

АЛГОРИТМЫ

(Лекции по дискретной математике)

М.И. Дехтярь

Содержание

1	Что такое алгоритм?	2
2	Структурированные программы	3
2.1	Задачи	6
3	Частично рекурсивные функции	7
3.1	Основные определения	7
3.1.1	Примеры	9
3.2	Программная вычислимость рекурсивных функций	11
3.3	Леммы о рекурсивных функциях	14
3.4	Задачи	17
4	Машины Тьюринга	19
4.1	Основные определения	19
4.2	Тьюрингово программирование	22
4.2.1	Стандартная заключительная конфигурация	23
4.2.2	Односторонние машины Тьюринга	24
4.2.3	Последовательная и параллельная композиции машин Тьюринга	25
4.2.4	Ветвление (условный оператор)	26
4.2.5	Повторение (цикл)	27
4.3	Задачи	28
5	Машины Тьюринга, ч.р.ф. и структурированные программы	30
5.1	Вычислимость частично рекурсивных функций по Тьюрингу	30
5.2	Моделирование структурированных программ машинами Тьюринга	32
5.3	Частичная рекурсивность функций, вычисляемых по Тьюрингу	33
5.4	Задачи	35

6	Тезис Тьюринга-Черча и алгоритмически неразрешимые проблемы	35
6.1	Задачи	42

1 Что такое алгоритм?

Первоначальной целью теории алгоритмов является классификация всех задач на алгоритмически разрешимые и неразрешимые, т.е. на те, для которых существуют решающие их алгоритмы, и те, для которых таких алгоритмов нет. Неформально под алгоритмом \mathcal{A} можно понимать выраженный в некотором языке *набор правил (предписание, рецепт, способ)*, позволяющий применить к исходным (входным) данным x из некоторого множества допустимых данных X последовательность дискретных действий (операций, команд), приводящую к определенному результату - выходным данным $y = \mathcal{A}(x)$ из некоторого множества Y . В этом случае говорят, что алгоритм \mathcal{A} вычисляет функцию $F(x) = \mathcal{A}(x)$ типа $X \rightarrow Y$. Это нестрогое определение вполне подходит в тех случаях, когда для некоторой функции нам предъявляется “объект”, называемый алгоритмом ее вычисления (например, алгоритм Эвклида для вычисления наибольшего общего делителя двух целых чисел), и можно легко проверить позволяет ли он действительно вычислить требуемую функцию. Однако оно совершенно не годится, для доказательства того, что для заданной функции никакого алгоритма нет.

Начиная с тридцатых годов XX века, был предпринят ряд исследований для формализации понятия алгоритма. Перечислим некоторые из предложенных разными авторами в разное время формальных моделей: машины Тьюринга-Поста, частично-рекурсивные функции (Гедель, Клини), ε -исчисление (Черч, Клини), итеративные автоматы Неймана, нормальные алгорифмы Маркова, счетчиковые автоматы Минского, автоматы на графах Колмогорова-Барздиня и др. Заложенные в них идеи в значительной степени повлияли затем на архитектуру и языки программирования реальных компьютеров (например, на базе λ -исчисления построен широко применяемый в задачах искусственного интеллекта язык ЛИСП, а из нормальных алгорифмов Маркова произошел хорошо подходящий для текстовой обработки язык РЕ-ФАЛ). Каждый из многочисленных языков программирования также задает некоторую формальную модель алгоритмов. Мы вначале рассмотрим один из простейших таких языков — простые структурированные программы. А затем сравним их с двумя другими моделями алгоритмов: описаниями частично рекурсивных функций и машинами Тьюринга.

Хотя алгоритмы в разных прикладных областях имеют дело с дискретными объектами различных видов: целыми и рациональными числами, строками, формулами, разного рода выражениями, графами, матрицами, таблицами, точечными изображениями и др. — мы в этой части курса будем рассматривать только задачи вычисления функций от натуральных аргументов, принимающих натуральные значения. Такие

функции часто называют *арифметическими*. Дело в том, что для любого естественного множества дискретных объектов (в частности, для всех перечисленных выше) имеется простое кодирование его элементов целыми числами. Поэтому задачи вычисления функций на этих множествах превращаются в задачи вычисления арифметических функций.

Напомним, что через \mathbf{N} обозначается множество натуральных чисел, т.е. $\mathbf{N} = \{0, 1, 2, \dots\}$. Для частичной n -местной арифметической функции $f : \mathbf{N}^n \rightarrow \mathbf{N}$ через δ_f обозначим область ее определения. Чтобы указать, что f не определена на некотором наборе чисел a_1, \dots, a_n будем писать $f(a_1, \dots, a_n) = \infty$, а если f на этом наборе определена, то будем писать $f(a_1, \dots, a_n) < \infty$. Таким образом, $\delta_f = \{(a_1, \dots, a_n) \mid f(a_1, \dots, a_n) < \infty\}$.

2 Структурированные программы

В этом разделе рассмотрим в качестве средства описания алгоритмов структурированные программы. Они вычисляют функции, используя минимальные средства: элементарные присваивания, условные операторы и циклы.

Определим вначале синтаксис структурированных программ. Зафиксируем для этого некоторое счетное множество имен переменных Var , которые будут использоваться в программах. Как обычно, будем считать, что оно включает имена $x, x_1, x_2, \dots, y, y_1, \dots, z, z_1, \dots$ и т.п. В последующих определениях x, y, z — это произвольные переменные из Var .

Определение 2.1. Оператор присваивания. *Присваивание — это выражение одного из следующих трех видов:*

- а) $x := x + 1$
- б) $x := 0$
- в) $x := y$.

Определение 2.2. Условие. *Условие — это выражение одного из двух видов:*

- а) $x = y$ или б) $x < y$.

Структурированные программы определяются индуктивно.

Определение 2.3. Структурированные программы.

- а) Каждое присваивание — это структурированная программа.
- б) Если Π_1 и Π_2 — структурированные программы, то $\Pi = \Pi_1; \Pi_2$ — это структурированная программа.
- в) Если Π_1 и Π_2 — структурированные программы, а Φ — это условие, то $\Pi = \text{если } \Phi \text{ то } \Pi_1 \text{ иначе } \Pi_2 \text{ конец}$ является структурированной программой.
- г) Если Π_1 — структурированная программа, а Φ — это условие, то

П = пока Φ делай P_1 все

является структурированной программой.

д) Других структурированных программ нет.

Конструкция в п. (б) называется *последовательным применением* или *композицией* программ P_1 и P_2 , конструкция в п. (в) называется *условным оператором*, а конструкция в п. (г) — это *оператор цикла*, Φ — это условие цикла, а P_1 — это тело цикла.

С помощью структурированных программ (далее называемых просто программами) вычисляются (частичные) функции от натуральных аргументов, принимающие натуральные значения. С каждой программой P свяжем естественным образом множество входящих в нее переменных Var_P (определите это множество индукцией по построению программы). В процессе работы программа изменяет значения этих переменных. *Операционная семантика* задает правила такого изменения.

Определение 2.4. Состояние — это отображение σ из множества переменных Var во множество \mathbf{N} .

Для $x \in Var$ через $\sigma(x)$ обозначим значение переменной x в состоянии σ .

Через \mathbf{S} обозначим множество всех состояний.

Разумеется, при рассмотрении конкретной программы P нас будут интересовать значения переменных из Var_P .

Определение 2.5. Операционная семантика программы программы P — это отображение (вообще говоря, частичное) типа $\mathbf{S} \rightarrow \mathbf{S}$, которое программа P индуцирует на множестве всех состояний. Через $P(\sigma)$ обозначим состояние — результат применения программы P к состоянию σ . Оно определяется индукцией по построению программы.

а) $(x := x + 1)(\sigma) = \sigma_1$, где $\sigma_1(y) = \sigma(y)$ при $y \neq x$, и $\sigma_1(x) = \sigma(x) + 1$.

б) $(x := 0)(\sigma) = \sigma_1$, где $\sigma_1(y) = \sigma(y)$ при $y \neq x$, и $\sigma_1(x) = 0$.

в) $(x := y)(\sigma) = \sigma_1$, где $\sigma_1(z) = \sigma(z)$ при $z \neq x$, и $\sigma_1(x) = \sigma(y)$

г) Пусть $P = P_1; P_2$. Тогда $P(\sigma) = (P_1; P_2)(\sigma) = P_2(P_1(\sigma))$, при этом, если $P_1(\sigma) = \infty$ или $\sigma_1 = P_1(\sigma)$ и $P_2(\sigma_1) = \infty$, то и $P(\sigma) = \infty$.

д) Пусть $P = \text{если } x = y \text{ то } P_1 \text{ иначе } P_2 \text{ конец}$. Тогда

$$P(\sigma) = \begin{cases} P_1(\sigma), & \text{если } P_1(\sigma) < \infty \text{ и } \sigma(x) = \sigma(y) \\ P_2(\sigma), & \text{если } P_2(\sigma) < \infty \text{ и } \sigma(x) \neq \sigma(y) \\ \infty & \text{в остальных случаях} \end{cases}$$

е) Пусть $P = \text{если } x < y \text{ то } P_1 \text{ иначе } P_2 \text{ конец}$. Тогда

$$P(\sigma) = \begin{cases} P_1(\sigma), & \text{если } P_1(\sigma) < \infty \text{ и } \sigma(x) < \sigma(y) \\ P_2(\sigma), & \text{если } P_2(\sigma) < \infty \text{ и } \sigma(x) \geq \sigma(y) \\ \infty & \text{в остальных случаях} \end{cases}$$

ж) Пусть $\Pi = \text{пока } x = y \text{ делай } \Pi_1 \text{ все}$. Тогда при $\sigma(x) \neq \sigma(y)$ $\Pi(\sigma) = \sigma$, а при $\sigma(x) = \sigma(y)$ $\Pi(\sigma)$ — это первое такое состояние σ_m в последовательности состояний $\sigma_0 = \sigma, \sigma_1 = \Pi(\sigma_0), \dots, \sigma_{i+1} = \Pi(\sigma_i), \dots$, что при $i \leq t$ все состояния σ_i определены, при $i < t$ имеет место $\sigma_i(x) = \sigma_i(y)$, и $\sigma_m(x) \neq \sigma_m(y)$.

з) Семантику для цикла с условием $x < y$ определите самостоятельно (см. задачу 2.1).

Пусть Π — программа, Var_Π — множество ее переменных. Выделим среди эти переменных некоторое подмножество *входных* переменных x_1, \dots, x_n и одну *результатирующую* (выходную) переменную y (она может быть одной из входных). Переменные из Var_Π , не являющиеся входными будем называть *вспомогательными*.

Определение 2.6. Программа Π с входными переменными x_1, \dots, x_n и результирующей переменной y вычисляет частичную функцию $F : \mathbf{N}^n \rightarrow \mathbf{N}$, если для любого набора значений аргументов a_1, \dots, a_n ($a_i \in \mathbf{N}$), она переводит начальное состояние σ , в котором $\sigma(x_i) = a_i$ при $1 \leq i \leq n$ и $\sigma(z) = 0$ при $z \in Var \setminus \{x_1, \dots, x_n\}$, в состояние $\sigma_1 = \Pi(\sigma)$ тогда и только тогда, когда $(a_1, \dots, a_n) \in \delta_F$ и $F(a_1, \dots, a_n) = \sigma_1(y)$.

Функцию, вычисляемую программой Π с входными переменными x_1, \dots, x_n в (результирующей) переменной y обозначим $\Phi_{\Pi,y}(x_1, \dots, x_n)$.

Арифметическая функция $F(x_1, \dots, x_n)$ программно вычислима, если она вычислима некоторой программой Π в некоторой переменной y при некотором разбиении переменных Var_Π на входные: x_1, \dots, x_n и вспомогательные.

Заметим, что в нашем языке нет понятия процедуры (подпрограммы). Для сокращения записи мы будем иногда использовать имя одной ранее написанной программы внутри текста другой: $\Pi = \dots \Pi_1 \dots$. Такая запись будет означать текстовую (in-line) подстановку текста (кода) программы Π_1 в соответствующее место программы Π . Подчеркнем, что при этом переменные Π_1 не переименовываются и программист сам должен заботиться о правильной инициализации переменных из Var_{Π_1} . Например, если Π использует отдельно написанную программу Π_+ из приведенного ниже примера 2.3 для сложения переменных a и b и получения результата в t , то “безопасный” и корректный способ сделать это может выглядеть так:

$\Pi : \dots; x' := x; y' := y; z' := z; x := a; y := b; \Pi_+; t := x; x := x'; y := y'; z := z'; \dots$,

т.е. вначале сохраняются текущие значения переменных x, y, z , используемых в Π_+ , затем входным переменным x и y присваиваются нужные значения a и b и вызывается Π_+ , ее результат передается в t , затем восстанавливаются значения x, y, z .

Рассмотрим несколько примеров программ.

Пример 2.1.

$\Pi_0 : x := 0$

Ясно, что $\Phi_{\Pi_0,x}(x)$ тождественно равна 0.

Пример 2.2.

$$\Pi_1 : x := x + 1$$

А здесь $\Phi_{\Pi_1, x}(x) = x + 1$ для любого x .

Пример 2.3.

$$\Pi_+ : \text{пока } z < y \text{ делай } z := z + 1; x := x + 1 \text{ все}$$

Зафиксируем входные переменные x, y и выходную переменную x (z - рабочая переменная). Легко показать, что $\Phi_{\Pi_+, x}(x, y) = x + y$.

Действительно, при $y = 0$ тело цикла не выполняется и выход равен $x = x + 0$. При $y \geq 1$ тело цикла выполняется y раз и при каждом его выполнении x увеличивается на 1.

Пример 2.4.

$$\Pi_4(n, i) :$$

$$x_1 := x_1; \dots; x_n := x_n; x_1 := x_i$$

$\Pi_4(n, i)$ вычисляет в x_1 функцию выбора i -го аргумента: $\Phi_{\Pi_4(n, i), x_1}(x_1, \dots, x_n) = x_i$.

Пример 2.5.

$$\Pi_5(n) :$$

$$x_1 := x_1; \dots; x_n := x_n; \text{пока } x_1 = x_1 \text{ делай } x_1 := x_1 \text{ все}$$

Нетрудно понять, что $\Pi_5(n)$ вычисляет нигде не определенную функцию от n переменных: $\Phi_{\Pi_5(n), x_1}(x_1, \dots, x_n) = \infty$.

2.1 Задачи

Задача 2.1. Определите (по аналогии с п. (ж)) определения 2.5 семантику для программ вида

$\Pi = \text{пока } x < y \text{ делай } \Pi_1 \text{ все}.$

Задача 2.2. Построить структурированные программы, вычисляющие в z следующие функции, и доказать их корректность:

а) $f_{\times}(x, y) = x * y$;

б) $f_{fact}(x) = x!$;

в) $f_{-1}(x) = x \dot{-} 1$, где $0 \dot{-} 1 = 0$ и $(x + 1) \dot{-} 1 = x$;

г) $f_{-}(x, y) = x \dot{-} y$, где $x \dot{-} y = x - y$, если $x \geq y$ и $x \dot{-} y = 0$, если $x < y$;

д) $f_{sqr}(x) = \lfloor \sqrt{x} \rfloor$;

е) $f_{exp}(x) = 2^x$;

ж) $f_{log}(x) = \lfloor \log_2 x \rfloor$;

з) $f_{/}(x, y) = \lfloor x/y \rfloor$.

Задача 2.3. Пусть Π — структурированная программа и $|\text{Var}_\Pi| = m$. Из определений следует, что при различной фиксации входных переменных и выходной переменной она может вычислять различные функции.

а) Каково максимальное число функций от $n \leq m$ переменных, которое может вычислять Π ? Сколько всего разных функций может вычислить Π ?

б) Постройте программу $\Pi(m, n)$, которая вычисляет максимальное число различных функций от $n \leq m$ переменных?

в) Постройте программу $\Pi(m)$ с $|\text{Var}_\Pi| = m$, которая для каждого $n \leq m$ вычисляет максимальное число различных функций от n переменных?

