равенство $F(2, 0) = 1$, получим схему примитивной рекурсии для определения функции $f_2(y)$:

$$\begin{cases} f_2(0) = 1, \\ f_2(y+1) = 2f_2(y) + 1. \end{cases}$$

Из этой схемы нетрудно получить, что

$$f_2(y) = \underbrace{2(\ldots 2(2}_{y} \cdot 1 + 1) + 1 + \ldots) + 1 = 2^y + 2^{y-1} + \ldots + 2^0 = 2^{y+1} - 1.$$

Итак,

$$F(2, y) = 2^{y+1} - 1.$$

Продолжая в том же духе и вводя обозначение $f_3(y) = F(3, y)$, будем иметь схему примитивной рекурсии для определения функции $f_3(y)$:

$$\begin{cases} f_3(0) = 1, \\ f_3(y+1) = f_2(f_3(y)) = 2^{f_3(y)+1} - 1, \end{cases}$$

т. е. функция $f_3(y)$ представляет собой итерацию функции f_2 в точке 1.

Вообще, рассуждая по индукции (обычная индукция по x), можно убедиться в том, что для всякого натурального x функция $f_{x+1}(y) = F(x+1, y)$ (как функция от y) является итерацией функции $f_x(y)$ в точке 1. Эти соображения наводят на мысль, что функция $F(x, y)$ слишком быстро растет, чтобы быть примитивно рекурсивной функцией. И действительно, довольно сложными рассуждениями можно показать, что функция $F(x, y)$ не входит в класс примитивно рекурсивных функций (именно по причине слишком быстрого роста). Вместе с тем достаточно понятно, что функция $F(x, y)$ алгоритмически вычислима — равенства (2.1) можно рассматривать как «программу» вычисления функции F.

Посмотрим на равенства (2.1) еще с одной точки зрения. Выше мы обнаружили, что функцию $F(x, y)$ можно определить в помощью двух индукций (одновременных, а не последовательных!): индукцией по x при переходе от функции f_x к функции f_{x+1} и индукцией по y при определении значений функции f_x. Это дает нам право говорить о том, что функция $F(x, y)$ определяется с помощью индукции специального вида (или рекурсии специального вида). Рекурсия (2.1) носит название *рекурсия второй ступени*.

Таким образом, класс алгоритмически вычислимых функций можно расширить, рассматривая наряду с операциями суперпозиции и примитивной рекурсии еще и рекурсию второй ступени (мы сознательно не стали приводить общую схему рекурсии второй ступени, поскольку она довольно громоздка).

Можно ли продвинуться еще дальше на этом пути определения рекурсивных функций? Да, можно. Существуют рекурсии третьей сту-

пени, четвертой ступени и т. д., которые позволяют определять все более сложные рекурсивные функции. Однако есть ряд обстоятельств принципиального характера, которые препятствуют построению легко определяемой и удобной в обращении иерархии типов рекурсии. Поэтому далее мы не будем рассматривать новые типы рекурсии, а сразу перейдем к частичным функциям и операции минимизации, существенно более сильной, нежели рекурсия.

§ 2.2. Частичные функции и операция минимизации

Появление частично определенных функций есть неизбежная плата за возможность рассматривать произвольные алгоритмически вычислимые функции. Как это отражено в названии, частично определенная функция $f(x_1, \dots, x_n)$ может быть не определена на некоторых наборах значений переменных $x_1, \dots, x_n$ (всюду определенные функции мы также будем считать частично определенными). Частично определенные функции — не редкость в математической практике. Например, функцию x/y естественно считать неопределенной при $y = 0$. То же самое относится к функции x^y при $x = y = 0$ и функции $\log_2 x$ при $x = 0$.

Частично определенные функции часто получаются из всюду определенных функций в результате применения тех или иных операций. Нас в первую очередь будут интересовать эффективные операции — операции, которые сохраняют алгоритмическую вычислимость функций. К таким операциям относится операция *минимизации*.

Пусть $g(x_1, \dots, x_n)$ — частично определенная функция. Определим функцию $f(x_1, \dots, x_n)$ — результат применения операции минимизации μ к функции g,

$$f(x_1, \dots, x_n) = (\mu y)(g(x_1, \dots, x_{n-1}, y) = x_n). \tag{2.2}$$

Считаем, что для произвольного набора $(a_1, \dots, a_n)$ значение $f(a_1, \dots, a_n)$ определено и равно числу b в том и только том случае, когда:

1) значение $g(a_1, \dots, a_{n-1}, b)$ определено и равно a_n;

2) при любом $z < b$ значение $g(a_1, \dots, a_{n-1}, z)$ определено и отлично от a_n.

Понятно, что с помощью операции минимизации находится наименьшее (по y) решение уравнения

$$g(x_1, \dots, x_{n-1}, y) = x_n. \tag{2.3}$$

Существенное ограничение здесь состоит в том, что в процессе поиска такого решения (конечно, если оно имеется) мы не можем «перескакивать» через те точки y, в которых функция g не определена. Может показаться, что это ограничение носит искусственный характер. Однако без этого ограничения операция минимизации перестает быть эффективной операцией. Иными словами, если бы операция минимизации

находила наименьшее решение уравнения (2.3) без всяких ограничений, то можно было бы указать алгоритмически вычислимую функцию g, из которой в результате применения такой операции получалась бы алгоритмически невычислимая функция.

Посмотрим на примерах, как действует операция минимизации. Пусть $g_1(x) = x + 1$ и

$$f_1(x) = (\mu y)(g_1(y) = x).$$

Тогда, как нетрудно видеть,

$$f_1(x) = \begin{cases} x - 1, & \text{если } x > 0, \\ \text{не определено}, & \text{если } x = 0. \end{cases}$$

Пусть $g_2(x)$ есть функция-константа a и

$$f_2(x) = (\mu y)(g_2(y) = x).$$

Тогда

$$f_2(x) = \begin{cases} 0, & \text{если } x = a, \\ \text{не определено}, & \text{если } x \neq a. \end{cases}$$

Интересный результат получается, если операцию минимизации применить к функции $f_1(x)$. Итак, положим

$$f_3(x) = (\mu y)(f_1(y) = x).$$

Тогда, как легко убедиться, функция $f_3(x)$ не будет определена ни при одном значении x. Это так называемая *нигде не определенная функция*. Она играет важную роль в теории алгоритмов. Может показаться странным, что функция $f_3(x)$ является алгоритмически вычислимой. Тем не менее это так. Об этом мы еще скажем в главе III. А пока отметим один важный момент: применение операции минимизации даже к весьма простым всюду определенным функциям может привести к «существенно» частичным функциям.

Рассмотрим еще несколько примеров применения операции минимизации. Пусть

$$f_4(x) = (\mu y)(y^2 = x), \quad f_5(x) = (\mu y)(2^y = x).$$

Тогда

$$f_4(x) = \begin{cases} \sqrt{x}\,, & \text{если } x \text{ есть полный квадрат}, \\ \text{не определено} & \text{в противном случае}, \end{cases}$$

$$f_5(x) = \begin{cases} \log_2 x, & \text{если } x \text{ есть степень числа 2}, \\ \text{не определено} & \text{в противном случае}. \end{cases}$$

Применение операции минимизации к одной и той же функции, но по разным переменным, может привести к существенно различным результатам. Так, если

$$f_6(x, y) = (\mu z)(z \div x = y), \quad f_7(x, y) = (\mu z)(x \div z = y),$$

то

$$f_6(x,y)=\begin{cases}0, & \text{если } y=0,\\ x+y, & \text{если } y\neq 0;\end{cases}$$

$$f_7(x,y)=\begin{cases}x-y, & \text{если } x\geqslant y,\\ \text{не определено}, & \text{если } x<y.\end{cases}$$

Помимо формы (2.2) операция миинимизации существует еще в нескольких формах. Приведем две из них:

$$f(x_1,\ldots,x_{n-1})=(\mu y)(g(x_1,\ldots,x_{n-1},y)=0),$$
$$f(x_1,\ldots,x_{n-1})=(\mu y)(g_1(x_1,\ldots,x_{n-1},y)=g_2(x_1,\ldots,x_{n-1},y)).$$

Упражнения

20. Докажите, что результат применения операции минимизации к всюду определенной функции есть функция, отличная от нигде не определенной функции.

21. Примените операцию минимизации к функциям $x+y$, $x\cdot y$, $x-y$ (значение этой функции не определено, если $x<y$) и x/y (значение этой функции считаем неопределенным, если либо $y=0$, либо $y\neq 0$ и x не является кратным y).

22. Подберите такие функции $g_1(x,y)$, $g_2(x,y)$, чтобы применение к ним операции минимизации давало функции $x+y$ и $x\cdot y$.

§ 2.3. Класс частично рекурсивных функций

Уточнением понятия алгоритмически вычислимой функции служит понятие *частично рекурсивной функции* [1]). Имеется несколько эквивалентных определений частично рекурсивной функции. Мы дадим определение, использующее операцию минимизации (иногда в этой связи говорят о μ-рекурсивных функциях).

Определение частично рекурсивной функции получается дальнейшим расширением определения примитивно рекурсивной функции. В качестве исходных частично рекурсивных функций берутся функции $0, x+1, I$, а к операциям суперпозиции и примитивной рекурсии добавляется операция минимизации.

Необходимо еще уточнить, как действуют операции суперпозиции и примитивной рекурсии на частичные функции. Пусть функция $f(x_1,\ldots,x_n)$ получается из (вообще говоря, частичных) функций (1.10) с помощью операции суперпозиции (1.11). Для любого набора

[1]) Термин *частично рекурсивная функция* является не вполне удачным переводом английского термина *partial recursive function*. Правильнее было бы говорить о *частичных рекурсивных функциях*.

$(a_1, \dots, a_n)$ значение $f(a_1, \dots, a_n)$ считаем определенным только в том случае, когда определены все значения

$$g_1(a_1, \dots, a_n), \dots, g_m(a_1, \dots, a_n) \tag{2.4}$$

и определено значение функции g_0 на наборе с компонентами (2.4).

Пусть теперь функция $f(x_1, \dots, x_n)$ получается из функций g, h с помощью операции примитивной рекурсии (1.6). Здесь значение $f(a_1, \dots, a_n)$ считаем определенным только в том случае, когда определено значение $g(a_1, \dots, a_{n-1})$ и для любого $b < a_n$ определено также значение

$$h(a_1, \dots, a_{n-1}, b, f(a_1, \dots, a_{n-1}, b))$$

(разумеется, значения $f(a_1, \dots, a_{n-1}, b)$ при этом вычисляются согласно равенствам (1.6)).

Понятно, что всякая примитивно рекурсивная функция одновременно является частично рекурсивной. Вместе с тем, как мы видели в § 2.2, среди частично рекурсивных функций действительно имеются частичные функции. Поэтому класс частично рекурсивных функций шире класса примитивно рекурсивных функций. На самом деле существуют всюду определенные частично рекурсивные функции (их иногда называют общерекурсивными), которые не входят в класс примитивно рекурсивных функций. У нас нет технических возможностей, чтобы доказать этот факт. Однако мы наметим пути для доказательства частичной рекурсивности функции $F(x, y)$, которая была введена в § 2.1.

Пусть $x > 0$ и $y > 0$. Чтобы найти значение $F(x, y)$, нам необходимо согласно равенствам (2.1) иметь определенное количество значений функции F в точках (s, v), где либо $s < x$ и, вообще говоря, $v > y$, либо $s = x$ и $v < y$. Мы не можем сказать заранее, для каких именно точек потребуются значения функции F. Поэтому запишем их пока в виде последовательности с неопределенными величинами $v_0, \dots, v_{x-1}$:

$$\begin{aligned} &F(0,0), F(0,1), \dots, F(0,v_0);\ F(1,0), F(1,1), \dots, F(1,v_1); \dots;\ F(x-1,0),\\ &\quad F(x-1,1), \dots, F(x-1,v_{x-1}); F(x,0), F(x,1), \dots, F(x,y-1). \end{aligned} \tag{2.5}$$

Часть значений из последовательности (2.5) дается непосредственно равенствами (2.1). Именно

$$\begin{aligned} &F(0,0) = 2, F(0,1) = 3, \dots, F(0,v_0) = v_0 + 2,\\ &F(1,0) = F(2,0) = \dots = F(x,0) = 1. \end{aligned} \tag{2.6}$$

Что касается остальных значений, то для их получения нам придется воспользоваться третьим из равенств (2.1).

Итак, найти значение $F(x, y)$ можно, если предварительно создать последовательность (2.5) с достаточно большими (и пока неопределенными) величинами $v_0, \dots, v_{x-1}$, в которой часть значений известна и задается равенствами (2.6), а остальные значения определяются рекурсивно с использованием третьего из равенств (2.1) (предполагается,

что необходимые для этого значения $F(x, y-1)$ и $F(x-1, F(x, y-1))$ присутствуют в последовательности (2.5)).

В последовательность (2.5) входит символ функции F. Тем самым мы как бы предполагаем, что нужные нам значения функции F уже найдены. Однако это не так. Поэтому последовательность (2.5) имеет смысл переписать несколько иначе:

$$(0,0,b_{00}),(0,1,b_{01}),\dots,(0,v_0,b_{0v_0});(1,0,b_{10}),(1,1,b_{11}),\dots,(1,v_1,b_{1v_1});\dots$$
$$\dots;(x-1,0,b_{x-1,0}),(x-1,1,b_{x-1,1}),\dots,(x-1,v_{x-1},b_{x-1,v_{x-1}});$$
$$(x,0,b_{x0}),(x,1,b_{x1}),\dots,(x,y-1,b_{x,y-1}). \quad (2.7)$$

В этой последовательности числа $b_{00}, b_{01}, \dots, b_{0v_0}$ и $b_{10}, \dots, b_{x-1,0}$ определяются согласно первым двум равенствам (2.1), а остальные числа — с использованием третьего равенства (2.1). Наборы из троек чисел, составляющие последовательность (2.7), мы закодируем далее числами из N (о способе кодирования чуть позже). В результате образуется последовательность кодов

$$c_{00}, c_{01}, \dots, c_{0v_0}; c_{10}, c_{11}, \dots, c_{1v_1}; \dots; c_{x-1,0}, c_{x-1,1}, \dots, c_{x-1,v_{x-1}};$$
$$c_{x0}, c_{x1}, \dots, c_{x,y-1}. \quad (2.8)$$

Наконец, эту последовательность также закодируем одним числом d (способ кодирования пока оставляем в стороне).

