Г л а в а 6. Вычислимые функции

§1. Примитивно-рекурсивные функции

Одно из главных требований к алгоритму, предназначенному для решения той или иной задачи T, состоит в том, чтобы он позволял для каждого допустимого набора исходных данных $x_1, x_2, \ldots, x_n$ получить правильный ответ $T(x_1, x_2, \ldots, x_n)$. Если $x_1, x_2, \ldots, x_n$ и $T(x_1, x_2, \ldots, x_n)$ – это числа, то задача T – это числовая функция, которая каждому набору чисел $x_1, x_2, \ldots, x_n$ из заранее известной области допустимых значений ставит в соответствие число $T(x_1, x_2, \ldots, x_n)$. В таком случае можно сказать, что алгоритм, решающий данную задачу T, вычисляет некоторую функцию. Функцию, для вычисления которой существует алгоритм, принято называть *интуитивно вычислимой*.

На первый взгляд может показаться, что все числовые функции интуитивно вычислимы, а если вычислимость какой-либо функции и вызывает сомнение, то лишь потому, что для её вычисления пока ещё не придумали подходящий алгоритм. Однако математическими методами удалось доказать, что некоторые функции не являются вычислимыми. Это доказательство стало возможным только после того, как было сформулировано строгое определение и построена математическая модель алгоритма. Рекурсивная функция – это одна из таких моделей. Она была разработана в рамках теории рекурсивных функций, возникшей в 1930-е годы благодаря трудам А.Черча, К.Геделя, Дж.Эрбрана, С.Клини. Другие известные модели алгоритмов – это машины Поста и Тьюринга и нормальные алгоритмы Маркова.

Далее мы будем рассматривать только числовые функции, заданные на множестве целых неотрицательных чисел $N_0 = \{0, 1, 2, \ldots\}$ и принимающие значения из этого же множества. Если функция $f(x_1, x_2, \ldots, x_n)$ определена на всех наборах $(x_1, x_2, \ldots, x_n)$, где $x_1, x_2, \ldots, x_n \in N_0$, то она называется *всюду определенной*. В противном случае будем называть её *частичной* функцией.

Простейшими рекурсивными функциями считаются следующие всюду определенные функции:

1) тождественная константа $O(x) \equiv 0$;
2) *функция следования* $S(x) = x + 1$;
3) *селекторные функции* $I_k^n(x_1, x_2, \ldots, x_n) = x_k$, где $k \in \{1, 2, 3, \ldots, n\}$, $n \in \{1, 2, 3, \ldots\}$.

Более сложные функции можно получать из простейших, исполь-

зуя специальные операторы. Один из них – *оператор суперпозиции*. Пусть имеются функции $h(x_1, x_2,\ldots, x_m)$, $g_1(x_1, x_2,\ldots, x_n)$, $g_2(x_1, x_2,\ldots, x_n)$, $\ldots$, $g_m(x_1, x_2,\ldots, x_n)$. Тогда суперпозицией этих функций будет функция

$$f(x_1, x_2,\ldots, x_n) = h(g_1(x_1, x_2,\ldots, x_n), g_2(x_1, x_2,\ldots, x_n), \ldots, g_m(x_1, x_2,\ldots, x_n)),$$

которая получается в результате подстановки функций g_1, g_2, $\ldots$, g_m вместо аргументов функции h. С помощью суперпозиции из функций $O(x)$ и $S(x)$ можно получить все тождественные константы $h(x) \equiv k$, где $k \in \{1, 2, 3, \ldots\}$. Действительно, $S(O(x)) \equiv 1$, $S(S(O(x))) \equiv 2$ и т.д. Кроме того, суперпозиция позволяет выполнять переименование, перестановку и отождествление аргументов функции, зависящей от нескольких переменных. Например,

$$f(y, x_2, x_3,\ldots, x_n) = f(I_1^1(y), x_2, x_3,\ldots, x_n),$$

$$f(x_2, x_1, x_3,\ldots, x_n) = f(I_2^2(x_1, x_2), I_1^2(x_1, x_2), x_3,\ldots, x_n),$$

$$f(x_1, x_1, x_3,\ldots, x_n) = f(I_1^2(x_1, x_2), I_1^2(x_1, x_2), x_3,\ldots, x_n).$$

Иногда бывает необходимо из функции от n аргументов получить функцию от $(n + 1)$ аргументов за счет введения новой фиктивной переменной. Это тоже можно сделать с помощью суперпозиции и селекторной функции. Например, тождественная константа $h(x_1, x_2) \equiv 0$ может быть получена как суперпозиция $O(I_1^2(x_1, x_2))$.

Ещё один оператор, который мы будем применять к числовым функциям, – это оператор *примитивной рекурсии*. Рассмотрим систему

$$\begin{cases} f(0) = k, \\ f(n+1) = h(n, f(n)), \end{cases} \tag{54}$$

где $k, n \in \mathrm{N}_0$, а h – некоторая числовая функция от двух аргументов. Говорят, что система (54) задает функцию $f(x)$ с помощью оператора примитивной рекурсии. Такая система напоминает способ задания функции с помощью известного метода математической индукции, когда указывают значение функции в некоторой начальной точке и закон, согласно которому значение функции в каждой последующей точке вычисляют через её значение в предыдущей точке. Например, значение $f(3)$ получается в результате последовательного вычисления

$$f(0) = k,$$
$$f(1) = h(0, f(0)),$$
$$f(2) = h(1, f(1)),$$
$$f(3) = h(2, f(2)).$$

Иногда всюду определенную функцию $f(x)$ интерпретируют как числовую последовательность $y_0, y_1, \ldots, y_n, \ldots$, где $y_n = f(n)$. В данном случае элементы этой последовательности удовлетворяют рекуррентному соотношению единичной глубины

$$y_{n+1} = h(n, y_n)$$

с начальным условием $y_0 = k$. В некоторых случаях, когда функция h не слишком сложна, это соотношение удается решить, т.е. выразить y_n непосредственно через n.