Задача 2.4. Построить структурированные программы, вычисляющие в z следующие функции:

$$a) \quad f_1(x, y) = \begin{cases} x^2 y^3, & \text{если } x < y \\ \lfloor (x+y)/2 \rfloor, & \text{в противном случае} \end{cases}$$

$$б) \quad f_2(x, y) = \begin{cases} 3^y, & \text{если } \log_2(x+1) \geq y \\ |x-y|, & \text{в противном случае} \end{cases}$$

$$в) \quad f_3(x, y) = \begin{cases} \lfloor x * y/2 \rfloor, & \text{если } \log_2 x \leq y+2 \\ x^{\lfloor y/2 \rfloor}, & \text{в противном случае} \end{cases}$$

$$г) \quad p(x) = \begin{cases} 1, & \text{если } x \text{ — простое число} \\ 0, & \text{в противном случае} \end{cases}$$

3 Частично рекурсивные функции

В этом разделе мы изучим алгебраический подход к определению класса вычислимых функций. Каждая вычислимая функция будет получаться из некоторых простейших очевидно вычислимых базисных функций с помощью некоторых операций, вычислимость которых также не вызывает сомнения. Операция, которая дала название этому подходу — *рекурсия* — это способ задания функции путем определения каждого ее значения в терминах ранее определенных ее значений и других уже определенных функций.

3.1 Основные определения

Мы будем рассматривать частичные арифметические функции $f^n(x_1, \dots, x_n) : \mathbf{N}^n \rightarrow \mathbf{N}$. Здесь верхний индекс n у имени функции f обозначает число ее аргументов (“арность”). Если арность ясна из контекста или несущественна, то этот индекс будем опускать. Определим вначале три оператора, позволяющих по одним функциям получать другие.

Определение 3.1. Суперпозиция. Пусть F^m и f_1^n, \dots, f_m^n – арифметические функции. Скажем, что функция G^n получена из F^m, f_1^n, \dots, f_m^n с помощью оператора суперпозиции (обозначение: $G^n = [F^m; f_1^n, \dots, f_m^n]$), если для всех наборов аргументов (x_1, \dots, x_n)

$$G^n(x_1, \dots, x_n) = F^m(f_1^n(x_1, \dots, x_n), \dots, f_m^n(x_1, \dots, x_n)).$$

При этом для каждого набора аргументов (a_1, \dots, a_n) функция $G^n(a_1, \dots, a_n) < \infty$ (т.е. определена), если определены все значения $f_1^n(a_1, \dots, a_n) = b_1, \dots, f_m^n(a_1, \dots, a_n) = b_m$ и $F^m(b_1, \dots, b_m) < \infty$.

Определение 3.2. Примитивная рекурсия. Скажем, что функция $F^{n+1}(x_1, \dots, x_n, y)$ получена с помощью оператора примитивной рекурсии из функций $g^n(x_1, \dots, x_n)$ и $h^{n+2}(x_1, \dots, x_n, y, z)$, если она может быть задана схемой примитивной рекурсии

$$\begin{cases} F^{n+1}(x_1, \dots, x_n, 0) = g^n(x_1, \dots, x_n) \\ F^{n+1}(x_1, \dots, x_n, y+1) = h^{n+2}(x_1, \dots, x_n, y, F^{n+1}(x_1, \dots, x_n, y)) \end{cases}$$

В этом случае будем писать $F^{n+1} = R(g^n, h^{n+2})$.

При этом $F(a_1, \dots, a_n, 0) < \infty \iff g^n(a_1, \dots, a_n) < \infty$ и для каждого b $F(a_1, \dots, a_n, b+1) < \infty \iff F(a_1, \dots, a_n, b) = c < \infty$ и $h^{n+2}(a_1, \dots, a_n, b, c) < \infty$.

В случае, когда $n = 0$, т.е. F зависит от одного аргумента y , а аргументов x_1, \dots, x_n нет, схема примитивной рекурсии принимает вид

$$\begin{cases} F^1(0) = a \\ F^1(y+1) = h^2(y, F^1(y)), \end{cases}$$

где $a \in \mathbf{N}$.

Заметим, что если исходные функции в операторах суперпозиции и примитивной рекурсии всюду определены, то и результирующие функции также всюду определены. Следующий оператор позволяет задавать не всюду определенные, т.е. *частичные*, функции.

Определение 3.3. Минимизация. Скажем, что функция $F^n(x_1, \dots, x_n)$ получена с помощью оператора минимизации (μ -оператора) из функции $g^{n+1}(x_1, \dots, x_n, y)$, если $F^n(x_1, \dots, x_n)$ определена и равна y тогда и только тогда, когда все значения $g^{n+1}(x_1, \dots, x_n, 0), \dots, g^{n+1}(x_1, \dots, x_n, y-1)$ определены и не равны 0, а $g^{n+1}(x_1, \dots, x_n, y) = 0$. В этом случае будем писать

$$F^n(x_1, \dots, x_n) = \mu y[g^{n+1}(x_1, \dots, x_n, y) = 0].$$

Определение 3.4. Простейшие функции. *Функция называется простейшей, если она является одной из следующих функций:*

- а) $o^1(x) = 0$ - тождественный нуль;
- б) $s^1(x) = x + 1$ - следующее число (плюс один);
- в) функции выбора аргумента $I_m^n(x_1, \dots, x_n) = x_m$ ($1 \leq m \leq n$).

Заметим, что все простейшие функции вычислимы в интуитивном смысле. Кроме того, операторы суперпозиции, примитивной рекурсии и минимизации также вычислимы: понятны алгоритмы, по которым из программ для исходных функций можно получить программы для результирующих. Следующее определение вводит интересный нас класс частично рекурсивных функций и его важные подклассы.

Определение 3.5. Частично рекурсивные функции. *Функция f называется частично рекурсивной функцией (ч.р.ф.), если она является одной из простейших функций или может получиться из них с помощью конечного числа применений операторов суперпозиции, примитивной рекурсии и минимизации, т.е. существует последовательность функций $f_1, f_2, \dots, f_n = f$, каждая из которых является либо простейшей, либо получена из предыдущих с помощью одного из указанных операторов. Указанная последовательность функций называется частично рекурсивным описанием функции f .*

Функция f называется общерекурсивной функцией (о.р.ф.), если она частично рекурсивна и всюду определена.

Функция f называется примитивно рекурсивной функцией (п.р.ф.), если она частично рекурсивна и для нее существует определение, использующее лишь операторы суперпозиции и примитивной рекурсии.

Нетрудно проверить, что каждая примитивно рекурсивная функция всюду определена, т.е. является общерекурсивной (обратное, вообще говоря, неверно).

3.1.1 Примеры

Приведем некоторые примеры частично рекурсивных функций.

Пример 3.1. Постоянные функции.

Пусть $f^n(x_1, \dots, x_n) = k$ для всех наборов аргументов (x_1, \dots, x_n) и числа $k \in \mathbf{N}$. Тогда

$$f^n = \underbrace{[s^1; [s^1; \dots [s^1; [o^1; I_1^n]]] \dots]}_{k \text{ раз}}.$$

Пример 3.2. Сложение: $+^2(x, y) = x + y$.

Функция сложения определяется следующей примитивной рекурсией.

$$\begin{cases} x + 0 = x = I_1^1(x) \\ x + (y + 1) = (x + y) + 1 = [s^1; (x + y)] = [s^1; I_3^3(x, y, x + y)] \end{cases}$$

Следовательно, $+^2 = R(I_1^1, [s^1; I_3^3])$.

Пример 3.3. Умножение: $\times^2(x, y) = x \times y$.

Используя сложение, умножение можно задать следующей примитивной рекурсией:

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

Следовательно, $\times^2 = R(o_1^1, [+; I_3^3, I_1^1])$.

Пример 3.4. Минус 1: $\dot{-}1(0) = 0$, $\dot{-}1(x + 1) = x$.

Нетрудно проверить, что $\dot{-}1 = R(0, I_1^2)$.

Пример 3.5. Вычитание : $x \dot{-} y = x - y$, если $x \geq y$ и $x \dot{-} y = 0$, если $x < y$.

Вычитание определяется следующей примитивно рекурсивной схемой:

$$\begin{cases} x \dot{-} 0 = x = I_1^1(x) \\ x \dot{-} (y + 1) = (x \dot{-} y) \dot{-} 1 = [\dot{-}1^1; I_3^3(x, y, x \dot{-} y)] \end{cases}$$

Следовательно, $\dot{-}^2 = R(I_1^1, [\dot{-}1^1; I_3^3])$.

Пример 3.6. Предикаты равенства и неравенства нулю:

$$sg(x) = \begin{cases} 0, & \text{если } x = 0 \\ 1, & \text{если } x \neq 0 \end{cases} \quad \overline{sg}(x) = \begin{cases} 1, & \text{если } x = 0 \\ 0, & \text{если } x \neq 0 \end{cases}$$

Примитивная рекурсивность этих функций следует из равенств $sg = R(0, [s^1; [o^1; I_1^2]])$ и $\overline{sg}(x) = 1 \dot{-} sg(x)$.

Пример 3.7. Модуль разности: $|x - y| = (x \dot{-} y) + (y \dot{-} x)$.

Пример 3.8. $rm(x, y) = \text{остаток от деления } y \text{ на } x$ (при $x = 0$ положим $rm(0, y) = y$).

Заметим, что

$$rm(x, y + 1) = \begin{cases} 0, & \text{если } rm(x, y) + 1 = x \\ rm(x, y) + 1, & \text{если } rm(x, y) + 1 \neq x \end{cases}$$

Тогда функцию $rm(x, y)$ можно задать примитивно рекурсивной схемой

$$\begin{cases} rm(x, 0) = 0, \\ rm(x, y + 1) = (rm(x, y) + 1)sg(|x - (rm(x, y) + 1)|) \end{cases}$$

Правую часть второго равенства легко представить как функцию $g(x, y, rm(x, y))$, полученную с помощью суперпозиции уже построенных примитивно рекурсивных функций.

Пример 3.9. *Нигде не определенная функция $\omega(x)$.*

Эта функция может быть задана, например, соотношением $\omega(x) = \mu y [s^1(y) + x = 0]$.

Отметим, что все функции в примерах 2.1 - 2.8 являются примитивно рекурсивными.

3.2 Программная вычислимость рекурсивных функций

В этом параграфе рассмотрим соотношение между программно вычислимыми и частично рекурсивными функциями. Справедлива следующая

Теорема 3.1. *Каждая частично рекурсивная функция программно вычислима.*

Доказательство индукцией по определению ч.р.ф.

Базис: программная вычислимость простейших функций была установлена в примерах 1.1, 1.2 и 1.4.

Индукционный шаг: покажем программную вычислимость операторов суперпозиции, примитивной рекурсии и минимизации.

Суперпозиция. Пусть F^m и f_1^n, \dots, f_m^n – арифметические функции, вычисляемые программами Π, Π_1, \dots, Π_m так, что $\Phi_{\Pi, x_1}(x_1, \dots, x_m) = F^m(x_1, \dots, x_m)$, и $\Phi_{\Pi_i, x_1}(x_1, \dots, x_n) = f_i^n(x_1, \dots, x_n)$ при $i = 1, \dots, m$. Пусть переменные $y_1, \dots, y_m, z_1, \dots, z_n$ не используются в программах Π, Π_1, \dots, Π_m . Кроме того, пусть все вспомогательные переменные этих программ – это w_1, \dots, w_r . Рассмотрим следующую программу P :

$z_1 := x_1; \dots; z_n := x_n;$

$\Pi_1; y_1 := x_1; x_1 := z_1; \dots; x_n := z_n;$

$w_1 := 0; \dots; w_r := 0;$

$\Pi_2; y_2 := x_1; x_1 := z_1; \dots; x_n := z_n;$
 $w_1 := 0; \dots; w_r := 0;$
 \dots
 $\Pi_m; y_m := x_1; x_1 := y_1; \dots; x_m := y_m;$
 $w_1 := 0; \dots; w_r := 0;$
 Π

В качестве входных переменных зафиксируем x_1, \dots, x_n , а выходной — x_1 . Пусть в исходном состоянии $x_1 = a_1, \dots, x_n = a_n$. Тогда в первой строке эти значения сохраняются в переменных z_1, \dots, z_n , которые своих значений далее не меняют. Поэтому для каждого $i = 1, \dots, m-1$ после выполнения фрагмента

$\Pi_i; y_i := x_1; x_1 := z_1; \dots; x_n := z_n;$
 $w_1 := 0; \dots; w_r := 0;$

значением переменной y_i является $f_i^n(a_1, \dots, a_n)$, $x_1 = a_1, \dots, x_n = a_n$, а значения всех вспомогательных переменных равны 0. Тогда после выполнения

$\Pi_m; y_m := x_1; x_1 := y_1; \dots; x_m := y_m;$

значением каждого x_i также является $f_i^n(a_1, \dots, a_n)$, а после выполнения Π значение x_1 равно $\Phi_{P, x_1}(x_1, \dots, x_m) = F^m(f_1^n(a_1, \dots, a_n), \dots, f_m^n(a_1, \dots, a_n))$. Таким образом, $\Phi_{P, x_1} = [F; f_1, \dots, f_n]$.

Примитивная рекурсия. Рассмотрим для простоты случай $n = 1$. Пусть функция $F^2(x_1, y)$ получена с помощью оператора примитивной рекурсии из функций $g^1(x_1)$ и $h^3(x_1, y, z)$, т.е. $F^2 = R(g^1, h^3)$. Предположим, что существуют программы Π_1 и Π_2 , вычисляющие функции g^1 и h^3 так, что $\Phi_{\Pi_1, x_1}(x_1) = g^1(x_1)$ и $\Phi_{\Pi_2, x_1}(x_1, y, z) = h^3(x_1, y, z)$. Пусть вспомогательные переменные Π_2 — это z_1, \dots, z_m и они не встречаются в Π_1 , а переменные u_1, y_1 и v не используются в программах Π_1 и Π_2 . Рассмотрим программу P :

$u_1 := x_1; y_1 := y; v := 0; \Pi_1;$

пока $v < y_1$ **делай** $z := x_1; x_1 := u_1; y := v; \Pi_2; z_1 := 0; \dots; z_m := 0; v := v + 1$ **все**

В качестве входных переменных P возьмем x_1 и y , а выходной — x_1 .

Рассмотрим работу P на исходном состоянии σ , в котором $\sigma(x_1) = a, \sigma(y) = b$. При $b = 0$ цикл не выполняется и в результирующем состоянии $\sigma_1 = P(\sigma)$ имеем $\sigma_1(x_1) = \Pi_1(\sigma)(x_1) = g^1(a) = F^2(a, 0)$. При $b > 0$ цикл будет выполняться b раз, так как в его теле v всякий раз увеличивается на 1, а значение $y_1 = b$ и не меняется. Перед первым выполнением Π_2 все ее рабочие переменные z_i равны 0, $x_1 = a, y = 0, z = F^2(a, 0)$, а после ее выполнения $x_1 = h^3(a, 0, F(a, 0)) = F(a, 1)$. Предположим теперь по индукции, что перед $(i+1)$ -ым выполнением Π_2 все ее рабочие переменные z_i равны 0, $x_1 = a, y = i$ и $z = F^2(a, i)$. После этого выполнения $x_1 = h^3(a, i, z) = h^3(a, i, F(a, i)) = F(a, i+1)$. Тогда присваивания $z_1 := 0; \dots; z_m := 0; v := v + 1$ после Π_2 и $z := x_1; x_1 := u_1; y := v$; перед ее следующим выполнением установят значения переменных Π_2 так, что все ее рабочие переменные z_i равны 0, $x_1 = a, y = i + 1$ и $z = F^2(a, i + 1)$. Следовательно, после b -го выполнения тела цикла $x_1 =$

$$h^3(a, b-1, F(a, b-1)) = F(a, b).$$

Минимизация. Предположим, что функция $F^n(x_1, \dots, x_n)$ получена с помощью оператора минимизации (μ -оператора) из функции $g^{n+1}(x_1, \dots, x_n, y)$, т.е.

$$F^n(x_1, \dots, x_n) = \mu y[g^{n+1}(x_1, \dots, x_n, y) = 0].$$

Пусть программа Π_1 вычисляет g^{n+1} , так что $\Phi_{\Pi_1, x_1}(x_1, \dots, x_n, y) = g^{n+1}(x_1, \dots, x_n, y)$, и пусть рабочие переменные Π_1 — это z_1, \dots, z_m . Зафиксируем переменные $x'_1, \dots, x'_n, y', u, z$, не входящие в Var_{Π_1} . Рассмотрим следующую программу Π :

$$x'_1 := x_1; \dots; x'_n := x_n; z := 0; u := 0; u := u + 1; y' := 0;$$

пока $z < u$ **делай**

$$x_1 := x'_1; \dots; x_n := x'_n; y := y'; \Pi_1; u := x_1; z_1 := 0; \dots; z_m := 0;$$

если $u = z$ **то** $x_1 := y'$ **иначе** $y' := y' + 1$ **конец**

все

Рассмотрим работу Π на входных значениях $x_i = a_i$ ($i = 1, \dots, n$). В первой строке они сохраняются в переменных x'_i , которые нигде в Π не изменяются, z получает значение 0, которое тоже не меняется по ходу вычисления, а u вначале получает значение 1. Поэтому условие цикла после первой строки выполнено и он хоть один раз выполняется. Докажем, что для каждого $i \geq 1$, $(i+1)$ -ая итерация цикла выполняется тогда и только тогда, когда $g(a_1, \dots, a_n, 0) = b_1 > 0, \dots, g(a_1, \dots, a_n, i-1) = b_{i-1} > 0$, Π останавливается после $(i+1)$ -ой итерации цикла с результатом $x_1 = i$ тогда и только тогда, когда $g(a_1, \dots, a_n, i) = 0$. При этом перед выполнением Π_1 входные переменные x_1, \dots, x_n, y имеют значения a_1, \dots, a_n, i , соответственно, $y' = i$, а все рабочие переменные z_j ($j = 1, \dots, m$) равны 0.