Теперь для вычисления значения $F(x, y)$ следует найти (наименьшее!) число d, которое есть код последовательности вида (2.8), в которой числа c_{ij} в свою очередь являются кодами троек (i, j, b_{ij}). Данные тройки должны образовывать последовательность (2.7), элементы которой подчиняются известным нам соотношениям.

Оказывается, что свойство числа d, которое изложено в предыдущем абзаце, можно выразить с помощью примитивно рекурсивной функции. Разумеется, на этом пути немало довольно серьезных технических трудностей. Тем не менее предложенный путь доказательства частичной рекурсивности функции $F(x, y)$ представляется одним из наиболее простых.

Обратимся теперь к вопросу о кодировании троек. Сначала определим примитивно рекурсивную функцию, кодирующую пары. В качестве такой функции возьмем функцию

$$c(x, y) = (x + y)^2 + x.$$

Несложно убедиться, что функция c различным парам (x, y) сопоставляет различные числа $c(x, y)$. В самом деле, пусть $(x_1, y_1) \neq (x_2, y_2)$. Если $x_1 + y_1 = x_2 + y_2$, то $x_1 \neq x_2$ и потому

$$(x_1 + y_1)^2 + x_1 \neq (x_2 + y_2)^2 + x_2.$$

Если же $x_1 + y_1 \neq x_2 + y_2$, то величины $c(x_1, y_1)$ и $c(x_2, y_2)$ расположены между соседними квадратами различных чисел: $(x_1 + y_1)^2$,

$(x_1 + y_1 + 1)^2$ и соответственно $(x_2 + y_2)^2$, $(x_2 + y_2 + 1)^2$. Поэтому $c(x_1, y_1) \neq c(x_2, y_2)$.

Отметим, что функция $c(x, y)$ принимает далеко не все значения из N. Именно, при любых x, y в ее область значений не входят числа

$$(x + y)^2 + x + 1, \ldots, (x + y)^2 + 2x + 2y.$$

Тем не менее по коду $v = c(x, y)$ пары (x, y) довольно просто вычислить элементы x и y. Это достигается с помощью примитивно рекурсивных функций $l(v)$ и $r(v)$:

$$l(v) = v \div [\sqrt{v}]^2, \quad r(v) = [\sqrt{v}] \div l(v).$$

Кодирование троек можно осуществить с помощью функции

$$c^3(x, y, z) = c(c(x, y), z).$$

«Обратными» функциями, доставляющими по коду $v = c^3(x, y, z)$ компоненты x, y, z, будут суперпозиции функций l и r: соответственно $l(l(v))$, $r(l(v))$ и $r(v)$.

Вообще, при любом $n \geqslant 3$ кодирование n-ок можно выполнить с помощью функции $c^n(x_1, \ldots, x_n)$, где

$$c^2(x_1, x_2) = c(x_1, x_2),\ c^{n+1}(x_1, \ldots, x_{n+1}) = c(c^n(x_1, \ldots, x_n), x_{n+1}).$$

Если $v = c^n(x_1, \ldots, x_n)$, то

$$x_1 = \underbrace{l(\ldots l}_{n-1}(v)\ldots),\ x_2 = r(\underbrace{l(\ldots l}_{n-2}(v)\ldots)), \ldots, x_{n-1} = r(l(v)),\ x_n = r(v).$$

Для кодирования последовательности (2.8) одним числом d также можно использовать подходящую функцию c^n. Чтобы при декодировании избежать многократных суперпозиций функции l, определим примитивно рекурсивные функции l' и l_1:

$$\begin{cases} l'(v, 0) = v, \\ l'(v, i + 1) = l(l'(v, i)), \end{cases} \qquad l_1(v, i) = r(l'(v, i)).$$

Если теперь $v = c^{n+1}(a_n, \ldots, a_0)$, то при любом $i < n$ будем иметь $l_1(v, i) = a_i$.

Упражнения

23. Пусть $a_1, \ldots, a_s$ — различные числа из N и

$$f(x) = \begin{cases} 1, & \text{если } x \text{ совпадает с одним из чисел } a_1, \ldots, a_s, \\ \text{не определено} & \text{в остальных случаях.} \end{cases}$$

Докажите, что функция $f(x)$ частично рекурсивна.

24. Докажите частичную рекурсивность функции $g(x)$, если

$$g(x) = \begin{cases} 0, & \text{если существует такое } i, \text{ что } l_1(x, i) = 1, \\ \text{не определено} & \text{в противном случае.} \end{cases}$$

25. Постарайтесь довести до конца доказательство частичной рекурсивности функции $F(x, y)$.

26. Разработайте такую (примитивно рекурсивную) нумерацию пар, чтобы каждое число из N было номером ровно одной пары.

§ 2.4. Рекурсивно перечислимые множества. Нормальная форма Клини

Подмножество M множества N называется *рекурсивно перечислимым* (английский термин *recursively enumerable set*), если существует частично рекурсивная функция f, область значений которой совпадает с множеством M.

Сделаем два замечания по поводу введенного определения. Во-первых, функция f может перечислять множество M с повторениями, т. е. принимать одинаковые значения на различных наборах значений переменных. Во-вторых, рекурсивно перечислимым согласно определению оказывается пустое множество $\varnothing$ (множество, не содержащее ни одного элемента), как множество значений нигде не определенной функции.

Ниже мы докажем, что за исключением пустого множества всякое рекурсивно перечислимое множество может быть перечислено подходящей примитивно рекурсивной функцией. Прежде чем доказать этот довольно неожиданный факт, мы приведем еще некоторые необходимые определения.

Прежде всего распространим понятие рекурсивно перечислимого множества на множества наборов. Именно, пусть $n > 1$ и M есть множество наборов $(a_1, \ldots, a_n)$, где $a_1, \ldots, a_n$ — элементы множества N. Будем считать множество M рекурсивно перечислимым, если рекурсивно перечислимым является множество номеров $c^n(a_1, \ldots, a_n)$ всех наборов $(a_1, \ldots, a_n)$ из M.

Пусть $f(x_1, \ldots, x_n)$ — произвольная функция (вообще говоря, частичная). *Графиком функции* f называется множество всех наборов вида $(a_1, \ldots, a_n, f(a_1, \ldots, a_n))$.

Индукцией по построению частично рекурсивной функции докажем, что график любой частично рекурсивной функции, отличной от нигде не определенной функции, можно перечислить подходящей примитивно рекурсивной функцией (перечисляется, конечно, множество номеров этого графика). Отсюда легко будет следовать наше основное утверждение, что всякое непустое рекурсивно перечислимое множество есть область значений некоторой примитивно рекурсивной функции.

Базис индукции — исходные частично рекурсивные функции 0, $x + 1$, I. Для них соответствующими примитивно рекурсивными функциями, перечисляющими графики, будут функции

$$c(x, 0),\ c(x, x + 1),\ c^{n+1}(x_1, \ldots, x_n, x_i) \quad (1 \leqslant i \leqslant n,\ n = 1, 2, \ldots).$$

Пусть функция $f(x_1, \dots, x_n)$ определена хотя бы на одном наборе и получена из функций $g_0, g_1, \dots, g_m$ с помощью операции суперпозиции (1.11). Тогда, разумеется, каждая из функций $g_0, g_1, \dots, g_m$ также определена хотя бы на одном наборе. Будем предполагать, что графики функций $g_0, g_1, \dots, g_m$ перечисляют соответственно примитивно рекурсивные функции $h_0, h_1, \dots, h_m$. Можно считать, что каждая из функций $h_0, h_1, \dots, h_m$ зависит только от одной переменной. В самом деле, если, например, функция h_j зависит от k переменных ($k > 1$), то $h_j(y_1, \dots, y_k)$ перечисляет такое же множество, как и примитивно рекурсивная функция

$$h_j(l_1(y, k-1), l_1(y, k-2), \dots, l_1(y, 0)),$$

поскольку при $y = c^{k+1}(0, y_1, \dots, y_k)$ набор $(y_1, \dots, y_k)$ совпадает с набором $(l_1(y, k-1), \dots, l_1(y, 0))$.

Чтобы перечислить теперь график функции f, следует с помощью функций $h_0, h_1, \dots, h_m$ перечислять элементы графиков функций $g_0, g_1, \dots, g_m$ и согласовывать их, имея в виду формулу (1.11). Поскольку функции $h_0, h_1, \dots, h_m$ перечисляют графики функций $g_0, g_1, \dots, g_m$ в неизвестном нам порядке, мы должны быть готовы к тому, что такое согласование не всегда возможно (например, некоторые из функций $g_1, \dots, g_m$ будут рассматриваться на различных наборах переменных). Поэтому выберем фиксированный элемент a графика функции f и будем перечислять его во всех случаях нарушения условий согласования.

Итак, определяем примитивно рекурсивную функцию h, перечисляющую график функции f:

$$h(z, z_1, \dots, z_m) = \begin{cases} h_0(z), & \text{если } h_0(z) \text{ имеет вид } c^{m+1}(y_1, \dots, y_m, y), \\ & \text{где } y_1 = r(h_1(z_1)), \dots, y_m = r(h_m(z_m)) \text{ и} \\ & l(h_1(z_1)) = \dots = l(h_m(z_m)), \\ a & \text{в остальных случаях.} \end{cases}$$

Перейдем к операции примитивной рекурсии. Пусть функция f определена хотя бы на одном наборе и получена из функций g, h с помощью операции примитивной рекурсии (1.6), и пусть примитивно рекурсивные функции j_0, j_1 перечисляют графики функций g, h, а число a является элементом графика функции f. Вновь будем предполагать, что функции j_0, j_1 зависят от одной переменной.

Чтобы определить значение $f(x_1, \dots, x_n)$, необходимо найти такую последовательность чисел $a_0, a_1, \dots, a_{x_n}$, что $a_0 = g(x_1, \dots, x_{n-1})$ и для всякого $i < x_n$ выполняется равенство

$$a_{i+1} = h(x_1, \dots, x_{n-1}, i, a_i).$$

Понятно, что последовательность $a_1, \dots, a_{x_n}$ есть последовательность некоторых значений функции h и $a_{x_n} = f(x_1, \dots, x_n)$. Для отыскания

этих значений следует воспользоваться функций j_1 и подобрать такие числа $b_0, \dots, b_{x_n-1}$, что для всякого $i < x_n$

$$j_1(b_i) = c^{n+2}(x_1, \dots, x_{n-1}, i, a_i, a_{i+1}).$$

В свою очередь последовательность $b_0, \dots, b_{x_n-1}$ можно представить одним числом

$$v = c^{x_n+1}(0, b_{x_n-1}, \dots, b_0)$$

(число 0 к последовательности $b_0, \dots, b_{x_n-1}$ добавляем только для того, чтобы в дальнейшем элементы $b_0, \dots, b_{x_n-1}$ получались в виде $l_1(v, 0), \dots, l_1(v, x_n - 1)$).

Итак, определяем примитивно рекурсивную функцию $j(x_1, \dots, x_n, z, v)$, перечисляющую график функции f.

Если $x_n = 0$, то проверяем, что число $j_0(z)$ имеет вид $c^n(x_1, \dots, x_{n-1}, a_0)$, т. е. проверяем выполнение равенства

$$l(j_0(z)) = c^{n-1}(x_1, \dots, x_{n-1}).$$

В случае положительного исхода проверки полагаем

$$j(x_1, \dots, x_{n-1}, 0, z, v) = c^{n+1}(x_1, \dots, x_{n-1}, 0, a_0).$$

В противном случае полагаем $j(x_1, \dots, x_{n-1}, 0, z, v) = a$.

Пусть $x_n > 0$. Вновь проверяем, что величина $j_0(z)$ имеет вид $c^n(x_1, \dots, x_{n-1}, a_0)$. Далее для всякого $i < x_n$ находим число $j_1(l_1(v, i))$, которое обозначим через d_i. Убеждаемся, что каждое число d_i имеет вид

$$c^{n+2}(x_1, \dots, x_{n-1}, i, a_i, a_{i+1}), \tag{2.9}$$

где $a_0 = r(j_0(z))$. В случае положительных исходов всех проверок полагаем

$$j(x_1, \dots, x_n, z, v) = r(d_{x_n-1}).$$

Во всех остальных случаях полагаем $j(x_1, \dots, x_n, z, v) = a$.

В определении функции j встречается проверка выполнения некоторых условий для всех $i < x_n$. Формально, в рамках примитивно рекурсивных функций, это можно выполнить следующим образом. Пусть, например, необходимо проверить, что величина $j_1(l_1(v, i))$ при всех $i < x_n$ имеет вид (2.9). Обозначим через $p(x_1, \dots, x_{n-1}, v, i)$ характеристическую функцию отношения

$$l(l(j_1(l_1(v, i)))) = c^n(x_1, \dots, x_{n-1}, i)$$

(это отношение, конечно, примитивно рекурсивно). Тогда характеристическая функция искомого отношения выражается формулой

$$\prod_{i<x_n} p(x_1, \dots, x_{n-1}, v, i) \cdot \overline{\operatorname{sg}}\,|r(j_1(l_1(v, i))) - r(l(j_1(l_1(v, i+1))))|.$$

Рассмотрим, наконец, операцию минимизации. Пусть функция f определена хотя бы на одном наборе значений переменных и получена из функции g с помощью операции минимизации (2.2). Предположим

далее, что примитивно рекурсивная функция $h(x)$ перечисляет график функции g. Построение примитивно рекурсивной функции j, перечисляющей график функции f, в основных чертах совпадает с соответствующим построением для случая операции примитивной рекурсии.

