Министерство образования и науки
Российской Федерации

Национальный исследовательский
ядерный университет «МИФИ»

А.Н. Тихомирова, Н.В. Сафоненко

Практикум по теории алгоритмов

Рекомендовано УМО
«Ядерные физика и технологии»
в качестве учебного пособия
для студентов высших учебных заведений

Москва 2011

УДК 519.712(075)
ББК 22.176.я7
Т46

Тихомирова А.Н., Сафоненко Н.В. **Практикум по теории алгоритмов:** Учебное пособие. М.: НИЯУ МИФИ, 2011. – 132 с.

Даны базовые понятия теории алгоритмов, основные определения, свойства и теоремы. Теоретическая часть изложена кратко и носит справочный характер, цель – дать основу для решения практических задач и подготовки к сдаче экзамена. В каждом разделе приведены типовые задачи и вопросы с подробными решениям. Материал ориентирован на темы, изучаемые на третьем семестре НИЯУ МИФИ в рамках дисциплины «Дискретная математика (Теория алгоритмов и сложность вычислений)». Приведенные задания для самостоятельной работы во многом покрывают контрольные экзаменационные задачи, а иногда и превосходят их по сложности.

Предназначено студентам, изучающим курс «Дискретная математика (Теория алгоритмов и сложность вычислений)», а также всем, кто интересуется классическими формальными моделями представления алгоритмов, вопросами алгоритмической разрешимости, эффективной перечислимостью и вычислимостью чисел и функций, рекурсивными функциями.

Подготовлено в рамках Программы создания и развития НИЯУ МИФИ.

Рецензент канд. физ.-мат. наук,
доц. А.З. Насыров

ISBN 978-5-7262-1468-9

ОГЛАВЛЕНИЕ

ГЛАВА 1. ФОРМАЛЬНЫЕ ОПИСАНИЯ АЛГОРИТМОВ

1.1. Классические машины Тьюринга

1.1.1. Теоретическая основа

Предписание считается ***алгоритмом***, если оно обладает следующими свойствами:

- определенностью, т.е. общепонятностью и точностью, не оставляющими место произволу;
- массовостью – возможностью исходить из меняющихся в известных пределах значений исходных данных;
- результативностью – направленностью на получение искомого результата;
- элементарностью шагов, на который разбивается последовательность операций.

Каждый алгоритм, в общем случае, должен задаваться следующими ***параметрами***:

- множеством допустимых исходных данных;
- начальным состоянием;
- множеством допустимых промежуточных состояний;
- правилами перехода из одного состояния в другое;
- множеством конечных результатов;
- конечным состоянием.

Машина Тьюринга – абстрактная (воображаемая) "вычислительная машина" некоторого точно охарактеризованного типа, дающая пригодное для целей математического рассмотрения уточнение общего интуитивного представления об алгоритме.

Тезис Тьюринга: любой алгоритм можно преобразовать в машину Тьюринга.

Классической моделью считается одноленточная машина Тьюринга – кибернетическое устройство, состоящее из следующих элементов:

- бесконечной ленты, разделенной на ячейки;

- управляющей головки, способной читать символы, содержащиеся в ячейках ленты, и записывать символы в эти ячейки;
- выделенной ячейки памяти, содержащей символ внутреннего алфавита, задающий состояние машины Тьюринга;
- механического устройства управления, обеспечивающего перемещение головки относительно ленты;
- функциональной схемы – области памяти, содержащей команды (программу) машины Тьюринга (эта область доступна только для чтения).

Текущим состоянием машины Тьюринга или ее ***конфигурацией*** будем называть совокупность, образованную содержимым текущей обозреваемой ячейки a_j и состоянием внутренней памяти S_i.

Обычно машина Тьюринга схематично изображается в виде ленты с отмеченной указателем ячейкой (рис. 1.1).

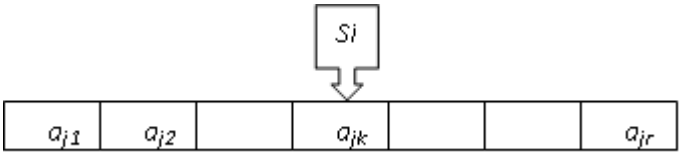

Рис. 1.1. Схематичное изображение одноленточной машины Тьюринга

1.1.2. Программы для одноленточных машин Тьюринга

Условимся обозначать буквой A внешний алфавит, т.е. те символы, из которых может состоять входное слово, σ – начальный символ, λ – пустой символ.

Условимся обозначать заглавными латинскими буквами с индексами внутренний алфавит (например, S_0, S_1, A_0, A_1 и т.д.), Ω – конечное состояние, Ω^1 – конечное состояние с выводом сообщения об ошибке.

Условимся записывать команду в следующем виде:

$A\ a \rightarrow b\ (R,L,H)\ B$,

где *A, B* – элементы внутреннего алфавита;

a, b – элементы внешнего алфавита;

(*R, L, H*) – одна из возможных команд перемещения управляющей головки:

R – перемещение управляющей головки на одну ячейку вправо;

L – перемещение управляющей головки на одну ячейку влево;

H – управляющая головка остаётся неподвижной.

При составлении алгоритма следует обратить внимание на следующие условия задачи:

- необходимо ли проверять, что входная строка не пустая;
- должно ли входное слово остаться неизменным в результате выполнения алгоритма.

Кроме того, желательно соблюдать несколько рекомендаций:

- выбирать названия элементов внутреннего алфавита в соответствии с действиями, которые должна выполнять машина, попав в данное состояние;
- минимизировать внутренний алфавит;
- по возможности не слишком сильно расширять внешний алфавит;
- минимизировать количество строк в алгоритме.

Задачи для самостоятельной работы

Задача 1.1. *A* = {*a, b*}. Заменить во входном слове все символы "*a*" на "*b*".

Пример (рис. 1.2):

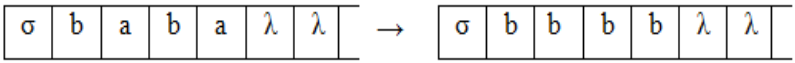

Рис. 1.2. Пример замены символов

Решение.

$S_0\,\sigma \rightarrow \sigma\, R\, S_0$ // символ начала ленты, сдвигаемся вправо без // изменения состояния;

$S_0\ b \rightarrow b\, R\, S_0$ // если текущий символ b, то оставляем без // изменения;

$S_0\ a \rightarrow b\, R\, S_0$ // если текущий символ a, то записываем на его // место b;

$S_0\,\lambda \to \lambda\, H\, \Omega$ // если слово пустое или слово закончилось, то // останавливаемся и завершаем программу.

Задача 1.2. $A = \{0, 1\}$. Записать в конце слова 1. Если слово пустое – вывести сообщение об ошибке (отметив заключительное состояние как Ω^1).

Пример (рис. 1.3):

σ	0	1	0	0	λ	λ

→

σ	0	1	0	0	1	λ	λ

Рис. 1.3. Пример записи в конце слова

Решение. Движемся вправо, пока не встретим символ пустой ячейки, что означает – слово закончилось, и записываем на его место 1.

$S_0\,\sigma \to \sigma\, R\, S_0$;

$S_0\,\lambda \to 1\, H\, \Omega^1$ // если слово пустое, то останавливаемся, вывод // сообщения об ошибке Ω^1;

$S_0\,1 \to 1\, R\, S_1$;

$S_0\,0 \to 0\, R\, S_1$;

$S_1\,1 \to 1\, R\, S_1$;

$S_1\,0 \to 0\, R\, S_1$;

$S_1\,\lambda \to 1\, H\, \Omega$.

Задача 1.3. $A = \{0, 1\}$. После слова записать первую букву слова. Если слово пустое – вывести сообщение об ошибке (отметив заключительное состояние как Ω^1).

Пример (рис. 1.4.):

σ	0	1	0	1	λ	λ

→

σ	0	1	0	1	0	λ	λ

Рис. 1.4. Пример записи после слова первого символа

Решение. В данной задаче нужно запомнить первый символ слова. Для этого используем разные состояния машины.

$S_0\,\sigma \to \sigma\, R\, S_0$;

$S_0\,\lambda \to \lambda\, H\, \Omega^1$ // пустое слово – сообщение об ошибке;

$S_0\ 0 \to 0\, R\, S_2$ // если символ 0, то переводим автомат в;

$S_2\ 0 \to 0\ R\ S_2$ // состояние S_2, в котором он двигается вправо;

$S_2\ 1 \to 1\ R\ S_2$ // и записывает в конце 0;

$S_2\ \lambda \to 0\ H\ \Omega$;

$S_0\ 1 \to 1\ R\ S_1$ // если символ 1, то переводим автомат в состояние;

$S_1\ 1 \to 1\ R\ S_1$ // S_1, в котором он двигается вправо и записывает;

$S_1\ 0 \to 0\ R\ S_1$ // в конце 1;

$S_1\ \lambda \to 1\ H\ \Omega$.

Многие программы удобно представлять в виде таблицы, где на пересечении соответствующего состояния и символа указываются действия, которые должна выполнить МТ. Обозначим × невозможность появления данной конфигурации. Решение задачи 1.3 в таком формате представлено в табл. 1.1.

Таблица 1.1

Табличное представление программы для задачи 1.3

Состояние	σ	0	1	λ
S_0	$\sigma\ R\ S_0$	$0\ R\ S_2$	$1\ R\ S_1$	$\lambda\ H\ \Omega^1$
S_1	×	$0\ R\ S_1$	$1\ R\ S_1$	$1\ H\ \Omega$
S_2	×	$0\ R\ S_2$	$1\ R\ S_2$	$0\ H\ \Omega$

Также программу можно оформлять в виде группировки по состояниям, решение задачи 1.3 в таком формате представлено в табл. 1.2.

Таблица 1.2

Табличное представление программы для задачи 1.3

S_0	S_1	S_2
$S_0\ \sigma \to \sigma\ R\ S_0$	$S_1\ 1 \to 1\ R\ S_1$	$S_2\ 1 \to 1\ R\ S_2$
$S_0\ \lambda \to \lambda\ H\ \Omega^1$	$S_1\ 0 \to 0\ R\ S_1$	$S_2\ 0 \to 0\ R\ S_2$

Продолжение табл. 1.2

S_0	S_1	S_2
$S_0\ 1 \rightarrow 1\ R\ S_1$	$S_1\ \lambda \rightarrow 1\ H\ \Omega$	$S_2\ \lambda \rightarrow 0\ H\ \Omega$
$S_0\ 0 \rightarrow 0\ R\ S_2$		

Отметим, что если в условии задачи не указана необходимость отдельно обрабатывать ситуацию с пустым словом, то текст программы будет короче. Также в качестве рекомендации можно отметить, что выбор названия для новых состояний – дело вкуса, и, как и в случае с наименованием переменных в современных языках программирования, читаемость текста программы во многом зависит от этого выбора. При решении задачи 1.3, например, вместо состояний S_1 и S_2 более удачным был бы выбор A_0 и A_1, так как в этом случае нижний индекс являлся бы индикатором того, какой символ запоминается с помощью этого состояния.

Ниже представлено решение задачи 1.3 для случая, когда не требуется выделение ситуации пустого входного слова.

$S_0\ \sigma \rightarrow \sigma\ R\ S_0$;

$S_0\ \lambda \rightarrow \lambda\ H\ \Omega$; // слово пустое, окончание программы;

$S_0\ 0 \rightarrow 0\ R\ A_0$ // если символ 0, то переводим автомат в;

$A_0\ 0 \rightarrow 0\ R\ A_0$ // состояние A_0, в котором он двигается вправо;

$A_0\ 1 \rightarrow 1\ R\ A_0$ // и записывает в конце 0;

$A_0\ \lambda \rightarrow 0\ H\ \Omega$;

$S_0\ 1 \rightarrow 1\ R\ A_1$ // если символ 1, то переводим автомат;

$A_1\ 0 \rightarrow 0\ R\ A_1$ // в состояние A_1, в котором он двигается;

$A_1\ 1 \rightarrow 1\ R\ A_1$ // вправо и записывает в конце 1;

$A_1\ \lambda \rightarrow 1\ H\ \Omega$.

Задача 1.4. $A = \{0, 1\}$. Записать в конце слова символ, который был последним в слове.

Пример (рис. 1.5):

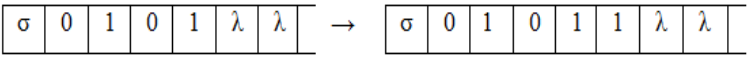

Рис. 1.5. Пример записи после слова последнего символа

Решение. Для 1 «резервируется» состояние A_1, для 0 – состояние A_0. Если в состоянии A_1 встречается 1 или в состоянии A_0 встречается 0, то, не меняя символа и состояния, движемся вправо. Если в состоянии A_1 встречается 0, то переходим в состояние A_0, если в состоянии A_0 встречается 1, то переходим в состояние A_1. Таким образом, индекс состояния указывает на последний прочитанный символ, таким образом дойдя до свободной ячейки (первой пустой после слова), мы автоматически знаем, какой символ был последним в слове.

$S_0\,\sigma \rightarrow \sigma\, R\, S_0$;

$S_0\,0 \rightarrow 0\, R\, A_0$;

$S_0\,1 \rightarrow 1\, R\, A_1$;

$S_0\,\lambda \rightarrow \lambda\, H\, \Omega$ // слово пустое, окончание программы;

$A_0\,0 \rightarrow 0\, R\, A_0$;

$A_0\,1 \rightarrow 1\, R\, A_1$;

$A_1\,0 \rightarrow 0\, R\, A_0$;

$A_1\,1 \rightarrow 1\, R\, A_1$;

$A_0\,\lambda \rightarrow 1\, H\, \Omega$;

$A_1\,\lambda \rightarrow 0\, H\, \Omega$.

В ряде задач существует несколько альтернативных подходов к идее алгоритма. Например, в данной задаче можно было бы дойти до конца слова, вернуться на шаг назад, считать последний символ и, снова перейдя на клетку правее, его записать. С точки зрения экономии количества тактов работы машины Тьюринга это менее эффективно, зато сам алгоритм получится на одну строчку короче (стоит отметить, что при этом потребуется ввести на одно состояние больше, чем в первой версии).

$S_0\,\sigma \rightarrow \sigma\, R\, S_0$;

$S_0\,0 \rightarrow 0\, R\, S_0$;

$S_0\,1 \rightarrow 1\, R\, S_0$;

$S_0\,\lambda \rightarrow \lambda\, L\, A$;

$A\,\sigma \rightarrow \sigma\, H\, \Omega$ // слово пустое;

$A\,0 \rightarrow 0\, R\, A_0$;

$A\,1 \rightarrow 1\, R\, A_1$;

$A_0\,\lambda \rightarrow 1\, H\, \Omega$;

$A_1\,\lambda \rightarrow 0\, H\, \Omega$.

Задача 1.5. $A = \{0, 1, 2\}$. Во входном слове заменить все комбинации “012” на звездочки.

Пример (рис. 1.6):

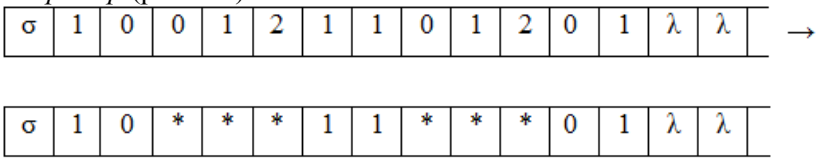

Рис. 1.6. Пример замены комбинации символов

Решение. Предлагается следующий алгоритм: сначала находится комбинация 012, затем она заменяется звездочками справа налево (показано жирным шрифтом), затем управляющая головка сдвигается правее поставленных звездочек, и снова запускается процесс поиска комбинации 012 (табл. 1.3).