Действительно, предположив это условие, получим, что после очередного выполнения фрагмента

$$\Pi_1; z_1 := 0; \dots; z_m := 0; u := x_1;$$

значение $u = x_1 = g(a_1, \dots, a_n, i)$, а рабочие переменные восстанавливают нулевые значения. Если $g(a_1, \dots, a_n, i) = 0$, то $u = z$ и в условном операторе x_1 получает значение $y' = i$. После этого условие цикла нарушено и Π завершает работу с выходным значением $x_1 = i = F(a_1, \dots, a_n)$. Если же $g(a_1, \dots, a_n, i) > 0$, то $u > z$ и в условном операторе y' увеличивает значение до $(i+1)$. Тогда условие цикла выполнено и перед $(i+2)$ -ым выполнением Π_1 ее входные переменные x_1, \dots, x_n, y имеют значения $a_1, \dots, a_n, i+1$, соответственно, $y' = i+1$, а все рабочие переменные равны 0.

Из доказанного утверждения непосредственно следует, что $\Phi_{\Pi, x_1}(a_1, \dots, a_n) = \mu y[g(a_1, \dots, a_n, y) = 0]$. \square

Имеет место и утверждение, обратное теореме 3.1, которое мы приводим здесь без доказательства.

Теорема 3.2. Каждая программно вычислимая функция является частично рекурсивной.

3.3 Леммы о рекурсивных функциях

В этом параграфе мы установим примитивную (частичную) рекурсивность некоторых важных классов функций - таблиц и нумераций, и расширим возможности определения функций с помощью разбора случаев и взаимной рекурсии.

Лемма 3.1. Рекурсивность табличных функций. *Пусть всюду определенная функция $f(x)$ на всех аргументах, кроме конечного числа, равна некоторой константе c (такую функцию назовем табличной). Тогда она является примитивно рекурсивной.*

Доказательство Пусть для функции f из условия леммы $n_f = \max\{x | f(x) \neq c\}$. Доказательство проведем индукцией по n_f .

При $n_f = 0$ функция f является постоянной и поэтому примитивно рекурсивной (пример 3.1 на стр. 9).

Предположим что все табличные функции g со значением $n_g \leq k$ примитивно рекурсивны и пусть $n_f = k + 1$ и $f(0) = a$. Определим табличную функцию $f'(x) = f(x + 1)$. Ясно, что $n_{f'} = k$, и по предположению индукции f' примитивно рекурсивна. Легко проверить, что тогда f задается следующей схемой:

$$\begin{cases} f(0) = a, \\ f(x + 1) = g(x, f(x)) = I_1^2(f'(x), f(x)) \end{cases}$$

и, следовательно, также примитивно рекурсивна. \square

Покажем замкнутость класса ч.р.ф. (п.р.ф.) относительно операций суммирования и произведения.

Лемма 3.2. Суммирование и произведение. *Пусть функция $f(x_1, \dots, x_n, y)$ является частично (примитивно) рекурсивной. Тогда и функции F^{n+1} и G^{n+1} , заданные следующими равенствами*

$$F(x_1, \dots, x_n, y) = \sum_{i=0}^y f(x_1, \dots, x_n, i),$$

$$G(x_1, \dots, x_n, y) = \prod_{i=0}^y f(x_1, \dots, x_n, i),$$

являются частично (примитивно) рекурсивными.

Доказательство Действительно, эти функции задаются следующими примитивно рекурсивными схемами:

$$\begin{cases} F(x_1, \dots, x_n, 0) = f(x_1, \dots, x_n, 0) \\ F(x_1, \dots, x_n, y + 1) = F(x_1, \dots, x_n, y) + f(x_1, \dots, x_n, y + 1) \end{cases}$$

$$\begin{cases} G(x_1, \dots, x_n, 0) = f(x_1, \dots, 0) \\ G(x_1, \dots, x_n, y+1) = G(x_1, \dots, x_n, y) \times f(x_1, \dots, x_n, y+1) \end{cases}$$

□

Приведем примеры использования леммы 3.2.

Пример 3.10. $\max_deg_div(x, y)$ = максимальная степень x , на которую нацело делится y .

Нетрудно проверить, что эта функция задается соотношением

$$\max_deg_div(x, y) = \sum_{i=0}^y \overline{sg}(rm(exp(x, i), y)) - 1$$

и, следовательно, является примитивно рекурсивной.

Пример 3.11. Ограниченная минимизация. Пусть примитивно рекурсивная функция $g(x, y)$ такова, что для каждого x найдется $y \leq x$, для которого $g(x, y) = 0$. Положим $F(x) = \mu y[g(x, y) = 0]$.

Тогда, по определению, $F(x)$ является частично рекурсивной функцией. Покажем, что, на самом деле, она примитивно рекурсивна. Действительно, определим $h(x, y) = \prod_{i=0}^y sg(g(x, i))$. По лемме 3.2 эта функция примитивно рекурсивна. Пусть для данного x $y_0 = \mu y[g(x, y) = 0]$. Тогда при $i < y_0$ имеем $h(x, i) = 1$, а при $i \geq y_0$ $h(x, i) = 0$. Поэтому искомая функция F задается равенством $F(x) = \sum_{i=0}^y h(x, i)$ и также является примитивно рекурсивной.

Лемма 3.3. Кусочное задание или разбор случаев. Пусть $h_1(x_1, \dots, x_n), \dots, h_k(x_1, \dots, x_n)$ — произвольные ч.р.ф., а всюду определенные ч.р.ф. $f_1(x_1, \dots, x_n), \dots, f_k(x_1, \dots, x_n)$ таковы, что на любом наборе аргументов (a_1, \dots, a_n) одна и только одна из этих функций равна 0. Тогда функция $g(x_1, \dots, x_n)$, определенная соотношениями:

$$g(x_1, \dots, x_n) = \begin{cases} h_1(x_1, \dots, x_n), & \text{если } f_1(x_1, \dots, x_n) = 0 \\ h_2(x_1, \dots, x_n), & \text{если } f_2(x_1, \dots, x_n) = 0 \\ \dots & \dots \\ h_k(x_1, \dots, x_n), & \text{если } f_k(x_1, \dots, x_n) = 0 \end{cases}$$

является частично рекурсивной.

Доказательство Действительно, g^n можно представить как сумму k произведений:

$$g(x_1, \dots, x_n) = h_1(x_1, \dots, x_n) \overline{sg}(f_1(x_1, \dots, x_n)) + \dots + h_k(x_1, \dots, x_n) \overline{sg}(f_k(x_1, \dots, x_n)).$$

Следующий класс функций, который нас будет интересовать, — это функции для однозначной нумерации пар и n -ок целых чисел и обратные им. Определим для

любой пары чисел (x, y) ее номер $c_2(x, y) = 2^x(2y + 1) - 1$. Например, $c(0, 0) = 0, c(1, 0) = 1, c(0, 1) = 2, c(1, 1) = 5, c(2, 1) = 19$. Из единственности разложения чисел на простые множители следует, что функция $c_2 : \mathbf{N}^2 \rightarrow \mathbf{N}$ взаимно однозначно нумерует пары целых чисел. Определим теперь обратные функции:

$c_{21}(z) =$ максимальная степень 2, на которую делится $z + 1$ и

$c_{22}(z) = [(\text{максимальное нечетное число, на которое делится } z + 1) - 1] / 2$.

Из этих определений непосредственно следует, что для любого z выполнено равенство $c_2(c_{21}(z), c_{22}(z)) = z$.

Определим теперь по индукции функции c_n нумерации n -ок чисел при $n > 2$ и обратные им координатные функции c_{ni} ($1 \leq i \leq n$):

$c_n(x_1, x_2, x_3, \dots, x_n) = c_{n-1}(c_2(x_1, x_2), x_3, \dots, x_n),$

$c_{n1}(z) = c_{21}(c_{(n-1)1}(z)),$

$c_{n2}(z) = c_{22}(c_{(n-1)1}(z)),$

$c_{n(i+2)}(z) = c_{(n-1)(i+1)}(z) \quad (i = 1, \dots, n-2).$

Из этих определений также непосредственно следует, что для любого z имеет место равенство $c_n(c_{n1}(z), c_{n2}(z), \dots, c_{nn}(z)) = z$. (Проверьте это свойство индукцией по n .)

Лемма 3.4. Рекурсивность нумерационных функций. Для любых $n \geq 2$ и $1 \leq i \leq n$ все определенные выше функции c_n и c_{ni} являются примитивно рекурсивными.

Доказательство Примитивная рекурсивность $c_2(x, y)$ устанавливается непосредственно (см. задачу 3.1(a)). Функция $c_{21}(z)$ задается равенством $c_{21}(z) = \max_deg_div(2, z + 1)$ и является примитивно рекурсивной (это показано в примере 3.10 на стр. 15). Для функции $c_{22}(z)$ справедливо определение $c_{22}(z) = \max_deg_div(2, \max_deg_div(2^{c_{21}(z)}, z + 1) - 1)$ (здесь мы используем примитивную рекурсивность функции целочисленного деления $div(x, y)$ из задачи 3.1(e)). Примитивная рекурсивность остальных нумерационных функций следует по индукции из их определений (см. задачу 3.10).□

В следующей лемме обобщается оператор примитивной рекурсии.

Лемма 3.5. (совместная рекурсия) Предположим, что функции $\alpha_i^n(x_1, \dots, x_n)$ ($1 \leq i \leq k$) и функции $\beta_i^{n+k+1}(x_1, \dots, x_n, y, y_1, \dots, y_k)$ ($1 \leq i \leq k$) примитивно рекурсивны. Тогда функции $f_1^{n+1}(x_1, \dots, x_n, y), \dots, f_k^{n+1}(x_1, \dots, x_n, y)$, определяемые следующей совместной рекурсией

$$\begin{cases} f_i(x_1, \dots, x_n, 0) = \alpha_i(x_1, \dots, x_n) & (1 \leq i \leq k) \\ f_i(x_1, \dots, x_n, y + 1) = \beta_i(x_1, \dots, x_n, y, f_1(x_1, \dots, x_n, y), \dots, f_k(x_1, \dots, x_n, y)) & (1 \leq i \leq k) \end{cases}$$

также являются примитивно рекурсивными.

Доказательство Обозначим (для простоты) через \bar{x} набор переменных x_1, \dots, x_n . Определим следующие примитивно рекурсивные функции: $g^n(\bar{x}) = c_k(\alpha_1(\bar{x}), \dots, \alpha_k(\bar{x}))$,

$\hat{\beta}_i(\bar{x}, y, z) = \beta_i(\bar{x}, y, c_{k1}(z), \dots, c_{kk}(z)), 1 \leq i \leq k$, и положим

$$\begin{cases} F^{n+1}(\bar{x}, 0) = g^n(\bar{x}) \\ F^{n+1}(\bar{x}, y+1) = c_k(\hat{\beta}_1(\bar{x}, y, F(\bar{x}, y)), \dots, \hat{\beta}_k(\bar{x}, y, F(\bar{x}, y))) \end{cases}$$

Функция F^{n+1} получена примитивной рекурсией из примитивно рекурсивных функций и, следовательно, сама примитивно рекурсивна. Справедливость леммы теперь следует из того, что для всякого $i \in [1, k]$ $f_i(\bar{x}, y) = c_k^i(F^{n+1}(\bar{x}, y))$.

3.4 Задачи

Задача 3.1. Показать, что следующие функции являются частично (примитивно) рекурсивными.

- а) $\exp(x, y) = x^y$;
- б) $\text{fact}(x) = x!$;
- в) $\min(x, y) = \text{наименьшее из } x \text{ и } y$;
- г) $\max(x, y) = \text{наибольшее из } x \text{ и } y$;
- д) $\text{div}(x, y) = \text{частное от деления } y \text{ на } x \text{ (пусть } \text{div}(0, y) = y)$.
- е) предикаты равенства и неравенства:

$$\text{eq}(x, y) = \begin{cases} 1, & \text{если } x = y \\ 0, & \text{если } x \neq y \end{cases} \quad \overline{\text{eq}}(x) = \begin{cases} 0, & \text{если } x = y \\ 1, & \text{если } x \neq y \end{cases}$$

ж) $f(x) = 2^{2^x}$.

Задача 3.2. Докажите, что если $f(x_1, \dots, x_n)$ является ч.р.ф. (н.р.ф.), то и функция $g(x_1, \dots, x_n) = f(x_{i_1}, \dots, x_{i_n})$ является ч.р.ф. (н.р.ф.) для любой перестановки

(i_1, \dots, i_n) чисел $1, 2, \dots, n$.

Задача 3.3. Оператор сдвига.

Пусть $g(x_1, \dots, x_n)$ — частично (примитивно) рекурсивная функция, а a и $b > 0$ — числа из \mathbf{N} . Тогда и функция

$$f(x_1, \dots, x_n) = \begin{cases} a, & \text{если } x_n \leq b \\ g(x_1, \dots, x_n - b), & \text{если } x_n > b \end{cases}$$

является частично (примитивно) рекурсивной.

Задача 3.4. Показать, что следующие функции являются примитивно рекурсивными.

- а) $\text{rt}(n, x) = \lfloor \sqrt[n]{x} \rfloor$ — корень n -ой степени из x (целая часть).

- б) $\log(i, x) = \lfloor \log_i x \rfloor$ (пусть при $i \in \{0, 1\}$ или $x = 0$ $\log(i, x) = 0$).
- в) $p(x) = 1$, если x — простое число, и $p(x) = 0$, если x составное.
- г) $pn(k)$ — k -ое простое число в порядке возрастания ($pn(0) = 0, pn(1) = 2, pn(2) = 3, pn(3) = 5 \dots$).
- д) $t(x)$ — число различных делителей числа x ($t(0) = 0$).
- е) $d(n, m, i)$ — i -ый знак в m -ичном разложении числа n , т.е. если $n = \sum_0^\infty a_i m^i$, то $d(n, m, i) = a_i$.

Задача 3.5. Пусть $F(x)$ задана соотношениями $F(0) = 1$, $F(1) = 2$, $F(x+2) = F(x-1) + F(x)$ (элементы последовательности $F(x)$ называются числами Фибоначчи). Покажите, что функция F^1 примитивно рекурсивна.
(Указание: покажите сначала, что функция $g(x) = 2^{F(x)} 3^{F(x+1)}$ примитивно рекурсивна.)

Задача 3.6. Докажите, что если значения общерекурсивной функции $f(x)$ изменить на конечном множестве, то получившаяся функция $f'(x)$ также будет общерекурсивной.

Задача 3.7. Доказать, что из функции $o(x) = 0$ и из функций выбора $I_m^n(x_1, \dots, x_n) = x_m$ с помощью суперпозиции и примитивной рекурсии нельзя получить функцию $s(x) = x + 1$ и функцию $d(x) = 2 * x$.

Задача 3.8. Пусть $g(x_1, \dots, x_n, y)$ — примитивно рекурсивна. Доказать, что функция

$$f(x_1, \dots, x_n, y, z) = \begin{cases} \sum_{i=0}^z g(x_1, \dots, x_n, y + i), & \text{при } y \leq z \\ 0, & \text{при } y > z \end{cases}$$

примитивно рекурсивна.