В самом деле, для того чтобы убедиться в выполнении равенства $f(x_1, \dots, x_n) = y$, необходимо найти значения функции $g(x_1, \dots, x_{n-1}, i)$ при всех $i \leqslant y$ и проверить, что $g(x_1, \dots, x_{n-1}, y) = x_n$ и $g(x_1, \dots, x_{n-1}, i) \neq x_n$ при всех $i < y$. Это достигается тем же приемом, что и в случае операции примитивной рекурсии. Именно, вводится новая переменная v, образуется последовательность

$$a_0 = l_1(v, 0),\ a_1 = l_1(v, 1), \dots, a_y = l_1(v, y),$$

которая порождает последовательность $h(a_0), h(a_1,) \dots, h(a_y)$ элементов графика функции g. Далее, величина $h(a_y)$ должна иметь вид $c^{n+1}(x_1, \dots,\ x_{n-1}, y, x_n)$, а величины $h(a_i)$ при $i < y$ — вид $c^{n+1}(x_1, \dots, x_{n-1}, i, b_i)$, где $b_i \neq x_n$. Детали построения функции j мы оставляем читателю.

Теперь мы в состоянии выполнить обещание, данное в начале параграфа, и доказать, что всякое непустое рекурсивно перечислимое множество перечисляется подходящей примитивно рекурсивной функцией. Действительно, пусть непустое рекурсивно перечислимое множество M перечисляется частично рекурсивной функцией f. По доказанному график функции f можно перечислить некоторой примитивно рекурсивной функцией g. Тогда примитивно рекурсивная функция $r(g)$ будет, очевидно, перечислять множество M.

Установим еще один важный результат, касающийся представления частично рекурсивных функций. Пусть $f(x_1, \dots, x_n)$ — частично рекурсивная функция, отличная от нигде не определенной функции. Частично рекурсивная функция

$$c^{n+1}(x_1, \dots, x_n, f(x_1, \dots, x_n)),$$

очевидно, перечисляет график функции f. Поскольку он непуст, существует примитивно рекурсивная функция $g(x)$, которая перечисляет этот график. Имеем согласно определению графика функции f: $f(x_1,\dots, x_n)=y$ тогда и только тогда, когда найдется такое число z, что

$$g(z) = c^{n+1}(x_1, \dots, x_n, y).$$

Положим

$$h(x_1, \dots, x_n) = (\mu v)(g(l(v)) = c^{n+1}(x_1, \dots, x_n, r(v))). \tag{2.10}$$

Тогда будем иметь

$$f(x_1, \dots, x_n) = r(h(x_1, \dots, x_n)). \tag{2.11}$$

Таким образом, *каждая частично рекурсивная функция, отличная от нигде не определенной функции, может быть получена из исходных функций* $0,\ x + 1,\ I$ *с помощью операций суперпозиции,*

примитивной рекурсии и не более чем однократным применением операции минимизации.

Соотношения (2.10), (2.11) показывают также, что частично рекурсивную функцию $f(x_1, \dots, x_n)$ можно представить в виде

$$f(x_1, \dots, x_n) = r((\mu v)(G(x_1, \dots, x_n, v) = 0)), \tag{2.12}$$

где G — примитивно рекурсивная функция. Это так называемая *нормальная форма Клини*. Отметим еще, что в представлении (2.12) «внешняя» функция r не зависит от функции f.

Упражнения

27. Пусть M_1, M_2 — рекурсивно перечислимые множества. Докажите, что рекурсивно перечислимыми будут объединение $M_1 \cup M_2$ множеств M_1, M_2 (множество, которое состоит из всех элементов, которые входят в множество M_1 или множество M_2) и пересечение $M_1 \cap M_2$ множеств M_1, M_2 (множество, состоящее из элементов, которые принадлежат одновременно обоим множествам M_1 и M_2).

28. Пусть M_1, M_2 — рекурсивно перечислимые множества и отношение «x входит в множество M_2» примитивно рекурсивно. Докажите, что рекурсивно перечислимым будет разность $M_1 \setminus M_2$ множеств M_1 и M_2 (множество всех тех элементов из M_1, которые не попали в множество M_2).

29. Пусть M — рекурсивно перечислимое множество. Докажите, что функция

$$f(x) = \begin{cases} 1, & \text{если } x \text{ входит в } M, \\ \text{не определено} & \text{в противном случае} \end{cases}$$

частично рекурсивна.

30. Докажите, что функция $f(x_1, \dots, x_n)$ представима в виде

$$f(x_1, \dots, x_n) = (\mu y)(G(x_1, \dots, x_n, y) = 0), \tag{2.13}$$

где G — примитивно рекурсивная функция, тогда и только тогда, когда отношение $f(x_1, \dots, x_n) = y$ примитивно рекурсивно.

Глава 3

ФУНКЦИИ, ВЫЧИСЛИМЫЕ НА МАШИНАХ ТЬЮРИНГА

Как уже говорилось в предисловии, понятие алгоритмически вычислимой функции получило точную математическую формулировку в середине 1930-х годов. Почти одновременно нескольким математикам удалось создать математический аппарат для решения этой задачи. Одним из них был английский математик А. Тьюринг (A. Turing, 1912–1954). Он предложил определение некоторого вычислительного устройства, которое позже было названо *машиной Тьюринга*. Справедливости ради следует отметить, что буквально несколькими месяцами позже почти такое же понятие абстрактной вычислительной машины было предложено американским математиком Э. Постом (о машине Поста подробно рассказывается в брошюре В. А. Успенского «Машина Поста». — М.: Наука, 1979). К настоящему времени предложено большое число вариантов машины Тьюринга–Поста. Об одном из них пойдет речь в этой главе.

§ 3.1. Машина Тьюринга

Машина Тьюринга состоит из *ленты*, считывающе-записывающей *головки* и *управляющего устройства* (см. рис. 1).

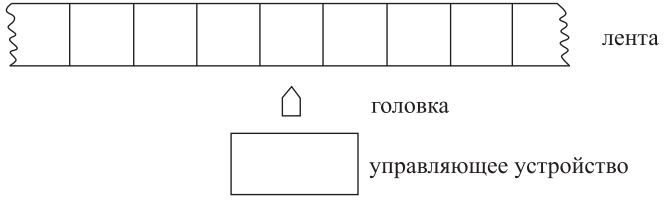

Рис. 1

Лента бесконечна влево и вправо и разбита на клетки. В каждой клетке ленты может быть записан один из символов (букв) конечного ленточного *алфавита* $A = \{a_0, a_1, \ldots, a_k\}$. Обычно в алфавите A выделяется так называемый «пустой» символ a_0.

Головка машины может:

двигаться по ленте влево или вправо, перемещаясь из одной клетки в другую, соседнюю с ней клетку;

читать символ, записанный в клетке, и записывать в нее любой символ алфавита A.

Управляющее устройство организует перемещение головки по ленте и запись символов в клетки ленты. Оно может находиться в одном из конечного числа *состояний* $q_0, q_1, \ldots, q_m$. В множестве состояний выделяются *начальное состояние* q_1 и *заключительное состояние* q_0.

Изменение положения головки на ленте, запись новых символов в клетки ленты и переход в другие состояния происходят согласно *программе* машины, которая состоит из *команд* вида

$$a_i q_j \to a_r M q_s, \tag{3.1}$$

где $j \neq 0$, а M есть один из символов движения: L, R, S. Для любых возможных значений i, j $(0 \leqslant i \leqslant k,\ 1 \leqslant j \leqslant m)$ программа машины содержит ровно одну команду (3.1) с левой частью $a_i q_j$. Обычно программу машины Тьюринга записывают в виде таблицы, строки которой помечены символами $a_0, a_1, \ldots, a_k$, а столбцы — символами $q_1, \ldots, q_m$. В клетке таблицы, отвечающей строке a_i и столбцу q_j, записана правая часть $a_r M q_s$ команды (3.1).

Работа машины Тьюринга протекает в дискретном времени $t = 1, 2, \ldots$. Перед началом работы машины Тьюринга $(t = 0)$ на ленте записывается некоторое слово w в алфавите $\{a_1, \ldots, a_k\}$, а вся остальная часть ленты (слева и справа от слова w) заполняется «пустым» символом a_0. Головка машины Тьюринга устанавливается в клетку, содержащую первую букву слова w, а управляющее устройство приводится в состояние q_1.

Каждый из последующих незаключительных шагов (тактов) работы машины Тьюринга выполняется согласно одному и тому же правилу: если в момент времени t головка считывает из обозреваемой клетки символ a_i, а управляющее устройство находится в состоянии q_j $(j \neq 0)$ и в программе машины имеется команда (3.1), то в момент времени $t + 1$:

в обозреваемую клетку ленты будет записан символ a_r (возможно, что $r = i$);

головка сдвинется на одну клетку влево $(M = L)$, вправо $(M = R)$ или останется в прежней клетке $(M = S)$;

управляющее устройство перейдет в состояние q_s (и здесь возможно равенство $s = j$).

Машина Тьюринга заканчивает работу, если управляющее устройство попадает в заключительное состояние q_0.

Сразу отметим, что, вообще говоря, не для всех слов w машина Тьюринга завершает свою работу. Для некоторых слов w она может работать неограниченно долго, не попадая в заключительное состояние

q_0. В случае окончания работы машины Тьюринга результатом применения машины к слову w, как правило, считают слово в алфавите $\{a_1, \dots, a_k\}$, которое оказывается записанным на ленте в заключительный момент времени. Если же машина Тьюринга не заканчивает работу, то результат применения машины к слову w не определен.

Рассмотрим примеры машин Тьюринга. Машина Тьюринга $\mathcal{M}_1$, начиная работу на первом символе a_1 слова $w = a_1 \dots a_1$, оставляет этот символ без изменения, сдвигает головку на одну клетку вправо и останавливается.

Машина Тьюринга $\mathcal{M}_2$ также оставляет первый символ a_1 слова $w = a_1 \dots a_1$ без изменения, однако сдвигает головку на одну клетку влево, записывает в ней еще один символ a_1 и останавливается. Машина Тьюринга $\mathcal{M}_3$ на любом слове работает неограниченно долго.

	q_1
a_0	$a_0 R q_1$
a_1	$a_1 R q_0$

$\mathcal{M}_1$

	q_1
a_0	$a_1 S q_0$
a_1	$a_1 L q_1$

$\mathcal{M}_2$

	q_1
a_0	$a_0 S q_1$
a_1	$a_1 S q_1$

$\mathcal{M}_3$

В дальнейшем нас в первую очередь будут интересовать функции, вычислимые на машинах Тьюринга. Чтобы определить эти функции, необходимо прежде всего условиться о способе представления чисел из N на ленте машины Тьюринга. Мы выберем способ, который на практике встречается редко, однако удобен в теоретических построениях. Именно, число m из N будем представлять словом, состоящим из $(m + 1)$ символов 1 («лишнюю» единицу добавляем, чтобы иметь возможность представлять число 0). В связи с этим ленточный алфавит A машины Тьюринга, предназначенной для вычисления некоторой функции, всегда будет содержать символ 1 (например, $a_1 = 1$), пустой символ 0 ($a_0 = 0$) и, возможно, другие символы.

Итак, число m из N записываем на ленте машины Тьюринга в виде массива из $(m + 1)$ единиц, а в остальных клетках ленты помещаем пустой символ 0. Если $n > 1$ и требуется представить на ленте машины Тьюринга набор чисел $(m_1, \dots, m_n)$, то образуем n массивов из $m_1 + 1, \dots, m_n + 1$ единиц соответственно и помещаем между соседними массивами разделительный символ 0. Такое представление набора чисел $(m_1, \dots, m_n)$ на ленте машины Тьюринга (включая случай $n = 1$) будем называть *основным кодом набора* $(m_1, \dots, m_n)$.

Перейдем собственно к определению функции, вычислимой на машине Тьюринга. Пусть $f(x_1, \dots, x_n)$ — функция (вообще говоря, частичная) и $\mathcal{M}$ — машина Тьюринга с ленточным алфавитом A, содержащим символы 0 и 1. Будем говорить, что *машина* $\mathcal{M}$ *вычисляет функцию* f, если для любого набора $(m_1, \dots, m_n)$ выполняются следующие условия.

1. Если значение $f(m_1, \dots, m_n)$ определено, то машина $\mathcal{M}$, начиная работу в состоянии q_1 на первой единице основного кода набора

$(m_1, \ldots, m_n)$, через конечное число тактов останавливается и в заключительный момент времени на ленте машины в основном коде представлено число $f(m_1, \ldots, m_n)$.

2. Если значение $f(m_1, \ldots, m_n)$ не определено, то, начиная работу из той же позиции, что и в п. 1, машина $\mathcal{M}$ либо никогда не останавливается, либо останавливается, но при этом на ленте машины не представлено в основном коде никакое число из N.

В дальнейшем по чисто техническим причинам нам будет удобно иметь определение вычислимой функции в более стандартизованном виде. Именно, мы хотим, чтобы в п. 1 определения машина $\mathcal{M}$ в заключительный момент времени останавливалась на первой единице основного кода числа $f(m_1, \ldots, m_n)$. А в п. 2 определения мы хотим отказаться от второй возможности, когда на ленте машины $\mathcal{M}$ отсутствует основной код числа из N. Такое модифицированное понятие вычисления функции f на машине $\mathcal{M}$ будем называть *правильным вычислением функции f на машине* $\mathcal{M}$. Сравнительно несложно показать (попытайтесь это сделать!), что второе определение вычислимости не сужает класса функций, вычислимых на машинах Тьюринга.

Обратимся к примерам вычисления функций на машинах Тьюринга:

	q_1
0	$1Sq_0$
1	$1Lq_1$

$\mathcal{M}_4$

	q_1
0	$1Sq_0$
1	$0Rq_1$

$\mathcal{M}_5$

	q_1	q_2
0	—	$1Sq_0$
1	$0Rq_2$	$1Sq_0$

$\mathcal{M}_6$

Нетрудно понять, что машина $\mathcal{M}_4$ правильно вычисляет функцию $x + 1$. Также нетрудно убедиться, что машина $\mathcal{M}_5$ правильно вычисляет функцию-константу 0 (рассматриваемую от одной переменной). Машина $\mathcal{M}_5$ пробегает в состоянии q_1 слева направо весь массив из единиц и заменяет их нулями. После этого вместо первого справа символа 0 она ставит 1 и останавливается.