В общем случае оператор примитивной рекурсии порождает из имеющихся функций $g(x_1, x_2, \ldots, x_m)$ и $h(x_1, x_2, \ldots, x_m, x_{m+1}, x_{m+2})$ новую функцию f от $(m + 1)$ аргументов, которую обычно задают системой равенств следующего вида:

$$\begin{cases} f(x_1, x_2, \ldots x_m, 0) = g(x_1, x_2, \ldots x_m), \\ f(x_1, x_2, \ldots x_m, n+1) = h(x_1, x_2, \ldots x_m, n, f(x_1, x_2, \ldots x_m, n)), \end{cases} \qquad (55)$$

где $n \in \mathrm{N}_0$. Здесь примитивная рекурсия применена по последнему аргументу функции f. Однако её можно применять по любому из аргументов. В любом случае значение функции f в каждой конкретной точке $(x_1, x_2, \ldots, x_m, n + 1)$ вычисляется через её же значение в предыдущей точке $(x_1, x_2, \ldots, x_m, n)$. Рассмотренная ранее примитивная рекурсия, определяемая системой (54), является простейшим случаем оператора примитивной рекурсии по единственному аргументу, когда $m = 0$, g – некоторая константа, а функция h зависит от двух аргументов. Можно доказать, что функция f, получающаяся в результате примитивной рекурсии, однозначно определяется парой исходных функций g и h.

Пример 1. Применим оператор примитивной рекурсии к функциям $g(x_1) = x_1$ и $h(x_1, x_2, x_3) = x_3 + 1$. В результате получим систему

$$\begin{cases} f(x_1, 0) = g(x_1) = x_1, \\ f(x_1, n+1) = h(x_1, n, f(x_1, n)) = f(x_1, n) + 1. \end{cases}$$

Найдем значение функции f в нескольких точках. Например,

$$f(x_1, 1) = f(x_1, 0) + 1 = x_1 + 1,$$
$$f(x_1, 2) = f(x_1, 1) + 1 = x_1 + 2,$$
$$f(x_1, 3) = f(x_1, 2) + 1 = x_1 + 3.$$

Нетрудно видеть, что $f(x_1, x_2) = x_1 + x_2$. Это согласуется с обоими равенствами системы, задающей функцию f.

Важно отметить, что, применяя операторы суперпозиции и примитивной рекурсии к всюду определенным функциям, мы получаем

также всюду определенные функции.

Определение. Функция f называется *примитивно-рекурсивной*, если она может быть получена из простейших рекурсивных функций $O(x)$, $S(x)$ и $I_k^n(x_1, x_2, ..., x_n)$ с помощью конечного числа применений операторов суперпозиции и примитивной рекурсии.

Класс всех примитивно-рекурсивных функций обозначим через $F_{пр}$. Примитивно-рекурсивны все тождественные константы, поскольку они могут быть получены с помощью суперпозиции из функций $O(x)$ и $S(x)$. Функция $f(x_1, x_2) = x_1 + x_2$ из примера 1 тоже примитивно-рекурсивна. Действительно, она получается с помощью примитивной рекурсии из функций $g(x_1) = x_1$ и $h(x_1, x_2, x_3) = x_3 + 1$, которые сами являются примитивно-рекурсивными, т.к.

$$g_1(x_1) = I_1^1(x_1),\ h(x_1, x_2, x_3) = S(I_3^3(x_1, x_2, x_3)).$$

Отождествляя аргументы функции $f(x_1, x_2) = x_1 + x_2$, получим новую функцию $g(x_1) = 2x_1$. Она также примитивно-рекурсивна, поскольку отождествление аргументов – это вариант оператора суперпозиции. Далее из функций $f(x_1, x_2) = x_1 + x_2$, $g(x_1) = 2x_1$ и тождественных констант, используя суперпозицию, можно получить $3x$, $4x$, …, а также все линейные функции вида $ax + b$, где $a, b \in \mathrm{N}_0$.

Пример 2. Покажем, что функция $p(x_1, x_2) = x_1 \cdot x_2$ примитивно-рекурсивна. Действительно, данную функцию можно получить с помощью операторов примитивной рекурсии и суперпозиции:

$$p(x_1, 0) \equiv 0 = O(x_1),$$

$$p(x_1, x_2 + 1) = x_1 \cdot (x_2 + 1) = x_1 \cdot x_2 + x_1 = x_1 + p(x_1, x_2) = f(x_1, p(x_1, x_2)),$$

где $f(x_1, x_2) = x_1 + x_2$ – примитивно-рекурсивная функция.

Отождествляя аргументы функции $p(x_1, x_2) = x_1 \cdot x_2$, получаем примитивно-рекурсивную функцию x_1^2, а далее с помощью подстановки – степенные функции x^3, x^4 и т.д. Из них, а также тождественных констант и примитивно-рекурсивной функции $f(x_1, x_2) = x_1 + x_2$, используя суперпозицию, можно получить любой полином вида

$$a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0,\ \text{где } a_0, \ldots, a_n \in \mathrm{N}_0.$$

Приведем ещё несколько примеров примитивно-рекурсивных функций, не доказывая строго их принадлежность классу $F_{пр}$.