Задача 3.9. Доказать, что если функции $f(x_1, \dots, x_n, y)$, $g(x_1, \dots, x_n, y)$ и $h(x_1, \dots, x_n, y)$ частично рекурсивны, то и функция $F(x_1, \dots, x_n) = \min\{y \mid f(x_1, \dots, x_n, y) = 0 \text{ или } g(x_1, \dots, x_n, y) > h(x_1, \dots, x_n, y)\}$ является частично рекурсивной.

Задача 3.10. Докажите, что определенные выше функция нумерации n -ок $c_n(x_1, \dots, x_n)$ и обратные ей функции выбора i -го элемента набора $c_{ni}(z)$ ($1 \leq i \leq n$) являются примитивно рекурсивными.

Задача 3.11. Предположим, что все пары (x, y) натуральных чисел упорядочены по возрастанию суммы $(x+y)$, а внутри группы пар с одинаковой суммой — по возрастанию x -координаты. Этот порядок выглядит так: $(0, 0), (0, 1), (1, 0), (0, 2), (1, 1), (2, 0), \dots, (0, x+y), (1, x+y-1), \dots, (x, y), \dots, (x+y, 0), \dots$. Пусть $d(x, y)$ — это номер пары (x, y) в этом порядке (будем считать, что пара $(0, 0)$ имеет номер 0). Тогда функция d^2 однозначно нумерует все пары.

- а) Докажите, что $d(x, y) = \frac{(x+y)(x+y+1)}{2} + x - 1$.
 б) Найдите обратные функции $d_1(z)$ и $d_2(z)$ такие, что $d_1(d(x, y)) = x$, $d_2(d(x, y)) = y$ и, следовательно, $d(d_1(z), d_2(z)) = z$.

4 Машины Тьюринга

4.1 Основные определения

Рассматриваемая в этом разделе модель алгоритмов была предложена английским математиком Тьюрингом в 1937г. еще до создания современных компьютеров¹. Он исходил из общей идеи моделирования работы вычислителя, оперирующего в соответствии с некоторым строгим предписанием. В машине Тьюринга расчленение процесса вычисления на элементарные шаги доведено в известном смысле до предела. Элементарным действием является замена одного символа в ячейке на другой и перемещение к соседней ячейке. При таком подходе процесс вычисления значительно удлиняется, но зато логическая структура процесса сильно упрощается и приобретает удобный для теоретического исследования вид.

Машина Тьюринга (м.т.) состоит из неограниченной в обе стороны ленты, разбитой на ячейки, по которой передвигается головка машины. Такая “бесконечность” ленты является математической абстракцией, отражающей потенциальную неограниченность памяти вычислителя. Разумеется, в каждом завершающемся вычислении используется только конечная часть этой памяти — конечное число ячеек. В каждой ячейке ленты записан один символ из конечного *внешнего алфавита* машины $\Sigma = \{a_0, a_1, \dots, a_m\}$. Головка машины представляет конечный автомат, который в каждый момент времени находится в одном из *внутренних состояний* $Q = \{q_0, q_1, \dots, q_n\}$. На каждом шаге головка в зависимости от своего внутреннего состояния и символа в ячейке, которую она наблюдает, изменяет свое внутреннее состояние и содержимое наблюдаемой ячейки и может сдвинуться на одну ячейку вправо или влево, либо остаться на месте.

Дадим более формальное определение.

Определение 4.1. *Машина Тьюринга — это система вида*

$$M = \langle Q, \Sigma, P, q_0, q_f \rangle,$$

включающая следующие компоненты:

$Q = \{q_0, q_1, \dots, q_n\}$ - *внутренний алфавит (алфавит состояний);*

$\Sigma = \{a_0, a_1, \dots, a_{m-1}\}$ - *внешний алфавит (алфавит ленты);*

P - *программа машины, в которой для каждой пары $q_i \in Q \setminus \{q_f\}$, $a_j \in \Sigma$ имеется*

¹Аналогичная модель вычислений была примерно в то же время определена Е. Постом. Поэтому иногда такие машины называют машинами Тьюринга-Поста.

(одна!) команда вида

$$q_i a_j \rightarrow q_k a_l C, \quad q_k \in Q, a_l \in \Sigma,$$

$C \in \{L, P, H\}$ задает сдвиг головки вправо, влево или на месте;

$q_0 \in Q$ — начальное состояние;

$q_f \in Q$ — заключительное состояние.

Выделим в алфавите Σ специальный пустой символ $a_0 = \Lambda$ и будем считать, что во всех ячейках ленты, кроме конечного их числа, в начальный и во все последующие моменты находится пустой символ.

Будем говорить, что некоторый символ *стирается*, если он заменяется на пустой. Два слова из Σ^* будем считать равными, если они совпадают после отбрасывания всех пустых символов слева и справа. Например, $\Lambda \Lambda ab \Lambda c \Lambda = \Lambda ab \Lambda c = ab \Lambda c$, но $ab \Lambda c \neq abc$.

Как и для конечных автоматов, программу P можно задавать с помощью таблицы. размера $n \times m$, строки которой соответствуют состояниям из Q , а столбцы — символам из входного алфавита Σ , в которой на пересечении строки q_i и столбца a_j стоит тройка $q_k a_l C$ — правая часть команды $q_i a_j \rightarrow q_k a_l C$.

Определение 4.2. Назовем конфигурацией м.т. \mathcal{M} в некоторый момент времени слово $K = w_a q_i a_j w_n$, где $w_a \in \Sigma^*$ — слово на ленте левее текущего положения головки, q_i — внутреннее состояние в данный момент, a_j — символ, обозреваемый головкой, $w_n \in \Sigma^*$ — слово на ленте правее текущего положения головки.

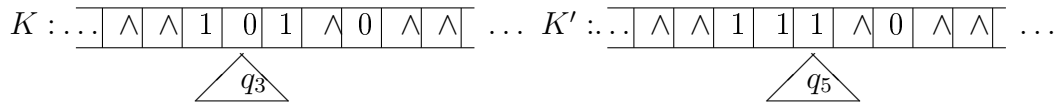
Будем считать, что слово $w_a a_j w_n$ содержит все значащие символы на ленте. Поэтому, с точностью до описанного выше равенства слов, конфигурация определена однозначно. В частности, если $w_a = \varepsilon$, т.е. пусто, то левее положения головки все ячейки пусты, а если $w_n = \varepsilon$, то правее положения головки все ячейки пусты.

Начальная конфигурация — это конфигурация вида $q_0 w$, т.е. в начальный момент времени головка в состоянии q_0 обозревает первый символ входного слова w . *Заключительная* конфигурация — это конфигурация вида $w_1 q_f w_2$, в которой машина находится в заключительном состоянии q_f .

Определение 4.3. Скажем, что конфигурация $K = w_1 q_i a_j w_2$ м.т. \mathcal{M} за один шаг (такт) переходит в конфигурацию $K' = w'_1 q_k a_{j'} w'_2$ ($K \vdash_{\mathcal{M}} K'$), если в программе имеется команда $q_i a_j \rightarrow q_k a_l C$ и при этом,

- (а) если $C=H$, то $w'_1 = w_1, w'_2 = w_2$ и $a_{j'} = a_l$;
- (б) если $C=L$, то $w_1 = w'_1 a$, $a_{j'} = a$, $w'_2 = a_l w_2$ (если $w_1 = \varepsilon$, то $w'_1 = \varepsilon$ и $a_{j'} = \Lambda$);
- (в) если $C=P$, то $w_2 = a w'_2$, $a_{j'} = a$, $w'_1 = w_1 a_l$ (если $w_2 = \varepsilon$, то $w'_2 = \varepsilon$ и $a_{j'} = \Lambda$).

Как обычно, через $\vdash_{\mathcal{M}}^*$ обозначим рефлексивное и транзитивное замыкание отношения $\vdash_{\mathcal{M}}$, а $K \vdash_{\mathcal{M}}^n K'$ будет означать, что конфигурация K за n шагов переходит в K' . (Если из контекста ясно, о какой машине идет речь, то индекс \mathcal{M} будем опускать).

Рис. 1: Выполнение команды $q_3 0 \rightarrow q_5 1\Pi$ **Пример 4.1.**

Например, ситуации, представленной на рис. 1 слева соответствует конфигурация $K = 1q_3 01 \wedge 0$. Предположим, что программа P содержит команду $q_3 0 \rightarrow q_5 1\Pi$. Тогда после выполнения этой команды K перейдет за один шаг в конфигурацию $K' = 11q_5 1 \wedge 0$, показанную на этом рисунке справа. Следовательно, $K \vdash K'$.

Определение 4.4. *Вычисление м.т. \mathcal{M} на входе w - это конечная или бесконечная последовательность конфигураций $K_0 \vdash K_1 \vdash \dots \vdash K_t \vdash K_{t+1} \dots$ такая, что $K_0 = q_0 w$ - начальная конфигурация. Эта последовательность конечна, когда ее последняя конфигурация $K_n = v_1 q_f v_2$ - заключительная. В этом случае вычисление назовем результативным, а слово $v = v_1 v_2$ - его результатом на входе w (всегда будем предполагать, что v не содержит пустых символов слева и справа).*

Определение 4.5. *Скажем, что м.т. \mathcal{M} вычисляет частичную словарную функцию $f : \Sigma^* \rightarrow \Sigma^*$, если для каждого слова w из области определения f существует результативное вычисление \mathcal{M} с результатом $f(w)$, а если $f(w)$ не определена ($f(w) = \infty$), то вычисление \mathcal{M} на входе w бесконечно.*

Скажем, что две м.т. \mathcal{M}_1 и \mathcal{M}_2 эквивалентны, если они вычисляют одинаковые функции.

Далее мы будем также рассматривать вычисления *арифметических* функций, т.е. функций с натуральными аргументами, принимающих натуральные значения. Для представления натуральных чисел используем *унарное* кодирование: число n будет представляться как слово из n палочек $|^n$, а последовательные аргументы будем отделять $*$.

Определение 4.6. *Скажем, что м.т. \mathcal{M} вычисляет частичную арифметическую функцию $f : \mathbf{N}^k \rightarrow \mathbf{N}$, если для любого набора чисел (x_1, x_2, \dots, x_k) , на котором f определена, существует результативное вычисление \mathcal{M} на входе $|^{x_1} * |^{x_2} * \dots * |^{x_k}$ с результатом $|^{f(x_1, x_2, \dots, x_k)}$, а если $f(x_1, x_2, \dots, x_k) = \infty$, то вычисление \mathcal{M} на соответствующем входе бесконечно.*

Аналогичное определение можно дать и для других способов кодирования чисел (двоичного, десятичного и др.). Ниже мы покажем, что класс вычислимых функций не зависит от выбора одного из таких кодирований.

4.2 Тьюрингово программирование

В этом разделе мы приведем примеры вычислений на машинах Тьюринга и рассмотрим некоторые общие приемы, позволяющие комбинировать программы различных м. Т. для получения более сложных вычислений. Будем считать, что ячейки ленты м.Т. занумерованы от $-\infty$ до $+\infty$, причем в начальной конфигурации головка находится в 1-ой ячейке:

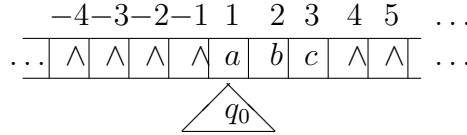


Рис. 2: Нумерация ячеек ленты машины Тьюринга

Пример 4.2. Функция $f(x) = x + 1$

а) *Унарное кодирование.*

Пусть м.Т. $\mathcal{M}_1 = \langle \{q_0, q_f\}, \{\wedge, |\}, P_1, q_0, q_f \rangle$, где $P_1 : q_0| \rightarrow q_0|П$, $q_0\wedge \rightarrow q_f|Н$. Ясно, что м.Т. \mathcal{M}_1 проходит по массиву палочек слева направо и записывает в первой пустой ячейке новую $|$.

б) *Бинарное кодирование.*

Пусть м.Т. $\mathcal{M}'_1 = \langle \{q_0, q_1, q_f\}, \{\wedge, 0, 1\}, P'_1, q_0, q_f \rangle$, где $P'_1 :$

$$q_00 \rightarrow q_00П, \quad q_01 \rightarrow q_01П, \quad q_0\wedge \rightarrow q_1\wedge Л,$$

$$q_10 \rightarrow q_f1Н, \quad q_11 \rightarrow q_10П, \quad q_1\wedge \rightarrow q_f1Н.$$

Нетрудно видеть, что эта машина в состоянии q_0 находит младший разряд двоичного входа, затем в состоянии q_1 , идя справа налево, заменяет единицы на нули до тех пор, пока не находит 0 (или \wedge) и заменяет его на 1. Следовательно, м.Т. \mathcal{M}'_1 вычисляет функцию $f(x) = x + 1$.

Пример 4.3. Копирование Рассмотрим функцию копирования (дублирования) слов в алфавите $\Sigma : \text{copy}(x) = x * x$ (мы предполагаем, что $*$ $\notin \Sigma$).

Для ее реализации используем один из типичных приемов Тьюрингова программирования - *расширение алфавита*. Пусть $\Sigma' = \{a' | a \in \Sigma\}$ и $\Sigma_1 = \Sigma \cup \Sigma'$. М.Т. \mathcal{M}_2 , копирующая вход, работает следующим образом:

1) отмечает 1-ый символ входа, идет направо, ставит $*$ после входа и возвращается в начало:

$$q_0a \rightarrow q_1a'П \ (a \in \Sigma); \quad q_1a \rightarrow q_1aП \ (a \in \Sigma \setminus \{\wedge\}); \quad q_1\wedge \rightarrow q_2*Л;$$

- $q_2a \rightarrow q_2aL$ ($a \in \Sigma \cup \{*\}$); $q_2a' \rightarrow q_2a'P$ ($a \in \Sigma$);
- 2) в состоянии q_a движется направо и записывает a в первую свободную ячейку:
 $q_ab \rightarrow q_abP$ ($b \in \Sigma \cup \{*\}$); $q_a\Lambda \rightarrow q_4aL$;
- 3) возвращается в отмеченную ячейку и передвигает метку $'$ на одну ячейку вправо, снова переходя в состояние q_2 :
 $q_4a \rightarrow q_4aL$ ($a \in \Sigma \cup \{*\}$); $q_4a' \rightarrow q_5aP$ ($a \in \Sigma$); $q_5a \rightarrow q_2a'H$ ($a \in \Sigma$);
- 4) увидев символ $*$ в состоянии q_5 , останавливается:
 $q_5* \rightarrow q_f * H$.

Из этого описания непосредственно следует, что $q_0x \vdash^* xq_f * x$ для любого $x \in \Sigma^*$.

4.2.1 Стандартная заключительная конфигурация

Назовем заключительную конфигурацию *стандартной*, если в ней головка наблюдает первый значащий символ результата, который находится в 1-ой ячейке (т.е. в той же ячейке, где начиналось входное слово).

Лемма 4.1. *Для всякой м.Т. \mathcal{M} можно построить эквивалентную м.Т. \mathcal{M}' , у которой все заключительные конфигурации стандартны.*

Доказательство. Пусть $\mathcal{M} = \langle Q, \Sigma, P, q_0, q_f \rangle$. Определим по ней м.Т. $\mathcal{M}' = \langle Q', \Sigma', P', q'_0, q'_f \rangle$, которая удовлетворяет требованиям леммы. Положим $\Sigma' = \Sigma \cup \{a' | a \in \Sigma\} \cup \{\#\}$, где $\#$ - новый символ. \mathcal{M}' работает следующим образом.

- 1) Отмечает символ в первой ячейке штрихом и переходит в начальное состояние \mathcal{M} : $q'_0a \rightarrow q_0a'H$.
- 2) Далее работает как \mathcal{M} , но сохраняет штрих в первой ячейке и вместо пустого символа Λ записывает $\#$. Для этого для каждой команды $q_ia_j \rightarrow q_ka_lC$ из P в P' добавляется ее дубликат $q_ia'_j \rightarrow q_ka'_lC$, в правых частях команд символ Λ всюду заменяется на $\#$ и для каждой команды вида $q_i\Lambda \rightarrow q_ka_lC$ в P' добавляется команда $q_i\# \rightarrow q_ka_lC$. После завершения этого этапа все посещенные в процессе работы головкой \mathcal{M} ячейки составляют непрерывный отрезок, не содержащий пустых символов.
- 3) Далее \mathcal{M}' стирает ненужные символы $\#$ слева и справа от блока ячеек, содержащего первую ячейку и все ячейки с символами результата, и переходит в одну из трех следующих конфигураций:

$$K_1 = q_a\#\#\dots\#w\Lambda, \quad K_2 = \Lambda q_nw\#\dots\#\#', \quad K_3 = \Lambda q_nw_1a'w_2\Lambda,$$

где w - результат работы \mathcal{M} (с заменой символов Λ внутри w на $\#$) и $w_1aw_2 = w$.