Таблица 1.3

Табличное представление программы для задачи 1.5

S_0	S_1	S_2	S_3
$S_0\,\sigma \rightarrow \sigma\, R\, S_0$	$S_1\,0 \rightarrow 0\, R\, S_1$	$S_2\,0 \rightarrow 0\, R\, S_1$	$S_3\,1 \rightarrow *\, L\, S_3$
$S_0\,0 \rightarrow 0\, R\, S_1$	$S_1\,1 \rightarrow 1\, R\, S_2$	$S_2\,2 \rightarrow *\, L\, S_3$	$S_3\,0 \rightarrow *\, H\, S_0$

Продолжение табл. 1.3

S_0	S_1	S_2	S_3
$S_0\,1 \to 1\ R\ S_0$	$S_1\,2 \to 2\ R\ S_0$	$S_2\,1 \to 1\ R\ S_0$	
$S_0\,2 \to 2\ R\ S_0$	$S_1\,\lambda \to \lambda\ H\ \Omega$	$S_2\,\lambda \to \lambda\ H\ \Omega$	
$S_0\,* \to *\ R\ S_0$			
$S_0\,\lambda \to \lambda\ H\ \Omega$			

В данной задаче самым тонким моментом является необходимость корректно отслеживать начало последовательности. Наиболее частая ошибка в решении подобных задач заключается в сбрасывании индекса состояния снова в S_0, когда не удается найти второй и третий символы нужной последовательности (ошибочные строки: $S_1\ 0 \to 0\ R\ S_0$ и $S_2\ 0 \to 0\ R\ S_0$). Однако фрагмент последовательности может выглядеть так: «0012» или «01012». За правильную обработку отвечают строки $S_1\ 0 \to 0\ R\ S_1$ и $S_2\,0 \to 0\ R\ S_1$.

Задача 1.6. $A = \{1\}$. На ленте в унарном коде записано натуральное число. Вычислить число, предшествующее данному в натуральном ряду. Примечание: в унарном коде число ноль кодируется одной единицей, число один – двумя единицами и т.д.

Пример (рис 1.7):

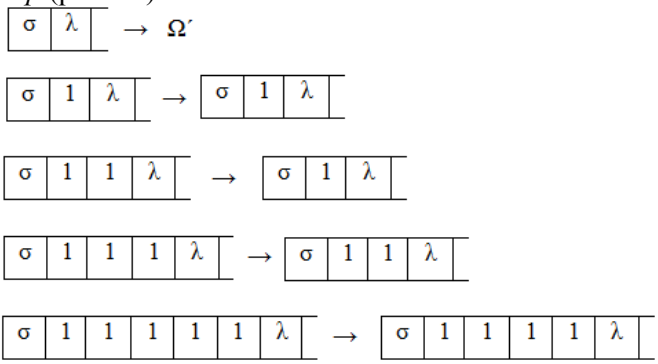

Рис. 1.7. Пример вычисления предыдущего числа

Решение. Если входное слово пустое или состоит из одной единицы (код числа ноль), выводим сообщение об ошибке. Если

входное слово состоит из двух единиц, оставляем одну единицу (код числа ноль). Если число единиц три или более, то стираем одну единицу справа.

$S_0\,\sigma \rightarrow \sigma\, R\, S_0$;

$S_0\,\lambda \rightarrow \lambda\, H\, \Omega^1$ // пустое слово – вывод сообщения об ошибке;

$S_0\,1 \rightarrow 1\, R\, S_1$;

$S_1\,\lambda \rightarrow \lambda\, H\, \Omega^1$ // слово состоит из одной единицы;

$S_1\,1 \rightarrow 1\, R\, S_2$ // если встречается вторая единица, то переходим // в состояние S_2;

$S_2\,1 \rightarrow 1\, R\, S_2$ // движемся вправо пока не доходим до конца;

$S_2\,\lambda \rightarrow \lambda\, L\, S_3$ // слово закончилось, передвигаемся влево;

$S_3\,1 \rightarrow \lambda\, H\, \Omega$ // стираем одну единицу.

Задача 1.7. $A = \{0, 1\}$. Скопировать слово в обратном порядке после символа *.

Пример (рис. 1.8):

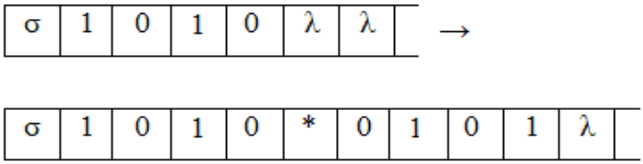

Рис. 1.8. Пример копирования слова в обратном порядке

Решение. Предлагается следующий алгоритм: сначала находится конец слова, ставится *, затем маркером помечается символ, стоящий слева от *, он копируется в конец слова, далее управляющая головка перемещается к предпоследнему символу, он также копируется в конец слова, и так до тех пор пока маркер сдвигается к началу слова.

$S_0\,\sigma \rightarrow \sigma\, R\, S_0$;

$S_0\,\lambda \rightarrow \lambda\, H\, \Omega^1$ // пустое слово – вывод сообщения об ошибке;

$S_0\,0 \to 0\,R\,S_1$ // двигаемся вправо, пока не дойдем до конца слова;

$S_0\,1 \to 1\,R\,S_1$;

$S_1\,0 \to 0\,R\,S_1$;

$S_1\,1 \to 1\,R\,S_1$;

$S_1\,\lambda \to *\,L\,S_2$ // дошли до конца слова – ставим *;

$A_0\,0 \to 0\,R\,A_0$ // двигаемся вправо, пока не дойдем до конца слова;

$A_0\,1 \to 1\,R\,A_0$;

$A_0\,* \to *\,R\,A_0$

$A_1\,0 \to 0\,R\,A_1$; // переходим через поставленный символ *;

$A_1\,1 \to 1\,R\,A_1$;

$A_1\,* \to *\,R\,A_1$;

// переходим через поставленный символ *;

$A_0\,\lambda \to 0\,L\,S_3$ // дошли до конца слова – ставим 0;

$A_1\,\lambda \to 1\,L\,S_3$ // дошли до конца слова – ставим 1;

$S_2\,0 \to 0'\,R\,A_0$ // если встретился ноль, то запоминаем A_0;

$S_2\,1 \to 1'\,R\,A_1$ // если встретилась единица, то запоминаем A_1;

$S_2\,\sigma \to \sigma\,H\,\Omega$ // слово скопировано;

$S_3\,0 \to 0\,L\,S_3$ // двигаемся влево;

$S_3\,1 \to 1\,L\,S_3$;

$S_3\,* \to *\,L\,S_3$;

$S_3\,0' \to 0\,L\,S_2$ // находим 0 или 1 с пометкой, снимаем отметку;

$S_3\,1' \to 1\,L\,S_2$.

Задача 1.8. $A = \{0, 1\}$. Скопировать слово в обратном порядке сразу вслед за исходным символом (без использования *).

Пример (рис. 1.9):

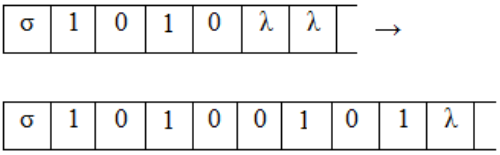

Рис. 1.9. Пример копирования слова в обратном порядке

Решение. Предлагается следующий алгоритм: сначала находится конец слова, затем маркером помечается последний символ и копируется в конце слова, маркер сдвигается к началу слова.

$S_0\ \sigma \rightarrow \sigma\ R\ S_0$;

$S_0\ \lambda \rightarrow \lambda\ H\ \Omega^1$ // пустое слово – вывод сообщения об ошибке;

$S_0\ 0 \rightarrow 0\ R\ S_1$;

$S_0\ 1 \rightarrow 1\ R\ S_1$;

$S_1\ 0 \rightarrow 0\ R\ S_1$ // двигаемся вправо;

$S_1\ 1 \rightarrow 1\ R\ S_1$ // пока не дойдем до конца слова;

$S_1\ \lambda \rightarrow \lambda\ L\ S_2$ // дошли до конца слова, вернулись к последнему символу;

$S_2\ \sigma \rightarrow \sigma\ H\ \Omega$ // это начало ленты – значит слово скопировано;

$S_2\ 0 \rightarrow 0'\ R\ A_0$ // если встретился ноль, то помечаем ноль и переходим в состояние A_0;

$S_2\ 1 \rightarrow 1'\ R\ A_1$ // если встретилась единица, то помечаем единицу и переходим в состояние A_1;

$A_0\ 0 \rightarrow 0\ R\ A_0$ // двигаемся вправо;

$A_0\ 1 \rightarrow 1\ R\ A_0$ // пока не дойдем до конца слова;

$A_1\ 0 \rightarrow 0\ R\ A_1$;

$A_1\ 1 \rightarrow 1\ R\ A_1$;

$A_0\ \lambda \rightarrow 0\ L\ S_3$ // дошли до конца слова – ставим 0;

$A_1\ \lambda \rightarrow 1\ L\ S_3$ // дошли до конца слова – ставим 1;

$S_3\ 0 \rightarrow 0\ L\ S_3$ // двигаемся влево, пока не найдем 0’ или 1’;

$S_3\ 1 \rightarrow 1\ L\ S_3$;

$S_3\ 0' \rightarrow 0\ L\ S_2$ // снимаем отметку;

$S_3\ 1' \rightarrow 1\ L\ S_2$.

Задача 1.9. $A = \{0, 1\}$. Скопировать слово в прямом порядке после символа *.

Пример (рис.1.10):

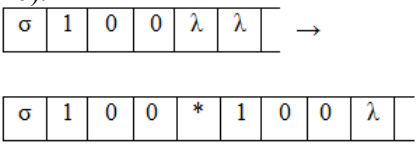

Рис. 1.10. Пример копирования слова в прямом порядке

Решение. Предлагается следующий алгоритм: сначала находится конец слова, ставится *, затем маркером помечается первый символ слова и этот символ копируется в конец слова, затем постепенно управляющая головка сдвигается на второй символ, копируется он, и далее к концу слова, пока не дойдёт до *.

$S_0\ \sigma \rightarrow \sigma\ R\ S_0$;

$S_0\ \lambda \rightarrow \lambda\ H\ \Omega^1$ // пустое слово – вывод сообщения об ошибке;

$S_0\ 0 \rightarrow 0\ R\ S_1$ //двигаемся вправо, пока не дойдем до конца слова;

$S_0\ 1 \rightarrow 1\ R\ S_1$;

$S_1\ 0 \rightarrow 0\ R\ S_1$;

$S_1\ 1 \rightarrow 1\ R\ S_1$;

$S_1\ \lambda \rightarrow *\ L\ S_2$ //дошли до конца слова – ставим *;

$S_2\ 0 \rightarrow 0\ L\ S_2$ // двигаемся влево, пока не найдем 1’ или 0’;

$S_2\ 1 \rightarrow 1\ L\ S_2$;

$S_2\ * \rightarrow *\ L\ S_2$;

$S_2\ 0' \rightarrow 0\ R\ S_3$ // снимаем отметку;

$S_2\ 1' \rightarrow 1\ R\ S_3$;

$S_2\ \sigma \rightarrow \sigma\ R\ S_3$ // дошли до начала слова;

$S_3\ 0 \rightarrow 0'\ R\ A_0$ // если встретился ноль, то помечаем ноль и переходим в состояние A_0;

$S_3\ 1 \rightarrow 1'\ R\ A_1$ // если встретилась единица, то помечаем единицу и переходим в состояние A_1;

$S_3\ * \rightarrow *\ H\ \Omega$ //слово скопировано;

$A_0\ 0 \rightarrow 0\ R\ A_0$ //двигаемся вправо, пока не дойдем до конца слова;

$A_0\ 1 \rightarrow 1\ R\ A_0$;

$A_0\ * \rightarrow *\ R\ A_0$;

$A_1\ 0 \rightarrow 0\ R\ A_1$;

$A_1\ 1 \rightarrow 1\ R\ A_1$;

$A_1\ * \rightarrow *\ R\ A_1$;

$A_0\ \lambda \rightarrow 0\ L\ S_2$ //дошли до конца слова – ставим 0;

$A_1\ \lambda \rightarrow 1\ L\ S_2$ //дошли до конца слова – ставим 1.

Задача 1.10. $A = \{0, 1\}$. Скопировать слово в прямом порядке без символа *.

Пример (рис.1.11):

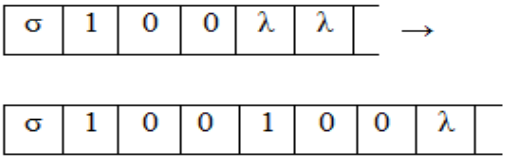

Рис. 1.11. Пример копирования слова без использования разделителя

Решение. Предлагается следующий алгоритм: копируем по символу, скопированные символы помечаем (и в исходном слове, и в скопированном), когда будет скопировано всё слово, отметки убираем.

$S_0\ \sigma \rightarrow \sigma\ R\ S_0$;

$S_0\ \lambda \rightarrow \lambda\ H\ \Omega^1$ // пустое слово – вывод сообщения об ошибке;

$S_0\ 0 \rightarrow 0'\ R\ A_0$ // помечаем ноль;

$S_0\ 1 \rightarrow 1'\ R\ A_1$ // помечаем единицу;

$S_0\ 0'' \rightarrow 0\ R\ S_3$ // слово скопировано целиком;

$S_0\ 1'' \rightarrow 1\ R\ S_3$; // осталось убрать пометки;

$A_0\ 0 \rightarrow 0\ R\ A_0$ // двигаемся вправо через

$A_0\ 1 \rightarrow 1\ R\ A_0$ //символы исходного слова;

$A_1\ 0 \rightarrow 0\ R\ A_1$ // ищем конец слова;

$A_1\ 1 \rightarrow 1\ R\ A_1$;

$A_0\ 0'' \rightarrow 0''\ R\ A_0$ // двигаемся вправо через символы копии;

$A_0\ 1'' \rightarrow 1''\ R\ A_0$ // ищем конец слова;

$A_1\ 0'' \rightarrow 0''\ R\ A_1$;

$A_1\ 1'' \rightarrow 1''\ R\ A_1$;

$A_0\ \lambda \rightarrow 0''\ L\ S_2$ // дошли до конца слова – ставим 0 с пометкой “;

$A_1\ \lambda \rightarrow 1''\ L\ S_2$ //дошли до конца слова – ставим 1 с пометкой “;

$S_2\ 0 \rightarrow 0\ L\ S_2$ // двигаемся влево пока не найдем 0’ или 1’;

$S_2\ 1 \rightarrow 1\ L\ S_2$;

$S_2\ 0'' \rightarrow 0''L\ S_2$;

$S_2\ 1'' \rightarrow 1''L\ S_2$;

$S_2\ 0' \rightarrow 0\ R\ S_0$ // снимаем пометку;

$S_2\ 1' \rightarrow 1\ R\ S_0$;

$S_3\ 0'' \rightarrow 0\ R\ S_3$ // снимаем отметку и двигаемся вправо;

$S_3\ 1'' \rightarrow 1\ R\ S_3$;

$S_3\ \lambda \rightarrow \lambda\ H\ \Omega$ // слово скопировано.