Пример 3. Следующие функции примитивно-рекурсивны:

а) *усеченная разность* $x \dot{-} y = \begin{cases} x - y, \text{ если } x \geq y, \\ 0 \quad \text{иначе}; \end{cases}$

б) степенно-показательная функция $f(x,y) = x^y$;

в) факториал $f(n) = n!$;

г) $f(x, y) = |x - y|$;

д) $f(x, y) = \max\{x, y\}$;

е) $f(x, y) = \min\{x, y\}$;

ж) функция *сигнум* $sg(x) = \begin{cases} 0, \text{ если } x = 0, \\ 1 \text{ иначе}; \end{cases}$

з) *отрицание функции сигнум* $\overline{sg}(x) = \begin{cases} 1, \text{ если } x = 0, \\ 0 \text{ иначе}; \end{cases}$

и) $q(x,y) = [x/y]$ – целая часть дроби x/y (условимся, что $q(x,0) = 0$);

к) $r(x,y) = \{x/y\}$ – остаток от деления x на y (положим $r(x,0) = x$).

Таким образом, класс примитивно-рекурсивных функций $F_{пр}$ оказывается достаточно широким. В него входят обычные функции сложения и умножения. Следовательно, сумма и произведение конечного числа примитивно-рекурсивных функций – примитивно-рекурсивная функция. Кроме того, класс $F_{пр}$ содержит «почти обычные» вычитание и деление, а также функции $sg(x)$ и $\overline{sg}(x)$, иногда используемые в обычной (непрерывной) математике. Последние две функции оказываются полезными, когда требуется одним уравнением задать функцию, которая обычно определяется системой равенств.

Пример 4. Рассмотрим кусочно-заданную функцию

$$f(x) = \begin{cases} x + 2, \text{ если } x = 0,1,2,3, \\ 3x, \text{ если } x = 4,5,6, \\ 5 \text{ иначе}. \end{cases}$$

Проверкой нетрудно убедиться, что эту же функцию можно задать одним уравнением

$$f(x) = (x+2) \cdot \overline{sg}(x \dot{-} 3) + 3x \cdot sg(x \dot{-} 3) \cdot \overline{sg}(x \dot{-} 6) + 5 \cdot sg(x \dot{-} 6).$$

Поскольку все функции в этом уравнении принадлежат классу $F_{пр}$, то функция f – примитивно-рекурсивна.

Важно подчеркнуть, что примитивно-рекурсивные функции являются всюду определенными, поскольку этим свойством обладают простейшие рекурсивные функции $O(x)$, $S(x)$ и $I_k^n(x_1, x_2, ..., x_n)$, а опе-

раторы суперпозиции и примитивной рекурсии, примененные к всюду определенным функциям, порождают тоже всюду определенные функции. По этой причине не является примитивно-рекурсивной, например, следующая частичная функция:

$$f(x) = \begin{cases} x/2, & \textit{если } x - \textit{четное}, \\ \text{не определено } \textit{иначе}. \end{cases}$$

Отметим ещё одну важную черту всех функций класса $F_{пр}$: любая примитивно-рекурсивная функция вычислима в том смысле, что для её вычисления имеется алгоритм. Это можно обосновать тем, что, во-первых, существуют программы для машины Тьюринга, вычисляющие простейшие примитивно-рекурсивные функции. Во-вторых, разработаны методы композиции машин Тьюринга, позволяющие моделировать действие операторов суперпозиции и примитивной рекурсии. Благодаря этому появилась возможность создавать программы для вычисления сложных примитивно-рекурсивных функций на основе уже имеющихся программ для более простых функций.

§2. Примитивно-рекурсивные операторы и предикаты

Два основных примитивно-рекурсивных оператора нам уже известны – это операторы суперпозиции и примитивной рекурсии. Однако для получения новых примитивно-рекурсивных функций из простейших можно использовать и другие *примитивно-рекурсивные операторы*. Так называются операторы, в результате применения которых к примитивно-рекурсивным функциям получаются тоже примитивно-рекурсивные функции. Иными словами, это операторы, относительно действия которых класс $F_{пр}$ является замкнутым. Рассмотрим ещё несколько примитивно-рекурсивных операторов.

Оператор *переопределения* изменяет значение исходной примитивно-рекурсивной функции в одной из точек. Результатом переопределения функции $f(x_1, x_2, \ldots, x_m)$ в точке $(a_1, a_2, \ldots, a_m)$ будет функция

$$g(x_1, x_2, \ldots, x_m) = \begin{cases} b, & \textit{если } (x_1, x_2, \ldots, x_m) = (a_1, a_2, \ldots, a_m), \\ f(x_1, x_2, \ldots, x_m) & \textit{иначе}. \end{cases}$$

На конкретном примере проверим, что полученная функция g тоже примитивно-рекурсивна.

Пример 1. Пусть исходная функция $f(x)$ – тождественная константа $O(x)$. Переопределим её в точке $x = 2$ следующим образом:

$$g(x)=\begin{cases}5, \text{ если } x=2,\\ 0 \text{ иначе.}\end{cases}$$

Полученную функцию $g(x)$ можно задать равенством

$$g(x)=5\cdot\overline{sg}(x\dot{-}2)\cdot sg(x\dot{-}1),$$

в правой части которого участвуют только функции из класса $F_{пр}$. Следовательно, $g(x)$ примитивно-рекурсивна.

Многократным применением оператора переопределения из простейшей функции $O(x)$ можно получить любую всюду определенную функцию, равную нулю на множестве N_0, за исключением конечного числа точек. Поэтому все такие функции примитивно-рекурсивны.