Рассмотрим чуть более сложный пример функции $x \div 1$. Машина Тьюринга $\mathcal{M}_6$ заменяет первую единицу основного кода нулем, переходит в состояние q_2 и сдвигает головку вправо. Находясь в состоянии q_2, она записывает во вторую клетку 1 и останавливается. Прочерк в одной из клеток таблицы для машины $\mathcal{M}_6$ указывает на то, что эту клетку можно заполнить произвольным образом.

Из простых вычислимых функций остановимся еще на функциях I_i^n. Все они вычисляются сходным образом: головка машины движется слева направо до i-го массива из единиц, стирая по пути все единицы (т. е. заменяя их нулями), оставляет без изменения i-й массив, затем, продвигаясь до конца основного кода, стирает единицы оставшихся массивов. В заключение головка машины возвращается на первую единицу бывшего i-го массива. Поскольку число состояний машины Тьюринга, вычисляющей функцию I_i^n, линейным образом зависит от параметров i, n, мы построим машину Тьюринга лишь для вычисления

функции $I_2^3(x_1, x_2, x_3)$. Программа этой машины представлена в нижеследующей таблице:

	q_1	q_2	q_3	q_4	q_5
0	$0Rq_2$	$0Rq_3$	$0Lq_4$	$0Lq_4$	$0Rq_0$
1	$0Rq_1$	$1Rq_2$	$0Rq_3$	$1Lq_5$	$1Lq_5$

Упражнения

31. Докажите, что для всякой машины Тьюринга существует эквивалентная ей (т.е. дающая тот же самый результат) машина Тьюринга, программа которой не содержит команд с символом S.

32. Постройте машину Тьюринга, правильно вычисляющую функцию $x + y$.

33. Может ли одна и та же машина Тьюринга правильно вычислять две различные функции?

34. Пусть программа машины Тьюринга $\mathcal{M}'$ получена из программы машины $\mathcal{M}$ заменой всех символов L символами R и всех символов R символами L. Можно ли в каком-либо смысле утверждать, что машина $\mathcal{M}'$ вычисляет те же функции, что и машина $\mathcal{M}$?

§ 3.2. Композиция и итерация машин Тьюринга

Обычно при построении достаточно сложных машин Тьюринга прибегают к известному на практике приему: разбивают алгоритм, реализуемый машиной Тьюринга, на отдельные простые части, строят для них машины Тьюринга и затем «собирают» из полученных «малых» машин Тьюринга требуемую машину Тьюринга. Если разбиение алгоритма на части и построение для них машин Тьюринга есть процесс в значительной степени творческий, то последующая «сборка» машин Тьюринга является делом весьма рутинным. Здесь существуют два основных приема конструирования из одних машин Тьюринга других машин Тьюринга. Это *композиция* и *итерация* машин Тьюринга.

Композиция машин Тьюринга формализует идею последовательного выполнения двух (реже нескольких) вычислительных процессов. В самом простейшем случае композиция двух машин Тьюринга определяется так. Пусть $\mathcal{M}_1, \mathcal{M}_2$ — машины Тьюринга с одним и тем же ленточным алфавитом. Будем предполагать, что множества состояний машин $\mathcal{M}_1, \mathcal{M}_2$ не имеют общих элементов. Мы хотим создать машину Тьюринга $\mathcal{M}$, которая работает следующим образом. На исходном слове w начинает работать машина $\mathcal{M}_1$. Если $\mathcal{M}_1$ заканчивает свою работу, то с «этого самого места» (т. е. с достигнутыми заполнением ленты и положением головки на ленте) запускается машина $\mathcal{M}_2$. Результат, полученный при этом машиной $\mathcal{M}_2$, и будет являться результатом применения машины $\mathcal{M}$ к слову w. Разумеется, если одна из машин

$\mathcal{M}_1, \mathcal{M}_2$ работает неограниченно долго, то и машина $\mathcal{M}$ будет работать неограниченно долго.

Довольно понятно, как из программ машин $\mathcal{M}_1, \mathcal{M}_2$ получить программу машины $\mathcal{M}$. Для этого необходимо во всех командах машины $\mathcal{M}_1$ заменить заключительное состояние начальным состоянием машины $\mathcal{M}_2$, а затем объединить полученные множества команд машин $\mathcal{M}_1$ и $\mathcal{M}_2$. Начальным состоянием машины $\mathcal{M}$ будет являться начальное состояние машины $\mathcal{M}_1$, а заключительным — заключительное состояние машины $\mathcal{M}_2$.

Вернемся к машине Тьюринга из предыдущего параграфа, которая правильно вычисляет функцию $I_2^3(x_1, x_2, x_3)$. Ее можно получить в виде последовательной композиции машин Тьюринга $\mathcal{M}_1$–$\mathcal{M}_5$ (первый индекс у состояния q указывает на номер машины):

	q_{11}
0	$0Rq_{10}$
1	$0Rq_{11}$

$\mathcal{M}_1$

	q_{21}
0	$0Rq_{20}$
1	$1Rq_{21}$

$\mathcal{M}_2$

	q_{31}
0	$0Lq_{30}$
1	$0Rq_{31}$

$\mathcal{M}_3$

	q_{41}
0	$0Lq_{41}$
1	$1Lq_{40}$

$\mathcal{M}_4$

	q_{51}
0	$0Rq_{50}$
1	$1Lq_{51}$

$\mathcal{M}_5$

Машина $\mathcal{M}_1$ стирает в основном коде набора (x_1, x_2, x_3) первый массив из единиц, пропускает пустую клетку между первым и вторым массивами и останавливается на первой единице второго массива. Машина $\mathcal{M}_2$, ничего не изменяя, пробегает второй массив из единиц, пропускает следующую за ним пустую клетку и останавливается на первой единице третьего массива. Действия машины $\mathcal{M}_3$ аналогичны действиям машины $\mathcal{M}_1$, за исключением последнего шага, на котором головка машины $\mathcal{M}_3$ смещается на одну клетку влево. Машина $\mathcal{M}_4$ движется влево по пустым клеткам до крайней правой единицы бывшего второго массива и останавливается в соседней с ней клетке. Наконец, машина $\mathcal{M}_5$ пробегает справа налево оставшийся массив из единиц, выходит на примыкающую к нему пустую клетку, возвращается на одну клетку назад и останавливается.

Тот тип композиции, который мы описали выше, можно было бы назвать безусловной композицией машин Тьюринга. Однако существуют задачи, где необходимо воспользоваться более сложными формами композиции. Предположим, например, что мы хотим получить из данных машин Тьюринга $\mathcal{M}_1$, $\mathcal{M}_2$ такую машину Тьюринга $\mathcal{M}$, которая в некоторых случаях работает как машина $\mathcal{M}_1$, а в других — как композиция машин $\mathcal{M}_1$ и $\mathcal{M}_2$. Для этого, исходя из содержательных соображений, мы выделим среди заключительных команд (т. е. команд, содержащих заключительное состояние) машины $\mathcal{M}_1$ такие команды, которые отвечают второй из рассматриваемых возможностей. А далее, как и выше, заменим в них заключительное состояние начальным состоянием машины $\mathcal{M}_2$. Все остальные команды обеих машин оставим без изменения.

Еще один тип композиции машин Тьюринга соответствует случаю, когда необходимо осуществить выбор из двух или более альтернатив. Пусть $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$ — машины Тьюринга с попарно не пересекающимися множествами состояний, и пусть содержательно машина $\mathcal{M}_1$ распознает некоторое свойство p. При этом в случае выполнения свойства p мы бы хотели запустить машину $\mathcal{M}_2$, а в случае невыполнения — машину $\mathcal{M}_3$. Формально свойство p характеризуется заключительными командами машины $\mathcal{M}_1$: часть заключительных команд соответствует выполнению свойства p, остальные — невыполнению свойства p. Обозначим заключительное состояние из первой части заключительных команд машины $\mathcal{M}_1$ через q_0', из второй части — через q_0''. Тогда указанное разветвление машин Тьюринга $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$ выполняется так: все заключительные состояния q_0' машины $\mathcal{M}_1$ заменяются начальным состоянием машины $\mathcal{M}_2$, а все заключительные состояния q_0'' — начальным состоянием машины $\mathcal{M}_3$. Заключительными состояними полученной машины Тьюринга являются заключительное состояние машины $\mathcal{M}_2$ и заключительное состояние машины $\mathcal{M}_3$.

Рассмотрим пример композиции машин Тьюринга последнего типа. Используя этот тип композиции, построим машину Тьюринга, которая правильно вычисляет функцию

$$\mathrm{rm}\,(x,2) = \begin{cases} 0, & \text{если } x \text{ четно,} \\ 1, & \text{если } x \text{ нечетно.} \end{cases}$$

Определим машины $\mathcal{M}_6$–$\mathcal{M}_8$:

	q_{11}	q_{12}
0	$0Sq_{10}''$	$0Sq_{10}'$
1	$0Rq_{12}$	$0Rq_{11}$

$\mathcal{M}_6$

	q_{21}
0	$1Sq_{20}$
1	—

$\mathcal{M}_7$

	q_{31}	q_{32}
0	$1Rq_{32}$	$1Lq_{30}$
1	—	—

$\mathcal{M}_8$

Машина $\mathcal{M}_6$, находясь в начальном состоянии q_{11} на первой единице массива из $(x+1)$ единиц, движется по массиву вправо, стирает все единицы и поочередно проходит через состояния q_{11}, q_{12}. В случае четного числа x (тогда пробегаемый массив состоит из нечетного числа единиц) машина $\mathcal{M}_6$ выходит на первую пустую клетку в состоянии q_{12} и останавливается в заключительном состоянии q_{10}'. В случае нечетного числа x аналогичный процесс заканчивается в заключительном состоянии q_{10}''. Таким образом, с помощью состояний q_{10}', q_{10}'' машина $\mathcal{M}_6$ «распознаёт» четность числа x.

Машина $\mathcal{M}_7$ записывает в пустую клетку 1 (основной код числа 0) и останавливается. Машина $\mathcal{M}_8$ записывает в двух пустых клетках единицы, возвращается на первую единицу и останавливается. Прочерки в таблицах для машин $\mathcal{M}_7$, $\mathcal{M}_8$ означают, что соответствующие клетки могут быть заполнены произвольным образом.

Образуем теперь композицию $\mathcal{M}$ машин $\mathcal{M}_6, \mathcal{M}_7, \mathcal{M}_8$ так, чтобы машина $\mathcal{M}_7$ начинала работу при попадании машины $\mathcal{M}_6$ в заключительное состояние q'_{10}, а машина $\mathcal{M}_8$ — в заключительное состояние q''_{10}. Соответствующая таблица для машины $\mathcal{M}$ будет выглядеть следующим образом:

	q_{11}	q_{12}	q_{21}	q_{31}	q_{32}
0	$0Sq_{31}$	$0Sq_{21}$	$1Sq_{20}$	$1Rq_{32}$	$1Lq_{30}$
1	$0Rq_{12}$	$0Rq_{11}$	—	—	—

Итерация машины Тьюринга формализует идею многократного регулируемого применения одного и того же алгоритма (нелишне здесь вспомнить операцию примитивной рекурсии и ее частный случай — операцию итерации).

Пусть задана машина Тьюринга $\mathcal{M}$ и мы хотим многократно применять эту машину (в духе композиции машин Тьюринга) до тех пор, пока не будет выполнено некоторое условие p. Само условие p, как и ранее, можно задать с помощью некоторых заключительных команд машины $\mathcal{M}$. Чтобы выделить эти заключительные команды, обозначим содержащееся в них заключительное состояние через q'_0. Все остальные вхождения заключительного состояния в программу машины $\mathcal{M}$ обозначим через q''_0. Теперь наше намерение итерировать работу машины Тьюринга $\mathcal{M}$ можно выразить более точно. Запускаем машину $\mathcal{M}$ в начальном состоянии q_1. Если через некоторое количество тактов работы она попадает в заключительное состояние q''_0, то «возвращаем» ее вновь в начальное состояние q_1, и т. д. Формально итерация машины $\mathcal{M}$ относительно условия p получается заменой всех вхождений заключительного состояния q''_0 начальным состоянием q_1.

Пусть, например, машина $\mathcal{M}$ задается табл. 1.

Таблица 1

	q_1	q_2	q_3	q_4
0	—	$0Rq_2$	$0Sq'_0$	$0Lq_4$
1	$0Rq_2$	$0Rq_3$	$0Lq_4$	$1Sq''_0$

Проанализируем работу машины $\mathcal{M}$. Предположим, что в начальный момент времени на ленте машины $\mathcal{M}$ записан основной код набора (x_1, x_2), а головка машины установлена на крайнюю правую единицу первого массива. Начиная работу в этой позиции, машина $\mathcal{M}$ совершает следующие действия.

Она заменяет обозреваемую единицу нулем, сдвигается вправо до первой единицы второго массива и также заменяет ее нулем. Если $x_2 = 0$, то машина $\mathcal{M}$ останавливается в следущей справа клетке в состоянии q'_0. В противном случае машина $\mathcal{M}$ заменяет вторую единицу

второго массива нулем и возвращается (если $x_1 \neq 0$) на крайнюю правую единицу первого массива, где и останавливается в состоянии q_0''.

Таблица 2

	q_1	q_2	q_3	q_4
0	—	$0Rq_2$	$0Sq_0'$	$0Lq_4$
1	$0Rq_2$	$0Rq_3$	$0Lq_4$	$1Sq_1$

Образуем итерацию машины $\mathcal{M}$ — заменим в ее программе заключительное состояние q_0'' начальным состоянием q_1 (см. табл. 2). Получим машину Тьюринга $\mathcal{M}'$. Нетрудно понять, как будет работать машина $\mathcal{M}'$. Она достигает заключительного состояния q_0' в том и только том случае, когда $x_2 = 2y$ и $x_1 \geqslant y$. В остальных случаях машина $\mathcal{M}'$ работает неограниченно долго.