- 4) Сдвигает в нужном направлении результат, совмещая его начало с ячейкой, помеченной штрихом, заменяет все $\#$ внутри w на Λ , снимает штрих в 1-ой ячейке и останавливается. Например, для K_1 это достигается с помощью следующих команд (мы предполагаем, что ни одно из используемых ниже состояний $q_a, q_1, p, p_1, p_2, p_3, p^a, p^{a'}$ ($a \in$

$\Sigma \cup \{\#\}$) не входит в Q) :

а) поиск левого конца w : $q_a a \rightarrow q_a a \Pi$ ($a \in \{\#', \#\}$); $q_a a \rightarrow q_1 a' \Pi$ ($a \in \Sigma$) (отметили первый символ w), $q_a \wedge \rightarrow p_3 \wedge \mathcal{L}$; (результат пуст);

б) поиск правого конца w : $q_1 a \rightarrow q_1 a \Pi$ ($a \in \Sigma \cup \{\#\}$), $q_1 \wedge \rightarrow p \wedge \mathcal{L}$ (в состоянии p наблюдает последний символ w);

в) сдвиг результата на 1 ячейку влево: $p a \rightarrow p^a \wedge \mathcal{L}$; $p^a b \rightarrow p^b a \mathcal{L}$; $p^a b' \rightarrow p^{b'} a \Pi$; $p^{b'} \# \rightarrow p_1 b' \Pi$;

г) возврат к правому концу и переход к следующему сдвигу: $p_1 a \rightarrow p_1 a \Pi$ ($a \neq \wedge$); $p_1 \wedge \rightarrow p \wedge \mathcal{L}$;

д) при сдвиге до 1-ой ячейки замена символов $\#$ на \wedge и удаление штриха:

$p^{a'} \# \rightarrow p_2 a' \mathcal{H}$; $p_2 b \rightarrow p_2 b \Pi$; $p_2 \wedge \rightarrow p_3 \wedge \mathcal{L}$; $p_3 \# \rightarrow p_3 \wedge \mathcal{L}$; $p_3 b \rightarrow p_3 b \Pi$ ($b \neq \#, b \neq a'$); $p_3 a' \rightarrow q_f a \mathcal{H}$.

Из построения непосредственно следует, что м.Т. \mathcal{M}' удовлетворяет требованиям леммы. \square

4.2.2 Односторонние машины Тьюринга

Машина Тьюринга \mathcal{M} называется *односторонней*, если в процессе вычисления ее головка никогда не сдвигается левее начальной ячейки (т.е. всегда находится в ячейках с положительными номерами).

Лемма 4.2. Для всякой м.Т. \mathcal{M} можно построить эквивалентную одностороннюю м.Т. \mathcal{M}' .

Доказательство. Пусть $\mathcal{M} = \langle Q, \Sigma, P, q_0, q_f \rangle$. Будем считать (используя лемму 1), что \mathcal{M} завершает работу в стандартных конфигурациях. Требуемая м.Т. $\mathcal{M}' = \langle Q', \Sigma', P', q'_0, q'_f \rangle$ будет моделировать работу \mathcal{M} , используя “многоэтажную” ленту. Содержимое ячеек на 1-ом (нижнем) этаже будет на каждом такте совпадать с содержимым тех же ячеек \mathcal{M} , на 2-ом этаже будет копироваться содержимое левой полуленты: на нем в i -ой ячейке \mathcal{M}' будет тот же символ, что и в $-i$ -ой ячейке \mathcal{M} . Кроме того, на 3-ем этаже в 1-ой ячейке будет стоять отмечающий ее символ $\#$. Таким образом, $\Sigma' = \Sigma \times \Sigma \times \{\wedge, \#\} \cup \Sigma$. Работа \mathcal{M}' будет происходить следующим образом.

1) На первом этапе отмечается 1-я ячейка и содержимое входа переписывается на 1-ый этаж трехэтажной ленты:

$$q'_0 a \rightarrow p_1 \left(\frac{\#}{a} \right) \Pi \quad (a \in \Sigma); \quad p_1 a \rightarrow p_1 \left(\frac{\wedge}{a} \right) \Pi \quad (a \in \Sigma \setminus \{\wedge\}); \quad p_1 \wedge \rightarrow p_2 \wedge \mathcal{L};$$

$$p_2 \left(\frac{\wedge}{a} \right) \rightarrow p_2 \left(\frac{\wedge}{a} \right) \mathcal{L} \quad (a \in \Sigma); \quad p_2 \left(\frac{\#}{a} \right) \rightarrow q_0 \left(\frac{\#}{a} \right) \mathcal{L} \quad (a \in \Sigma).$$

2) Затем \mathcal{M}' моделирует работу \mathcal{M} , используя для работы на 2-ом этаже дубликаты состояний (со штрихами) и команды со сдвигами в обратном направлении. Для команды $q a \rightarrow r b \mathcal{C}$ из P в P' поместим команды:

$$q \left(\frac{\wedge}{c} \right) \rightarrow r \left(\frac{\wedge}{b} \right) C \quad \text{и} \quad q' \left(\frac{\wedge}{c} \right) \rightarrow r' \left(\frac{\wedge}{b} \right) C', \quad \text{где} \quad C' = \begin{cases} \Pi & \text{при} \quad C = L \\ L & \text{при} \quad C = \Pi \\ H & \text{при} \quad C = H \end{cases}$$

для всех $c \in \Sigma$. Кроме того, для $a = \wedge$ сохраним и старые команды для работы на впервые посещаемых ячейках:

$$q \wedge \rightarrow r \left(\frac{\wedge}{b} \right) C \quad \text{и} \quad q' \wedge \rightarrow r' \left(\frac{\wedge}{b} \right) C'.$$

Сдвиги \mathcal{M} из 1-ой ячейки налево в -1-ю и обратно моделируются переходом с одного этажа на другой в 1-ой ячейке \mathcal{M}' :

$$q \left(\frac{\#}{a} \right) \rightarrow \bar{r} \left(\frac{\#}{b} \right) \bar{C}, \quad \text{где при} \quad C = L \quad \bar{r} = r', \quad \bar{C} = H, \quad \text{а при} \quad C \neq L \quad \bar{r} = r,$$

$$\bar{C} = C$$

$$q' \left(\frac{\#}{a} \right) \rightarrow \bar{r} \left(\frac{\#}{b} \right) \bar{C}, \quad \text{где при} \quad C = \Pi \quad \bar{r} = r, \quad \bar{C} = H, \quad \text{а при} \quad C \neq \Pi \quad \bar{r} = r',$$

$$\bar{C} = C.$$

3) После завершения моделирования \mathcal{M} результат записан в начальных ячейках на 1-ом этаже. \mathcal{M}' переводит его в первоначальный алфавит Σ .

$$q_f \left(\frac{c}{a} \right) \rightarrow q_f a \Pi \quad (a \in \Sigma \setminus \{\wedge\}, \quad c \in \{\wedge, \#\}), \quad q_f \wedge \rightarrow q'_f \wedge H.$$

Проверка правильности работы м.Т. \mathcal{M}' предоставляется читателю (см. задачу 4.4).
□.

4.2.3 Последовательная и параллельная композиции машин Тьюринга

Используя возможность моделирования произвольной м.Т. на м.Т. со стандартными заключительными конфигурациями, легко установить справедливость следующей леммы о *последовательной композиции* машин Тьюринга.

Лемма 4.3. (Последовательная композиция) Пусть м.Т. \mathcal{M}_1 вычисляет функцию $f(x)$, а м.Т. \mathcal{M}_2 - функцию $g(x)$. Тогда существует м.Т. \mathcal{M} , вычисляющая функцию $h(x) = f(g(x))$.

Доказательство. Действительно, пусть $\mathcal{M}_1 = \langle Q_1, \Sigma_1, P_1, q_0^1, q_f^1 \rangle$, а $\mathcal{M}_2 = \langle Q_2, \Sigma_2, P_2, q_0^2, q_f^2 \rangle$. Используя лемму 4.1, будем считать, что у \mathcal{M}_2 заключительные конфигурации стандартны. Тогда легко проверить, что функция h вычисляется следующей м.Т. $\mathcal{M} = \langle Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, P, q_0^2, q_f^1 \rangle$, где $P = P_1 \cup P_2 \cup \{q_f^2 a \rightarrow q_0^1 a \mid a \in \Sigma_2\}$. □

Покажем, что работу двух м.Т. можно комбинировать так, чтобы в заключительной конфигурации содержались результаты работы каждой из них над независимыми входами.

Лемма 4.4. (Параллельная композиция) Пусть м.Т. \mathcal{M}_1 вычисляет функцию $f(x)$, а м.Т. \mathcal{M}_2 - функцию $g(y)$ и символ $*$ не входит в алфавит м.Т. \mathcal{M}_1 . Тогда существует м.Т. \mathcal{M} , которая по любому входу вида $x*y$ выдает результат $f(x)*g(y)$, т.е. вычисляет функцию $H(x*y) = f(x)*g(y)$.

Доказательство. Пусть $\mathcal{M}_1 = \langle Q_1, \Sigma_1, P_1, q_0^1, q_f^1 \rangle$, и $\mathcal{M}_2 = \langle Q_2, \Sigma_2, P_2, q_0^2, q_f^2 \rangle$ - м.Т. их условия. Не ограничивая общности, будем считать, что эти машины односторонние (по Лемме 2). Определим теперь м.Т. $\mathcal{M} = \langle Q_1 \cup Q_2 \cup Q', \Sigma_1 \cup \Sigma_2 \cup \{*\}, P_1 \cup P_2 \cup P', p_0, p_f \rangle$, которая работает следующим образом.

- 1) Начав в конфигурации $(p_0 x * y)$, находит 1-ый символ y и переходит в конфигурацию $(x * q_0^2 y)$.
- 2) Работая как \mathcal{M}_2 , вычисляет $g(y)$ и переходит при этом в конфигурацию $(x * q_f^2 g(y))$.
- 3) Переписывает $*x$ после $g(y)$ и переходит в конфигурацию $g(y) * q_0^1 x$.
- 4) Работая как \mathcal{M}_1 , вычисляет $f(x)$ и переходит при этом в конфигурацию $(g(y) * q_f^1 f(x))$.
- 5) Меняет $g(y)$ и $f(x)$ местами и останавливается.

Корректность этапов 2 и 4 следует из односторонности \mathcal{M}_2 и \mathcal{M}_1 , а реализация этапов 1, 3 и 5 достаточно очевидна (см. задачу 4.6). \square

Построенную в этой лемме м.Т. \mathcal{M} , полученную в результате параллельной композиции \mathcal{M}_1 и \mathcal{M}_2 , будем обозначать как $\text{par}_*(\mathcal{M}_1, \mathcal{M}_2)$. Здесь индекс $*$ указывает символ, которым отделяются аргументы \mathcal{M}_1 и \mathcal{M}_2 на ленте \mathcal{M} . Этот символ может быть любым символом, не входящим в алфавит машины \mathcal{M}_1 . Например, $\text{par}_\#(\mathcal{M}_1, \mathcal{M}_2)$ будет обозначать параллельную композицию машин \mathcal{M}_1 и \mathcal{M}_2 , в которой их аргументы отделены символом $\#$.

Конструкцию параллельной композиции можно обобщить на произвольное конечное число машин Тьюринга.

Следствие. Пусть $\mathcal{M}_1, \dots, \mathcal{M}_m$ - машины Тьюринга, вычисляющие функции f_1, \dots, f_m , соответственно. Пусть символ $*$ не входит в алфавиты этих машин. Тогда существует м.Т. \mathcal{M} , перерабатывающая любой вход вида $x_1 * x_2 * \dots * x_m$ ($x_i \in \Sigma_i^*, i = 1, \dots, m$) в выход $f_1(x_1) * f_2(x_2) * \dots * f_m(x_m)$.

Действительно, в качестве \mathcal{M} можно взять м.Т., определяемую выражением $\text{par}_*(\mathcal{M}_1, \text{par}_*(\mathcal{M}_2, \text{par}_*(\mathcal{M}_3, \dots \text{par}_*(\mathcal{M}_{m-1}, \mathcal{M}_m) \dots)))$.

Будем обозначать эту машину Тьюринга как $\text{par}_*(\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_m)$.

4.2.4 Ветвление (условный оператор)

Машину Тьюринга Φ будем называть *распознающей*, если для некоторого алфавита Σ и каждого входа $x \in \Sigma^*$, на котором Φ останавливается, ее результат $\Phi(x) \in \{0, 1\}$, т.е. Φ вычисляет некоторую двужначную функцию (возможно частичную) на словах из Σ .

Лемма 4.5. Пусть Φ - распознающая м.Т., м.Т. \mathcal{M}_1 вычисляет функцию $f(x)$, а м.Т. \mathcal{M}_2 - функцию $g(x)$. Тогда существует м.Т. \mathcal{M} , вычисляющая функцию

$$h(x) = \begin{cases} f(x) & \text{при } \Phi(x) = 0 \\ g(x) & \text{при } \Phi(x) = 1 \end{cases}$$

Доказательство. Требуемая м.Т. \mathcal{M} вначале копирует вход x и получает на ленте слово $x*x$, затем вычисляет параллельную композицию функций $\Phi(x)$ и $g(x) = x$ и переходит в конфигурацию $p\Phi(x)*x$. Выбор между f и g происходит по следующим командам:

$$p0 \rightarrow p_0 \wedge \Pi; \quad p1 \rightarrow p_1 \wedge \Pi; \quad p_0* \rightarrow q_0^1 \wedge \Pi; \quad p_1* \rightarrow q_0^2 \wedge \Pi$$

Кроме того, обеспечим переход в новое заключительное состояние:

$$q_f^1 a \rightarrow q_f a \ H; \quad q_f^2 a \rightarrow q_f a \ H \quad (a \in \Sigma).$$

□

Таким образом, мы реализовали в терминах машин Тьюринга обычный в языках программирования оператор ветвления:

$$\text{if } \Phi(x) = 0 \text{ then } f(x) \text{ else } g(x) \text{ endif}.$$

4.2.5 Повторение (цикл)

Используя конструкцию для ветвления легко реализовать в терминах машин Тьюринга и оператор цикла.

Лемма 4.6. Пусть Φ - распознающая м.Т., а м.Т. \mathcal{M}_1 вычисляет функцию $f(x)$. Тогда существует м.Т. \mathcal{N} , вычисляющая функцию, задаваемую выражением:

$$g(x) = \text{'while } \Phi(x) = 0 \text{ do } x := f(x) \text{ enddo'}$$

Доказательство. Действительно, пусть м.Т. \mathcal{M}_2 - вычисляет тождественную функцию $g(x) = x$. Построим по м.Т. Φ, \mathcal{M}_1 и \mathcal{M}_2 м.Т. \mathcal{M} , реализующую ветвление как в лемме 4.5. Тогда искомая м.Т. \mathcal{N} получается из \mathcal{M} заменой команд $q_f^1 a \rightarrow q_f a \ H$ ($a \in \Sigma$) на соответствующие команды $q_f^1 a \rightarrow q_0 a \ H$ ($a \in \Sigma$), обеспечивающие закливание. □

Реализованные выше операции над машинами Тьюринга и вычислимыми функциями позволяют получать программы новых м.Т., используя обычные конструкции языка программирования “высокого” уровня: последовательную и параллельную композицию, ветвление и цикл. Пусть $M, M_1, M_2, \dots, M_m, \Phi$ - машины Тьюринга. Последовательную композицию M_1 и M_2 будем обозначать $M_1; M_2$, параллельную композицию M_1, M_2, \dots, M_m обозначаем как $\text{par}_b(M_1, M_2, \dots, M_m)$ (здесь b — это символ, разделяющий аргументы и результаты этих машин), ветвление - **if** Φ **then** M_1 **else** M_2 **endif**, цикл - **while** Φ **do** M **enddo**.