Задача 1.11. $A = \{0, 1\}$. Скопировать слово в обратном порядке, записав результат вместо слова.

Решение. Предлагается следующий алгоритм: копируем по очереди символ из начала слова на место в конце слова, запоминая, какой символ стоял на этом месте, затем возвращаемся к началу слова и записываем этот символ, сдвигаемся вправо, копируем следующие символ на место предпоследнего, и так далее, пока все символы не будут скопированы в обратном порядке. Обратите внимание: в алгоритме учитывается, что слово может быть и чётной, и нечётной длины.

$S_0\,\sigma \to \sigma\, R\, S_0$;

$S_0\,0 \to 0`R\, A$ // помечаем 0 и запоминаем состоянием A;

$S_0\,1 \to 1`R\, B$ // помечаем 1 и запоминаем состоянием B;

$S_0\,\lambda \to \lambda\, H\, \Omega^1$ // пустое слово – вывод сообщения об ошибке;

$A\,0 \to 0\, R\, A$ //двигаемся вправо, пока не дойдем до конца слова;

$A\,1 \to 1\, R\, A$;

$B\,0 \to 0\, R\, B$;

$B\,1 \to 1\, R\, B$;

$A\,\lambda \to \lambda\, L\, A_1$ // переходим к последнему символу слова;

$B\,\lambda \to \lambda\, L\, B_1$;

$A_1\,0 \to 0"L\, A_2$ // дважды помечаем 0 и запоминаем состоянием A_2;

$A_1\,1 \to 0"L\, B_2$ // заменяем 1 на 0"и запоминаем состоянием B_2;

$A_2\,0 \to 0\, L\, A_2$ // двигаемся влево, пока не дойдем до помеченного ‘ символа в начале слова;

$A_2\,1 \to 1\, L\, A_2$;

$B_2\,0 \to 0\, L\, B_2$;

$B_2\,1 \to 1\, L\, B_2$;

$B_1\,0 \to 1"\, L\, A_2$ // заменяем 0 на 1" и запоминаем состоянием A_2;

$B_1\,1 \to 1"LB_2$ // дважды помечаем 1 и запоминаем состоянием B_2;

$A_2\,0` \to 0\, R\, S_0$ // снимаем ‘ с нуля и снова переходим к S_0;

$A_2\,1\text{`}\rightarrow 0\;R\;S_0$ // заменяем 1’ на ноль и снова переходим к S_0;

$B_2\,0\text{`}\rightarrow 1\;R\;S_0$ // заменяем 0’ на единицу и снова переходим к S_0;

$B_2\,1\text{`}\rightarrow 1\;R\;S_0$ // снимаем ‘ с единицы и снова переходим к S_0;

$S_0\,0\text{”}\rightarrow 0\;H\;\Omega$ // слово скопировано;

$S_0\,1\text{”}\rightarrow 1\;H\;\Omega$;

$A\;0\text{”}\rightarrow 0\;L\;A_1$ // дошли до первого скопированного символа;

$A\;1\text{”}\rightarrow 1\;L\;A_1$ // снимаем “ и возвращаемся к предыдущему;

$B\;0\text{”}\rightarrow 0\;L\;B_1$;

$B\;1\text{”}\rightarrow 1\;L\;B_1$;

$A_1 0\text{`}\rightarrow 0\;H\;\Omega$ // слово скопировано;

$A_1\,1\text{`}\rightarrow 1\;H\;\Omega$;

$B_1\,0\text{`}\rightarrow 0\;H\;\Omega$;

$B_1\,1\text{`}\rightarrow 1\;H\;\Omega$.

Задача 1.12. $A = \{0, 1\}$. Определить, является ли слово палиндромом. Если слово палиндром, завершить работу нормально (в состоянии Ω), если нет – аварийно (в состоянии Ω^1).

Решение. Предлагается следующий алгоритм: сравниваем по очереди символ из начала слова и соответствующий символ в конце слова: если они не совпадают, то слово не палиндром, если совпадают, то сравнение продолжается, пока все символы не будут проверены. Обратите внимание: в алгоритме учитывается, что слово может быть и чётной, и нечётной длины.

$S_0\;\sigma\rightarrow\sigma\;R\;S_0$;

$S_0\,0\rightarrow 0\text{`}\,R\;A$ // помечаем 0 и запоминаем состоянием A;

$S_0\,1\rightarrow 1\text{`}\,R\;B$ // помечаем 1 и запоминаем состоянием B;

$S_0\;\lambda\rightarrow\lambda\;H\;\Omega^1$ // пустое слово, вывод сообщения об ошибке;

$S_0\,0\text{”}\rightarrow 0\;H\;\Omega$ // слово является палиндромом;

$S_0\,1\text{”}\rightarrow 1\;H\;\Omega$;

$A\;0\rightarrow 0\;R\;A$ // двигаемся вправо, пока не дойдем до конца слова;

$A\;1\rightarrow 1\;R\;A$;

$B\;0\rightarrow 0\;R\;B$;

$B\;1\rightarrow 1\;R\;B$;

$A\;\lambda\rightarrow\lambda\;L\;A_1$ // переходим к последнему символу слова;

$B\;\lambda\rightarrow\lambda\;L\;B_1$;

$A_1\;0\rightarrow 0\text{”}\;LC$ // дважды помечаем 0 и переходим в состояние *C;*

$A_1\;1\rightarrow 1\;L\;D$ // слово не палиндром, двигаемся влево;

// чтобы снять пометку с нуля;

$B_1\;0\rightarrow 0\;L\;D$ // слово не палиндром, двигаемся влево;

// чтобы снять пометку с единицы;

$B_1 1\rightarrow 1\text{”}\;L\;C$ // помечаем 1 и переходим в состояние *C;*

// для движения влево;

$C\;0\rightarrow 0\;\;LC$ // двигаемся влево, пока не дойдем;

$C\;1\rightarrow 1\;L\;C$ // до помеченного ‘символа в начале слова;

$D\;0\rightarrow 0\;L\;D$;

$D\;1\rightarrow 1\;L\;D$;

$D\;0\text{`}\rightarrow 0\;H\;\Omega^1$ // слово не является палиндромом;

$D\;1\text{`}\rightarrow 1\;H\;\Omega^1$;

$C\;0\text{`}\rightarrow 0\;R\;S_0$ // снимаем ‘ с нуля и снова переходим к S_0;

$C\;1\text{`}\rightarrow 1\;R\;S_0$ // снимаем ‘ с единицы и снова переходим к S_0;

$A\;0\text{”}\rightarrow 0\;L\;A_1$ // дошли до первого скопированного символа;

$A\;1\text{”}\rightarrow 1\;L\;A_1$ // снимаем “ и возвращаемся к предыдущему;

$B\;0\text{”}\rightarrow 0\;L\;B_1$;

$B\ 1'' \rightarrow 1\ L\ B_1$;

$A_1\ 0` \rightarrow 0\ H\ \Omega$ // слово является палиндромом;

$A_1\ 1` \rightarrow 1\ H\ \Omega$;

$B_1\ 0` \rightarrow 0\ H\ \Omega$;

$B_1\ 1` \rightarrow 1\ H\ \Omega$.

Задача 1.13. $A = \{1\}$. На ленте в унарном коде записана псевдоразность чисел a и b (числа разделены знаком «$\dot{-}$»), записать результат вместо псевдоразности. Примечание: если число a больше b, то результат псевдоразности равен $a \dot{-} b$, если число a меньше или равно b, то результат псевдоразности равен нулю.

Решение. Предлагается следующий алгоритм: по очереди отмечаются единицы у числа справа от знака «$\dot{-}$» и соответствующие единицы слева. Если число справа оказывается больше, то на ленте записываем одну единицу (псевдоразность равна нулю).

$S_0\ \sigma \rightarrow \sigma\ R\ S_0$;

$S_0\ \lambda \rightarrow \lambda\ H\ \Omega^1$ // некорректное условие, нет знака «$\dot{-}$»;

$S_0\ 1 \rightarrow 1\ R\ S_0$ // двигаемся вправо, до знака «$\dot{-}$»;

$S_0\ \dot{-} \rightarrow \dot{-}\ R\ A$;

$A\ 1 \rightarrow 1`\ L\ B$ // помечаем первую непомеченную единицу второго числа и возвращаемся к первому числу;

$A\ \lambda \rightarrow \lambda\ L\ C$ // во втором числе не осталось непомеченных единиц (т.е. число a больше или равно b);

$B\ \dot{-} \rightarrow \dot{-}\ L\ B$ // двигаемся влево до первой непомеченной единицы;

$B\ 1 \rightarrow 1`\ R\ S_0$ // помечаем ближайшую единицу первого числа;

$B\ \sigma \rightarrow \sigma\ R\ E$ // в первом числе не осталось непомеченных единиц (т.е. число a меньше или равно b);

$A\ 1` \rightarrow 1`\ R\ A$ // пропускаем помеченные единицы;

$B\ 1` \rightarrow 1`\ R\ B$ // двигаемся вправо;

$S_0\ 1` \rightarrow 1`\ R\ S_0$

$C\ 1` \rightarrow \lambda\ L\ C$ // стираем все помеченные единицы;

$C\ \div \rightarrow \lambda\ L\ C$ // стираем знак «÷»;

$C\ 1 \rightarrow 1\ H\ \Omega$ // псевдоразность записана;

$C\ \sigma \rightarrow \sigma\ R\ F$ // не осталось единичек в первом числе;

$F\ \lambda \rightarrow 1\ H\ \Omega$ // записали единичку – унарный код числа ноль;

$D\ \lambda \rightarrow \lambda\ H\ \Omega$;

$C\ \lambda \rightarrow \lambda\ H\ \Omega$;

$E\ 1' \rightarrow 1\ R\ D$ // оставляем одну единицу, так как псевдоразность;

$E\ \div \rightarrow 1\ R\ D$ // равна нулю;

$D\ 1` \rightarrow \lambda\ R\ D$ // стираем все помеченные единицы;

$D\ \div \rightarrow \lambda\ R\ D$ // стираем знак «÷»;

$D\ 1 \rightarrow \lambda\ R\ D$ // стираем все помеченные единицы.

Задача 1.14. $A = \{0, 1\}$. Определить, каких символов в слове больше: ноликов или единиц. Если больше единичек, то в конце слова поставить символ «+», если ноликов – символ «–». Если символов равное количество – знак «=».

Решение. Предлагается следующий алгоритм: для каждого (первого встреченного) символа ищем парный, если он не найден, то таких символов в слове меньше, если все символы в слове оказались помечены, то их равное количество.

$S_0\ \sigma \rightarrow \sigma\ R\ S_0$;

$S_0\ 0 \rightarrow 0`\ R\ B$ // помечаем 0 и запоминаем состоянием B;

$S_0\ 1 \rightarrow 1`\ R\ A$ // помечаем 1 и запоминаем состоянием A;

$S_0\ \lambda \rightarrow \lambda\ H\ \Omega^1$ // пустое слово – вывод сообщения об ошибке;

$A\ 0 \rightarrow 0`\ L\ C$ // помечаем 0 и запоминаем состоянием C, что пара найдена;

$A\ 1 \rightarrow 1\ R\ A$ // ищем 0, двигаемся вправо;

$A\ 0` \rightarrow 0`\ R\ A$;

$A\ 1` \rightarrow 1`\ R\ A$;

$A\ \lambda \rightarrow +\ L\ D$ // единиц больше, двигаемся назад, чтобы;

// стереть штрихи;

$B\ 0 \rightarrow 0\ R\ B$ // ищем 1, двигаемся вправо;

$B\ 0` \rightarrow 0`\ R\ B$;

$B\ 1` \rightarrow 1`\ R\ B$;

$B\ 1 \rightarrow 1`\ L\ C$ // помечаем 1 и запоминаем состоянием C, что пара найдена;

$C\ 1` \rightarrow 1`L\ C$ // двигаемся в начало;

$C\ 0` \rightarrow 0`L\ C$;

$C\ 1 \rightarrow 1\ L\ C$;

$C\ 0 \rightarrow 0\ L\ C$;

$D\ 1` \rightarrow 1\ L\ D$;

$D\ 0` \rightarrow 0\ L\ D$;

$D\ 1 \rightarrow 1\ L\ D$;

$D\ 0 \rightarrow 0\ L\ D$;

$B\ \lambda \rightarrow -\ L\ D$ // нулей больше, двигаемся назад, чтобы стереть штрихи;

$D\ \sigma \rightarrow \sigma\ H\ \Omega$ // нулей больше, конец программы;

$C\ \sigma \rightarrow \sigma\ R\ S$ // перешли к началу слова;

$S\ 0 \rightarrow 0`\ R\ B$ //помечаем 0 и запоминаем состоянием B;

$S\ 1 \rightarrow 1`\ R\ A$ // помечаем 1 и запоминаем состоянием A;

$S\ 0` \rightarrow 0`R\ S$ // ищем непомеченный символ;

$S\ 1` \rightarrow 1`R\ S$;

$S\ \lambda \rightarrow = L\ D$ // нулей и единиц равное количество, двигаемся назад, чтобы стереть штрихи.

1.2. Многоленточные машины Тьюринга

1.2.1. Теоретическая основа

Многоленточная машина Тьюринга для каждой ленты в общем случае может иметь свой внешний алфавит. Ленты в машине движутся независимо друг от друга. Однако состояние для всех лент машины единое, по сути, это состояние управляющего механизма.

Для удобства предположим, что управляющий механизм подсоединен к набору окошек, в которые обозреваются непосредственно символы на лентах. Тогда считаем, что эти окошки на каждой из лент движутся независимо друг от друга. Виртуально этот процесс становится хоть как-то представимым, если вообразить наличие неких пружинок, которые соединяют между собой окошки и привязывают их к единому управляющему механизму. В этом случае нотация для записи команд в программе машины Тьюринга остается неизменной.

Многоленточная машина Тьюринга схематично изображается в виде лент с отмеченными окошками ячейками (рис. 1.12).

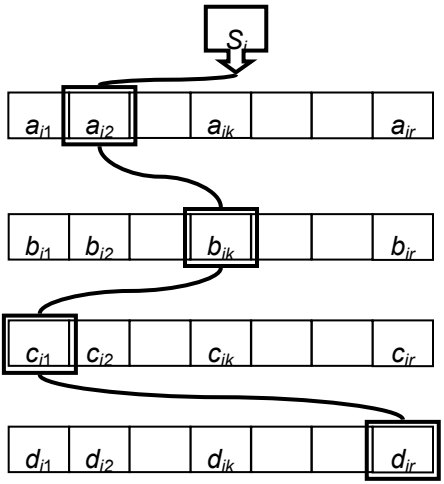

Рис. 1.12. Схематичное изображение многоленточной машины Тьюринга

1.2.2. Программы для многоленточных машин Тьюринга

Условимся обозначать буквой *A* внешний алфавит, т.е. те символы, из которых может состоять входное слово, σ – начальный символ, λ – пустой.

Условимся обозначать заглавными латинскими буквами с индексами внутренний алфавит (например, S_0, S_1, A_0, A_1 и т.д.), Ω – конечное состояние, Ω^1 – конечное состояние с выводом сообщения об ошибке.

Условимся записывать команду в следующем виде:

$A\ \{a, b, c\} \rightarrow \{a', b', c'\}\ \{(R, L, H), (R, L, H), (R, L, H)\}\ B,$

где: *A, B* – элементы внутреннего алфавита,

a, b, c, a' ,b', c' – элементы внешнего алфавита для данной ленты;

(*R, L, H*) – одна из возможных команд перемещения для каждого окошка:

R – перемещение окошка на одну ячейку вправо;

L – перемещение окошка на одну ячейку влево;

H – окошко остаётся неподвижным.