Пусть дана примитивно-рекурсивная функция $f(x_1, x_2, \ldots, x_m, x_{m+1})$. Через $g(x_1, x_2, \ldots, x_m, n)$ обозначим сумму n слагаемых вида

$$g(x_1,x_2,...,x_m,n)=f(x_1,x_2,...,x_m,0)+f(x_1,x_2,...,x_m,1)+$$
$$+f(x_1,x_2,...,x_m,2)+...+f(x_1,x_2,...,x_m,n-1)=\sum_{k=0}^{n-1}f(x_1,x_2,...,x_m,k).$$

Если положить $g(x_1, x_2, \ldots, x_m, 0) = 0$, то функция g будет всюду определенной, поскольку всюду определена исходная функция f. Про полученную функцию g говорят, что она является результатом применения оператора *ограниченного суммирования* к функции f. Этот оператор является примитивно-рекурсивным. Действительно,

$$\begin{cases}g(x_1,x_2,...,x_m,0)=0,\\ g(x_1,x_2,...,x_m,n+1)=g(x_1,x_2,...,x_m,n)+f(x_1,x_2,...,x_m,n),\end{cases}$$

т.е. значение функции g в точке $(x_1, x_2, \ldots, x_m, n + 1)$ может быть вычислено через её значение в предыдущей точке $(x_1, x_2, \ldots, x_m, n)$, исходную функцию f и примитивно-рекурсивную функцию $x + y$.

Пример 2. Применим оператор ограниченного суммирования по аргументу x к примитивно-рекурсивной функции $f(a, x) = a^x$. Будем считать, что $0^0 = 1$. Поскольку при $a \neq 1$

$$\sum_{k=0}^{n-1}a^k=(a^n-1)/(a-1),$$

то полученную в результате функцию $g(a, x)$ можно задать системой

$$g(a,n)=\begin{cases}0, & \text{если } n=0,\\ n, & \text{если } a=1,\\ (a^n-1)/(a-1) & \text{иначе.}\end{cases}$$

В силу примитивной рекурсивности оператора ограниченного суммирования функция g является примитивно-рекурсивной.

Похожим образом действует оператор *ограниченного умножения*. Он ставит в соответствие примитивно-рекурсивной функции $f(x_1, x_2,\ldots, x_m, x_{m+1})$ новую функцию $h(x_1, x_2,\ldots, x_m, n)$ по следующему правилу:

$$h(x_1,x_2,\ldots,x_m,n)=f(x_1,x_2,\ldots,x_m,0)\cdot f(x_1,x_2,\ldots,x_m,1)\times$$

$$\times f(x_1,x_2,\ldots,x_m,2)\cdot\ldots\cdot f(x_1,x_2,\ldots,x_m,n-1)=\prod_{k=0}^{n-1} f(x_1,x_2,\ldots,x_m,k),$$

где по определению полагают $h(x_1, x_2,\ldots, x_m, 0) = 1$. Поскольку исходная функция f примитивно-рекурсивна, то и h является примитивно-рекурсивной функцией, т.к.

$$\begin{cases}g(x_1,x_2,\ldots,x_m,0)=1,\\ g(x_1,x_2,\ldots,x_m,n+1)=g(x_1,x_2,\ldots,x_m,n)\cdot f(x_1,x_2,\ldots,x_m,n).\end{cases}$$

Это означает, что оператор ограниченного умножения является примитивно-рекурсивным.

Пример 3. Если оператор ограниченного умножения применить к функции следования $S(x)$, получим функцию $n!$. Действительно, по определению оператора ограниченного умножения при $n = 0$ получаем $0!=1$. Если же $n > 0$, то

$$\prod_{k=0}^{n-1} S(k)=1\cdot 2\cdot 3\cdot\ldots\cdot n=n!.$$

В определении операторов ограниченного суммирования и умножения параметр n, ограничивающий число слагаемых или множителей, не обязательно является константой. Он может быть любой примитивно-рекурсивной функцией от переменных $x_1, x_2,\ldots, x_m$. Доказано, что полученная функция всё равно будет принадлежать классу $F_{пр}$.

Рассмотрим ещё один примитивно-рекурсивный оператор – *оператор взаимной* (или *совместной*) *рекурсии*. Он позволяет из примитивно-рекурсивных функций $g_1, g_2,\ldots, g_k$ и $h_1, h_2, \ldots, h_k$ получить целое семейство новых функций $f_1, f_2, \ldots, f_k$ по следующей схеме:

$$\begin{cases} f_i(x_1, x_2, ..., x_m, 0) = g_i(x_1, x_2, ..., x_m), \\ f_i(x_1, x_2, ..., x_m, n+1) = h_i(x_1, x_2, ..., x_m, n, f_1(x_1, x_2, ..., x_m, n), \\ \qquad\qquad f_2(x_1, x_2, ..., x_m, n), ..., f_k(x_1, x_2, ..., x_m, n)), \end{cases}$$

где $i = 1, 2, \ldots, k$. Например, с помощью взаимной рекурсии из констант $g_1 = g_2 = 1$ и функций $h_1(x,y,z) = z$, $h_2(x,y,z) = y + z$ получаются функции $f_1(x)$ и $f_2(x)$ такие, что

$$f_1(0) = f_2(0) = 1,$$
$$f_1(n + 1) = f_2(n),$$
$$f_2(n + 1) = f_1(n) + f_2(n).$$

Нетрудно видеть, что функция f_1 удовлетворяет линейному однородному рекуррентному соотношению

$$f_1(n + 1) = f_1(n - 1) + f_1(n)$$

с начальными условиями $f_1(0) = f_1(1) = 1$. Значения функции $f_1(0)$, $f_1(1)$, $f_1(2)$, ... образуют последовательность чисел Фибоначчи 1, 1, 2, 3 и т.д.