Упражнения

35. Представьте в виде композиции более простых машин Тьюринга машины, правильно вычисляющие функции $\operatorname{sg} x$, $\overline{\operatorname{sg}}\, x$, $3 \div x$.

36. Применив итерацию машины Тьюринга, постройте машину, правильно вычисляющую функцию $[x/2]$.

§ 3.3. Моделирование машин Тьюринга

В теории алгоритмов довольно распространенным является прием, когда с помощью одного вычислительного устройства моделируется работа другого вычислительного устройства. Здесь мы рассмотрим лишь моделирование машин Тьюринга, работающих в алфавите $\{0, 1, a_2, \ldots, a_k\}$, где $k \geqslant 2$, машинами Тьюринга, имеющими ленточный алфавит $\{0, 1\}$. Более того, нас будут интересовать только функции, правильно вычислимые теми и другими машинами Тьюринга. Мы установим, что эти классы функций совпадают.

Идея моделирования машины Тьюринга, работающей в алфавите $A = \{0, 1, a_2, \ldots, a_k\}$, чрезвычайно проста. Необходимо закодировать буквы алфавита A словами в алфавите $\{0, 1\}$ и затем оперировать с этими кодами так же, как исходная машина Тьюринга оперирует с буквами $0, 1, a_2, \ldots, a_k$.

Существует большое количество разнообразных кодирований. Мы выберем блочное кодирование, когда каждой букве алфавита A сопоставляется двоичное слово (блок) одной и той же длины l. Понятно, что для однозначности последующего декодирования число l должно удовлетворять неравенству $2^l \geqslant k + 1$ (количество двоичных слов длины l должно быть не меньше числа букв алфавита A). Теперь каждой букве a алфавита A можно сопоставить двоичное слово $\nu(a)$ длины l, причем разным буквам a следует сопоставить различные слова $\nu(a)$.

В дальнейшем нам будет удобно считать, что кодирование ν ставит в соответствие букве 0 слово из l нулей, а букве 1 — слово из l единиц.

Пусть $\mathcal{M}$ — произвольная машина Тьюринга с ленточным алфавитом A и множеством состояний $\{q_0, q_1, \ldots, q_m\}$. Мы хотим сначала выполнить центральную часть моделирования машины $\mathcal{M}$ — построить машину Тьюринга $\mathcal{M}_1$ с ленточным алфавитом $\{0, 1\}$, которая работает над кодами слов так же, как машина $\mathcal{M}$ работает над исходными словами в алфавите A. С этой целью к состояниям $q_0, q_1, \ldots, q_m$ машины $\mathcal{M}$ добавим еще три группы состояний.

Состояния первой группы будем для наглядности записывать в виде $[b_1 \ldots b_p, j]$, где $1 \leqslant p \leqslant l$, $1 \leqslant j \leqslant m$ и $b_1, \ldots, b_p$ — символы 0 или 1. В состояниях первой группы машина $\mathcal{M}_1$ проводит «расшифровку» кодов $\nu(a)$, «помня» при этом текущее состояние машины $\mathcal{M}$.

Предположим, что в некоторый момент времени машина $\mathcal{M}_1$, моделируя работу машины $\mathcal{M}$, находится в состоянии q_j (так же, как и машина $\mathcal{M}$), а ее головка обозревает первую букву кода некоторой буквы алфавита A. Тогда следующие команды в программе машины $\mathcal{M}_1$ обеспечивают чтение этого кода:

$$bq_j \to bR[b, j], \quad b[b_1 \ldots b_p, j] \to bR[b_1 \ldots b_p b, j] \quad (1 \leqslant p \leqslant l-2),$$
$$b[b_1 \ldots b_{l-1}, j] \to bS[b_1 \ldots b_{l-1} b, j]$$

(здесь b, b_i — символы 0 или 1).

Пусть теперь в программе машины $\mathcal{M}$ имеется команда (3.1), $\nu(a_i) = b_1 \ldots b_l$, $\nu(a_r) = c_1 \ldots c_l$ и машина $\mathcal{M}_1$ в некоторый момент времени находится на последней букве b_l кода $\nu(a_i)$ в состоянии $[b_1 \ldots b_l, j]$. Состояния второй группы, которые мы обозначим как $[b_1 \ldots b_p, i, j]$, предназначены для моделирования «половины» выполнения машиной $\mathcal{M}$ команды (3.1), которое состоит в замене кода $b_1 \ldots b_l$ кодом $c_1 \ldots c_l$. С этими состояниями связаны следующие команды программы машины $\mathcal{M}_1$:

$$b_l[b_1 \ldots b_l, j] \to c_l L[b_1 \ldots b_{l-1}, i, j],$$
$$b_p[b_1 \ldots b_p, i, j] \to c_p L[b_1 \ldots b_{p-1}, i, j] \quad (2 \leqslant p \leqslant l-1),$$
$$b_1[b_1, i, j] \to c_1 S\{M, s\}$$

(здесь символы M, s взяты из правой части команды (3.1), а $\{M, s\}$ есть обозначение нового состояния, которое мы отнесем к третьей группе).

Теперь, находясь в состоянии $\{M, s\}$, машина $\mathcal{M}_1$ должна промоделировать выполнение оставшейся части команды (3.1), т. е. $\mathcal{M}_1$ должна сдвинуться на l клеток влево (если $M = L$) или вправо (если $M = R$) и перейти в состояние q_s. В случае $M = S$ машина $\mathcal{M}_1$, не сдвигая головку, сразу переходит в состояние q_s. Для выполнения этих действий служат состояния третьей группы и соответствующие

команды:

$$b\{L,s\} \to bL\{L,s,1\},\ b\{L,s,p\} \to bL\{L,s,p+1\} \quad (1 \leqslant p \leqslant l-1),$$
$$b\{L,s,l\} \to bSq_s;$$
$$b\{R,s\} \to bR\{R,s,1\},\ b\{R,s,p\} \to bR\{R,s,p+1\} \quad (1 \leqslant p \leqslant l-1),$$
$$b\{R,s,l\} \to bRq_s;$$
$$b\{S,s\} \to bSq_s.$$

Чтобы полностью промоделировать работу машины $\mathcal{M}$, необходимо определить еще две машины Тьюринга $\mathcal{M}_0$ и $\mathcal{M}_2$. Машина $\mathcal{M}_0$ «растягивает» основной код исходного набора в l раз, а машина $\mathcal{M}_2$, наоборот, «сокращает» основной код числа в l раз. Поскольку действия машин $\mathcal{M}_0$, $\mathcal{M}_2$ в известном смысле обратны друг другу, мы рассмотрим лишь машину $\mathcal{M}_0$.

Итак, машина $\mathcal{M}_0$ должна «растянуть» основной код любого набора в l раз. Это означает, что каждая единица основного кода должна быть заменена l единицами, а каждый нуль, стоящий между масивами из единиц, — l нулями. Для простоты ограничимся случаем, когда машина $\mathcal{M}$ вычисляет функцию от одной переменной.

Машину Тьюринга $\mathcal{M}_0$ можно представить в виде композиции и итерации следующих машин $\mathcal{M}_3$–$\mathcal{M}_6$. Машина $\mathcal{M}_3$, находясь на первой единице основного кода числа x, сдвигается вправо на одну клетку и анализирует содержащийся в ней символ a. Если $a = 0$, то запускается машина $\mathcal{M}_4$, которая дописывает к имеющейся на ленте единице еще $(l-1)$ единиц и останавливается. Если $a = 1$, то запускается машина $\mathcal{M}_5$, которая стирает первую единицу основного массива, сдвигается влево на одну клетку и записывает массив из l единиц. Далее действует машина $\mathcal{M}_6$, которая будет подвергнута итерации.

Машина $\mathcal{M}_6$ пробегает слева направо массив из единиц, затем массив из нулей, встречает первую единицу следующего массива и анализирует символ a, стоящий непосредственно справа от этой единицы. Если $a = 1$, то стирается первая единица второго массива, машина $\mathcal{M}_6$ движется далее налево, пробегает массив из нулей, затем массив из единиц и приписывает к последнему массиву слева новые l единиц. После этого $\mathcal{M}_6$ переходит в начальное состояние (цикл итерации). Если же $a = 0$, то машина $\mathcal{M}_6$ действует аналогично до момента перехода в начальное состояние, после чего она $\mathcal{M}_6$ должна завершить работу в заключительном состоянии.

Построение машин $\mathcal{M}_3$ – $\mathcal{M}_6$ не представляет большой трудности, и мы его опускаем.

Упражнения

37. Постройте программу машины Тьюринга $\mathcal{M}_6$.

§ 3.4. Вычисление частично рекурсивных функций на машинах Тьюринга

В этом параграфе мы докажем важное утверждение.

Любая частично рекурсивная функция вычислима на подходящей машине Тьюринга.

Доказательство проведем индукцией по построению частично рекурсивных функций.

Вычислимость исходных частично рекурсивных функций 0, $x+1$, I установлена в § 3.1. Остается показать, что операции суперпозиции, примитивной рекурсии и минимизации сохраняют вычислимость функций на машинах Тьюринга.

Итак, пусть функции $g_0, g_1, \ldots, g_m$ правильно вычислимы на машинах Тьюринга $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_m$, а функция $f(x_1, \ldots, x_n)$ получается из функций $g_0, g_1, \ldots, g_m$ с помощью операции суперпозиции (1.11). Будем предполагать, что все машины $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_m$ работают в алфавите $\{0, 1\}$. С принципиальной точки зрения вычислимость функции f не вызывает сомнений: достаточно последовательно с помощью машин $\mathcal{M}_1, \ldots, \mathcal{M}_m$ вычислить значения функций $g_1, \ldots, g_m$ (если, конечно, все вычисления заканчиваются) и затем к полученному набору значений применить машину Тьюринга $\mathcal{M}_0$. Однако при реализации этого плана возникают технические трудности, которые мы попытаемся обрисовать и преодолеть.

Первая трудность возникает сразу же, как только мы приступаем к вычислению значения функции g_1. Действительно, при вычислении значения $g_1(x_1, \ldots, x_n)$ основной код набора $(x_1, \ldots, x_n)$ будет, вообще говоря, утерян. Как после этого вычислять значения функций $g_2, \ldots, g_m$?

Для преодоления этой трудности нам придется воспользоваться дополнительными символами на ленте. Начнем с того, что несколько изменим процесс вычисления функций $g_1, \ldots, g_m$ на машинах $\mathcal{M}_1, \ldots, \mathcal{M}_m$. Именно, добавим к символам 0, 1 ленточного алфавита машин $\mathcal{M}_1, \ldots, \mathcal{M}_m$ новый символ 2. Он будет играть роль дополнительного «пустого» символа, отмечая в процессе вычисления те клетки ленты, в которых хотя бы раз побывала головка машины Тьюринга и которые изначально содержали символ 0.

В программе каждой из машин Тьюринга $\mathcal{M}_1, \ldots, \mathcal{M}_m$ заменим любую команду вида

$$aq_i \to 0Mq_j$$

командой

$$aq_i \to 2Mq_j.$$

После этого к каждой команде вида

$$0q_i \to bMq_j$$

добавим «дублирующую» команду

$$2q_i \to bMq_j.$$

Полученные в результате этих преобразований машины Тьюринга обозначим через $\mathcal{M}'_1, \ldots, \mathcal{M}'_m$.

Понятно, что машины $\mathcal{M}'_1, \ldots, \mathcal{M}'_m$ будут воспринимать символы 0 и 2 одинаково, как раньше воспринимали символ 0 машины $\mathcal{M}_1, \ldots, \mathcal{M}_m$. Главное отличие вычислений на машинах $\mathcal{M}'_1, \ldots, \mathcal{M}'_m$ от аналогичных вычислений на машинах $\mathcal{M}_1, \ldots, \mathcal{M}_m$ состоит в том, что после окончания вычисления (если оно действительно заканчивается) символом 2 будут отмечены все клетки ленты, где хотя бы раз побывала головка машины Тьюринга и которые в момент окончания вычисления являются пустыми (т. е. не содержат символ 1). Это свойство вычислений на машинах $\mathcal{M}'_1, \ldots, \mathcal{M}'_m$ поможет нам в дальнейшем создать основной код набора значений функций $g_1, \ldots, g_m$ для последующего применения машины $\mathcal{M}_0$.

Преобразование машин $\mathcal{M}_1, \ldots, \mathcal{M}_m$ в машины $\mathcal{M}'_1, \ldots, \mathcal{M}'_m$ пока не решает основную задачу: как на одной ленте организовать вычисление нескольких функций, сохраняя при этом результаты предыдущих вычислений. Для решения этой задачи служат так называемые *дорожки* на ленте машины Тьюринга.

Представим себе, что каждая клетка ленты машины Тьюринга разделена на m «этажей» так, что на каждом «этаже» можно записать один из символов 0, 1, 2. Части клеток ленты, принадлежащие одному и тому же «этажу», образуют *дорожку*. Введение дорожек, разумеется, представляет собой всего лишь наглядный технический прием, с помощью которого удобно описывать моделирование нескольких машин Тьюринга на одной ленте. Формально при введении дорожек мы переходим от алфавита $\{0, 1, 2\}$, в котором работают машины $\mathcal{M}'_1, \ldots, \mathcal{M}'_m$, к алфавиту B, состоящему из 3^m букв. Эти буквы можно представлять себе в виде наборов $(b_1, \ldots, b_m)$ длины m, где каждая компонента b_i есть буква алфавита $\{0, 1, 2\}$. Компоненты b_i букв $(b_1, \ldots, b_i, \ldots, b_m)$ как раз записываются на i-й дорожке ленты.