При формировании программы многоленточной машины следует обратить внимание на те же условия задачи, что и для классической машины Тьюринга:

- необходимо ли проверять, что входная строка не пустая;
- должно ли входное слово остаться неизменным в результате выполнения алгоритма.

Кроме того, желательно соблюдать те же рекомендации:

- выбирать названия элементов внутреннего алфавита в соответствии с действиями, которые должна выполнять машина, попав в данное состояние;
- минимизировать внутренний алфавит;
- по возможности не слишком сильно расширять внешний алфавит;
- минимизировать количество строк в алгоритме.

Задачи для самостоятельной работы

Задача 1.15. $A = \{0, 1\}$. Машина двухленточная. Дано слово, оно записано на первой ленте. Скопировать это слово на вторую ленту в обратном порядке.

Решение. Предлагается следующий алгоритм: находим конец слова и копируем, посимвольно продвигаясь по первой ленте влево и по второй вправо.

$S_0\ \{\sigma, \sigma\} \rightarrow \{\sigma, \sigma\}\ \{R, R\}\ S_0$	// сдвигаемся вправо без изменения состояния;
$S_0\ \{\lambda, \lambda\} \rightarrow \{\lambda, \lambda\}\ \{H, H\}\ \Omega^1$	// обе ленты пустые, остановка, вывод сообщения об ошибке;
$S_0\ \{0, \lambda\} \rightarrow \{0, \lambda\}\ \{R, H\}\ S_1$	// сдвигаемся до конца слова по первой ленте;
$S_0\ \{1, \lambda\} \rightarrow \{1, \lambda\}\ \{R, H\}\ S_1;$	
$S_1\ \{0, \lambda\} \rightarrow \{0, \lambda\}\ \{R, H\}\ S_1;$	
$S_1\ \{1, \lambda\} \rightarrow \{1, \lambda\}\ \{R, H\}\ S_1;$	
$S_1\ \{\lambda, \lambda\} \rightarrow \{\lambda, \lambda\}\ \{L, H\}\ S_2$	// дошли до конца слова, начинаем копирование;
$S_2\ \{0, \lambda\} \rightarrow \{0, 0\}\ \{L, R\}\ S_2$	// копируем слово в обратном

порядке, по первой ленте;

$S_2\ \{1, \lambda\} \rightarrow \{1, 1\}\ \{L, R\}\ S_2$ // сдвигаемся влево по первой ленте, по второй – вправо;

$S_0\ \{\sigma, \lambda\} \rightarrow \{\sigma, \lambda\}\ \{H, H\}\ \Omega$ // слово скопировано.

Задача 1.16. $A = \{0, 1\}$. Машина трехленточная. Даны два числа в двоичном коде, они записаны на первой и второй ленте соответственно. Считается, что слова записаны нормально: т.е. оба слово всегда есть и начинаются они с единицы (исключение только для числа ноль). Определить, чему равна сумма этих двух чисел, и результат записать на третью ленту.

Решение. Предлагается следующий алгоритм: суммирование будем проводить поразрядно, начиная с последнего разряда. Если в ходе суммирования разрядов не хватит, то результат будем сдвигать на одну ячейку вправо.

$S_0\ \{\sigma, \sigma, \sigma\} \rightarrow \{\sigma, \sigma, \sigma\}\ \{R, R, R\}\ S_0$ // сдвигаемся вправо без изменения состояния;

$S_0\ \{\lambda, \lambda, \lambda\} \rightarrow \{\lambda, \lambda, \lambda\}\ \{H, H, H\}\ \Omega^1$ // обе ленты пустые, остановка, сообщение об ошибке;

$S_0\ \{\lambda, 0, \lambda\} \rightarrow \{\lambda, 0, \lambda\}\ \{H, H, H\}\ \Omega^1$
$S_0\ \{\lambda, 1, \lambda\} \rightarrow \{\lambda, 1, \lambda\}\ \{H, H, H\}\ \Omega^1$ // первая лента пустая, остановка, сообщение об ошибке;

$S_0\ \{0, \lambda, \lambda\} \rightarrow \{0, \lambda, \lambda\}\ \{H, H, H\}\ \Omega^1$
$S_0\ \{1, \lambda, \lambda\} \rightarrow \{1, \lambda, \lambda\}\ \{H, H, H\}\ \Omega^1;$ // вторая лента пустая, остановка, сообщение об ошибке;

$S_0\ \{0, 0, \lambda\} \rightarrow \{0, 0, \lambda\}\ \{R, R, R\}\ S_1$
$S_0\ \{0, 1, \lambda\} \rightarrow \{0, 1, \lambda\}\ \{R, R, R\}\ S_1;$
$S_0\ \{1, 0, \lambda\} \rightarrow \{1, 0, \lambda\}\ \{R, R, R\}\ S_1;$
$S_0\ \{1, 1, \lambda\} \rightarrow \{1, 1, \lambda\}\ \{R, R, R\}\ S_1;$
$S_1\ \{0, 0, \lambda\} \rightarrow \{0, 0, \lambda\}\ \{R, R, R\}\ S_1;$ // сдвигаемся вправо по всем лентам;

$S_1\ \{0, 1, \lambda\} \rightarrow \{0, 1, \lambda\}\ \{R, R, R\}\ S_1$;

$S_1\ \{1, 0, \lambda\} \rightarrow \{1, 0, \lambda\}\ \{R, R, R\}\ S_1$;

$S_1\ \{1, 1, \lambda\} \rightarrow \{1, 1, \lambda\}\ \{R, R, R\}\ S_1$;

$S_1\ \{\lambda, 0, \lambda\} \rightarrow \{\lambda, 0, \lambda\}\ \{H, R, R\}\ S_1$ // сдвигаемся вправо по 2 и 3 лентам;

$S_1\ \{\lambda, 1, \lambda\} \rightarrow \{\lambda, 1, \lambda\}\ \{H, R, R\}\ S_1$

$S_1\ \{0, \lambda, \lambda\} \rightarrow \{0, \lambda, \lambda\}\ \{R, H, R\}\ S_1$ // сдвигаемся вправо по 1 и 3 лентам;

$S_1\ \{1, \lambda, \lambda\} \rightarrow \{1, \lambda, \lambda\}\ \{R, H, R\}\ S_1$;

$S_2\ \{0, 0, \lambda\} \rightarrow \{0, 0, 0\}\ \{L, L, L\}\ S_2$ // сдвигаемся влево по всем лентам;

$S_2\ \{0, 1, \lambda\} \rightarrow \{0, 1, 1\}\ \{L, L, L\}\ S_2$;

$S_2\ \{1, 0, \lambda\} \rightarrow \{1, 0, 1\}\ \{L, L, L\}\ S_2$;

$S_2\ \{1, 1, \lambda\} \rightarrow \{1, 1, 0\}\ \{L, L, L\}\ S_3$;

$S_1\ \{\lambda, \lambda, \lambda\} \rightarrow \{\lambda, \lambda, \lambda\}\ \{L, L, L\}\ S_2$ //переходим к суммированию;

$S_2\ \{\sigma, 0, \lambda\} \rightarrow \{\sigma, 0, 0\}\ \{H, L, L\}\ S_2$ // сдвигаемся влево по 2 и 3 лентам;

$S_2\ \{\sigma, 1, \lambda\} \rightarrow \{\sigma, 1, 1\}\ \{H, L, L\}\ S_2$;

$S_2\ \{0, \sigma, \lambda\} \rightarrow \{0, \sigma, 0\}\ \{L, H, L\}\ S_2$ // сдвигаемся влево по 1 и 3 лентам) ;

$S_2\ \{1, \sigma, \lambda\} \rightarrow \{1, \sigma, 1\}\ \{L, H, L\}\ S_2$;

$S_2\ \{\sigma, \sigma, \sigma\} \rightarrow \{\sigma, \sigma, \sigma\}\ \{H, H, H\}\ \Omega$ // операция суммирования выполнена;

$S_3\ \{0, 0, \lambda\} \rightarrow \{0, 0, 1\}\ \{L, L, L\}\ S_2$ // сдвигаемся влево по всем лентам;

$S_3\ \{0, 1, \lambda\} \rightarrow \{0, 1, 0\}\ \{L, L, L\}\ S_3$;

$S_3\ \{1, 0, \lambda\} \rightarrow \{1, 0, 0\}\ \{L, L, L\}\ S_3$;

$S_3\ \{1, 1, \lambda\} \rightarrow \{1, 1, 1\}\ \{L, L, L\}\ S_3$;

$S_3\ \{\sigma, 0, \lambda\} \rightarrow \{\sigma, 0, 1\}\ \{H, L, L\}\ S_2$ // сдвигаемся влево по 2 и 3 лентам;

$S_3\ \{\sigma, 1, \lambda\} \rightarrow \{\sigma, 1, 0\}\ \{H, L, L\}\ S_3$;

$S_3\ \{0, \sigma, \lambda\} \rightarrow \{0, \sigma, 1\}\ \{L, H, L\}\ S_2$ // сдвигаемся влево по 1 и 3

$S_3\ \{1, \sigma, \lambda\} \rightarrow \{1, \sigma, 0\}\ \{L, H, L\}\ S_3$; лентам;

$S_3\ \{\sigma, \sigma, \sigma\} \rightarrow \{\ \sigma, \sigma, \sigma\}\ \{H, H, R\}\ A_1$ //если в ходе сложения разрядов не хватило;

$A_1\ \{\sigma, \sigma, 0\} \rightarrow \{\ \sigma, \sigma, 1\}\ \{H, H, R\}\ A_0$ // записали 1, сдвигаем 0;

$A_1\ \{\sigma, \sigma, 1\} \rightarrow \{\ \sigma, \sigma, 1\}\ \{H, H, R\}\ A_1$ // записали 1, сдвигаем 1;

$A_0\ \{\sigma, \sigma, 0\} \rightarrow \{\ \sigma, \sigma, 0\}\ \{H, H, R\}\ A_0$ // записали 0, сдвигаем 0;

$A_0\ \{\sigma, \sigma, 1\} \rightarrow \{\ \sigma, \sigma, 0\}\ \{H, H, R\}\ A_1$ // записали 0, сдвигаем 1;

$A_1\ \{\sigma, \sigma, \lambda\} \rightarrow \{\ \sigma, \sigma, 1\}\ \{H, H, H\}\ \Omega$ // записали 1, операция суммирования выполнена;

$A_0\ \{\sigma, \sigma, \lambda\} \rightarrow \{\ \sigma, \sigma, 0\}\ \{H, H, H\}\ \Omega$ // записали 0, операция суммирования выполнена.

Задача 1.17. $A = \{1\}$. Машина трехленточная. Даны два числа в унарном коде, они записаны на первой и второй ленте соответственно. Определить, чему равно произведение этих двух чисел и результат записать на третью ленту.

Решение. Предлагается следующий алгоритм: умножение будем проводить с учётом того, что число n кодируется $n + 1$ единицей.

$S_0\ \{\sigma, \sigma, \sigma\} \rightarrow \{\sigma, \sigma, \sigma\}\ \{R, R, R\}\ S_0$ // сдвигаемся вправо без изменения состояния;

$S_0\ \{\lambda, \lambda, \lambda\} \rightarrow \{\lambda, \lambda, \lambda\}\ \{H, H, H\}\ \Omega^1$ // обе ленты пустые, остановка, вывод сообщения об ошибке;

$S_0\ \{\lambda, 1, \lambda\} \rightarrow \{\lambda, 1, \lambda\}\ \{H, H, H\}\ \Omega^1$ // вторая лента пустая, остановка, вывод сообщения об ошибке;

$S_0\ \{1, \lambda, \lambda\} \rightarrow \{1, \lambda, \lambda\}\ \{H, H, H\}\ \Omega^1$ // первая лента пустая,

	остановка, вывод сообщения об ошибке;
$S_0\ \{1, 1, \lambda\} \rightarrow \{1, 1, 1\}\ \{R, R, H\}\ S_1$	// сдвигаемся вправо по первой и второй лентам, так как число n представлено n+1 единицей;
$S_1\ \{1, 1, \lambda\} \rightarrow \{1, 1', 1\}\ \{H, R, R\}\ S_1$	// сдвигаемся вправо по второй и третьей лентам, копируя единицы со второй ленты на третью;
$S_1\ \{\lambda, \lambda, \lambda\} \rightarrow \{\lambda, \lambda, \lambda\}\ \{H, L, H\}\ S_3$	// возвращаемся влево по второй ленте;
$S_1\ \{1, \lambda, \lambda\} \rightarrow \{1, \lambda, \lambda\}\ \{H, L, H\}\ S_2$;	
$S_2\ \{1, 1', \lambda\} \rightarrow \{1, 1, \lambda\}\ \{H, L, H\}\ S_2$	// снимаем пометки с единиц;
$S_2\ \{1, 1, \lambda\} \rightarrow \{1, 1, \lambda\}\ \{R, R, H\}\ S_1$	// вернулись в начало, сдвигаемся на одну ячейку по первой ленте;
$S_3\ \{1, 1', \lambda\} \rightarrow \{1, 1, \lambda\}\ \{H, L, H\}\ S_3$	// снимаем пометки с единиц;
$S_3\ \{1, 1, \lambda\} \rightarrow \{1, 1, \lambda\}\ \{H, H, H\}\ \Omega$	// операция умножения выполнена.

1.3. Нормальные алгоритмы

1.3.1. Теоретическая основа

Основная операция при работе нормальных алгоритмов Маркова – переработка слов в некотором алфавите. Эта переработка заключается в производстве некоторого количества замен определенных последовательностей символов. Данные замены совершаются в СТРОГО определенном порядке, а именно: *после каждой замены алгоритм читается с самого начала, а слово анализируется с самого первого символа.* В отличие от машин

Тьюринга, алгоритмы Маркова выполняются без какого-либо устройства, осуществляющего движения и имеющего внутреннюю память. В данном случае мы можем оперировать только ленточными символами. Сама лента в этом случае не разделяется на строгие ячейки, а имеет гибкую основу, что позволяет ей растягиваться и сжиматься исходя из того, увеличивается в слове число символов или уменьшается.

1.3.2. Программы в формате алгоритмов Маркова

Условимся записывать команду в следующем виде:

$$\{a_i\} \rightarrow \{b_j\}\ [\bullet],$$

где $\{a_i\}$ – последовательность символов, которая ищется в слове;

$\rightarrow$ – знак перехода к операции записи;

$\{b_j\}$ – последовательность символов, которая записывается вместо найденной;

[•] – знак принудительного окончания алгоритма (необязательный параметр).

Программа (алгоритм) представляет собой совокупность строк указанного вида, причем порядок строк имеет важнейшее значение. Окончание работы алгоритма происходит в тот момент, когда выполняется строка, содержащая знак принудительной остановки, либо тогда, когда более ни одна строка не может быть выполнена (в слове нет ни одной из искомых последовательностей символов).

Например, алгоритм, состоящий из одной строки вида

$$0 \rightarrow *$$

будучи примененным к слову в алфавите {0,1}, заменит все нули на звездочки.

В свою очередь алгоритм

$$0 \rightarrow * \bullet$$

будучи примененный к слову в алфавите {0, 1}, заменит на звездочку только первый встреченный ноль.