Пример 4.4. Рассмотрим в качестве примера задачу перевода чисел из унарной системы счисления в двоичную. Пусть $f^{ub}(|^n) = n_{(2)}$ для всех $n \in \mathbb{N}$, где $n_{(2)}$ - двоичная запись числа n .

Пусть M_1 - м.Т., которая начальную конфигурацию $q_0 |^n$ переводит в конфигурацию $q_1 0 * |^n$; M_2 - м.Т., которая прибавляет 1 к двоичному числу-аргументу (см. пример 4.2 на стр. 22); M_3 - м.Т., которая вычитает 1 из унарного числа; Φ - м.Т., которая на аргументе вида $x * |^y$ выдает 0, если число $y > 0$, и выдает 1 при $y = 0$ (т.е. на аргументе $x * \wedge$); M_4 - м.Т., которая стирает $*$ в аргументе вида $x*$ и останавливается. Реализация каждой из указанных м.Т. очевидна. Теперь требуемая м.Т. M_{ub} , вычисляющая f^{ub} , получается как M_1 ; **while** Φ **do** **par** $_*$ (M_2, M_3) **enddo**; M_4 .

Действительно, после работы M_1 получаем конфигурацию $q_1 0 * |^n$. Предположим теперь по индукции, что после i ($i < n$) итераций цикла **while** получается конфигурация $q_1 i_{(2)} * |^{n-i}$. Тогда на $(i+1)$ -ой итерации цикла после параллельного применения M_2 к $i_{(2)}$ и M_3 к $|^{n-i}$ получаем конфигурацию $q_1 (i+1)_{(2)} * |^{n-i-1}$. Поэтому после n итераций получится конфигурация $q_1 n_{(2)} * \wedge$. На ней Φ выдаст 1, и цикл завершится с записью $n_{(2)} * \wedge$ на ленте, из которой M_4 сотрет $*$ и оставит требуемый результат $n_{(2)}$.

Отметим, что из приведенного примера и из задачи 4.7(а) следует, что класс вычислимых на м.Т. арифметических функций не зависит от выбора унарного или двоичного кодирования аргументов и результатов. Это же справедливо и для троичной, десятичной и других позиционных систем счисления (почему?).

4.3 Задачи

Задача 4.1. Постройте м.Т. для функции копирования, не увеличивая исходный алфавит Σ .

Задача 4.2. Постройте программу м.Т., которая выполняла бы перенос непустого слова в заданное место ленты, т.е. для любого слова $w \in (\Sigma \setminus \{\wedge\})^*$ и $n > 0$ выполняла преобразование конфигураций: $q_1 w \wedge^n \# \vdash^* \wedge^n q_2 \# w$.

Задача 4.3. Постройте программу м.Т. M' из леммы 4.1 на стр. 23 на этапах 3 и 4.

Задача 4.4. Докажите, что односторонняя м.Т. M' , построенная в лемме 4.2 на стр. 24 корректно моделирует исходную м.Т. M .

Задача 4.5. Другой, по сравнению с конструкцией леммы 4.2, подход к моделированию двухсторонней ленты на односторонней заключается в том, чтобы содержимое правой полуленты M хранить в четных ячейках M' , а содержимое левой

полупленты - в нечетных, поместив в 1-ю ячейку специальный маркер. Постройте программу, реализующую этот подход (ее достоинство - увеличение алфавита ленты всего на 1 символ).

Задача 4.6. Постройте программу м.т. М из леммы 4.4 на этапах 1, 3 и 5.

Задача 4.7. Построить программы машин Тьюринга, вычисляющих следующие функции.

- а) Перевод из двоичной системы в унарную: $f^{bu}(n_{(2)}) = |^n$.
- б) Сложение и вычитание в двоичной системе: $sum(n * m) = n + m$ и $sub(n * m) = n - m$, ($-$ совпадает с $-$ при $n \geq m$ и $n - m = 0$ при $m > n$).
- в) Умножение в двоичной системе: $mul(n * m) = n \times m$. (Реализуйте алгоритм умножения "в столбик".)
- г) Возведение в степень: $exp(n * m) = n^m$.
- д) Извлечение квадратного корня: $sqr(n) = \lfloor \sqrt{n} \rfloor$.
- е) Логарифмирование: $log(n) = \lfloor \log_2 n \rfloor$.
- ж) Деление: $div(n * m) = \lfloor \frac{n}{m} \rfloor$ ($m \neq 0$).
- з) Остаток от деления: $rest(n * m) = n \bmod m$.
- и) Функция выбора аргумента: $I_m^n(|^{x_1} * |^{x_2} * \dots * |^{x_n}) = |^{x_m}$ ($1 \leq m \leq n$).

Задача 4.8. Используя машины Тьюринга из предыдущей задачи, построить программы машин Тьюринга, вычисляющих следующие функции.

- а)
$$f_1(x, y) = \begin{cases} x^2 y^3, & \text{если } x < y \\ \lfloor (x + y)/2 \rfloor, & \text{в противном случае} \end{cases}$$
- б)
$$f_2(x, y) = \begin{cases} \lfloor \sqrt{x} \rfloor, & \text{если } x + 1 \geq y \\ 2(x + y), & \text{в противном случае} \end{cases}$$
- в)
$$f_3(x, y) = \begin{cases} \lfloor \log_2(1 + \lfloor \frac{x}{y} \rfloor) \rfloor, & \text{если } x > 2y \\ xy, & \text{в противном случае} \end{cases}$$
- г)
$$f(x, y) = \begin{cases} \lfloor \sqrt{x} \rfloor, & \text{если } 2x \geq y \\ (x \bmod y), & \text{в противном случае} \end{cases}$$

Задача 4.9. Докажите, что всякую арифметическую функцию $f(x)$, вычислимую на некоторой м. т. М, можно также вычислить на м. т. М', алфавит ленты которой содержит лишь два символа \wedge и $|$. (Указание: используйте для моделирования одного символа М блок из нескольких подряд идущих ячеек, содержащих его код в алфавите $\{\wedge, |\}$ и замените каждую команду М группой команд, обрабатывающих соответствующий блок ячеек).

Задача 4.10. Построить машину Тьюринга, определяющую по слову x в алфавите $\{1, 2\}$ симметрично ли оно, т. е. вычисляющую функцию:

$$f(x) = \begin{cases} 1, & \text{если слово } x \text{ симметрично} \\ 2, & \text{в противном случае} \end{cases}$$

Задача 4.11. Построить машину Тьюринга, сравнивающую два слова $x = x_1x_2 \dots x_n$ и $y = y_1y_2 \dots y_m$ в алфавите $\{1, 2, 3\}$ лексикографически: $x \prec y \Leftrightarrow \exists i \leq n[(x_1 = y_1) \& (x_2 = y_2) \& \dots (x_{i-1} = y_{i-1}) \& (x_i < y_i)]$ или для некоторого непустого слова x' выполнено $y = xx'$. Эта машина Тьюринга должна вычислять функцию:

$$f(x, y) = \begin{cases} 1, & \text{если } x \prec y \\ 2, & \text{если } x = y \\ 3, & \text{если } y \prec x \end{cases}$$

5 Машины Тьюринга, ч.р.ф. и структурированные программы

В этом разделе мы установим, что классы частично рекурсивных функций, функций, вычисляемых структурированными программами, и функций, вычисляемых машинами Тьюринга, совпадают. Напомним, что в теореме 3.1 на стр. 11 мы уже показали, что каждая ч.р.ф. вычислима некоторой структурированной программой.

5.1 Вычислимость частично рекурсивных функций по Тьюрингу

Теорема 5.1. Для всякой ч.р.ф. f существует м.т. \mathcal{M}_f , вычисляющая функцию f .

Доказательство. Доказательство проведем индукцией по определению частично рекурсивной функции f .

Базис. Вычислимость простейших функций машинами Тьюринга очевидна.

Индукционный шаг. Покажем, что операторы суперпозиции, примитивной рекурсии и минимизации сохраняют вычислимость по Тьюрингу. Все используемые м.т. будем предполагать односторонними со стандартными заключительными конфигурациями.

Суперпозиция. Пусть F^m и f_1^n, \dots, f_m^n - ч.р.ф., вычисляемые на м.т. $\mathcal{M}_F, \mathcal{M}_{f_1}, \dots, \mathcal{M}_{f_m}$, соответственно. Пусть функция G^n получена из них с помощью суперпозиции: $G^n = [F^m; f_1^n, \dots, f_m^n]$. Тогда м.т. \mathcal{M}_G , вычисляющая G , работает следующим образом:

- 1) m раз копирует вход $|^{x_1} * \dots * |^{x_n}$, отделяя одну копию от другой символом $\#$;
- 2) на полученном слове вида $|^{x_1} * \dots * |^{x_n} \# \dots \# |^{x_1} * \dots * |^{x_n}$ запускает параллельную

композицию машин $\mathcal{M}_{f_1}, \dots, \mathcal{M}_{f_n}$ и получает конфигурацию вида $|^{y_1} \# \dots \# |^{y_n}$, где $y_i = f_i(x_1, \dots, x_n)$ ($i \in [1, m]$).

3) заменяет все символы $\#$ на $*$;

4) затем запускает программу м.Т. \mathcal{M}_F на получившемся после этапа 3) входе вида $|^{y_1} * \dots * |^{y_n}$, и вычисляет требуемое значение $G(x_1, \dots, x_n) = F(y_1, \dots, y_m)$.

Если обозначить м.Т., выполняющую копирование на этапе (1), через Kon^m , а м.Т., выполняющую замену $\#$ на $*$ на этапе (3), через $\text{Зам}_*^\#$, то требуемую для суперпозиции м.Т. \mathcal{M}_G можно представить как

$$\text{Kon}^m; \text{par}_\#(\mathcal{M}_{f_1}, \dots, \mathcal{M}_{f_n}); \text{Зам}_*^\#; \mathcal{M}_F.$$

Примитивная рекурсия. Пусть функция $F^{n+1}(x_1, \dots, x_n, y)$ получена с помощью оператора примитивной рекурсии из функций $g^n(x_1, \dots, x_n)$ и $f^{n+2}(x_1, \dots, x_n, y, z)$, которые вычислимы на м.Т. \mathcal{M}_g и \mathcal{M}_f . Определим вспомогательные м.Т.:

- \mathcal{M}_1 , используя \mathcal{M}_g , строит по входу вида $|^{x_1} * \dots * |^{x_n} * |^y$ конфигурацию на ленте $|^y * |^{x_1} * \dots * |^{x_n} * \wedge * |^{g(x_1, \dots, x_n)}$;
- \mathcal{M}_2 , используя \mathcal{M}_f , строит по входу вида $|^y * |^{x_1} * \dots * |^{x_n} * |^u * |^z$ конфигурацию $|^y * |^{x_1} * \dots * |^{x_n} * |^{u+1} * |^{f(x_1, \dots, x_n, u, z)}$;
- \mathcal{M}_3 на входе вида $|^y * |^{x_1} * \dots * |^{x_n} * |^u * |^z$ выдает в качестве результата $|^z$;
- Φ на входе вида $|^y * |^{x_1} * \dots * |^{x_n} * |^u * |^z$ проверяет условие $y \neq u$.

Построение каждой из указанных м.Т. достаточно очевидно. Из них можно получить, используя определенные в предыдущем разделе конструкции “языка программирования” для машин Тьюринга, требуемую м.Т. \mathcal{M}_F :

$$\mathcal{M}_1; \text{while } \Phi \text{ do } \mathcal{M}_2 \text{ enddo}; \mathcal{M}_3$$

Минимизация. Пусть $f^n(x_1, \dots, x_n) = \mu y[g^{n+1}(x_1, \dots, x_n, y) = 0]$ и м.Т. \mathcal{M}_g вычисляет функцию g^{n+1} . Определим следующие вспомогательные м.Т.:

\mathcal{N}_1 приписывает аргумент 0 ко входу, т.е. вход вида $|^{x_1} * \dots * |^{x_n}$ переводит в конфигурацию на ленте $|^{x_1} * \dots * |^{x_n} * \wedge$ (напомним, что при унарном кодировании 0 соответствует пустой символ).

\mathcal{N}_2 копирует свой вход с разделителем $\#$, т.е. по любому входу w выдает $w\#w$.

Через E обозначим м.Т., которая ничего не делает.

Пусть $\mathcal{N}_3 = \text{par}_\#(E, \mathcal{M}_g)$, т.е. вход вида $|^{x_1} * \dots * |^{x_n} * |^y \# |^{x_1} * \dots * |^{x_n} * |^y$ машина \mathcal{N}_3 перерабатывает, используя \mathcal{M}_g , в $|^{x_1} * \dots * |^{x_n} * |^y \# |^z$, где $z = g(x_1, \dots, x_n, y)$

Φ на входе вида $w\#v$ проверяет непустоту v (т.е. условие $v > 0$).

$$\Phi(w\#v) = \begin{cases} 0, & \text{если } v \neq \wedge \\ 1 & \text{если } v = \wedge \end{cases}$$

Таким образом, при $v = g(x_1, \dots, x_n, y)$ машина Φ проверяет условие $g(x_1, \dots, x_n, y) \neq 0$.

\mathcal{N}_4 по входу вида $|^{x_1} * \dots * |^{x_n} * |^y \# w$ стирает $\# w$ и прибавляет к y единицу, т.е. выдает результат: $|^{x_1} * \dots * |^{x_n} * |^{y+1}$.

Наконец, \mathcal{N}_5 по входу $|^{x_1} * \dots * |^{x_n} * |^y \# w$ выдает $|^y$, стирая ненужные блоки символов.

Ясно, что каждая из перечисленных м.Т. $\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3, \mathcal{N}_4, \mathcal{N}_5$ и Φ легко реализуема. Построим теперь с их помощью следующую м.Т. \mathcal{M}_f :

$\mathcal{N}_1; \mathcal{N}_2; \mathcal{N}_3;$

while Φ **do** $\mathcal{N}_4; \mathcal{N}_2; \mathcal{N}_3$ **enddo**;

\mathcal{N}_5 .

Из этого определения непосредственно следует, что \mathcal{M}_f вычисляет функцию $f^n(x_1, \dots, x_n)$, заданную с помощью оператора минимизации. \square

5.2 Моделирование структурированных программ машинами Тьюринга

На первый взгляд могло показаться, что машины Тьюринга с их примитивными элементарными действиями являются более слабыми вычислительными моделями, чем структурированные программы. Но после того, как мы научились реализовывать с их помощью операторы “высокого уровня” — условия и циклы, уже не удивительно, что они позволяют вычислить не меньше, чем структурированные программы.

Теорема 5.2. *Всякая арифметическая функция, вычислимая некоторой структурированной программой, может быть вычислена также некоторой машиной Тьюринга.*

Доказательство. Пусть структурированная программа Π вычисляет арифметическую функцию $f(x_1, \dots, x_n)$. Не ограничивая общности, будем считать, что $Var_\Pi = \{x_1, \dots, x_n, x_{n+1}, \dots, x_m\}$ и что результирующей переменной является x_1 .

М.Т. M_Π , моделирующая Π , будет иметь m -этажную ленту с алфавитом $\Sigma = \{\wedge, |, *\} \cup \{\wedge, |\}^m$. Обозначим конфигурацию ленты M_Π , в которой на i -ом этаже, начиная с 1-ой ячейки записано слева направо k_i символов ‘|’ ($i = 1, 2, \dots, m$), а далее идут “пустышки” \wedge , как (k_1, k_2, \dots, k_m) . Тогда состоянию $\sigma : Var_\Pi \rightarrow \mathbf{N}$ программы Π будет соответствовать конфигурация ленты M_Π : $K_\sigma = (\sigma(x_1), \sigma(x_2), \dots, \sigma(x_m))$.

M_Π получается с помощью конструкций последовательной композиции, условного оператора и цикла из простых машин Тьюринга, реализующих элементарные присваивания и условия структурированных программ.