Имея теперь m дорожек на ленте (или, что эквивалентно, алфавит B из 3^m букв), нетрудно организовать процесс последовательного вычисления значений функций $g_1, \ldots, g_m$. Сначала основной код набора $(x_1, \ldots, x_n)$ переводим на все m дорожек. При этом можно считать, что символам 0 и 1 исходного алфавита отвечают буквы $(0, \ldots, 0)$ и $(1, \ldots, 1)$ алфавита B. Затем на первой дорожке запускаем машину $\mathcal{M}'_1$. Она воспринимает только символы, записанные на первой дорожке, полностью игнорируя все, что записано на остальных $(m-1)$ дорожках. Если машина $\mathcal{M}'_1$ заканчивает вычисление, запускаем машину $\mathcal{M}'_2$ на второй дорожке. Вот здесь нам понадобятся символы 2, которые расставила машина $\mathcal{M}'_1$ на первой дорожке. С их помощью (а также с помощью символов 1) мы отыскиваем начало основного кода на второй дорожке — достаточно посетить лишь те клетки, у которых первая

дорожка содержит символы 1 или 2. Если машина $\mathcal{M}'_2$ заканчивает работу, то запускаем машину $\mathcal{M}'_3$ на третьей дорожке, и т. д.

Если значения всех функций $g_1, \dots, g_m$ определены, то в результате описанного процесса моделирования машин $\mathcal{M}'_1, \dots, \mathcal{M}'_m$ на m дорожках ленты будут записаны в основном коде значения

$$g_1(x_1, \dots, x_n), \dots, g_m(x_1, \dots, x_n). \tag{3.2}$$

Теперь необходимо «выровнять» эти записи, т. е. расположить их на соответствующих дорожках так, чтобы последовательность значений функций $g_1, \dots, g_m$ выглядела как основной код набора (3.2). Здесь нам вновь понадобятся символы 2, которые могут находиться на любой из дорожек. Они упростят процесс «выравнивания» записей на дорожках, позволяя определить потенциальные границы этих записей.

Остается совсем немного для завершения вычисления значений функций $g_1, \dots, g_m$. Следует только выполнить обратный переход из алфавита B в алфавит $\{0, 1\}$, т.е. сформировать на ленте основной код набора (3.2). Завершающий этап вычисления значения функции f состоит в том, что к полученному основному коду набора (3.2) применяется машина Тьюринга $\mathcal{M}_0$.

Подробно описанный алгоритм вычисления функции f на машине Тьюринга позволяет построить эту машину Тьюринга в виде композиции более простых машин Тьюринга, включающих, в частности, машины $\mathcal{M}_0, \mathcal{M}'_1, \dots, \mathcal{M}'_m$.

Перейдем к операции примитивной рекурсии. Пусть функции g и h правильно вычислимы на машинах Тьюринга $\mathcal{M}_0, \mathcal{M}_1$, а функция $f(x_1, \dots, x_n)$ получается из функций g, h с помощью операции примитивной рекурсии (1.6). Как и в случае операции суперпозиции, введем в ленточный алфавит машин $\mathcal{M}_0, \mathcal{M}_1$ символ 2, чтобы отмечать пустые клетки, в которых в процессе вычисления побывали головки машин $\mathcal{M}_0, \mathcal{M}_1$. Соответствующим образом модифицированные машины Тьюринга обозначим через $\mathcal{M}'_0$ и $\mathcal{M}'_1$.

Для вычисления функции f нам понадобятся три дорожки на ленте. На первой дорожке будет постоянно находиться исходный набор значений переменных $x_1, \dots, x_n$, на второй дорожке — «текущее» значение последней переменной (по которой ведется рекурсия), а на третьей дорожке будут вычисляться значения функций g и h. В связи с этим распределением дорожек первое действие машины Тьюринга, вычисляющей функцию f, будет состоять в переписывании основного кода набора $(x_1, \dots, x_n)$ на первую дорожку и создании основного кода числа 0 на второй дорожке. Третью дорожку пока оставляем пустой.

Второй этап вычислений заключается в переносе с первой дорожки на третью чисел $x_1, \dots, x_{n-1}$, формировании на третьей дорожке основного кода набора $(x_1, \dots, x_{n-1})$ и применении к этому коду машины $\mathcal{M}'_0$. Все дальнейшие действия по вычислению значения функции f будут содержаться в цикле.

Итак, сравниваем число, записанное на второй дорожке, с числом x_n, имеющимся на первой дорожке. Если эти числа равны, то на третьей дорожке представлено в основном коде искомое значение функции f и нам необходимо лишь убрать дорожки и получить основной код результата вычислений. В противном случае пусть на второй и третьей дорожках записаны в основном коде числа y и z. Добавим 1 к числу y на второй дорожке, образуем на третьей дорожке основной код набора $(x_1, \ldots, x_{n-1}, y, z)$ и применим к этому набору машину $\mathcal{M}'_1$. После этого вернемся к началу цикла.

Операция минимизации рассматривается в значительной степени аналогично операции примитивной рекурсии. Поэтому соответствующие рассуждения мы опускаем.

Упражнения

38. Постарайтесь подробно описать работу машины Тьюринга, которая осуществляет применение операции минимизации (2.2) к функции g.

§ 3.5. Частичная рекурсивность функций, вычислимых на машинах Тьюринга

В этом параграфе мы докажем, что

всякая функция, вычислимая на машине Тьюринга, является частично рекурсивной.

Таким образом, класс частично рекурсивных функций оказывается равным классу функций, вычислимых на машинах Тьюринга.

Пусть машина Тьюринга $\mathcal{M}$ правильно вычисляет функцию f. Поскольку в дальнейшем доказательстве частичной рекурсивности функции f число переменных у нее принципиальной роли не играет, будем предполагать, что функция f зависит от одной переменной.

Коротко изложим план предстоящего доказательства. С машиной $\mathcal{M}$ свяжем три функции $l(x,t), r(x,t)$ и $q(x,t)$. Пусть в начальный момент времени на ленте машины $\mathcal{M}$ в основном коде записано число x и головка машины $\mathcal{M}$ установлена на первую единицу кода. Функция $l(x,t)$ будет давать код части L_t ленты, которая в момент времени t расположена слева от головки машины $\mathcal{M}$ (см. рис. 2).

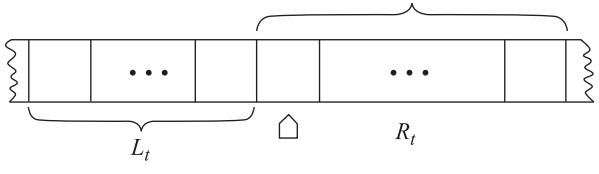

Рис. 2

Аналогичный смысл имеет функция $r(x,t)$ (отметим, что в правую часть R_t ленты входит клетка, обозреваемая головкой машины в момент t). Функция $q(x,t)$ дает номер состояния машины $\mathcal{M}$ в момент t.

Тройка чисел

$$l(x,t), \quad r(x,t), \quad q(x,t) \tag{3.3}$$

полностью определяет работу машины $\mathcal{M}$ после момента времени t. Вместо тройки (3.3) будем рассматривать одно число

$$\mathrm{Cod}\,(x,t) = c^3(l(x,t), r(x,t), q(x,t)),$$

где c^3 — нумерационная функция, введенная в § 2.3.

Далее мы покажем, что функция $\mathrm{Cod}\,(x,t)$ примитивно рекурсивна. Поэтому функция

$$T(x) = (\mu t)(r(\mathrm{Cod}\,(x,t)) = 0) \tag{3.4}$$

будет частично рекурсивной. Содержательно функция $T(x)$ определяет наименьший момент времени, в который машина $\mathcal{M}$ достигает заключительного состояния q_0 (если, конечно, функция f определена для аргумента x и соответственно машина $\mathcal{M}$ для данного x заканчивает работу). Значит, частично рекурсивная функция

$$\mathrm{Cod}\,(x, T(x)) \tag{3.5}$$

дает номер тройки (3.3) в заключительный момент времени. Тогда

$$r(l(\mathrm{Cod}\,(x, T(x)))) \tag{3.6}$$

есть код правой части ленты в момент окончания работы машины $\mathcal{M}$. Из него мы легко извлечем и само значение функции f.

Приступим к реализации намеченного плана. Для начала определим L_t и R_t. Считаем, что L_t есть наименьшая конечная связная (т. е. без пропусков) часть ленты, которая расположена строго слева от клетки, обозреваемой головкой машины $\mathcal{M}$ в момент t, и включает все клетки, содержащие в момент t символ 1. Аналогично определяем R_t, предполагая только, что самой левой клеткой R_t всегда является клетка, обозреваемая головкой машины в момент t.

Кодом части L_t будем считать число, двоичная запись которого находится в клетках части L_t. Код R_t определим так же, только двоичные разряды в R_t будем рассматривать слева направо (слева младшие разряды, справа старшие).

Определим значения функций (3.3) в начальный момент $t = 0$. Поскольку в этот момент вся лента, расположенная слева от первой единицы основного кода числа x, пуста (т. е. заполнена сплошь нулями), имеем $l(x,0) = 0$. Часть R_0 состоит из $(x+1)$ единиц. В двоичной системе счисления это есть запись числа $2^{x+1} - 1$. Значит, $r(x,0) = 2^{x+1} \div 1$. Перед началом вычисления машина $\mathcal{M}$ устанавливается в состояние q_1. Поэтому $q(x,0) = 1$. В итоге получаем

$$\mathrm{Cod}\,(x,0) = c^3(0, 2^{x+1} \div 1{,}1)$$

(мы дважды написали $2^{x+1} \div 1$, хотя можно было бы оставить $2^{x+1} - 1$, поскольку данная функция неотрицательна и, как легко понять, примитивно рекурсивна).

Мы хотим далее определить функцию $\mathrm{Cod}\,(x,t)$ примитивной рекурсией по t. Сразу это сделать весьма затруднительно, поскольку при переходе от $\mathrm{Cod}\,(x,t)$ к $\mathrm{Cod}\,(x,t+1)$ приходится обращаться к программе «произвольной» машины $\mathfrak{M}$. Поэтому мы поступим так: путем разбора случаев (согласно программе машины $\mathfrak{M}$) определим значения компонент

$$l(x,t+1), \quad r(x,t+1), \quad q(x,t+1) \tag{3.7}$$

функции $\mathrm{Cod}\,(x,t+1)$, используя при этом известные значения (3.3). Напомним в связи с этим, что по определению

$$\begin{aligned} l(x,t) &= l(l(\mathrm{Cod}\,(x,t))),\\ r(x,t) &= r(l(\mathrm{Cod}\,(x,t))),\\ q(x,t) &= r(\mathrm{Cod}\,(x,t)). \end{aligned}$$

Начнем со значения $l(x,t+1)$. Чтобы определить $l(x,t+1)$, необходимо знать $l(x,t)$, символ a_i, обозреваемый головкой машины $\mathfrak{M}$ в момент t, и состояние, в котором машина $\mathfrak{M}$ находится в момент t. Символ a_i есть крайний левый символ в R_t, т. е.

$$a_i = \mathrm{rm}\,(r(x,t),2).$$

Номер j состояния q_j машины $\mathfrak{M}$ в момент t дается равенством $j = q(x,t)$. При $j \neq 0$ паре (a_i, q_j) в программе машины $\mathfrak{M}$ отвечает единственная команда (3.1). Если $M = S$, то, очевидно, $L_{t+1} = L_t$ и потому $l(x,t+1) = l(x,t)$. Пусть $M = L$. Если L_t состоит из двух или более клеток ленты, то L_{t+1} получается из L_t отбрасыванием последней (правой) клетки. Если же L_t состоит из одной клетки, то L_{t+1} будет также состоять из одной (и при этом пустой) клетки. В обоих случаях получаем

$$l(x,t+1) = \left[\frac{l(x,t)}{2}\right].$$

Пусть теперь $M = R$. Тогда L_{t+1} получается из L_t присоединением справа клетки с символом a_r (именно этот символ записывает машина $\mathfrak{M}$ вместо обозреваемого символа a_i). Нетрудно понять, что в этом случае будет

$$l(x,t+1) = 2l(x,t) + a_r.$$

Похожим образом определяется значение $r(x,t+1)$. Наиболее простым является случай, когда $M = R$. Здесь

$$r(x,t+1) = \left[\frac{r(x,t)}{2}\right].$$

Если $M = S$, то в R_t необходимо заменить крайний левый символ a_i символом a_r. Это достигается следующим преобразованием:

$$r(x, t+1) = 2\left[\frac{r(x,t)}{2}\right] + a_r.$$

Более сложным является случай, когда $M = L$. Здесь необходимо сначала заменить в R_t крайний левый символ a_i символом a_r, а затем добавить слева крайний правый символ из L_t. Соответствущая формула для определения значения $r(x, t+1)$ имеет вид

$$r(x, t+1) = 2\left(2\left[\frac{r(x,t)}{2}\right] + a_r\right) + \operatorname{rm}(l(x,t), 2).$$

Если известны a_i и q_j, то при $j \neq 0$ величина $q(x, t+1)$ определяется в соответствии с командой (3.1), т. е. $q(x, t+1) = s$.

Таким образом, имея значение $\operatorname{Cod}(x,t)$, находим тройку (3.3), величины $a_i = \operatorname{rm}(l(x,t), 2)$, $j = q(x,t)$, команду (3.1) с левой частью $a_i q_j$ в программе машины $\mathfrak{M}$ и с помощью описанных выше действий вычисляем значения (3.7). Нетрудно понять, что все эти действия выражаются примитивно рекурсивными функциями. Это дает основание утверждать, что $\operatorname{Cod}(x,t)$ является примитивно рекурсивной функцией. Тогда функции (3.4)–(3.6) будут частично рекурсивными.

Как отмечалось, если значение $f(x)$ определено, то величина (3.6) равна коду части R_{t_0} в заключительный момент времени t_0. Вспоминая теперь определение функции $r(x,t)$, получаем

$$f(x) = [\log_2 r(l(\operatorname{Cod}(x, T(x))))],$$

где функция $T(x)$ определяется равенством (3.4).

Отметим, что попутно мы еще раз установили, что всякую частично рекурсивную функцию можно получить из исходных функций $0,\ x+1,\ I$ с помощью операций суперпозиции, примитивной рекурсии и не более чем однократным применением операции минимизации (см. соотношение (3.4)).