Довольно сложная для реализации на машинах Тьюринга задача сортировки слова по возрастанию, допустим в алфавите {0, 1, 2}, решается при помощи Маркова весьма изящно:

$$20 \rightarrow 02$$
$$10 \rightarrow 01$$
$$21 \rightarrow 12$$

Существует также специальный зарезервированный символ, именуемый «пустое слово», он обозначается как Λ. Этот символ служит для обозначения пустого слова, а также, по своему смыслу, содержится между всеми символами слова, причем в неограниченном количестве. Таким образом, можно считать, что входная строка *abcde* формально представима как Λ*a*Λ*b*Λ*c*Λ*d*Λ*e*Λ, причем при удалении символа Λ посредством строки вида

$$\Lambda \rightarrow$$

ничего не изменится. Иными словами, добавление или удаление символа Λ является совершенно бессмысленной процедурой, он как бы «прозрачный» и при этом «самовоспроизводящийся». По сути, этот символ обозначает «пустоту», «ничто» и поэтому есть везде. Единственный практический смысл данного символа проявляется тогда, когда необходимо написать вместо пустого слова какой-либо символ или необходимо какой-либо символ написать строго в начале данного слова. Оба эти действия выполняются с помощью команды

$$\Lambda \rightarrow 0.$$

При составлении алгоритма следует обратить внимание на то, должно ли по условию задачи входное слово остаться неизменным в результате выполнения алгоритма.

Кроме того, желательно соблюдать рекомендации:

- выбирать названия дополнительных элементов алфавита в соответствии с логикой производимых замен или текущего действия;
- по возможности не слишком сильно расширять имеющийся алфавит;
- минимизировать количество строк в алгоритме.

Помимо этих, уже привычных правил следует обратить особое внимание на то, в правильном ли порядке находятся строки алгоритма. Это один из важнейших аспектов работы алгоритма и источник большинства ошибок. Как правило, схема такая:

1) строки, содержащие условия принудительного окончания алгоритма;
2) основное тело программы (алгоритма);

3) начальные условия работы алгоритма (добавление флага, поиск начала слова и т.д.).

Разрабатывать такие программы проще «с конца», т.е. сначала пишется блок инициализации (его место в программе скорее всего будет в самом низу), затем основное тело (в большинстве случаев вставляется перед блоком инициализации), и в конце дописываются строки окончания (они при этом будут в самом верху программы, т.е. считываться при работе алгоритма первыми).

Задачи для самостоятельной работы

Задача 1.18. $A = \{0, 1\}$. Заменить произвольное слово на символ «0».

Решение.

$00 \rightarrow 0$ // два стоящих подряд нуля меняются на один, строчка работает до тех пор, пока в слове все нули не станут одиночными;

$1 \rightarrow 0$ // когда все нули станут одиночными, сработает данная строка – первая единица заменится на ноль (после этого снова один раз сработает первая строка, затем один раз вторая и т.д.

Такой, казалось бы, странный способ предложен для того, чтобы алгоритм завершал работу без принудительной остановки независимо от того, есть ли в исходном слове нули или единицы.

Можно написать программу иначе:

$0 \rightarrow$ // уничтожаем все нули;

$1 \rightarrow$ // уничтожаем все единицы;

$\Lambda \rightarrow 0$ // на пустом месте ставим ноль.

Задача 1.19. $A = \{0, 1\}$. Удвоить все символы в слове.

Решение. Предлагается следующий алгоритм: найти начало слова, затем удваивать все символы.

0→00 // * пробегает по слову слева направо, удваивая нолики;

1→11 // * пробегает по слову слева направо, удваивая единицы;

$* \rightarrow \bullet$ // слово закончилось;

$\Lambda \rightarrow *$ // находим начало слова, эта строка выполнится первой.

Задача 1.20. $A = \{0, 1\}$. Удалить каждый третий символ.

Решение. Предлагается следующий алгоритм: найти начало слова, затем отсчитывать каждый третий символ, меняя вид челнока.

$A0 \rightarrow 0B$ // прошли первый символ, челнок изменился на B;

$A1 \rightarrow 1B$

$B0 \rightarrow 0C$ // прошли второй символ, челнок изменился на C;

$B1 \rightarrow 1C$

$C0 \rightarrow A$ // удалили третий символ, вернули челноку исходный вид;

$C1 \rightarrow A$

$A \rightarrow \bullet$ // слово закончилось;

$B \rightarrow \bullet$

$C \rightarrow \bullet$

$\Lambda \rightarrow A$ // находим начало слова, создаем челнок A.

Задача 1.21. $A = \{0, 1\}$. Поставить в конце слова «+», если в нем есть хотя бы один ноль, поставить «–», если это не так.

Решение. Предлагается следующий алгоритм: найти начало слова (поставить «–»), затем, двигаясь по слову, искать ноль.

$-0 \rightarrow 0+$ // нашли ноль, изменяем челнок с «–» на «+»;

$-1 \rightarrow 1-$ // сдвигаем челнок вправо;

$+0 \rightarrow 0+$ // сдвигаем челнок вправо;

$+1 \rightarrow 1+$

$- \rightarrow - \bullet$ // слово закончилось, нулей нет;

$+ \rightarrow + \bullet$ // слово закончилось, есть не менее одного нуля;

Λ → – //находим начало слова, создаем челнок -.

Задача 1.22. *A* = {0, 1}. Переработать слово в «+», если в нем есть хотя бы два нуля (неважно расположены они рядом или нет), и в «–», если это не так (т.е. исходное слово удаляется).

Решение. Предлагается следующий алгоритм: найти начало слова (поставить «–»), затем, удаляя символы, искать ноль.

–0 → * // нашли первый ноль;

–1 → – // сдвигаем челнок вправо;

*0 → + // нашли второй ноль;

*1 → * // сдвигаем челнок вправо;

+0 → +

+1 → +

– → – • // слово закончилось, нулей нет;

* → – • // слово закончилось, ноль только один;

+ → + • // слово закончилось, есть не менее двух нулей;

Λ → – // находим начало слова, создаем челнок «–».

Задача 1.23. *A* = {0, 1}. Поставить в конце слова «+», если в нем рядом встречаются хотя бы два нуля (т.е. они соседние) и «–» если это не так.

Решение. Предлагается следующий алгоритм: найти начало слова, двигаясь по слову, искать пару нулей, поставить «+», если пара есть. Если пары нулей в слове нет, то поставить «–».

–00 → 00+ // нашли два соседних нуля, изменили челнок;

–1 → 1– // сдвигаем челнок вправо;

–0 → 0–

+0 → +

+1 → +

– → – • // слово закончилось, пары нулей нет;

+ →+ • // слово закончилось, есть пара нулей;

Λ → – // находим начало слова, создаем челнок «–».

Задача 1.24. $A = \{0, 1\}$. Переработать произвольное слово в «+», если в нем рядом встречаются точно два нуля (т.е. они соседние) и больше нулей в слове нет и в «–», если это не так.

Решение. Предлагается следующий алгоритм: найти начало слова, двигаясь по слову, искать пару нулей, поставить «+», если пара есть. Если пары нулей в слове нет или встречаются ещё нули, то поставить «–».

*00 → + // нашли два соседних нуля, изменили челнок;

*1 → * // сдвигаем челнок вправо;

*0 → – // сдвигаем челнок вправо, изменили челнок;

+0 → –

+1 → + // сдвигаем челнок вправо, найдена пара нулей;

–0 → – // сдвигаем челнок вправо;

–1 → –

– → – • // слово закончилось, нулей больше одной пары;

* → – • // слово закончилось, нулей нет;

+ → + • // слово закончилось, есть ровно одна пара нулей;

Λ → * // находим начало слова, создаем челнок *.

Задача 1.25. $A = \{a, b\}$. Скопировать слово в прямом порядке после слова.

Решение. Предлагается следующий алгоритм: найти начало слова, двигаясь по слову, дублировать каждый символ, затем сдвигать копию вправо.

$* a \quad \rightarrow a\, a_1 *$ // копируем символы;

$* b \rightarrow b\, b_1 *$

$a_1 b \rightarrow b\, a_1$ // сдвигаем скопированные символы вправо;

$a_1\, a \rightarrow a\, a_1$

$b_1\, b \rightarrow b\, b_1$

$b_1\, a \rightarrow a\, b_1$

$a_1 \rightarrow a$ // убираем индексы у скопированных символов;

$b_1 \rightarrow b$

$a * \rightarrow a \bullet$ // слово скопировано;

$b * \rightarrow b \bullet$

$\Lambda \rightarrow *$ // находим начало слова, создаем челнок *.

Задача 1.26. $A = \{a, b\}$. Скопировать слово в обратном порядке после слова.

Решение. Предлагается следующий алгоритм: найти начало слова, двигаясь по слову, дублировать каждый символ, в итоге в конце слова будет стоять символ *. Затем символы-копии перемещаются в конец слова в том же порядке (т.е. пока в прямом) и останавливаются у символа *. Затем за звездочку «перепрыгивает» самый правый символ копии, после этого с него снимается пометка. Теперь следующий символ после перехода правее *» сможет перейти и через имеющиеся символы и дойти до конца слова – таким образом будет реализовано копирование именно в обратном порядке

$* a \rightarrow a\, a_1 *$ // копируем символы;

$* b \rightarrow b\, b_1 *$

$a_1 b \rightarrow b\, a_1$ // сдвигаем скопированные символы вправо;

$a_1\, a \rightarrow a\, a_1$

$b_1\, b \rightarrow b\, b_1$

$b_1\, a \rightarrow a\, b_1$

$*a_1 \rightarrow *a$ // убираем индексы у символа, который правее *;

$*b_1 \rightarrow *b$

$a_1* \rightarrow * a_1$ // символ «перепрыгивает» за звездочку;

$b_1 * \rightarrow * b_1$

$a * \rightarrow a \bullet$ // слово скопировано;

$b * \rightarrow b \bullet$

$\Lambda \rightarrow *$ // находим начало слова, создаем челнок *.

Задача 1.27. $A = \{1\}$. Вычислить сумму двух натуральных чисел в унарном коде. Результат записать вместо слова.

Решение. В унарном коде каждое число представлено числом единиц в количестве, на одну больше номинального значения. Значит всего имеется таких единиц на две больше, чем нужно. Убираем плюс и стираем лишнюю единицу

$$1+1 \rightarrow 1$$

Задача 1.28. $A = \{1\}$. Вычислить псевдоразность двух натуральных чисел в унарном коде. Результат записать вместо слова.

Решение. Из левого и правого слова синхронно убираем одинаковое количество единиц. Если в левом слове единиц было больше, то стираем значок $\div$ и дописываем к ответу одну единицу ввиду унарного кодирования. Иначе – стираем все единицы, значок $\div$ и одну единицу, кодируя таким образом ноль.

$1 \div 1 \rightarrow \div$ // убираем по 1 из каждого числа;

$\div 1 \rightarrow \div$ // если число справа больше числа слева;

$\div \rightarrow 1\bullet$ // добавляем единицу.

Задача 1.29. $A = \{1\}$. Вычислить разность двух натуральных чисел в унарном коде. Результат записать вместо слова.

Решение. Аналогично алгоритму в задаче 1.28, но если в правом слове единичек больше, то в начале слова записываем – 1.

1–1 → – // убираем по 1 из каждого числа;

1– → 11• // если число слева больше числа справа;

–1 → –11 • // если число справа больше числа слева;

– → 1• // добавляем единицу.

Задача 1.30. *A* = {*a, b*}. Подсчитать количество символов *a*. Результат записать после слова в унарном коде.

Решение. Двигаясь сначала слова челночком *, после каждой буквы *a* порождаем «1», затем все единички перемещаются к концу слова. Дополнительно записываем одну единичку для унарного кодирования и останавливаем алгоритм.

* *a* → *a* 1 * // находим *a* – записываем 1;

* *b* → *b* * // сдвигаем челнок вправо;

1 *b* → *b* 1 //сдвигаем единицу вправо;

1 *a* → *a* 1

1 * → 11 • // добавляем к количеству а одну единицу;

Λ → * // находим начало слова, добавляем челнок.

Задача 1.31. *A* = {*a, b*}. Переработать слово. Определить, каких символов *a* или *b* в слове больше. Если больше *a*, то записать вместо слова «+», если больше *b*, то записать вместо слова «–». Если поровну – записать вместо слова «=».

Решение. Устраняем все парные символы, после этого остаются либо только *a*, либо только *b*. Последовательность одинаковых символов постепенно преобразуется в тот же одиночный символ, который заменяется на «+» или «–». Если символов не осталось, значит, их было равное количество, и надо записать «=».

a b → // убираем символы, для которых есть пара;

b a →

$b\ b \rightarrow b$ //убираем оставшиеся одинаковые символы;

$a\ a \rightarrow a$

$a \rightarrow +\bullet$ //оставшийся символ показывает, каких символов было в слове больше;

$b \rightarrow -\bullet$

$\Lambda \rightarrow =\bullet$ //если символов не осталось, значит, их было равное количество.

Задача 1.32. $A = \{a, b, c\}$. Любое количество идущих подряд одинаковых символов заменить на тройку этих символов (один символ тоже заменяется).

Решение. Сначала удаляем все последовательности одинаковых символов, формируя скелет слова (все символы в нем одиночные), затем каждый одиночный символ заменяется на тройку помеченных символов, в конце с помощью челнока * удаляем пометки.

$*\ c_1 \rightarrow c\ *$ // убираем пометки (единички) с символов;

$*\ b_1 \rightarrow b\ *$

$*\ a_1 \rightarrow a\ *$

$* \rightarrow \bullet$ //замена выполнена;

$a\ a \rightarrow a$ //пары заменяем на один символ;

$b\ b \rightarrow b$

$c\ c \rightarrow c$

$a \rightarrow a_1\ a_1\ a_1$ // символ заменяем на тройку помеченных символов;

$b \rightarrow b_1\ b_1\ b_1$

$c \rightarrow c_1\ c_1\ c_1$

$\Lambda \rightarrow *$ //находим начало слова, создаем челнок.

1.4. Преобразования машин Тьюринга

Пусть в исходной машине Тьюринга **A** n состояний и m символов.

Теорема 1.1 (теорема Шеннона № 1). Всякая машина Тьюринга **A** может быть преобразована в эквивалентную машину **B** не более чем с двумя внутренними состояниями.

При этом сокращение количества состояний компенсируется расширением внешнего алфавита: кроме образов исходных m символов, добавляется $4mn$ специальных символов. Таким образом, в машине **B** будет $4mn + m$ символов.

Теорема 1.2 (теорема Шеннона № 2). Всякая машина Тьюринга **A** может быть преобразована в эквивалентную машину **C** не более чем с двумя знаками внешнего алфавита.

При этом сокращение количества символов компенсируется расширением внутреннего алфавита: кроме образов исходных n состояний, добавляется $8mn$ специальных состояний. Таким образом, в машине **C** будет $8mn + n$ состояний.

1.5. Алгоритмически неразрешимые задачи

После того как Тьюрингом была предложена столь удобная формальная модель алгоритма, возник вопрос о границах ее применимости. Если любой алгоритм можно преобразить в машину Тьюринга, то возникает вопрос о том, для любой ли задачи, вообще говоря, существует алгоритм решения. В курсе рассматриваются примеры алгоритмически неразрешимых проблем и доказательства их неразрешимости.

Теорема 1.3. Задача об остановке произвольной машины Тьюринга на произвольном входном слове алгоритмически неразрешима.