В теории алгоритмов вычислительные задачи часто разделяют на два класса. Один из них состоит из задач, в которых требуется найти значение числовой функции на конкретном наборе входных данных. В задачах другого класса нужно получить ответ («да» либо «нет») на вопрос типа: «Верно ли, что объект, описываемый конкретными входными данными, обладает заданным свойством?». В этом случае говорят, что задача состоит в проверке истинности предиката. *Предикат* – это логическая функция одного или нескольких переменных, которая на каждом конкретном допустимом наборе аргументов принимает одно из двух возможных значений: «истина» или «ложь». Всякий предикат $Q(x_1, x_2, \ldots, x_n)$ описывает некоторое свойство элементов $x_1, x_2, \ldots, x_n$ (или, что то же самое, отношение между ними). *Областью истинности* предиката называется множество всех таких наборов, на которых этот предикат принимает истинное значение. Каждому предикату соответствует своя *характеристическая функция*. Она равна единице только на наборах из области истинности предиката. На остальных наборах она равна нулю. Например, областью истинности арифметического предиката

$$Q(x, y) = \begin{cases} \text{истина}, & \textit{если } x < y, \\ \text{ложь} & \textit{иначе}, \end{cases}$$

является множество всех пар (x, y), для которых $x < y$. Это двуместный предикат, описывающий отношение «первый элемент пары меньше второго». Его характеристическая функция

$$\chi_Q(x,y)=\begin{cases}1, & \text{если } x<y,\\ 0 & \text{иначе},\end{cases}$$

примитивно-рекурсивна, т.к. для неё справедливо равенство

$$\chi_Q(x,y)=sg(y\dot{-}x).$$

Предикат будем называть *примитивно-рекурсивным*, если его характеристическая функция примитивно-рекурсивна. Большинство арифметических предикатов примитивно-рекурсивны.

Пример 4. Примитивно-рекурсивными являются, например, следующие предикаты:

а) двуместный предикат $P(x,y)$, описывающий отношение равенства

$$P(x,y)=\begin{cases}1, & \text{если } x=y,\\ 0 & \text{иначе},\end{cases}$$

с характеристической функцией

$$\chi_P(x,y)=\overline{sg}(|x-y|);$$

б) одноместный предикат $R(x)$, описывающий свойство x «быть четным»

$$R(x)=\begin{cases}1, & \text{если } x-\text{четное},\\ 0 & \text{иначе};\end{cases}$$

его характеристическая функция

$$\chi_R(x)=\overline{sg}(x\dot{-}2\cdot[x/2]).$$

Используя примитивно-рекурсивные предикаты и уже имеющиеся функции из класса $F_{пр}$, можно порождать новые функции с помощью *оператора условного задания функции*. Он состоит в следующем: пусть заданы примитивно-рекурсивные предикаты $P_1(x_1, x_2, \ldots, x_n)$, $P_2(x_1, x_2, \ldots, x_n)$, …, $P_m(x_1, x_2, \ldots, x_n)$, причем никакие два из них не должны быть одновременно истинными в одной и той же точке $(x_1, x_2, \ldots, x_n)$. Пусть функции $f_1(x_1, x_2,\ldots, x_n)$, $f_2(x_1, x_2,\ldots, x_n)$, …, $f_m(x_1, x_2,\ldots, x_n)$ – примитивно-рекурсивны. Тогда результатом применения оператора условного задания функции будет считаться функция $g(x_1, x_2,\ldots, x_n)$, заданная системой

$$g(x_1,x_2,\ldots,x_n)=\begin{cases}f_1(x_1,x_2,\ldots,x_n), & \text{если } P_1(x_1,x_2,\ldots,x_n) \text{ истинно},\\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & \\ f_m(x_1,x_2,\ldots,x_n), & \text{если } P_m(x_1,x_2,\ldots,x_n) \text{ истинно},\\ 0 & \text{иначе}.\end{cases}$$

Эта функция тоже примитивно-рекурсивна, поскольку её можно представить в виде суммы попарных произведений примитивно-рекурсивных функций

$$g(x_1, x_2, ..., x_n) = \sum_{k=1}^{m} f_k(x_1, x_2, ..., x_n) \cdot \chi_{P_k}(x_1, x_2, ..., x_n),$$

где $\chi_{P_k}(x_1, x_2, ..., x_n)$ – характеристическая функция предиката P_k.

Важно отметить, что все рассмотренные выше примитивно-рекурсивные операторы сводятся к основным операторам – суперпозиции и примитивной рекурсии. Иными словами, любой примитивно-рекурсивный оператор можно выразить через два основных оператора – суперпозицию и примитивную рекурсию. Поэтому, хотя класс $F_{пр}$ получается из простейших функций $O(x)$, $S(x)$ и $I_k^n(x_1, x_2, ..., x_n)$ с помощью суперпозиции и примитивной рекурсии, однако он не станет шире от того, что нам будет разрешено использовать и другие примитивно-рекурсивные операторы. Другая черта, объединяющая все примитивно-рекурсивные операторы, заключается в том, что в результате их применения к всюду определенным функциям получаются тоже всюду определенные функции.