§ 3.6. Универсальная машина Тьюринга

Универсальность — явление, характерное для теории алгоритмов. Большинство самых известных результатов теории алгоритмов так или иначе связаны с универсальностью.

Понятие универсальности проще всего объяснить на примере функций одной переменной. Пусть $f_0(x), f_1(x), \ldots$ — последовательность функций, заданных на N. Это могут быть, например, примитивно рекурсивные либо частично рекурсивные функции.

Функция $U(n, x)$ называется *универсальной для последовательности функций* $f_0, f_1, \ldots$, если выполняются два условия:

во-первых, для любого n из N функция $U(n,x)$ как функция от x совпадает с некоторой функцией $f_m(x)$;

во-вторых, для любой функции $f_m(x)$ найдется такое n (их может быть и несколько), что функция $U(n,x)$ как функция от x совпадает с функцией $f_m(x)$.

Это определение универсальной функции можно выразить несколько иначе, если сказать, что совокупность функций $\{U(0,x), U(1,x), \ldots\}$ совпадает с совокупностью $\{f_0(x), f_1(x), \ldots\}$ (подчеркнем еще раз, что в последовательности $U(0,x), U(1,x), \ldots$ функции $f_0(x), f_1(x), \ldots$ располагаются в произвольном порядке и, возможно, с повторениями).

Понятие универсальной машины Тьюринга представляет собой вариант понятия универсальной функции. Для простоты ограничимся только машинами Тьюринга, имеющими ленточный алфавит $\{0,1\}$.

Назовем машину Тьюринга $\mathcal{U}$ *универсальной*, если она вычисляет функцию $U(n,x)$, которая является универсальной для последовательности функций одной переменной, вычислимых на машинах Тьюринга с ленточным алфавитом $\{0,1\}$.

В двух предыдущих параграфах мы установили, что класс частично рекурсивных функций совпадает с классом функций, вычислимых на машинах Тьюринга. Поэтому универсальная машина Тьюринга (если она существует) вычисляет частично рекурсивную функцию $U(n,x)$, которая является универсальной для последовательности всех частично рекурсивных функций одной переменной.

Далее мы продемонстрируем, как можно построить универсальную машину Тьюринга. К сожалению, нам не удастся выписать программу универсальной машины Тьюринга (она содержит более сотни команд), однако принципы ее устройства довольно просты и мы постараемся их изложить.

Прежде всего мы хотим перенумеровать все машины Тьюринга, работающие в алфавите $\{0,1\}$, т. е. сопоставить каждой машине Тьюринга некоторое число из N. При этом способ нумерации должен быть таким, чтобы по номеру машины Тьюринга можно было бы однозначно восстановить ее программу.

Сначала закодируем команды машин Тьюринга. Команде (3.1) сопоставим слово в алфавите $\{0,1,2\}$, имеющее вид

$$2a_i2d(j)2a_r2d(M)2d(s)2, \tag{3.8}$$

где $d(m)$ — двоичное представление числа m и $d(L)=0$, $d(R)=1$, $d(S)=01$.

Если программа машины Тьюринга $\mathcal{M}$ имеет p команд и им уже сопоставлены слова $w_1, \ldots, w_p$ вида (3.8) в алфавите $\{0,1,2\}$, то всей программе машины $\mathcal{M}$ сопоставим слово в алфавите $\{0,1,2,3\}$, которое имеет вид

$$w_1 3 w_2 3 \ldots 3 w_p \tag{3.9}$$

(порядок слов $w_1, \ldots, w_p$ в (3.9) роли не играет). Номером машины $\mathcal{M}$ теперь будем считать число, имеющее представление (3.9) в четверичной системе счисления.

Понятно, что в результате каждая машина Тьюринга получит некоторый номер (и даже несколько номеров, поскольку слова $w_1, \ldots, w_p$ в (3.9) можно переставлять). При этом по номеру машины Тьюринга сравнительно просто восстановить программу машины: достаточно лишь представить этот номер в четверичной системе счисления. Отметим еще, что далеко не каждое натуральное число будет являться номером некоторой машины Тьюринга. Однако, пользуясь алгоритмом кодирования машин Тьюринга, не составляет особого труда проанализировать четверичное представление натурального числа и выяснить, определяет ли оно программу машины Тьюринга.

Перейдем теперь к построению универсальной машины Тьюринга $\mathcal{U}$. Собственно, мы проведем лишь описание функционирования универсальной машины $\mathcal{U}$. Детали построения программ для отдельных частей машины $\mathcal{U}$ мы оставляем читателю.

Итак, пусть на ленте машины $\mathcal{U}$ в основном коде представлена пара чисел n, x. Как и в § 3.4, выделим на ленте машины $\mathcal{U}$ три дорожки. Первая дорожка будет служить для записи программы машины Тьюринга, имеющей номер n (обозначим эту машину через $\mathcal{M}_n$). На второй дорожке будет храниться двоичный номер «текущего» состояния машины $\mathcal{M}_n$. А на третьей дорожке будет собственно моделироваться процесс применения машины $\mathcal{M}_n$ к аргументу x.

В соответствии с распределением ролей дорожек машина $\mathcal{U}$ прежде всего переносит число n на первую дорожку и создает на ней четверичную запись n (для этого можно воспользоваться «школьным» алгоритмом деления на 4 с остатком). На вторую дорожку машина $\mathcal{U}$ записывает число 1, а на третью — число x в основном коде. Кроме того, на третьей дорожке особым символом (меткой) помечается та клетка, которую в данный момент времени обозревает головка моделируемой машины $\mathcal{M}_n$.

Используя структуру представлений (3.8), (3.9), машина $\mathcal{U}$ на первой дорожке проверяет, является ли четверичная запись числа n номером некоторой машины Тьюринга. Если нет, то машина $\mathcal{U}$ реализует какую-либо простую функцию (например, константу 0).

Предположим, что на первой дорожке действительно записан код машины Тьюринга $\mathcal{M}_n$. В этом случае машина $\mathcal{U}$ шаг за шагом моделирует процесс применения машины $\mathcal{M}_n$ к аргументу x. Для этого с третьей дорожки считывается символ a_i, который в данный (моделируемый) момент времени обозревает головка машины $\mathcal{M}_n$. Затем машина $\mathcal{U}$ последовательно просматривает код машины $\mathcal{M}_n$, представленный на первой дорожке, и разыскивает в нем слово (3.8), где j — номер «текущего» состояния машины $\mathcal{M}_n$, записанный на второй дорожке. Найдя это слово, машина $\mathcal{U}$ осуществляет преобразования на второй и третьей дорожках ленты, соответствующие тройке (a_r, M, s). Имен-

но, на второй дорожке двоичная запись числа j заменяется двоичной записью числа s, а на третьей дорожке обозреваемый машиной $\mathcal{M}_n$ символ a_i заменяется символом a_r и сдвигается метка, указывающая положение головки машины $\mathcal{M}_n$ на ленте. Далее машина $\mathcal{U}$ переходит к моделированию следующего шага работы машины $\mathcal{M}_n$.

Если машина $\mathcal{M}_n$ заканчивает работу над аргументом x, то машина $\mathcal{U}$ преобразует результат вычислений машины $\mathcal{M}_n$, полученный на третьей дорожке, в основной код. В противном случае машина $\mathcal{U}$, как и машина $\mathcal{M}_n$, работает неограниченно долго.

Из существования универсальной машины Тьюринга, как мы уже отмечали, следует существование частично рекурсивной функции $U(n,x)$, универсальной для множества всех частично рекурсивных функций одной переменной. В свою очередь из последнего факта вытекает ряд следствий, составляющих ядро теории алгоритмов. Приведем некоторые из них.

Рассмотрим функцию $U(x,x)+1$. Очевидно, что она частично рекурсивна. Значит, для некоторого n и всех x должно выполняться равенство

$$U(n,x) = U(x,x) + 1.$$

Подставляя в него вместо x число n, получим «противоречивое» равенство

$$U(n,n) = U(n,n) + 1. \tag{3.10}$$

В чем здесь ошибка? Ошибки нет. Равенство (3.10) показывает лишь, что значение $U(n,n)$ не определено (напомним, что функция $U(n,x)$ частичная).

Таким образом, имея универсальную частично рекурсивную функцию U, мы можем точно указать набор (именно набор (n,n), где n есть номер функции $U(x,x)+1$ в нумерации частично рекурсивных функций, даваемой функцией U), на котором функция U заведомо не определена. Итак, функция $U(x,x)+1$ частично рекурсивна и не всюду определена.

Однако бывают такие частично рекурсивные функции (например функция $[\log_2 x]$), которые легко доопределить до всюду определенных частично рекурсивных функций. Покажем, что функция $U(x,x)+1$ этим свойством не обладает, т. е.

не существует такой всюду определенной частично рекурсивной функции $V(x)$, которая совпадает с функцией $U(x,x)+1$ при всех x, для которых значение $U(x,x)$ определено.

Доказательство проведем от противного. Предположим, что функция $V(x)$ с указанным свойством существует. Тогда по определению универсальной функции для некоторого n и всех x выполняется равенство

$$V(x) = U(n,x).$$

В частности, оно верно при $x=n$. Однако функция V по предположению всюду определена. Следовательно, для данного n значение $U(n,n)$

определено и совпадает со значением

$$V(n) = U(n, n) + 1.$$

Противоречие покзывает, что наше предположение неверно и всюду определенной частично рекурсивной функции $V(x)$, доопределяющей функцию $U(x, x) + 1$, не существует.

И еще один интересный результат связан с функцией U. Рассмотрим множество M всех тех чисел n, для которых функция $U(n, x)$ как функция от x всюду определена. Может ли множество M быть рекурсивно перечислимым? Оказывается, нет.

В самом деле, предположим, что множество M рекурсивно перечислимо. Тогда (см. § 2.4) найдется примитивно рекурсивная функция f, которая перечисляет множество M. Рассмотрим функцию

$$V(x) = U(f(x), x) + 1. \tag{3.11}$$

Поскольку функция f принимает лишь значения из множества M, функция V будет всюду определенной частично рекурсивной функцией. Покажем, что она отличается от любой всюду определенной частично рекурсивной функции $g(x)$. Действительно, согласно определению универсальной функции найдется такое n, что для всех x выполняется равенство

$$g(x) = U(n, x).$$

Пусть число m таково, что $f(m) = n$. Тогда из соотношения (3.11) следует, что функция V отличается от функции g в точке $x = m$. Получили противоречие. Значит, множество M не является рекурсивно перечислимым множеством.

Упражнения

39. Докажите, что функция $U(x, 2x)$ не имеет всюду определенного частично рекурсивного доопределения.

40. Пусть M есть множество всех тех n, для которых функция $U(n, x)$ как функция от x определена хотя бы в одной точке. Докажите, что множество M рекурсивно перечислимо.

Для более глубокого изучения рекурсивных функций читателю можно порекомендовать следующие книги.

Мальцев А. И. Алгоритмы и рекурсивные функции. — М.: Наука, 1986.
Верещагин Н. К., *Шень А.* Вычислимые функции. — М.: МЦНМО, 1999.
Марченков С. С. Элементарные рекурсивные функции. — М.: МЦНМО, 2003.

Ответы, решения, указания

1. Имеем $0! = 1$, $(x+1)! = x! \cdot (x+1)$ и $[0/2] = [1/2] = 0$, $[(x+2)/2] = [x/2] + 1$ (здесь базис индукции состоит из двух равенств).

2. $h_1(x,y) = 2x + y + 1$, $h_2(x,y) = 3x^2 + 3x + y + 1$.

3. Любое число s, где $1 \leqslant s \leqslant r$.

4. Можно воспользоваться нумерационной функцией $c(x,y)$ (см. § 2.3) и образовать вспомогательную функцию

$$f'(x_1, \ldots, x_n) = c(f(x_1, \ldots, x_n), f(x_1, \ldots, x_{n-1}, x_n + 1)).$$

6. Четвертая функция $f(x,y)$ этого ряда может определяться следующей примитивной рекурсией:

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

В этом случае

$$f(x,y) = \overbrace{x^{x^{\cdot^{\cdot^{\cdot^{x}}}}}}^{(y+1) \text{ раз}}.$$

7. Можно считать, что $d > 1$. Функция $\operatorname{rm}(x,d) = x \div d \cdot [x/d]$ (см. упр. 5) периодическая с периодом d. Любая периодическая функция $f(x)$ с периодом d может быть получена из функции $\operatorname{rm}(x,d)$ подходящей заменой ее значений $0, 1, \ldots, d-1$ (так, как это делается, например, в формуле (1.14)).

8. Да, можно.

9. Следует n раз применить операцию ограниченного суммирования к функции $\overline{\operatorname{sg}}\,|P(x_1, \ldots, x_n) - Q(x_1, \ldots, x_n)|$.

10. Пусть $g(x,i)$ — характеристическая функция отношения $2^i | x$. Тогда $\exp_2(x) = \sum\limits_{i \leqslant x} g(x, i+1)$.

11. Пусть $h(y,z)$ — характеристическая функция отношения $g(z) = y$. Тогда

$$f(x,y) = \sum_{z \leqslant x} (z \cdot \overline{\operatorname{sg}} \sum_{i<z} h(y,i))$$

(операция $\sum\limits_{i<z}$ определяется аналогично операции $\sum\limits_{i \leqslant z}$).