Иными словами, нельзя придумать универсальный алгоритм, в результате выполнения которого будет получен однозначный ответ: «да» – если произвольная машина T остановится на ленте с произвольным входным словом t и «нет» – иначе.

Теорема 1.4. Задача об остановке произвольной машины Тьюринга на пустой ленте алгоритмически неразрешима.

Теорема 1.5. Задача о печатании данного символа на чистой ленте бесконечно много раз алгоритмически неразрешима.

Можно обобщить эти проблемы, для чего приведем следующие определения.

Две машины Тьюринга, имеющие один и тот же внешний алфавит, будем называть ***взаимозаменяемыми***, если, каково бы ни было слово в их общем алфавите, не содержащее пустого символа, они либо перерабатывают его в одно и то же слово, либо обе к нему неприменимы (т.е. не останавливаются при его обработке).

Свойство машин Тьюринга называется ***инвариантным***, если любые две взаимозаменяемые машины либо обе обладают этим свойством, либо не обладают.

Свойство машин Тьюринга называется ***нетривиальным***, если существуют как машины, обладающие этим свойством, так и не обладающие им.

Теорема 1.6 (теорема Райса). Ни для какого нетривиального инвариантного свойства машин Тьюринга не существует алгоритма, позволяющего для любой машины Тьюринга узнать, обладает ли она этим свойством.

1.6. Эффективная перечислимость и распознаваемость множеств

Множество (по Тьюрингу) – это объединение в одно общее объектов, хорошо различимых нашей интуицией или нашей мыслью.

Множество (по Кантору) – совокупность объектов безразлично какой природы, неизвестно существующих ли, рассматриваемая как единое целое.

Эффективно перечислимым множеством называется множество, элементы которого можно перечислить по алгоритму (пронумеровать натуральным рядом без пропусков и повторений).

Подмножество B множества A ***эффективно распознается*** в A, если существует алгоритм, позволяющий однозначно для каждого элемента множества A определить, принадлежит ли данный элемент множеству B или дополнению B до A.

Теорема 1.7. Множество машин Тьюринга эффективно перечислимо.

Для доказательства этого используется следующая идея: описать произвольную машину Тьюринга некоторым числом, которое эффективно распознаётся среди натуральных чисел. Тогда, применив алгоритм распознавания к натуральному ряду, удастся перенумеровать все машины Тьюринга. А это будет означать, что они эффективно перечислимы. Произведём кодировку, для чего перечислим и пронумеруем состояния (символы внутреннего алфавита) и закодируем их единичками, пронумеруем ленточные знаки (символы внешнего алфавита) и закодируем их двойками. Символы, определяющие движение управляющей головки получат коды, состоящие из троек. Стрелку в строке таблицы обозначим четверкой, а разделитель между строк (переход на новую строку) – пятеркой. Например, программа состоящая из одной строки вида $A\ a \to b\ \mathrm{L}\ B$ получит код 111222422223311115.

Теорема 1.8. Множество алгоритмов Маркова эффективно перечислимо.

Для этого используется следующая идея: описать произвольный алгоритм некоторым числом, которое эффективно распознаётся среди натуральных чисел. Тогда, применив алгоритм распознавания к натуральному ряду, удастся перенумеровать все нормальные алгоритмы. А это будет означать, что они эффективно перечислимы. Произведём кодировку. Пронумеруем символы алфавита. Первые три символа служебные, для них зарезервируем числа 1, 2, 3: символ $\to$ закодируем «1», пустой символ – «2», символ принудительной остановки – «3». Остальные символы получают следующие вакантные номера: a (x_1) – «4», b (x_2) – «5», …, (x_k) – «k + 3». Нормальный алгоритм представляет собой набор строк, например:

$ab \to c$

$\Lambda \to d$ •

Возьмём первую строчку алгоритма Маркова и представим её следующим образом: $2^4 \cdot 3^5 \cdot 5^1 \cdot 7^6 = A$. Полученное натуральное число A – код данной строчки (такая нумерация называется Гёделевой). Аналогично поступаем со второй строчкой алгоритма: $2^2 \cdot 3^1 \cdot 5^7 \cdot 7^3 = B$ – код второй строчки. Далее формируем кодовое

число всего алгоритма. Для этого выпишем ряд простых чисел и возведем каждое число в соответствующую данной строчке степень, а затем, перемножив эти числа, получим кодовый номер алгоритма: $2^A \bullet 3^B \ldots = K$.

Таким образом, алгоритм представлен в виде произведения простых чисел, взятых в степенях, соответствующих коду каждой строки. Полученное число K является кодом данного алгоритма. При этом по данному коду можно восстановить весь алгоритм.

Теорема 1.9 (теорема Поста). Если множество A эффективно перечислимо, то подмножество B эффективно распознается в A тогда и только тогда, когда B и $A \backslash B$ оба эффективно перечислимы.

Теорема 1.10. Множество останавливающихся машин Тьюринга эффективно перечислимо.

Теорема 1.11. Множество не останавливающихся машин Тьюринга невозможно эффективно перечислить.

Проверочные задания и вопросы

Задача П.1.1. Запишите конфигурацию машины Тьюринга с данной функциональной схемой на третьем такте её работы для входного слова {σ001122}.

$S_0\ \sigma \rightarrow \sigma\ R\ S_a$;
$S_a\ 0 \rightarrow 1\ R\ S_a$;
$S_a\ 1 \rightarrow 0\ R\ S_a$;
$S_a\ 2 \rightarrow 2\ H\ \Omega$;
$S_a\ \lambda \rightarrow 2\ H\ \Omega$.

Задача П.1.2. Программа машины Тьюринга выглядит следующим образом:

$S_0\ \sigma \rightarrow \sigma\ R\ S_a$;
$S_a\ 0 \rightarrow 0\ R\ S_b$;
$S_a\ 1 \rightarrow 0\ R\ S_b$;
$S_b\ 0 \rightarrow 0\ R\ S_c$;
$S_b\ 1 \rightarrow 0\ R\ S_c$;
$S_c\ 0 \rightarrow 1\ R\ S_a$;
$S_c\ 1 \rightarrow 1\ R\ S_a$;
$S_a\ \lambda \rightarrow 0\ R\ S_b$;
$S_b\ \lambda \rightarrow 0\ R\ S_c$;

$S_c\,\lambda \rightarrow 1\ R\ S_a$.

Какую последовательность печатает данная машина, если начинает работать на пустой ленте?

Задача П.1.3. Машину Тьюринга А с 5 символами алфавита и 3 состояниями преобразовали в эквивалентную машину В по первой теореме Шеннона. Какое максимальное количество состояний будет иметь машина В?

Задача П.1.4. Машину Тьюринга А с 8 символами алфавита и 5 состояниями преобразовали в эквивалентную машину В по первой теореме Шеннона. Какое максимальное количество символов будет иметь машины В?

Задача П.1.5. Машину Тьюринга А с 65 символами алфавита и 4 состояниями преобразовали в эквивалентную машину С по второй теореме Шеннона. Сколько клеток на ленте машины С понадобится для одного символа машины А?

Задача П.1.6. Машину Тьюринга А с 5 символами алфавита и 6 состояниями преобразовали в эквивалентную машину С с двумя символами внешнего алфавита. Какое максимальное количество состояний будет иметь машина С?

Задача П.1.7. Машину Тьюринга А с 33 символами алфавита и 8 состояниями преобразовали в эквивалентную машину С по второй теореме Шеннона. Сколько клеток на ленте машины С понадобится для записи одного символа машины А?

Задача П.1.8. Какие действия выполнит следующий алгоритм для произвольного входного слова в заданном алфавите {0,1}?

1) $b0 \rightarrow 1a$
2) $b1 \rightarrow 0a$
3) $a1 \rightarrow 1b$
4) $a0 \rightarrow 0b$
5) $a \rightarrow \bullet$
6) $b \rightarrow \bullet$
7) $\Lambda \rightarrow a$

Задача П.1.9. Пусть A – множество натуральных чисел x, таких что $x > 4$. Что можно сказать про эффективное распознавание подмножества простых чисел B = {5, 11, 17, 23, 29, 41, 47, 53, 59, 71, 83, 89, 101} в множестве A?

Задача П.1.10. Пусть K – некоторое число, являющееся кодом алгоритма Маркова при Гёделевой нумерации. Символы алфавита имеют следующие кодовые значения:

код «1» соответствует символу «→»;
код «2» – символу «Λ»;
код «3» – символу «•»;
код «4» – символу «0»;
код «5» – символу «1»;

$K = 2^A$, где A = 150 000. Какой программе соответствует данное число K?

Задача П.1.11. Нотация кодировки некоторого нормального алгоритма Маркова задана следующим образом:

код «1» соответствует символу «→»;
код «2» – символу «Λ»;
код «3» – символу «•»;
код «4» – символу «*a*»;
код «5» – символу «*b*»;
код «6» – символу «*q*».

Код алгоритма Маркова имеет вид: $K = 2^A \cdot 3^B \cdot 5^C \cdot 7^D$, где

$A = 2^6 \cdot 3^4 \cdot 5^1 \cdot 7^5 \cdot 11^6$;
$B = 2^6 \cdot 3^5 \cdot 5^1 \cdot 7^4 \cdot 11^6$;
$C = 2^6 \cdot 3^1 \cdot 5^3$;
$D = 2^2 \cdot 3^1 \cdot 5^6$.

Какую задачу решает данный алгоритм Маркова, если исходное слово задано в алфавите {*a*, *b*}, а при работе алгоритма используется служебный символ *q*?

Задача П.1.12. Нотация кодировки некоторой машины Тьюринга задана следующим образом.

Внутренние состояния:

код «1» соответствует состоянию «S_0»;
код «11» – состоянию «Ω».

Символы внешнего алфавита:

код «2» соответствует символу «σ»;
код «22» – символу «λ»;
код «222» – символу «•»;
код «2222» – символу «*a*»;
код «22222» – символу «*b*».

Служебные символы:

код «3» соответствует символу «*R*» (движение вправо);
код «33» – символу «*L*» (движение влево);

код «333» – символу «*H*» (без движения);
код «4» – символу «→»;
код «5» – начало новой строки.

К ряду натуральных чисел применен алгоритм распознавания, определяющий, является ли натуральное число *m* кодом машины Тьюринга в данной нотации. На каком из следующих натуральных чисел алгоритм даст положительный результат?

a) 12423151222222222351331;
b) 124231512222422222315122222422223151224233311;
c) 113332411333251312222222225124235151242315222;
d) 124231512242315231411151121141111411151121141111.

Задача П.1.13. Какую задачу решает машина Тьюринга из предыдущего вопроса?

Задача П.1.14. Нотация кодировки некоторой машины Тьюринга задана следующим образом:

внутренние состояния:

код «1» соответствует состоянию «S_0»;
код «11» – состоянию «Ω»;
код «111» – состоянию «S_1»;

символы внешнего алфавита:

код «2» соответствует символу «σ»;
код «22» – символу «λ»;
код «222» – символу «*a*»;
код «2222» – символу «*b*».

служебные символы:

код «3» соответствует символу «*R*» (движение вправо);
код «33» – символу «*L*» (движение влево);
код «333» – символу «*H*» (без движения);
код «4» – символу «→»;
код «5» – начало новой строки.

Известно, что некоторой машине Тьюринга T соответствует кодовое значение:

12423151222422231512222422223111511122242223151122422
2242223151224233311511122422333 11.

Что в результате работы данной машины Тьюринга T будет на ленте, если исходное слово имеет вид «σ*abba*λ»?

ГЛАВА 2. ЧИСЛОВЫЕ МНОЖЕСТВА

2.1. Классификация и мощность множеств

Кратко приведены некоторые определения и операции над множествами, знание которых может потребоваться при выполнении контрольных заданий и сдаче экзамена (основная часть материала изучалась на первом семестре в рамках дисциплины «Дискретная математика»).

Множество, которое не имеет ни одного элемента, называется ***пустым*** и обозначается Ø.

Множество M_1 называется ***подмножеством*** множества M тогда и только тогда, когда любой элемент множества M_1 принадлежит множеству M.

Множества называются ***равными***, если они имеют одни и те же элементы.

Подмножество M_1 множества M называется ***собственным*** подмножеством множества M, если M_1 является его подмножеством, но при этом существует хотя бы один элемент, принадлежащий M, но не принадлежащий M_1.

Пусть A и B – два множества. Множество $M = A \cup B$ такое, что каждый его элемент принадлежит A или B (а возможно, и A, и B), называется ***суммой*** или ***объединением*** множеств A и B.

Пусть A и B – два множества. Множество $M = A \cap B$ такое, что каждый его элемент принадлежит и A и B одновременно, называется ***пересечением*** множеств A и B.

Пусть A и B – два множества. Множество $M = A \setminus B$ такое, что оно состоит из тех элементов множества A, которых нет во множестве B, называется ***разностью*** множеств A и B, или дополнением B до A.

Пусть A и B – два множества. Множество $M = A \times B$ такое, что оно образовано из всех пар (a, b) таких, что a принадлежит A и b принадлежит B, называется ***декартовым произведением*** множеств A и B.

Пусть A – множество. Множество M, элементами которого являются подмножества множества A, включая само A и пустое множество, называется множеством всех подмножеств множества

A или ***булеаном*** A и обозначается $P(A)$. Пусть $A = \{a, b, c\}$. Тогда $M = P(A)=\{Ø, (a), (b), (c), (a,b), (a,c), (b,c), (a,b,c)\}$

Отображением f множества A в множество B называется некое правило, по которому каждому элементу множества A ставят в соответствие элемент множества B. Множество всех отображений множества A в B обозначается как B^A (B в степени A).

Определений «мощности множества» тоже два, одно из них принадлежит уже упоминавшемуся Кантору.

Мощность множества (по Кантору) – это та общая идея, которая остается у нас, когда мы, мысля об этом множестве, отвлекаемся как от всех свойств его элементов, так и от их порядка.

Мощность множества – характеристика, которая объединяет данное множество с другими множествами, применение процедуры сравнения к которым дает основание предполагать, что каждый элемент одного множества имеет парный элемент из другого множества и наоборот.

Далее мощность будем называть ***кардинальным числом*** множества.

Уточним кардинальные числа некоторых множеств и их свойства.

- мощность пустого множества равна 0: $| Ø | = 0$.
- мощность множества из одного элемента равна 1: $|\{a\}| = 1$.
- множества равномощны ($A \sim B$), тогда и только тогда, когда их кардинальные числа равны: $|A| = |B|$.
- мощность булеана множества A равна $2^{|A|}$: $|P(A)| = 2^{|A|}$.
- мощность множества всех отображений A в B равна $|B|^{|A|}$: $|B^A| = |B|^{|A|}$.

Можно классифицировать множества, опираясь на такой признак, как конечность.

Конечное множество – множество, состоящее из конечного числа элементов, его кардинальное число совпадает с одним из натуральных чисел. В противном случае множество называется бесконечным.

Тогда все множества делятся на два класса: конечные и бесконечные, которые в свою очередь делятся на два подкласса: счетно-бесконечные и несчетные (рис. 2.1).