Команду $x_i := 0$ ($i = 1, \dots, m$) программы Π реализует м.Т. M_0^i , обнуляющая i -ый этаж M , т.е. переводящая любую конфигурацию $(k_1, \dots, k_{i-1}, k_i, k_{i+1}, \dots, k_m)$ в конфигурацию $(k_1, \dots, k_{i-1}, 0, k_{i+1}, \dots, k_m)$. Команду $x_i := x_i + 1$ ($i = 1, \dots, m$) программы Π реализует м.Т. M_{+1}^i , добавляющая один символ ‘|’ справа на i -ом этаже, т.е. переводящая любую конфигурацию $(k_1, \dots, k_{i-1}, k_i, k_{i+1}, \dots, k_m)$ в конфигурацию $(k_1, \dots, k_{i-1}, k_i + 1, k_{i+1}, \dots, k_m)$. Команду $x_i := x_j$ ($i, j = 1, \dots, m$) программы Π реализует м.Т. M^{ij} , переписывающая содержимое j -го этажа на i -ый,

т.е. переводящая любую конфигурацию $(k_1, \dots, k_i, \dots, k_j, \dots, k_m)$ в конфигурацию $(k_1, \dots, k_j, \dots, k_j, \dots, k_m)$.

Условие $x_i = x_j$ реализуется машиной $\Phi_{=}^{ij}$, которая, работая на конфигурации $(k_1, \dots, k_i, \dots, k_j, \dots, k_m)$ выдает 0, если $k_i = k_j$, и 1 — в противном случае.

Условие $x_i < x_j$ реализуется машиной $\Phi_{<}^{ij}$, которая, работая на конфигурации $(k_1, \dots, k_i, \dots, k_j, \dots, k_m)$ выдает 0, если $k_i < k_j$, и 1 — в противном случае.

Далее по индукции: пусть Π_1 и Π_2 — структурированные программы, для которых построены соответствующие машины Тьюринга M_{Π_1} и M_{Π_2} , а Φ — некоторое условие, реализуемое м.т. M_Φ . Тогда программа $\Pi = \Pi_1; \Pi_2$ реализуется машиной $M_\Pi = M_{\Pi_1}; M_{\Pi_2}$ программа $\Pi = \text{если } \Phi \text{ то } \Pi_1 \text{ иначе } \Pi_2 \text{ конец}$ реализуется машиной $M_\Pi = \text{if } M_\Phi \text{ then } M_{\Pi_1} \text{ else } M_{\Pi_2} \text{ endif}$, а программа $\Pi = \text{пока } \Phi \text{ делай } \Pi_1 \text{ все}$ реализуется машиной $M_\Pi = \text{while } M_\Phi \text{ do } M_{\Pi_1} \text{ enddo}$.

Используя доказанные выше свойства конструкций машин Тьюринга, нетрудно проверить по индукции следующее

Утверждение 1. Пусть м.т. M_Π реализует в соответствии с приведенными определениями структурированную программу Π . Тогда для любого состояния σ программы Π $\Pi(\sigma) = \sigma_1$ тогда и только тогда, когда м.т. M_Π , начав работу в конфигурации K_σ завершит ее в конфигурации K_{σ_1} .

Теперь для завершения доказательства теоремы достаточно взять в качестве результирующей следующую м.т.: $M = M_{start}; M_\Pi; M_{end}$, где м.т. M_{start} переводит одноэтажную начальную конфигурацию $|^{x_1} * |^{x_2} * \dots * |^{x_n}$ в m -этажную конфигурацию $(x_1, x_2, \dots, x_n, 0, \dots, 0)$, а м.т. M_{end} заключительную m -этажную конфигурацию $(x_1, 0, \dots, 0)$ переводит в одноэтажную заключительную конфигурацию $|^{x_1}$. \square

5.3 Частичная рекурсивность функций, вычислимых по Тьюрингу

В этом параграфе покажем, как можно промоделировать работу машины Тьюринга, используя частично рекурсивные определения.

Теорема 5.3. Всякая арифметическая функция, вычислимая на машинах Тьюринга, является частично рекурсивной функцией.

Доказательство этой теоремы — дополнительный материал, который можно при первом чтении опустить.

Доказательство Пусть м.т. $\mathcal{M} = \langle Q, \Sigma, P, q_0, q_f \rangle$ вычисляет функцию $f(x_1, \dots, x_n)$. Пусть также $Q = \{q_0, q_1, \dots, q_{k-1}\}$, $q_f = q_1$ и $\Sigma = \{a_0 = \wedge, a_1, \dots, a_{R-1} = |\}$. Предположим также, не ограничивая общности, что \mathcal{M} никогда не пишет пустой символ \wedge (как перестроить программу произвольной м.т., чтобы она удовлетворяла этому условию?).

Определим кодирование элементов конфигураций \mathcal{M} целыми числами. Кодом символа $a_j \in \Sigma$ будет число j , кодом состояния q_i — число i . Пусть конфигурация \mathcal{M} имеет вид $K = (w_1, q_i, a_j, w_2)$, где $w_1 = a_{i_m} a_{i_{m-1}} \dots a_{i_0}$ и $w_2 = a_{j_0} a_{j_1} \dots a_{j_p}$. Тогда w_1 и w_2 можно

рассматривать как числа в R -ичной системе счисления, читаемые в противоположных направлениях (из наших предположений следует, что $i_m \neq 0$ при $m > 0$ и $j_p \neq 0$ при $p > 0$) :

$$code_1(w_1) = \sum_{l=0}^m R^l i_l \quad code_2(w_2) = \sum_{l=0}^p R^l j_l .$$

Например, если $\Sigma = \{\wedge, *, |\}$, то для конфигурации $K = (|*, q_3, |, *|)$ имеем $code_1(w_1) = 3^0 1 + 3^1 1 + 3^2 2 = 22 = [211]_3 = 22$ и $code_2(w_2) = 3^0 1 + 3^1 2 + 3^2 2 = 19 = [221]_3 = 25$. По программе P определим следующие табличные функции, кодирующие ее команды:

$A(i, j)$ — код символа, который пишет M , когда она в состоянии q_i видит символ a_j ;

$Q(i, j)$ — код состояния, в которое переходит M , когда она в состоянии q_i видит символ a_j ;

$C(i, j)$ — код направления сдвига головки M , когда она в состоянии q_i видит символ a_j (0 - на месте, 1 - вправо, 2 - влево).

Пусть при $i \geq k$ или $j \geq R$ эти функции принимают какое-нибудь фиксированное значение (например, 0). Тогда по лемме 1 все они примитивно рекурсивны.

Определим функции, которые по кодам компонент одной конфигурации $K = (w_1, q_i, a_j, w_2)$ вычисляют коды компонент следующей конфигурации $K' = (w'_1, q_m, a_p, w'_2)$.

$$lf(code_1(w_1), i, j, code_2(w_2)) = code_1(w'_1);$$

$$rt(code_1(w_1), i, j, code_2(w_2)) = code_2(w'_2);$$

$$q(code_1(w_1), i, j, code_2(w_2)) = m;$$

$$a(code_1(w_1), i, j, code_2(w_2)) = p.$$

Покажем, что все эти функции примитивно рекурсивны. Для q это следует из того, что для любых i, j $q(l, i, j, m) = Q(i, j)$. Определения остальных трех функций зависят от сдвига. При $C(i, j) = 0$ имеем $lf(l, i, j, r) = l$, $rt(l, i, j, r) = r$, $a(l, i, j, r) = A(i, j)$.

Если $C(i, j) = 2$, то $lf(l, i, j, r) = l : R$, $rt(l, i, j, r) = rR + A(i, j)$, $a(l, i, j, r) = rm(R, l)$. Если же $C(i, j) = 1$, то $lf(l, i, j, r) = lR + A(i, j)$, $rt(l, i, j, r) = r : R$, $a(l, i, j, r) = rm(R, r)$.

Объединяя эти случаи получаем, что

$$lf(l, i, j, r) = eq(C(i, j), 0)l + eq(C(i, j), 2)(l : R) + eq(C(i, j), 1)(lR + A(i, j)).$$

Аналогичные представления справедливы и для функций $rt(l, i, j, r)$ и $a(l, i, j, r)$. Следовательно, все эти функции примитивно рекурсивны.

Пусть из данной конфигурации K через t тактов получается конфигурация K^t . Определим коды компонент K^t как функции от компонент K и t :

$$A(l, i, j, r, 0) = j, \quad Q(l, i, j, r, 0) = i, \quad Lf(l, i, j, r, 0) = l, \quad Rt(l, i, j, r, 0) = r,$$

$$A(l, i, j, r, t+1) = a(Lf(l, i, j, r, t), Q(l, i, j, r, t), A(l, i, j, r, t), Rt(l, i, j, r, t)),$$

$$Q(l, i, j, r, t+1) = q(Lf(l, i, j, r, t), Q(l, i, j, r, t), A(l, i, j, r, t), Rt(l, i, j, r, t)),$$

$$Lf(l, i, j, r, t+1) = lf(Lf(l, i, j, r, t), Q(l, i, j, r, t), A(l, i, j, r, t), Rt(l, i, j, r, t)),$$

$$Rt(l, i, j, r, t+1) = rt(Lf(l, i, j, r, t), Q(l, i, j, r, t), A(l, i, j, r, t), Rt(l, i, j, r, t)).$$

Это определение задает функции $A^{(4)}, Q^{(4)}, Lf^{(4)}, Rt^{(4)}$ с помощью совместной рекурсии.

Следовательно, по лемме 4 они примитивно рекурсивны.

Пусть м.Т. \mathcal{M} вычисляет функцию $f(x)$, (т.е. $n = 1$). Тогда для начальной конфигурации $K_x = K^0 = (\wedge, q_0, |, |^{x-1})$ $code_1(w_1) = 0$, $code(q_0) = 0$, $code(|) = R - 1$, $code_2(w_2) = (R - 1)R^{x-2} + (R - 1)R^{x-3} + \dots + (R - 1)R^0 = R^{x-1} - 1$. Положим $\bar{Q}(x, t) = Q(0, 0, R - 1, R^{x-1} - 1)$ и $\bar{R}t(x, t) = Q(0, 0, R - 1, R^{x-1} - 1)$. Тогда функция $\tau(x) = \mu t(\bar{Q}(x, t) = 1)$ задает число шагов до перехода \mathcal{M} в заключительное состояние на входе x . Эта функция, очевидно, частично рекурсивна. Тогда функция $\hat{R}t(x) = \bar{R}t(x, \tau(x))$ задает код правой части заключительной конфигурации, имеющий вид $R^{f(x)-1} - 1$. Отсюда получаем, что

$$f(x) = 1 + \log_R(\hat{R}t(x) + 1) = 1 + \log_R(Rt(0, 0, R - 1, R^{x-1}) + 1),$$

и следовательно, функция $f(x)$ частично рекурсивна.

5.4 Задачи

Задача 5.1. Докажите, что машины Тьюринга \mathcal{M}_F и \mathcal{M}_f , определенные в доказательстве теоремы 5.1 на стр. 30 для примитивной рекурсии и минимизации действительно правильно реализуют указанные операторы.

Задача 5.2. Постройте машины Тьюринга M_0^i , M_{+1}^i , M^{ij} , $\Phi_{=}^{ij}$, $\Phi_{<}^{ij}$, M_{start} и M_{end} , определенные в доказательстве теоремы 5.2 на стр. 32.

Задача 5.3. Докажите утверждение 1, сформулированное в доказательстве теоремы 5.2 на стр. 32, используя индукцию по построению программы Π и соответствующей м.Т. \mathcal{M}_Π .

Задача 5.4. В доказательстве теоремы 5.3 на стр. 33 рассмотрен случай функций от одного аргумента $f(x)$. Покажите, что теорема верна и в общем случае для функций $f(x_1, \dots, x_n)$ при любом n .

6 Тезис Тьюринга-Черча и алгоритмически неразрешимые проблемы

Мы рассмотрели три математические модели для описания алгоритмов и вычисляемых ими функций, отражающие различные аспекты и представления о работе абстрактного вычислителя. Из теорем 3.1 на стр. 11, 5.2 на стр. 32 и 5.3 на стр. 33 непосредственно получаем

Следствие. Классы функций, вычислимых с помощью структурированных программ, машин Тьюринга и частично рекурсивных описаний, совпадают.

Естественно возникает вопрос о том, насколько общим является этот результат? Верно ли, что каждый алгоритм может быть задан одним из рассмотренных способов? На эти вопросы теория алгоритмов отвечает следующей гипотезой.

Тезис Тьюринга-Черча: Всякий алгоритм может быть задан в виде соответствующей машины Тьюринга или частично рекурсивного определения, а класс

вычислимых функций совпадает с классом частично рекурсивных функций и с классом функций, вычислимых на машинах Тьюринга.

Значение этого тезиса заключается в том, что он уточняет общее неформальное определения “всякого алгоритма” и “вычислимой функции” через точные формальные понятия машины Тьюринга, частично рекурсивного определения и соответствующих им классов функций. После этого можно осмысленно ставить вопрос о существовании или несуществовании алгоритма, решающего тот или иной класс задач. Теперь этот вопрос следует понимать как вопрос о существовании или несуществовании соответствующей машины Тьюринга или (что эквивалентно) структурированной программы или частично рекурсивного определения соответствующей функции.

Можно ли доказать этот тезис как теорему? Нет, поскольку в его формулировке речь идет о неточных понятиях “всякого алгоритма” и “вычислимой функции”, которые не могут быть объектами математических рассуждений. На чем же тогда основана уверенность в справедливости тезиса Тьюринга-Черча? В первую очередь на опыте. Все известные алгоритмы, придуманные за многие века математиками могут быть заданы с помощью машин Тьюринга. Для всех многочисленных моделей алгоритмов, появившихся за последние 70 лет (некоторые из них мы упоминали в начале лекций), была доказана их равносильность машинам Тьюринга. В качестве доводов в пользу тезиса Тьюринга-Черча можно также рассматривать замкнутость класса машин Тьюринга и ч.р.ф. относительно многочисленных естественных операций над алгоритмами и функциями. Отметим также, что тезис Тьюринга-Черча обращен и в будущее: он предполагает, что какие бы новые формальные определения алгоритмов не были предложены (а таковыми, например, являются новые языки программирования) все они не выйдут из класса алгоритмов, задаваемых машинами Тьюринга.

Чтобы показать связь теории алгоритмов с “практическим” программированием рассмотрим некоторые алгоритмические проблемы, связанные со структурированными программами.

Зафиксируем конечный алфавит $A = \{a_0, a_1, \dots, a_{m-1}\}$, включающий все символы латинского алфавита, цифры, знак пробела (пусть это будет a_0), знаки ‘;’, ‘=’, ‘<’, ‘:=’, а также знаки-ключевые слова **если**, **то**, **конец**, **пока**, **делай** и **все**. Тогда каждая структурированная программа Π представляет собой некоторое слово $w_\Pi = a_{i_1} a_{i_2} \dots a_{i_k}$ в алфавите A . Не ограничивая общности, будем считать, что это слово начинается не с пробела, т.е. $i_1 > 0$. Тогда слово w_Π однозначно определяет натуральное число n_Π , m -ичной записью которого оно является, т.е. $n_\Pi = \sum_{j=1}^k i_j m^{k-j}$. Назовем это число номером программы Π . По тексту программы Π ее номер n_Π определяется однозначно. Рассмотрим теперь обратное соответствие. Конечно, не каждое число является номером некоторой структурированной программы. Поэтому сопоставим каждому числу $n \in \mathbf{N}$ структурированную программу Π_n следующим образом: если $n = n_\Pi$ для некоторой программы Π , то $\Pi_n = \Pi$, иначе, т.е. когда n не является “естественным” номером никакой программы, сопоставим ему в качестве

Π_n некоторую никогда не останавливающуюся программу P (например, программу $\Pi_5(1)$: $x_1 := x_1$; **пока** $x_1 = x_1$ **делай** $x_1 := x_1$ **все** из примера 2.5 на стр. 6).

Проблема самоприменимости заключается в проверке для каждой программы Π с входной переменной x и выходной переменной y того, остановится ли Π на собственном номере n_Π , т.е. в вычислении всюду определенной функции

$$F_s(x) = \begin{cases} 1, & \text{если } \Phi_{\Pi,x}(n_{\Pi_x}) < \infty \\ 0 & \text{в противном случае} \end{cases}$$

Теорема 6.1. *Проблема самоприменимости алгоритмически неразрешима, т.е. не существует структурированной программы, вычисляющей функцию $F_s(x)$.*

Доказательство от противного. Предположим, что существует программа P , вычисляющая функцию $F_s(x)$. Без ограничения общности, можно считать, что ее выходная переменная есть y (почему?) и поэтому $\Phi_{P,y}(x) = F_s(x)$ для всех x . Пусть переменная z не входит в P . Рассмотрим следующую программу P' :

P ; $z := 0$; **если** $z < y$ **то** $\Pi_5(1)$ **иначе** $y := z$ **конец**.