12. Легко показать, что для всякого x число $p(x+1)$ не превосходит величины $p(x)! + 1$. Поэтому с использованием упр. 11 функцию $p(x)$ можно определить следующей примитивной рекурсией:

$$\begin{cases} p(0) = 2, \\ p(x+1) = \text{наименьшему } z, \text{ такому, что } p(x) < z \leqslant p(x)! + 1 \\ \qquad\qquad \text{и истинно } \Pr(z). \end{cases}$$

13. Очевидно, что $f(0) = 4$. Далее, $f(x+1)$ равно такому (единственному) числу $a, 0 \leqslant a \leqslant 9$, что

$$\left(1 + \frac{f(0)}{10} + \ldots + \frac{f(x)}{10^{x+1}} + \frac{a}{10^{x+2}}\right)^2 < 2$$

и

$$\left(1 + \frac{f(0)}{10} + \ldots + \frac{f(x)}{10^{x+1}} + \frac{a+1}{10^{x+2}}\right)^2 > 2.$$

Эти два соотношения можно переписать в виде

$$(10^{x+2} + f(0) \cdot 10^{x+1} + \ldots + f(x) \cdot 10 + a)^2 < 2 \cdot 10^{2(x+2)},$$
$$(10^{x+2} + f(0) \cdot 10^{x+1} + \ldots + f(x) \cdot 10 + a + 1)^2 > 2 \cdot 10^{2(x+2)}.$$

Отсюда уже нетрудно получить схему примитивной рекурсии, определяющую функцию $f(x)$.

14. Пусть $g_1(x,i)$ — характеристическая функция отношения $i|x$, $g_2(x)$ — характеристическая функция отношения «x не делится нацело на 2» ($g_2(x) = \overline{\text{sg}}\ g_1(x,2)$). Обе функции принадлежат классу **S**. Положим

$$g_3(x) = \overline{\text{sg}}\left(\sum_{i \leqslant x} g_1(x, i+2) g_2(i+2)\right).$$

Тогда $g_3(x)$ есть характеристическая функция отношения «x является степенью числа 2». Чтобы получить теперь функцию $[\log_2 x]$, достаточно просуммировать при $i \leqslant x$ произведение $g_3(i) \cdot (i \div 1)$.

15. Следует принять во внимание, что двоичное представление числа x содержит единицу на $(i+1)$-м месте в том и только том случае, когда $\text{rm}\,(x, 2^{i+1}) \geqslant 2^i$. Далее необходимо использовать функцию g_3 из упр. 14 для выделения чисел вида 2^{i+1} и 2^i.

16. Рассмотрим случай $b = 1$. Определим вспомогательные функции f_1–f_3:

$$f_1(x_1, \ldots, x_n) = \overline{\text{sg}}\left(\sum_{i \leqslant x_n} \overline{\text{sg}}\ g(x_1, \ldots, x_{n-1}, i)\right),$$

$$f_2(x_1, \ldots, x_n) = \sum_{i \leqslant x_n} g(x_1, \ldots, x_{n-1}, i) \cdot \text{sg}\,(g(x_1, \ldots, x_{n-1}, i) \div 1),$$

$$f_3(x_1, \ldots, x_n) = f_2(x_1, \ldots, x_n) + \overline{\text{sg}}\ f_2(x_1, \ldots, x_n).$$

Функция $f_1(x_1, \ldots, x_n)$ совпадает с функцией $\text{sg} \prod_{i \leqslant x_n} g(x_1, \ldots, x_{n-1}, i)$, функция $f_3(x_1, \ldots, x_n)$ дает отличное от 0 и 1 значение из ряда

$$g(x_1, \ldots, x_{n-1}, 0),\ g(x_1, \ldots, x_{n-1}, 1), \ldots, g(x_1, \ldots, x_{n-1}, x_n),$$

если такое значение существует, и 1 в противном случае. Функция (1.20) есть произведение $f_1(x_1,\dots,x_n)\cdot f_3(x_1,\dots,x_n)$. Случай $b>1$ рассматривается аналогично случаю $b=1$.

17. Примитивной рекурсией определим в классе **E** функцию f_1:

$$\begin{cases} f_1(x_1,\dots,x_{n-1},0,y)=\begin{cases} g(x_1,\dots,x_{n-1}), & \text{если } g(x_1,\dots,x_{n-1})\leqslant y,\\ y+1 & \text{в противном случае;}\end{cases}\\ f_1(x_1,\dots,x_{n-1},x_n+1,y)=\begin{cases} h(x_1,\dots,x_n,f_1(x_1,\dots,x_n,y)), \text{ если}\\ \qquad h(x_1,\dots,x_n,f_1(x_1,\dots,x_n,y))\leqslant y,\\ y+1 \text{ в противном случае.}\end{cases}\end{cases}$$

Далее получаем

$$f'(x_1,\dots,x_n,y)=\begin{cases} f_1(x_1,\dots,x_n,y), & \text{если } f_1(x_1,\dots,x_n)\leqslant y,\\ 0, & \text{если } f_1(x_1,\dots,x_n,y)=y+1.\end{cases}$$

18. Проще всего воспользоваться вторым определением класса **K**, приведенным в конце § 1.5. Легко заметить, что при любом x выполняется неравенство $F(x)\leqslant 2^x$. Следовательно, достаточно написать схему примитивной рекурсии, определяющую функцию $F(x)$ через функции класса **E**. Это можно сделать с помощью приема, который применялся при решении упр. 4.

19. Пусть $f(x)=\prod\limits_{i\leqslant x} g(i)$. Постарайтесь, не используя операцию ограниченного мультиплицирования, выразить характеристическую функцию отношения

$$y=p_0^{g(0)+1}\cdot p_1^{g(0)\cdot g(1)+1}\cdot\ldots\cdot p_x^{g(0)\cdot\ldots\cdot g(x)+1},$$

где $p_0,p_1,\dots,p_x$ — последовательные простые числа.

21.

$$(\mu z)(x+z=y)=\begin{cases} y-x, & \text{если } y\geqslant x,\\ \text{не определено}, & \text{если } y<x,\end{cases}$$

$$(\mu z)(x\cdot z=y)=\begin{cases} 0, & \text{если } y=0,\\ y/x, & \text{если } y\neq 0 \text{ и } y \text{ делится нацело на } x,\\ \text{не определено} & \text{в остальных случаях,}\end{cases}$$

$$(\mu z)(x-z=y)=\begin{cases} x-y, & \text{если } x\geqslant y,\\ \text{не определено}, & \text{если } x<y,\end{cases}$$

$$(\mu z)(z-x=y)=\begin{cases} y, & \text{если } x=0,\\ \text{не определено}, & \text{если } x\neq 0,\end{cases}$$

$(\mu z)(x/z=y)$ есть нигде не определенная функция,

$$(\mu z)(z/x=y)=\begin{cases} xy, & \text{если } x\neq 0,\\ \text{не определено}, & \text{если } x=0.\end{cases}$$

22. $g_1(x,y)=|x-y|$, $g_2(x,y)=[x/y]$ при условии, что $[x/0]=0$.

23. Определим примитивно рекурсивную функция $g(x)$:

$$g(x)=a_1\cdot\overline{\operatorname{sg}}\,|x-1|+\ldots+a_s\cdot\overline{\operatorname{sg}}\,|x-s|+a_1\cdot\operatorname{sg}|x-1|\cdot\ldots\cdot\operatorname{sg}|x-s|.$$

Она принимает лишь значения $a_1,\ldots,a_s$. Имеем

$$f(x)=\operatorname{sg}((\mu y)(g(y)=x)+1).$$

24. Имеем $g(x)=\overline{\operatorname{sg}}\,((\mu y)(l_1(x,y)=1)+1)$.

26. Расположим все пары чисел из N в следующем порядке:

$$(0,0);\ (0,1),(1,0);\ (0,2),(1,1),(2,0);\ (0,3),\ldots$$

Соответствующая этому порядку нумерующая функция будет

$$x+\frac{(x+y)(x+y+1)}{2}=\frac{(x+y)^2+3x+y}{2}.$$

Убедитесь, что «обратные» функции $l(v)$ и $r(v)$ определяются формулами

$$l(v)=v\div\frac{1}{2}\left[\frac{[\sqrt{8v+1}\,]+1}{2}\right]\left[\frac{[\sqrt{8v+1}\,]\div 1}{2}\right],$$

$$r(v)=\left[\frac{[\sqrt{8v+1}\,]+1}{2}\right]\div(l(v)+1).$$

27. Если одно из множеств M_1,M_2 пусто, например M_1, то $M_1\cup M_2=M_2$ и $M_1\cap M_2=\varnothing$. Пусть поэтому оба множества M_1,M_2 непусты и $f_1(x),f_2(x)$ — примитивно рекурсивные функции, перечисляющие множества M_1,M_2. Тогда примитивно рекурсивная функция

$$f_1(x)\cdot\operatorname{sg}\operatorname{rm}(x,2)+f_2(x)\cdot\overline{\operatorname{sg}}\,(\operatorname{rm}(x,2))$$

перечисляет множество $M_1\cup M_2$. Если множества M_1,M_2 не имеют общих элементов, то $M_1\cap M_2=\varnothing$. В противном случае пусть a — фиксированный элемент из $M_1\cap M_2$. Тогда множество $M_1\cap M_2$ перечисляется примитивно рекурсивной функцией

$$f_1(l(x))\cdot\overline{\operatorname{sg}}\,|f_1(l(x))-f_2(r(x))|+a\cdot\operatorname{sg}|f_1(l(x))-f_2(r(x))|.$$

28. Если $M_1=\varnothing$ либо все элементы множества M_1 входят в множество M_2, то $M_1\setminus M_2=\varnothing$. Предположим поэтому, что $M_1\setminus M_2\neq\varnothing$, $f_1(x)$ — примитивно рекурсивная функция, перечисляющая множество M_1, а $f_2(x)$ — характеристическая функция отношения «x входит в множество M_2». Пусть a — фиксированный элемент множества $M_1\setminus M_2$. Тогда множество $M_1\setminus M_2$ перечисляет примитивно рекурсивная функция $f_1(x)\cdot\overline{\operatorname{sg}}\ f_2(f_1(x))+a\cdot f_2(f_1(x))$.

29. Если $M = \varnothing$, то $f(x)$ — нигде не определенная функция. Пусть $M \neq \varnothing$ и $g(x)$ — примитивно рекурсивная функция, перечисляющая множество M. Тогда

$$f(x) = \operatorname{sg}((\mu y)(g(y) = x) + 1).$$

30. Пусть функция f представима в виде (2.13) с примитивно рекурсивной функцией G. Тогда характеристической функцией отношения $f(x_1, \ldots, x_n) = y$ будет функция

$$\overline{\operatorname{sg}}\,(G(x_1, \ldots, x_n, y)) \cdot \operatorname{sg}\left(\prod_{i<y} G(x_1, \ldots, x_n, i)\right).$$

Обратно, пусть $g(x_1, \ldots, x_n, y)$ — примитивно рекурсивная характеристическая функция отношения $f(x_1, \ldots, x_n) = y$. Тогда

$$f(x_1, \ldots, x_n) = (\mu y)(\overline{\operatorname{sg}}\,(g(x_1, \ldots, x_n, y)) = 0).$$

31. Пусть программа машины Тьюринга $\mathcal{M}$ содержит команду (3.1), где $M = S$. Добавим к множеству состояний машины $\mathcal{M}$ новое состояние q'_s, а команду (3.1) заменим серией из $(k+2)$ команд

$$a_i q_j \to a_r L q'_s, \quad a_l q'_s \to a_l R q_s \quad (l = 0, 1, \ldots, k).$$

Проделав это преобразование со всеми командами машины $\mathcal{M}$, содержащими символ S, получим программу машины Тьюринга $\mathcal{M}'$, которая эквивалентна машине $\mathcal{M}$.

32. Реализовать следующий алгоритм вычисления: стереть первую единицу основного кода, пробежать слева направо оставшийся массив из единиц (если он есть), заменить разделительный нуль единицей, вернуться к началу полученного массива из единиц, стереть в нем первую единицу, сдвинуться вправо на одну клетку и остановиться.

33. Может, однако эти функции должны зависеть от различного числа переменных. Одна из простейших машин такого типа правильно вычисляет функции $x + 1$ и $x + y + 2$. Для этого она пробегает слева направо первый массив из единиц, заменяет стоящий справа 0 на 1 и затем возвращается на первую единицу образовавшегося массива.

34. Можно. Достаточно для машины $\mathcal{M}'$ поменять ролями «лево» и «право». Иными словами, основной код набора $(x_1, \ldots, x_n)$ следует преобразовать в основной код набора $(x_n, \ldots, x_1)$, а в начальный момент времени головку машины $\mathcal{M}'$ установить на самую правую единицу основного кода.

36. Сначала следует записать символ 1 слева или справа от основного кода, отступив на одну пустую клетку. Затем (эти действия будут циклически повторяться) необходимо в основном коде числа x заменить две крайние единицы нулями (соблюдайте осторожность: они

могут оказаться последними!), добавить единицу к ранее поставленной единице и т. д.

37. Для случая $l = 3$ программа машины $\mathcal{M}_6$ выглядит так:

	q_1	q_2	q_3	q_4	q_5	q_6	q_6^1	q_6^2
0	$0Rq_2$	$0Rq_2$	$0Lq_7$	—	$0Lq_5$	$1Lq_6^1$	$1Lq_6^2$	$1Sq_1$
1	$1Rq_1$	$1Rq_3$	$1Lq_4$	$0Lq_5$	$1Lq_6$	$1Lq_6$	—	—

	q_7	q_8	q_9	q_9^1	q_9^2
0	—	$0Lq_8$	$1Lq_9^1$	$1Lq_9^2$	$1Sq_0$
1	$0Lq_8$	$1Lq_9$	$1Lq_9$	—	—

Жирными чертами отделены части машины $\mathcal{M}_6$, которые можно рассматривать как самостоятельные машины Тьюринга.

39. Пусть функция $U(x, 2x)$ имеет всюду определенное частично рекурсивное доопределение $V(x)$. Тогда функция $V(x) + 1$ также всюду определена и частично рекурсивна. Возьмем такое n, чтобы частично рекурсивная функция $U(n, 2x)$ как функция от x совпадала с функцией $V(x) + 1$, т. е. $U(n, 2x) = V(x) + 1$. Если положим теперь $x = n$, то получим противоречие.

40. Как установлено в § 2.4, график функции $U(n, x)$ есть рекурсивно перечислимое множество. Пусть примитивно рекурсивная функция $f(x)$ перечисляет это множество. Тогда множество M будет перечисляться примитивно рекурсивной функцией $l(f(x))$.