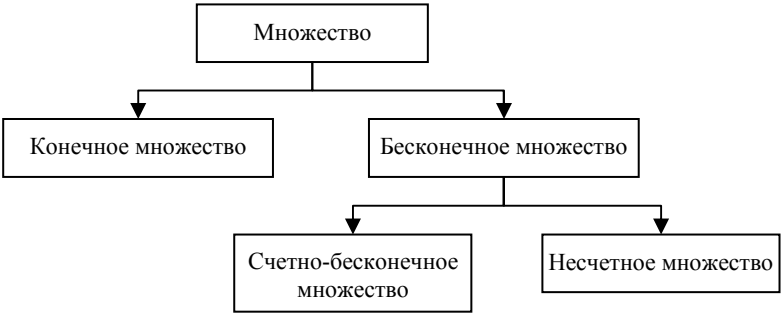

Рис. 2.1. Классификация множеств по признаку конечности

Счетно-бесконечные – бесконечные множества, равномощные множеству натуральных чисел (их элементы можно пронумеровать натуральными числами без пропусков и повторений).

Несчетные – бесконечные множества, не равномощные множеству натуральных чисел.

Можно классифицировать множества и по другому признаку: счетности.

Счетное множество – это множество, являющееся конечным или счетно-бесконечным.

Тогда все множества делятся на два класса: счетные и несчетные. Счетные множества в свою очередь делятся на два подкласса: конечные и счетно-бесконечные (рис. 2.2).

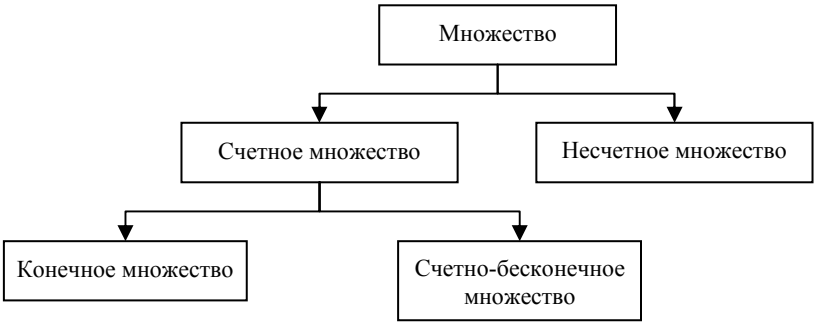

Рис. 2.2. Классификация множеств по признаку счетности

Важное свойство конечных множеств: конечные множества не равномощны никакому своему собственному подмножеству.

Важное свойство бесконечных множеств: бесконечное собственное подмножество бесконечного множества может быть равномощно самому множеству (внимание, именно «может быть», а вовсе не «всегда»; пример тому – несчетные множества, рассмотренные далее).

Иллюстрацией данного факта могут служить два известных парадокса.

Теорема 2.1 (парадокс Галилея). Хотя большинство натуральных чисел не является квадратами, всех натуральных чисел не больше, чем квадратов (если сравнивать данные множества по мощности). Иллюстрация этого парадокса приведена на рис. 2.3.

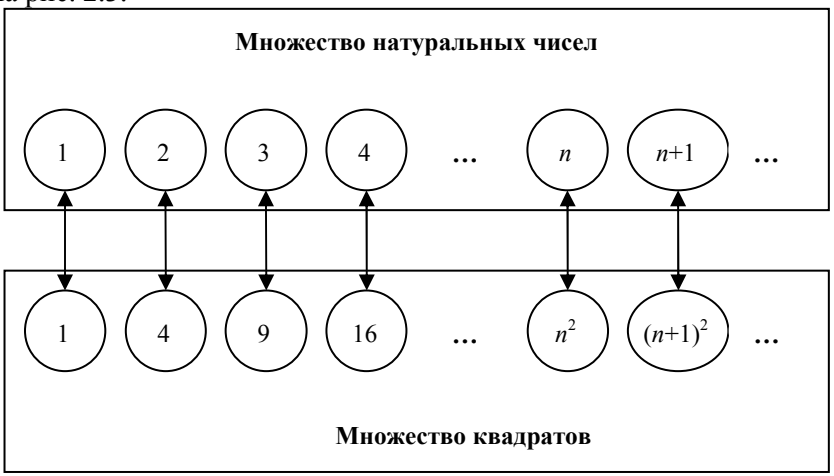

Рис. 2.3. Иллюстрация парадокса Галилея

Теорема 2.2. (парадокс Гильберта). Если гостиница с бесконечным количеством номеров полностью заполнена, в неё можно поселить ещё посетителей, даже бесконечное число. Иллюстрация этого парадокса приведена на рис. 2.4.

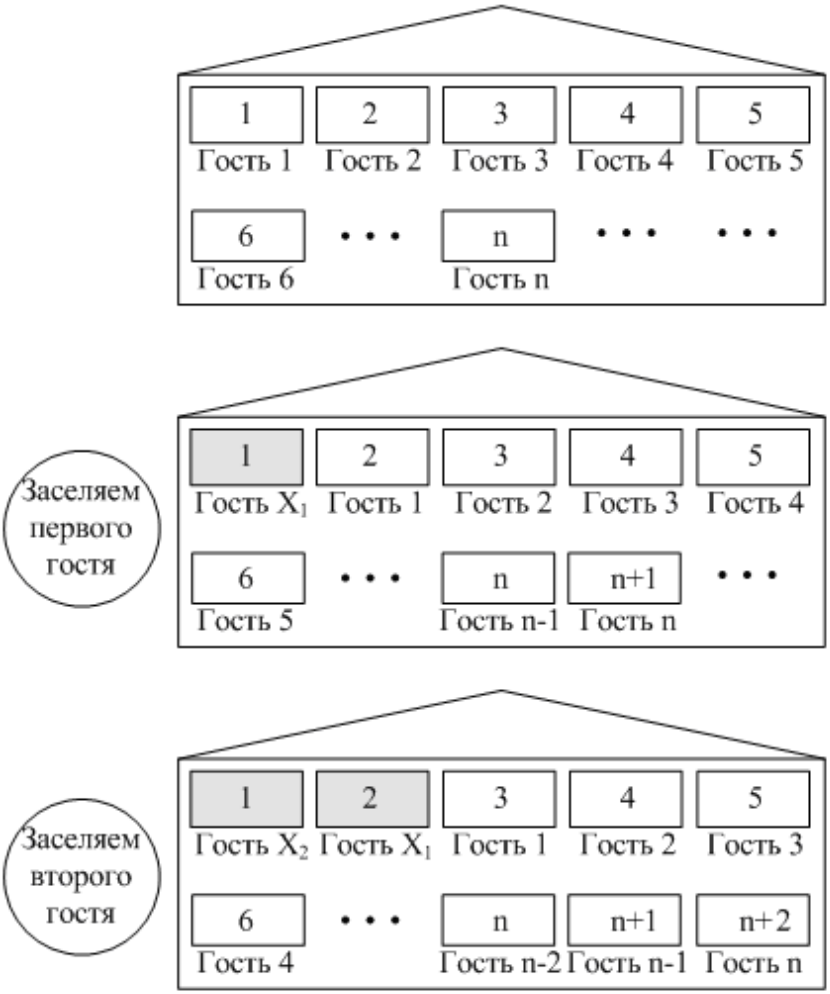

Рис. 2.4. Иллюстрация парадокса Гильберта

Трансфинитное число (finis – “конец”, лат.) – кардинальное число бесконечного множества.

Алеф-нуль χ_0 – первое трансфинитное число. По определению это мощность множества всех натуральных чисел, наименьшая бесконечная мощность.

2.2. Счетность и эффективная перечислимость числовых множеств

Эффективно перечислимым множеством называется множество, элементы которого можно перечислить по алгоритму (пронумеровать натуральным рядом без пропусков и повторений).

В парадоксе Галилея упоминается понятие натурального ряда или, иначе, натурального числа. Хотя этот термин знаком всем еще с 5-6 класса общеобразовательной школы, необходимо ввести важное уточнение. В общем смысле можно полагать, что натуральные (или естественные) числа – это числа, возникающие естественным образом при счёте (как в смысле перечисления, так и в смысле исчисления). В настоящее время в науке существуют два подхода к определению натуральных чисел.

Определение 1. ***Натуральные числа*** – это числа, используемые при перечислении (нумеровании) предметов (первый, второй, третий…). Данный подход общепринят в большинстве стран мира (в том числе и в России).

Определение 2. ***Натуральные числа*** – числа, используемые при обозначении количества предметов (нет предметов, один предмет, два предмета…). Принят в трудах Бурбаки, где натуральные числа определяются как мощности конечных множеств.

В любом случае отрицательные и нецелые числа натуральными числами не являются. В данном пособии в целях устранения двусмысленности вводится два понятия.

Множество натуральных чисел состоит из чисел вида 1, 2, 3, …, n, n+1 (соответствует определению 1).

Расширенное множество натуральных чисел состоит из чисел вида 0, 1, 2, 3, …, n, n+1 (соответствует определению 2).

Множество натуральных чисел обозначим латинской буквой N. Расширенное множество натуральных чисел обозначим N*. Множество натуральных чисел эффективно перечислимо по

определению. Расширенное множество натуральных чисел также окажется эффективно перечислимым, в силу того, что нумерацию его элементов можно провести обычными натуральными числами (рис. 2.5).

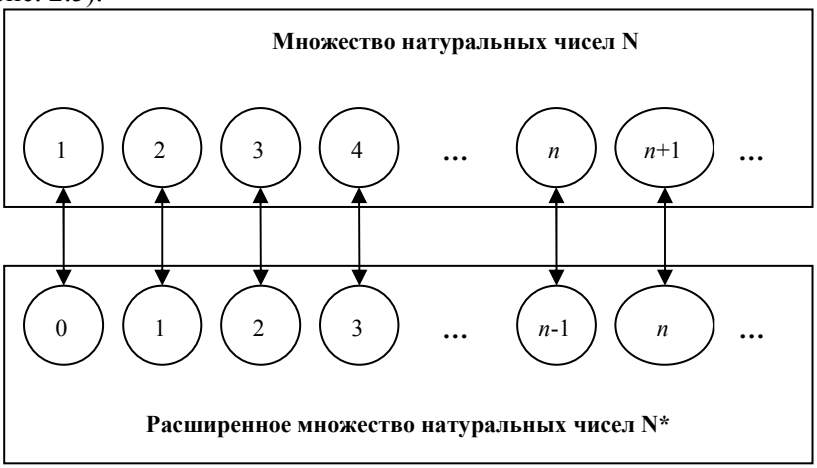

Рис. 2.5. Доказательство эффективной перечислимости расширенного множества натуральных чисел

Множество ***целых*** чисел – множество, состоящее из натуральных чисел, числа ноль и чисел, построенных на основе натуральных только со знаком «минус» (отрицательных чисел).

Множество целых чисел обозначим латинской буквой *Z*.

Теорема 2.3. Множество целых чисел счетно и эффективно перечислимо.

Два элемента *a* и *b* называют ***упорядоченной парой***, если указано, какой их этих элементов первый, а какой второй и при этом $((a,b) = (c,d)) \Leftrightarrow (a = c)\wedge(b = d)$. Упорядоченную пару элементов обозначают (a,b).

Теорема 2.4. Множество упорядоченных пар натуральных чисел счетно и эффективно перечислимо.

Упорядоченная n-ка натуральных чисел – это набор из *n* элементов вида $(m_1, m_2, \ldots, m_n)$, где m_i – натуральное число.

Теорема 2.5. Множество упорядоченных n-ок натуральных чисел счетно и эффективно перечислимо.

Конечные комплексы натуральных чисел – это элементы вида (p_1), (p_1, p_2), (p_1, p_2, p_3), …, $(p_1, p_2, \ldots, p_k)$, где k и p_i пробегают все натуральные числа.

Теорема 2.6. Множество конечных комплексов натуральных чисел счетно и эффективно перечислимо.

Рациональное число – число вида $q = \frac{n}{m}$, где n – целое число, m – натуральное.

Множество рациональных чисел обозначим латинской буквой Q.

Теорема 2.7. Множество рациональных чисел счетно и эффективно перечислимо.

Алгебраическое действительное число – действительный корень алгебраического уравнения ненулевой степени с рациональными коэффициентами.

Множество алгебраических действительных чисел обозначим латинской буквой A.

Общий вид алгебраического уравнения:

$$a_0 + a_1 \cdot x^1 + a_2 \cdot x^2 + \ldots + a_n \cdot x^n = 0,$$

где a_0, a_1,… a_n – рациональные коэффициенты. Исходя из определения, можно утверждать, что рассмотренные ранее классы натуральных, целых и рациональных чисел являются подмножествами множества алгебраических чисел.

Теорема 2.8. Множество алгебраических чисел счетно и эффективно перечислимо.

Теорема 2.9. Множество элементов, которые можно представить с помощью конечного числа счетной системы знаков, счетно.

Теорема 2.10. Множество действительных чисел несчётно.

Множество действительных чисел в дальнейшем будем обозначать латинской буквой R.

***Алеф* (χ – *второе трансфинитное число*)**. По определению это мощность континуума (всех действительных чисел), вторая по величине бесконечная мощность.

***Континуум-гипотеза**:* с точностью до эквивалентности, существуют только два типа бесконечных числовых множеств: счетное и континуум. Другая широко распространенная формулировка этой же гипотезы: «Любое бесконечное подмножество континуума является либо счётным, либо континуальным».

По сути, континуум-гипотеза утверждает, что не существует множеств, мощность которых находится в интервале между χ_0 и χ, но не совпадает ни с одной из указанных границ.

Если вернуться к определению булеана множества, это равносильно утверждению, что мощность булеана множества натуральных чисел равна мощности множества действительных чисел: $2^{\chi_0} = \chi$.

Трансцендентное число – действительное число, не являющееся алгебраическим.

Множество трансцендентных чисел обозначим латинской буквой *T*.

Теорема 2.13. Множество трансцендентных чисел несчетно.

Диаграмма, иллюстрирующая включение друг в друга различных числовых множеств, представлена на рис. 2.6.

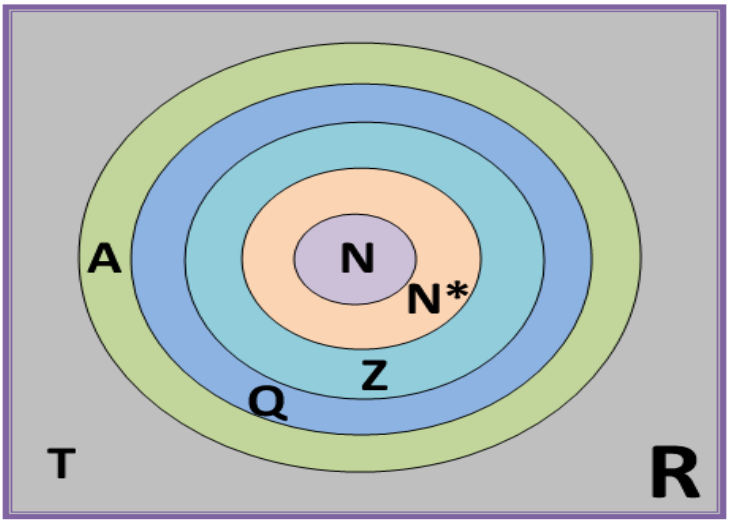

Рис. 2.6. Диаграмма включения числовых множеств

Комплексное число задается парой (r_1, r_2), где r_1, r_2 принадлежат множеству действительных чисел.

Множество комплексных чисел обозначим латинской буквой C.

Теорема 2.11. Множество комплексных чисел несчетно.

Иррациональным называется действительное число, не являющееся рациональным.

Множество иррациональных чисел в дальнейшем будем обозначать латинской буквой I.

Теорема 2.12. Множество иррациональных чисел несчетно.