Легко проверить, что если P на входе x выдает результат $y = 1$, то P' на этом входе не останавливается, а если P выдает результат $y = 0$, то P' останавливается (и тоже выдает 0). Пусть $n' = n_{P'}$ — номер программы P' . Чему тогда равно значение $\Phi_{P,y}(n')$? Если оно равно 1, то на входе $x = n'$ программа P' не остановится, т.е. $\Phi_{P',y}(n') = \infty$, но тогда $F_s(n') = 0 \neq \Phi_{P,y}(n')$. Если же $\Phi_{P,y}(n') = 0$, то P' на входе $x = n'$ останавливается с результатом 0, т.е. $\Phi_{P',y}(n') < \infty$. Но тогда $F_s(n') = 1 \neq \Phi_{P,y}(n') = 0$. Во всех случаях получили, что $F_s \neq \Phi_{P,y}$ и, следовательно, предположение о существовании программы для вычисления функции F_s неверно.

Заметим, что на самом деле мы доказали отсутствие структурированной программы для вычисления функции F_s . Но тезис Тьюринга-Черча и эквивалентность структурированных программ машинам Тьюринга и ч.р.ф. позволяют сделать вывод об алгоритмической неразрешимости рассматриваемой проблемы.

Проблема самоприменимости может показаться не очень интересной с практической (“программистской”) точки зрения. Но оказывается, что ее можно использовать для доказательства алгоритмической неразрешимости многих других алгоритмических проблем, более тесно связанных с практикой программирования.

1. *Проблема останова*: по произвольной структурированной программе Π определить завершится ли вычисление Π на входе 0, т.е. вычислить всюду определенную функцию

$$F_{h0}(n) = \begin{cases} 1, & \text{если } \Phi_{\Pi,n}(0) < \infty \\ 0 & \text{в противном случае} \end{cases}$$

В более общем виде проблема останова состоит в вычислении следующей функции:

$$F_h(n, a) = \begin{cases} 1, & \text{если } \Phi_{\Pi,n}(a) < \infty \\ 0 & \text{в противном случае} \end{cases}$$

Из этого определения следует, что программа Π_n останавливается на входе a тогда и только тогда, когда $F_h(n, a) = 1$.

2. *Проблема тотальности:* по произвольной структурированной программе Π определить завершает ли она работу при всех значениях входной переменной, т.е. вычислить всюду определенную функцию

$$F_t(n) = \begin{cases} 1, & \text{если для всех } a \Phi_{\Pi, y}(a) < \infty \\ 0 & \text{в противном случае} \end{cases}$$

3. *Проблема эквивалентности:* по произвольным двум структурированным программам Π и Π' определить эквивалентны ли они, т.е. вычисляют ли они одну и ту же функцию:

$$F_{eq}(n, m) = \begin{cases} 1, & \text{если для всех } a \Phi_{\Pi, y}(a) = \Phi_{\Pi', y}(a) \\ 0 & \text{в противном случае} \end{cases}$$

4. *Проблемы оптимизации текста программы.* Одна из возможных оптимизаций (текста) программы состоит в удалении из нее операторов присваивания, которые никогда не работают, а другая — в замене условных операторов вида

если φ то Π_1 иначе Π_2 конец,

на Π_1 в случае, когда условие φ истинно на любых входных данных, и — на Π_2 , если оно на любом входе ложно. Определим соответствующие этим оптимизациям функции:

$$F_{opt1}(n, m) = \begin{cases} 1, & \text{если существует вход } a, \text{ при работе на котором} \\ & \text{в программе } \Pi_n \text{ срабатывает } m\text{-ый по счету} \\ & \text{оператор присваивания} \\ 0 & \text{в противном случае} \end{cases}$$

Из этого определения следует, что при $F_{opt1}(n, m) = 0$ программу Π_n можно оптимизировать, удалив из нее m -ый оператор присваивания. Назовем задачу вычисления функции $F_{opt1}(n, m)$ *проблемой лишнего присваивания*.

$$F_{opt2}(n, m) = \begin{cases} 1, & \text{если существует вход } a, \text{ на котором при некотором} \\ & \text{срабатывании } m\text{-го по счету условного оператора из} \\ & \text{программы } \Pi_n \text{ его условие истинно} \\ 0 & \text{в противном случае} \end{cases}$$

Ясно, что при $F_{opt2}(n, m) = 0$ программу Π_n можно оптимизировать, заменив ее m -ый условный оператор его второй альтернативой. Назовем задачу вычисления функции $F_{opt2}(n, m)$ *проблемой лишнего условия*.

Заметим, что проблема самоприменимости и все проблемы, перечисленные в пп. 1-4 выше, связаны с вычислением функций, принимающих два значения 0 и 1. Эти

функции являются *характеристическими функциями* соответствующих множеств. Например, $F_{h0}(n)$ является характеристической функцией множества номеров программ, останавливающихся на входе 0. Напомним, что для множества $A \subseteq \mathbf{N}^k$ его характеристическая функция c_A^k определяется следующим образом:

$$c_A^k(x_1, \dots, x_k) = \begin{cases} 1, & \text{если } (x_1, \dots, x_k) \in A \\ 0 & \text{в противном случае} \end{cases}$$

На множества переносится понятия разрешимости и неразрешимости.

Определение 6.1. *Множество $A \subseteq \mathbf{N}^k$ называется разрешимым (или рекурсивным), если его характеристическая функция c_A^k вычислима, т.е. является общерекурсивной функцией, в противном случае, оно (и связанная с ним проблема) неразрешимо.*

Используя это определение, теорему 6.1 на стр. 37 можно переформулировать так:

Множество номеров программ, останавливающихся на собственном номере, $M_s = \{n \mid \Phi_{n,y}(n) < \infty\}$ неразрешимо.

Обычно доказательства неразрешимости проблем используют метод сведения. Неформально его идею можно сформулировать следующим образом: “Если решение некоторой неразрешимой проблемы A можно получить, используя решение проблемы B , то тогда проблема B тоже неразрешима.”

Определим отношение сводимости более формально. Напомним, что нумерационные функции позволяют вместо наборов (векторов, n -ок) целых чисел рассматривать их номера. Для множества $B \subseteq \mathbf{N}^r$ обозначим через $c_r(B)$ множество номеров входящих в B наборов: $c_r(B) = \{c_r(a_1, \dots, a_r) \mid (a_1, \dots, a_r) \in B\}$ (при $r = 1$, разумеется, $c_1(B) = B$).

Определение 6.2. *Множество (проблема) $A \subseteq \mathbf{N}^k$ сводится к множеству (проблема) $B \subseteq \mathbf{N}^r$, если существует общерекурсивная функция $f : \mathbf{N}^k \rightarrow \mathbf{N}^r$ такая, что $(x_1, \dots, x_k) \in A \Leftrightarrow f(x_1, \dots, x_k) \in c_r(B)$. В этом случае будем писать $A \leq_m B$ посредством f .*

Содержательно, “ A сводится к B посредством f ” означает, что для выяснения того входит ли x в A , можно эффективно преобразовать x в такие входные данные $y = f(x)$ проблемы B , что при $y \in B$ имеем $x \in A$, а если $y \notin B$, то и $x \notin A$.

Как мы уже отмечали, доказательство неразрешимости можно основывать на следующем утверждении.

Лемма 6.1. *Если A сводится к B и проблема A неразрешима, то и проблема B неразрешима.*

Доказательство. Пусть $A \leq_m B$ посредством f . Тогда из определения сводимости следует, что для всех x имеет место равенство $c_A(x) = c_B(f(x))$. Поэтому, если бы B была разрешима, то ее характеристическая функция c_B была бы общерекурсивна и c_A также была бы общерекурсивна. Но это противоречит неразрешимости проблемы A .

Теорема 6.2. *Все проблемы, перечисленные выше в пунктах 1-4 являются алгоритмически неразрешимыми.*

Доказательство. Нам потребуются следующие вспомогательные программы $\Pi_{x:=n}$: $x := 0; x := x + 1; \dots; x := x + 1$ (присваивание $x := x + 1$ повторяется n раз). Понятно, что для любого начального состояния σ после выполнения $\Pi_{x:=n}$ имеем $\Pi_{x:=n}(\sigma)(x) = n$.

1) Докажем неразрешимость проблемы останова: по произвольной структурированной программе Π определить завершится ли вычисление Π на входе 0. Пусть $M_{h_0} = \{n \mid \Phi_{\Pi, y}(0) < \infty\}$. Докажем, что множество номеров самоприменимых программ M_s сводится к M_{h_0} . Пусть n — номер программы Π_n . преобразуем ее в программу Π' : $\Pi_{x:=n}; \Pi_n; y := 0$. Таким образом, Π' вначале заносит в x номер n программы Π_n , а затем применяет Π_n к этому номеру и, если Π_n на n останавливается, выдает результат $y = 0$. Поэтому Π' останавливается на любом аргументе (в том числе и на 0) тогда и только тогда, когда $n \in M_s$. Преобразование программы Π_n в программу Π' осуществляется эффективно. Поэтому (на основании тезиса Тьюринга-Черча) существует такая о.р.ф. f , которая по n вычисляет номер m программы $\Pi' = \Pi_m = \Pi_{f(n)}$. Эта функция и будет сводить M_s к M_{h_0} , так как $n \in M_s \Leftrightarrow f(n) \in M_{h_0}$. Следовательно, по лемме 6.1 на стр. 39 проблемы останова M_{h_0} неразрешима.

Очевидно, что и более общая форма проблемы останова $M_h = \{(n, a) \mid \Phi_{\Pi_n, y}(a) < \infty\}$ также неразрешима, поскольку к ней сводится M_{h_0} : $n \in M_{h_0} \Leftrightarrow (n, 0) \in M_h$.

2) Для сведения M_s к множеству M_t номеров программ, вычисляющих всюду определенные функции можно также использовать функцию f из пункта 1. Действительно, Π_n останавливается на входе n тогда и только тогда, когда $\Pi' = \Pi_{f(n)}$ останавливается на всех входах, т.е. $n \in M_s \Leftrightarrow f(n) \in M_t$. Следовательно, проблема тотальности M_t неразрешима.

3) Рассмотрим теперь проблему эквивалентности. Пусть

$$M_{eq} = \{(n, m) \mid \text{для всех } x \Phi_{\Pi_n, y}(x) = \Phi_{\Pi_m, y}(x)\}.$$

Зафиксируем следующую программу P^0 : $x := x; y := 0$. Очевидно, что она вычисляет функцию тождественно равную нулю, т.е. $\Phi_{P^0, y}(x) = 0$ для всякого x . Пусть ее номер $n(P^0)$ равен k_0 . Для произвольного n рассмотрим пару $(f(n), k_0)$. Из определения f следует, что Π_n останавливается на входе n тогда и только тогда, когда $\Pi' = \Pi_{f(n)}$ останавливается на всех входах и выдает результат 0: $\Phi_{\Pi_{f(n)}, y}(x) = 0$ для всех x , т.е. $\Pi_{f(n)}$ и Π_{k_0} эквивалентны. Тогда $n \in M_s \Leftrightarrow (f(n), k_0) \in M_{eq}$. Положим $g(n) = c_2(f(n), k_0)$. Тогда g является о.р.ф. и $n \in M_s \Leftrightarrow g(n) \in c_2(M_{eq})$. Следовательно, M_s сводится к M_{eq} посредством g и проблема M_{eq} неразрешима.

4) Для доказательства неразрешимости проблемы лишнего присваивания:

$M_{opt1} = \{(n, m) \mid \text{на некотором входе } a \text{ в программе } \Pi_n \text{ срабатывает } m\text{-ый по счету оператор присваивания}\}$

снова используем функцию f из пункта 1. Напомним, что $\Pi_{f(n)} : \Pi_{x:=n}; \Pi_n; y := 0$. По n и соответствующей программе Π_n можно легко определить номер m последнего присваивания $y := 0$ в $\Pi_{f(n)}$:

$m = (n + 1) + (\text{число присваиваний в } \Pi_n) + 1$.

Пусть $g(n)$ — это о.р.ф., вычисляющая по n этот номер m . Тогда $n \in M_s \Leftrightarrow (f(n), g(n)) \in M_{opt1}$. Положим $h(n) = c_2(f(n), g(n))$. Тогда h является о.р.ф. и $n \in M_s \Leftrightarrow h(n) \in c_2(M_{opt1})$. Следовательно, M_s сводится к M_{opt1} посредством h и проблема M_{opt1} неразрешима.

Рассмотрим теперь проблему лишнего условия:

$M_{opt2} = \{(n, m) \mid \text{существует вход } a, \text{ на котором при некотором срабатывании } m\text{-го по счету условного оператора в программе } \Pi_n \text{ его условие истинно}\}$.

Для доказательства ее неразрешимости определим по n программу $\Pi'' : \Pi'; \text{если } y = 0 \text{ то } y := y \text{ иначе } y := y + 1 \text{ конец}$ (здесь Π' — программа из п. 1). И в этом случае программа Π'' строится по программе Π_n эффективно. Пусть ее номер вычисляется о.р.ф. f' , т.е. $\Pi_{f'(n)} = \Pi''$ и пусть о.р.ф. $g'(n)$ определяет номер последнего условного оператора в программе $\Pi_{f'(n)}$. Тогда $n \in M_s \Leftrightarrow$ в программе $\Pi_{f'(n)}$ последний условный оператор выполняется (на любом входе) и при этом $y = 0$, т.е. его условие истинно, а это означает, что $(f'(n), g'(n)) \in M_{opt2}$. Положив $h'(n) = c_2(f'(n), g'(n))$, получим, что $n \in M_s \Leftrightarrow h'(n) \in c_2(M_{opt2})$. Следовательно, M_s сводится к M_{opt2} посредством h' и проблема M_{opt2} также неразрешима.

Теорема доказана.

Какой же вывод можно сделать из того, что некоторая алгоритмическая проблема оказалась неразрешимой? Для программистов из такого утверждения извлекаются “две новости: плохая и хорошая”. “Плохая новость” состоит в том, что невозможно построить алгоритм (программу) для автоматического решения такой проблемы. Например, из теоремы 6.2 на стр. 40 следует, что невозможно автоматически проверить входит ли некоторый вход в область определения вычислимой функции, нельзя определить корректность программы, т.е. то, что она вычисляет требуемую функцию, нет способа проверять эквивалентность программ (не только структурированных, но и написанных на Паскале, Си, ассемблере, Яве и других языках программирования), не существует алгоритмов для оптимизаций, связанных с удалением лишних присваиваний и условий и т.п. Но неразрешимость проблемы не означает, что она не может быть решена для некоторых отдельных входных данных. Например, в предыдущих разделах мы построили достаточно много программ и доказали их корректность. Поэтому “хорошая новость” для программистов и математиков состоит в том, что их труд при решении неразрешимых проблем в каждом отдельном случае является творческим — никакой программой их не заменить. Появление каждой новой содер-

жательно интересной неразрешимой проблемы только расширяет область их творчества, заставляет искать все более и более широкие алгоритмы, позволяющие решать все более обширные подклассы относящихся к этой проблеме индивидуальных задач.

6.1 Задачи

Задача 6.1. Доказать алгоритмическую неразрешимость следующих проблем.

- а) По произвольной программе Π определить является ли вычисляемая ей функция $\Phi_{\Pi,y}(x)$ постоянной константой.
- б) По произвольной программе Π и числам a и b проверить равенство $\Phi_{\Pi,y}(a) = b$.
- в) По произвольной программе Π определить является ли множество значений вычисляемой ею функции $\Phi_{\Pi,y}(x)$ бесконечным.
- г) По произвольной паре программ Π и Π' проверить, что для всех x имеет место неравенство $\Phi_{\Pi,y}(x) > \Phi_{\Pi',y}(x)$.

Задача 6.2. Докажите, что

- а) пересечение двух разрешимых множеств является разрешимым множеством.
- б) объединение двух разрешимых множеств является разрешимым множеством.

Задача 6.3. Докажите, что для двух разрешимых множеств A и B их “сумма” $A + B = \{x + y \mid x \in A, y \in B\}$ также является разрешимым множеством.

Задача 6.4. Пусть A — разрешимое множество, а $g(x)$ и $h(x)$ являются о.р.ф. Докажите, что функция

$$F(x) = \begin{cases} g(x), & \text{если } x \in A \\ h(x), & \text{в противном случае} \end{cases}$$

также является общерекурсивной.