Как известно, каждая примитивно-рекурсивная функция является всюду определенной. Но не любая всюду определенная функция примитивно-рекурсивна. Одна из них – так называемая *функция Аккермана* $A(x,y)$. Её можно задать системой

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

В этом описании используется двойная рекурсия, т.е. рекурсия по двум аргументам сразу. Анализ системы показывает, что значение функции Аккермана в любой точке (x,y) может быть вычислено через её значения в предыдущих точках. Например,

$$A(2,1) = A(1,A(2,0)) = A(1,sg(1)) = A(1,1) = A(0,A(1,0)) = A(0,0) = 2.$$

Это говорит в пользу того, что функция Аккермана вычислима. В то же время строго доказано, что её нельзя получить из простейших примитивно-рекурсивных функций $O(x)$, $S(x)$ и $I_k^n(x_1, x_2, ..., x_n)$ с помощью примитивно-рекурсивных операторов. Причиной этого является слишком большая скорость роста функции $A(x,y)$. Например,

$$A(0,x)=x+2,\ A(1,x)=2x,\ A(2,x)=2^x,\ A(3,x)=\underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{x\ раз},$$

а величина $A(4,x)$ растет настолько быстро, что её нельзя записать с помощью обычной математической формулы. Такое быстрое возрастание функции не могут обеспечить никакие примитивно-рекурсивные операторы. Это, кстати, означает, что двойная рекурсия не сводится к действию рассмотренных выше примитивно-рекурсивных операторов.

§3. Частично-рекурсивные функции

Пример функции Аккермана показывает, что существует более широкий класс функций, чем $F_{пр}$. Его можно построить, если к примитивно-рекурсивным функциям применить новый оператор, действие которого не сводится к многократному использованию суперпозиции и примитивной рекурсии. Это так называемый оператор минимизации.

Пусть задана числовая (не обязательно примитивно-рекурсивная) функция $f(x_1, x_2,\ldots, x_n)$. При фиксированных значениях $x_1, x_2,\ldots, x_n$ рассмотрим уравнение с неизвестным y вида

$$f(x_1, x_2,\ldots, x_{n-1}, y)=x_n. \tag{56}$$

Через $\mu_y((f(x_1, x_2,\ldots, x_{n-1}, y)=x_n)$ обозначим минимальное целое неотрицательное решение уравнения (56), если оно существует. *Оператор минимизации* по переменной x_n преобразует исходную функцию $f(x_1, x_2,\ldots, x_n)$ в новую функцию $g(x_1, x_2,\ldots, x_n)$, значение которой в каждой конкретной точке $(x_1, x_2,\ldots, x_n)$ вычисляется по следующему правилу:

1) если уравнение (56) имеет решение, и функция f определена во всех точках $(x_1, x_2,\ldots, x_{n-1}, k)$, где $0\le k\le\mu_y((f(x_1, x_2,\ldots, x_{n-1}, y)=x_n)$, то полагают $g(x_1, x_2,\ldots, x_n)=\mu_y((f(x_1, x_2,\ldots, x_{n-1}, y)=x_n)$;
2) если уравнение (56) имеет решение, но функция f не определена хотя бы в одной из точек $(x_1, x_2,\ldots, x_{n-1}, k)$, где $0\le k<\mu_y((f(x_1, x_2,\ldots, x_{n-1}, y)=x_n)$, то значение $g(x_1, x_2,\ldots, x_n)$ считается неопределенным;
3) если уравнение (56) не имеет решений, то значение $g(x_1, x_2,\ldots, x_n)$ считается неопределенным.

Оператор минимизации иногда ещё называют оператором поиска, т.к. для вычисления значения $g(x_1, x_2,\ldots, x_n)$ он отыскивает минимальный корень уравнения (56). Отыскать этот корень можно с помощью следующей процедуры: вычисляем $f(x_1, x_2,\ldots, x_{n-1}, 0)$ и проверяем ра-

венство $f(x_1, x_2, \ldots, x_{n-1}, 0) = x_n$. Если оно истинно, то $y = 0$ – искомый минимальный корень уравнения (56). В противном случае вычисляем $f(x_1, x_2, \ldots, x_{n-1}, 1)$ и проверяем равенство $f(x_1, x_2, \ldots, x_{n-1}, 1) = x_n$. Если оно истинно, то минимальный корень $y = 1$, иначе вычисляем $f(x_1, x_2, \ldots, x_{n-1}, 2)$ и т.д. Возможно, этот процесс поиска никогда не завершится, или окажется, что на некотором шаге значение $f(x_1, x_2, \ldots, x_{n-1}, y)$ не определено. В обоих этих случаях функция g в точке $(x_1, x_2, \ldots, x_{n-1}, x_n)$ будет считаться неопределенной. Таким образом, само описание оператора минимизации допускает возможность того, что в результате его применения может получиться не всюду определенная функция, даже если исходная функция была всюду определена.

Пример 1. Найдем функцию $g(x)$, которая получится в результате применения оператора минимизации к всюду определенной функции $f(x) = [x/2]$. При каждом конкретном x значение $g(x)$ равно минимальному корню уравнения $[y/2] = x$. Это уравнение имеет два корня: $2x$ и $2x + 1$. Поэтому $g(x) = 2x$.

Заметим, что в этом примере полученная функция $g(x)$ оказалась всюду определенной и в некотором смысле обратной по отношению к исходной функции $f(x)$. Поэтому оператор минимизации, примененный к функции одного аргумента $f(x)$, напоминает оператор нахождения обратной функции $f^{-1}(x)$ в обычной (непрерывной) математике.

Пример 2. Пусть требуется применить оператор минимизации по аргументу x_2 к функции $f(x_1, x_2) = x_1 \dot{-} x_2$. Значения искомой функции $g(x_1, x_2)$ будем получать отдельно для случаев $x_2 = 0$ и $x_2 > 0$.

Пусть $x_2 = 0$, тогда уравнение (56) будет иметь вид $x_1 \dot{-} y = 0$. Решая его относительно y, получаем бесконечное множество корней $\{x_1, x_1 + 1, x_1 + 2, \ldots\}$. Поскольку минимальным корнем является x_1, то при всех x_1 значение $g(x_1, 0) = x_1$.

Пусть теперь $x_2 > 0$. Тогда получаем уравнение $x_1 \dot{-} y = x_2$, которое при $x_2 > 0$ равносильно уравнению $x_1 - y = x_2$. Последнее уравнение имеет единственное решение $y = x_1 - x_2$, если $x_1 \geq x_2$, либо вообще не имеет корней, если $x_1 < x_2$. В итоге получаем следующую функцию:

$$g(x_1, x_2) = \begin{cases} x_1 - x_2, \textit{ если } x_1 \geq x_2, \\ \text{не определено } \textit{иначе.} \end{cases}$$

Заметим, что функция $g(x_1, x_2)$ оказалась частичной (т.е. не всюду определенной), хотя исходная функция $f(x_1, x_2)$ была всюду определена. Поэтому функция $g(x_1, x_2)$ не является примитивно-рекурсивной.

Определение. Функция f называется *частично-рекурсивной*, если

она может быть получена из простейших рекурсивных функций $O(x)$, $S(x)$ и $I_k^n(x_1,x_2,...,x_n)$ с помощью конечного числа применений операторов суперпозиции, примитивной рекурсии и минимизации.

Класс всех частично-рекурсивных функций обозначим через $F_{чр}$. Функция g из примера 2 принадлежит этому классу, т.к. она является результатом оператора минимизации по аргументу x_2 функции f, которая примитивно-рекурсивна. Поскольку любая примитивно-рекурсивная функция частично-рекурсивна, но, как показывает пример 2, существуют частично-рекурсивные функции, не являющиеся примитивно-рекурсивными, то справедливо строгое включение классов $F_{пр} \subset F_{чр}$.

Определим ещё один класс функций, также являющийся подмножеством класса $F_{чр}$. *Общерекурсивной функцией* будем называть всюду определенную частично-рекурсивную функцию. Доказано, что общерекурсивной является, например, рассмотренная выше функция Аккермана $A(x,y)$. Класс всех общерекурсивных функций обозначим через $F_{ор}$. Поскольку примитивно-рекурсивные функции являются всюду определенными, но не каждая всюду определенная функция примитивно-рекурсивна (например, функция Аккермана), то выполняется соотношение $F_{пр} \subset F_{ор}$. Оно подчеркивает несводимость оператора минимизации к примитивно-рекурсивным операторам, поскольку расширение класса $F_{пр}$ до $F_{ор}$ произошло именно благодаря применению к функциям из класса $F_{пр}$ оператора минимизации.

Функция из примера 2 не является общерекурсивной, хотя она частично-рекурсивна. Значит, $F_{ор} \subset F_{чр}$. Таким образом, между классами рекурсивных функций выполняется соотношение $F_{пр} \subset F_{ор} \subset F_{чр}$.

В начале главы 6 мы ввели понятие интуитивно вычислимой функций. Класс всех интуитивно вычислимых функций обозначим через $F_{выч}$. Этот класс, в отличие от $F_{пр}$, $F_{ор}$ и $F_{чр}$, не является четко определенным, поскольку единственное требование к функции этого класса – это существование единого алгоритма для её вычисления, а само понятие алгоритма не формализовано.

Все примитивно-рекурсивные операторы и оператор минимизации алгоритмичны, т.к. правила их применения к исходным функциям можно задать в виде алгоритмического процесса. Получается, что, с одной стороны, любая частично-рекурсивная функция интуитивно вычислима, поскольку вычислимы простейшие функции $O(x)$, $S(x)$ и $I_k^n(x_1,x_2,...,x_n)$, и алгоритмичен сам процесс применения к ним указанных операторов. Следовательно, класс $F_{чр}$ содержится в классе $F_{выч}$.

С другой стороны, мы уже видели, что многие часто встречающиеся вычислимые функции частично-рекурсивны (или даже примитивно-рекурсивны), если результат вычислений не выходит за рамки множества целых неотрицательных чисел N_0. Возникает вопрос: все ли интуитивно вычислимые функции частично-рекурсивны? Ответ на этот вопрос содержится в тезисе Черча.

Тезис Черча. *Любая интуитивно вычислимая функция частично-рекурсивна.*

Тезис Черча – это не математическая теорема, поскольку в его формулировке содержится понятие «интуитивной вычислимости», не имеющее точного математического определения. Поэтому тезис Черча невозможно строго доказать. Однако его справедливость подтверждается косвенно теоретическими исследованиями математиков, а также практической деятельностью специалистов, занятых разработкой и анализом вычислительных алгоритмов. С одной стороны, известные математики (Черч, Гедель, Клини, Тьюринг, Пост, Марков), поставившие перед собой цель найти точное определение понятия «вычислимой функции», независимо друг от друга разными путями пришли к одному и тому же классу функций – классу $F_{чр}$. С другой стороны, анализ существующих вычислительных алгоритмов, включая и те, которые использовались людьми ещё задолго до появления вычислительной техники, показывает, что каждый такой алгоритм, по сути, задает правила вычисления некоторой частично-рекурсивной функции.

Из тезиса Черча следует, что $F_{чр} = F_{выч}$, т.е. классы интуитивно вычислимых и частично-рекурсивных функций совпадают. Через $F_ч$ обозначим множество всех числовых функций (не обязательно всюду определенных), заданных на множестве целых неотрицательных чисел N_0 и принимающих значения из этого множества. Очевидно, это множество содержит внутри себя все классы рекурсивных функций $F_{пр}$, $F_{ор}$, $F_{чр}$ и класс вычислимых функций $F_{выч}$, который совпадает с $F_{чр}$. Соотношение между перечисленными классами можно задать диаграммой (рис. 94).

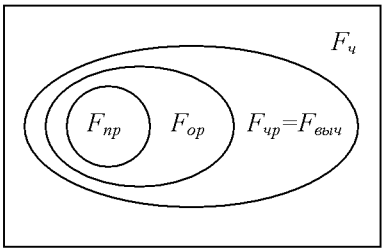

Рис. 94

Заметим, что класс числовых функций $F_ч$ шире класса вычислимых функций $F_{выч}$. Это можно обосновать следующим рассуждением: любую вычислимую функцию можно задать алгоритмом для её вы-

числения, причем разным функциям соответствуют разные алгоритмы. Сам алгоритм может быть записан в виде конечной последовательности символов из некоторого конечного универсального алфавита, единого для всех алгоритмов (например, в виде программы на одном из языков программирования). Поскольку количество всех возможных последовательностей конечной длины в этом алфавите счетно, то число различных алгоритмов не более, чем счетно (т.е. их не больше, чем натуральных чисел). Однако функций в классе $F_ч$ значительно больше. Действительно, область значений всякой функции из класса $F_ч$ – это некоторое подмножество множества $N_0 = \{0, 1, 2, \ldots\}$. И наоборот, всякое подмножество множества N_0 является областью значений некоторой функции из $F_ч$, причем разным подмножествам соответствуют разные функции. Как известно, количество таких подмножеств более, чем счетно (т.е. их строго больше, чем натуральных чисел). Поэтому и количество различных функций в классе $F_ч$ более, чем счетно. По сути это означает, что не для каждой функции из $F_ч$ найдется алгоритм, её вычисляющий. Рассмотрим одну из таких функций.

Пусть нам удалось пронумеровать все вычислимые функции от одного аргумента. Их счетное число, т.к. все тождественные константы 0, 1, 2 и т.д. можно отнести к вычислимым функциям одного аргумента. В результате мы получили последовательность

$$f_0(x), f_1(x), f_2(x), \ldots, f_n(x), \ldots . \qquad (57)$$

Функцию $g(x)$ зададим системой

$$g(n) = \begin{cases} f_n(n) + 1, & \textit{если } f_n(n) \textit{ определено,} \\ 0 & \textit{иначе.} \end{cases}$$

Функция $g(x)$ всюду определена. Является ли она вычислимой? Заметим, что она отличается от каждой из функций последовательности (57) хотя бы в одной точке. Например, $g(x)$ отличается от функции $f_n(x)$ в точке $x = n$. Действительно, если $f_n(x)$ не определена в точке $x = n$, то $g(x)$ в этой точке равна 0. Если же $f_n(x)$ определена при $x = n$, то её значение в этой точке равно $f_n(n)$, а функция $g(x)$ в этой точке равна $f_n(n) + 1$. Следовательно, эти функции различны. Таким образом, функция $g(x)$ не попала в последовательность (57) и, значит, не является вычислимой, поскольку по нашему предположению последовательность (57) содержит все вычислимые функции одного аргумента.

В начале главы 6 отмечалось, что всякой вычислительной задаче можно поставить в соответствие некоторую числовую функцию. Если эта функция интуитивно вычислима, то данную задачу называют *алго-*

ритмически разрешимой. Поскольку существуют и невычислимые функции, то можно предположить, что некоторым задачам соответствуют невычислимые функции. Такие задачи логично было бы назвать *алгоритмически неразрешимыми проблемами*.

Важно отметить принципиальную разницу в методах доказательства алгоритмической разрешимости и неразрешимости проблемы. Если для обоснования алгоритмической разрешимости некоторой задачи достаточно придумать решающий её алгоритм, то для доказательства алгоритмической неразрешимости проблемы нужно показать, что соответствующая ей числовая функция не является частично-рекурсивной. Тогда согласно тезису Черча это будет означать, что данная проблема алгоритмически неразрешима.

В теории алгоритмов существует множество задач, алгоритмическая неразрешимость которых строго доказана с использованием тезиса Черча. Например, алгоритмически неразрешима проблема, состоящая в том, чтобы по заданному описанию частично-рекурсивной функции узнать, обращается ли где-нибудь эта функция в ноль.

Алгоритмическая неразрешимость задачи не означает, что она вообще не может быть решена никакими методами ни при каких входных данных. Вполне возможно, что если ограничить множество допустимых входных наборов, то, зная об этих ограничениях, мы сможем придумать метод решения *частного случая* исходной задачи. Однако нельзя будет воспользоваться придуманным методом для решения задачи *в общем случае*. Поэтому, встречаясь с алгоритмически неразрешимыми проблемами на практике, обычно пытаются найти её разрешимые частные случаи.

Часто по тексту программы, записанной на некотором языке программирования, требуется выяснить, правильно ли она решает поставленную задачу (т.е. ту ли функцию она вычисляет). Обычно это удается выяснить с помощью отладки и разного рода логических рассуждений. Однако общего метода (алгоритма) анализа произвольной программы не существует из-за алгоритмической неразрешимости проблемы смыслового анализа алгоритма. Если знание физических законов сохранения предостерегает инженера от бесплодных попыток создания вечного двигателя, то знание основных алгоритмически неразрешимых проблем помогает программисту, пытающемуся придумать алгоритм для решения той или иной задачи, прежде всего понять, существует ли вообще такой алгоритм.