Особый интерес представляют две следующие теоремы, из которых первая кажется более-менее очевидной, тогда как вторая представляется малореалистичной, особенно для тех, кто усвоил фундаментальное различие между множеством рациональных чисел (оно счетно) и множеством иррациональных чисел (оно, напротив, несчетно).

Теорема 2.14. Между любыми двумя различными рациональными числами всегда найдется множество иррациональных чисел мощности континуума.

Теорема 2.15. Между любыми двумя различными иррациональными числами всегда найдется счетное множество рациональных чисел.

Отметим, что иррациональных чисел намного больше, чем рациональных, и, следовательно, расположены на числовой оси они гораздо более «плотно» (рис. 2.7).

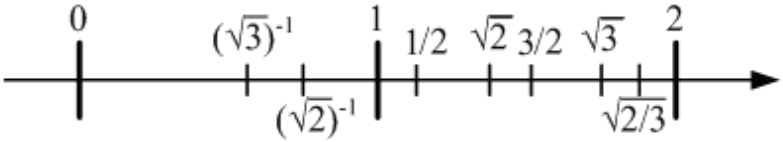

Рис. 2.7. Рациональные и иррациональные числа на числовой оси

Однако несмотря на это, удаётся «втиснуть» между любыми двумя различными иррациональными числами какое-нибудь рациональное число, хуже того, таких рациональных чисел в любой заданный интервал «поместится» счетное (и, главное, бесконечное множество).

На этом «чудеса» не заканчиваются. Вопрос о том, есть ли еще более обширные множества, чем множество действительных чисел или континуум, имеет ответ «да». Этому посвящены следующие теоремы.

Теорема 2.16 (теорема Кантора). Для любого кардинального числа α справедливо $\alpha < 2^{\alpha}$.

Теорема 2.17. Для любого множества A найдется множество B, мощность которого больше A.

Алеф-один χ_1 – третье трансфинитное число. По определению, это мощность множества всех подмножеств континуума. Таким образом, $2^{\chi} = \chi_1$.

Однако дальнейшее движение к всё более обширным множествам приводит к парадоксам, приведем два наиболее известных (напомним, что $P(U)$ – булеан множества U).

Теорема 2.18 (парадокс Кантора). Кардинальное число множества всех подмножеств $P(U)$ множества всех множеств U не больше чем $|U|$.

Теорема 2.19 (парадокс Рассела). Пусть B – множество всех множеств, которые не содержат самих себя в качестве своих собственных элементов. Тогда можно доказать две теоремы: B принадлежит B, и B не принадлежит B.

2.3. Вычислимость чисел

Действительное ***число вычислимо***, если существует алгоритм его вычисления с любой степенью точности

Множество вычислимых действительных чисел обозначим сокращенно ВДЧ, а множество вычислимых трансцендентных чисел – ВТЧ.

Какие же числа являются вычислимыми? Рассмотрим определенные ранее множества чисел.

Натуральные числа вычислимы по определению.

Рассмотрим множество целых чисел. По определению, это такое множество Z, которое содержит в себе: все натуральнее числа, все числа, противоположные натуральным и число 0 (ноль). При этом противоположными называются такие числа a и b, что $a + b = 0$. Необходимость использования множества Z обусловлена тем, что результат вычитания двух натуральных чисел не всегда является натуральным числом. В таком случае говорят: «множество N не замкнуто относительно вычитания». Так, $5-3 = 2$ – это натуральное число, но $8-17 = -9$ – число не натуральное. Множество Z, являясь расширением множества N, замкнуто относительно операции вычитания. Более того, множество Z − минимально возможное расширение множества N, которое обладает свойством замкнутости. Отсюда следует, что существует алгоритм порождения любого целого числа из двух натуральных, а значит, все *целые числа* вычислимы.

Аналогично, учитывая, что рациональное число – это число вида $q = \frac{n}{m}$, где n – целое число, m – натуральное, а для операции деления существует четкий алгоритм, получим, что все *рациональные числа* вычислимы.

Следующий класс – алгебраические числа. Исходя из определения алгебраического числа как действительного корня

алгебраического уравнения ненулевой степени с рациональными коэффициентами и с учетом того факта, что для решения таких уравнений существует алгоритм (точнее, несколько алгоритмов, основанных на методах Ньютона, Лобачевского, Лина и других), можно также утверждать, что все *алгебраические числа* вычислимы.

Чтобы разобраться с вычислимость чисел дальше, обратим внимание на следующий факт. Для вычисления каждого вычислимого числа по определению существует алгоритм, возможно не один. Алгоритмов существует счетное множество. Отсюда получим, что вычислимых чисел никак не больше, чем алгоритмов. Отсюда получаем следующую теорему.

Теорема 2.20. Множество вычислимых действительных чисел счетно.

Однако всего действительных чисел – несчетное множество (рис. 2.8).

Следовательно, верна теорема 2.21.

Теорема 2.21. Существуют невычислимые действительные числа и их несчетное множество.

Приведем пример невычислимого действительного числа. Пусть имеются $T_i,\ldots\ldots,T_n,\ldots$, где T_i – i-я машина Тьюринга. Действительное число X можно представить так:

$X = 0,a_1,\ldots\ldots,a_n\ldots$, где

$$a_i = \begin{cases} 1, \text{ если } T_i \text{ останавливается на чистой ленте;} \\ 0, \text{ в противном случае.} \end{cases}$$

Такое число является невычислимым, так как задача об остановке машины T_i на чистой ленте алгоритмически не разрешима.

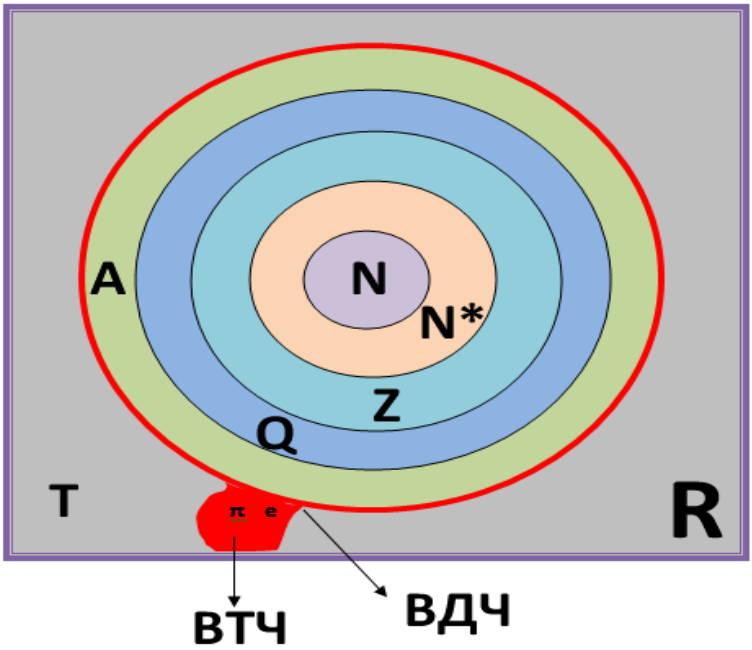

Рис.2.8. Диаграмма включения числовых множеств с определением зоны вычислимости

Проверочные задания и вопросы

Задача П.2.1. Для проверки и систематизации полученных знаний заполните табл. 2.1. В столбцах «Счетность» и «Эффективная перечислимость» напишите «да» или «нет», в столбце «Мощность» укажите конкретное кардинальное число, в столбце «Вычислимость» напишите «+», если все числа множества вычислимы, «–», если все числа множества невычислимы, и «+/–» если в множестве присутствуют различные с точки зрения вычислимости числа.

Таблица 2.1

Счетность и эффективная перечислимость основных числовых множеств

№	Множество	Счетность	Эффективная перечислимость	Мощность	Вычислимость
1	*N*				
2	*N**				
3	*Z*				
4	*Q*				
5	*A*				
6	*R*				
7	*T*				
8	ВДЧ				
9	*I*				
10	*C*				

Задача П.2.2. Основываясь на табл. 2.1 и операциях над числовыми множествами (объединения, пересечения и разности), заполните табл. 2.2. В столбцах «Счетность» и «Эффективная перечислимость» напишите «да» или «нет», в столбце «Мощность» укажите конкретное кардинальное число.

Таблица 2.2

Счетность и эффективная перечислимость других числовых множеств

№	Множество	Счетность	Эффективная перечислимость	Мощность
1	*R* \ ВДЧ			
2	*T* \ ВТЧ			
3	*A* ∪ ВТЧ			

Продолжение табл. 2.2

№	Множество	Счетность	Эффективная перечислимость	Мощность
4	$Q \setminus I$			
5	$N \cup Z$			
6	$A \cap N$			
7	$T \cap A$			
8	$Z \setminus N$			
9	$Q \cup T$			
10	$R \cap N$			

Задача П.2.3. Основываясь на табл. 2.1, операциях над числовыми множествами (декартова произведения и возведения в степень), заполните табл. 2.3. В столбцах «Счетность» и «Эффективная перечислимость» напишите «да» или «нет», в столбце «Мощность» укажите конкретное кардинальное число.

Таблица 2.3

Счетность и эффективная перечислимость прочих множеств

№	Множество	Счетность	Эффективная перечислимость	Мощность
1	$N \times N$			
2	$T \times T$			
3	$Q \times R$			
4	$N \times N \times N$			
5	N^N			
6	R^N			
7	N^R			

Продолжение табл. 2.3

№	Множество	Счетность	Эффективная перечислимость	Мощность
8	Точек на декартовой плоскости			
9	Точек в декартовом 3-мерном пространстве			
10	Точек в декартовом χ_0-мерном пространстве			

Задача П.2.4. Основываясь на табл. 2.1, операциях над числовыми множествами и определении булеана, заполните табл. 2.4. В столбце «Счетность» и «Эффективная перечислимость» напишите «да» или «нет», в столбце «Мощность» укажите конкретное кардинальное число.

Таблица 2.4

Счетность и эффективная перечислимость булеанов числовых множеств

№	Множество	Счетность	Эффективная перечислимость	Мощность
1	$P(N)$			
2	$P(T)$			
3	$P(N \times N)$			
4	$P(\varnothing)$			
5	$P(P(\varnothing))$			
6	$P(P(P(\varnothing)))$			
7	Фигур на плоскости			

Продолжение табл. 2.4

№	Множество	Счетность	Эффективная перечислимость	Мощность
8	Тел в 3-мерном пространстве			
9	Тел в n-мерном пространстве			
10	Тел в χ_0-мерном пространстве			

Задача П.2.5. Заполните табл. 2.5. В столбцах «Счетность» и «Эффективная перечислимость» напишите «да» или «нет», в столбце «Мощность» укажите конкретное кардинальное число.

Таблица 2.5

Счетность и эффективная перечислимость некоторых других множеств

№	Множество	Счетность	Эффективная перечис-лимость	Мощность
1	Множество всех кулинарных рецептов, написанных на русском языке			
2	Множество элементов, которые можно представить с помощью конечного числа символов счетно-бесконечной системы знаков			
3	Множество элементов, которые можно представить с помощью конечного числа символов конечной системы знаков			

Продолжение табл. 2.5

4	Множество всех конечных подмножеств счетного множества			
5	Множество всех четырехугольников на плоскости, координаты всех вершин которых рациональны			
6	Множество всех многоугольников на плоскости, координаты всех вершин которых рациональны			
7	Множество всех многочленов с действительными коэффициентами			
8	Множество всех действительных чисел, в десятичном разложении которых встречается цифра 3			
9	Множество всех действительных чисел, в десятичном разложении которых не встречается цифра 6			
10	Множество всех сыгранных когда-либо кем-либо музыкальных произведений			

Задача П.2.6. Произведите операции над трансфинитными числами и впишите ответ в табл. 2.6. Важно! Результатом должно быть элементарное трансфинитное число, без использования степенных функций (χ_0, χ, χ_1).

Таблица 2.6

Операции над трансфинитными числами

№	Операция	Результат
1	$\chi_1 + \chi_0$	
2	2^{χ}	
3	χ^2	
4	$\chi^2 - \chi_0$	
5	$\chi_0 + \chi + \chi_1$	
6	2^{χ_0}	
7	$\sqrt{\chi}$	
8	2^{χ}	
9	$100000\chi_0$	
10	$\chi_1 - \chi - \chi_0$	

Задача П.2.7. Определите принадлежность элементов к указанным множествам, поставив «+» или «–» в соответствующих клетках таблицы.

Таблица 2.7

Принадлежность элементов к множествам

№	Число	$R\backslash Q$	$T\cap A$	$P((Z\backslash N^*)\cup N)$	$P(R^*R)$	$(R\backslash Z)\backslash T$	Q	$P((N^*\backslash N)\cap T)$
1	5/6							
2	0							

Продолжение табл.2.7

№	Число	$R\backslash Q$	$T\cap A$	$P((Z\backslash N^*)\cup N)$	$P(R^*R)$	$(R\backslash Z)\backslash T$	Q	$P((N^*\backslash N)\cap T)$
3	$\sqrt{3}$							
4	π							
5	(8,2)							
6	Ø							
7	{2}							
8	{1/3}							
9	{(2,π)}							
10	{8,2}							
11	4							
12	-3							
13	{1, 3, 5}							
14	$2\sqrt{3}$							
15	$2^{\sqrt{3}}$							

ГЛАВА 3. АРИФМЕТИЧЕСКИЕ ВЫЧИСЛЕНИЯ

3.1. Арифметические и частичные арифметические функции

Расширенное множество натуральных чисел, помимо обычного множества натуральных чисел, включает также число ноль (ранее это множество обозначалось *N**).

Арифметическая функция (АФ) – функция, определенная на расширенном множестве натуральных чисел и принимающая значения из множества натуральных чисел.

Теорема 3.1. Множество арифметических функций *n*-переменных несчетно.

Частичная арифметическая функция (ЧАФ) **–** это функция, определенная на некотором подмножестве *M* расширенного множества натуральных чисел *N** и принимающая значения из всего расширенного множества *N**.

Класс частичных арифметических функций шире класса арифметических функций, так как любая арифметическая функция является всюду определенной частичной арифметической функцией и, кроме того, существуют не всюду определенные частичные арифметические функции. Диаграмма вхождения рассмотренных множеств друг в друга представлена на рис. 3.1.

Теорема 3.2. Множество частичных арифметических функций несчетно.

С количественной точки зрения можно оценить мощность множества арифметических функций. Для наглядности сделаем это для функций одной переменной, т.е. функций вида $f(x)$. Функции этого класса в качестве аргумента могут иметь любой натуральное число, таких чисел χ_0. Значением функции в каждой точке также может быть любое натуральное число, таких чисел χ_0. Согласно правилам комбинаторики общее количество различных упорядоченных наборов из χ_0 элементов составит: $\chi_0^{\chi_0}$. Это число равно χ.

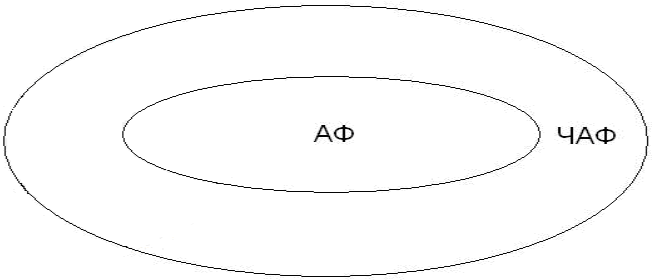

Рис. 3.1. Диаграмма вхождения классов АФ и ЧАФ

Приведем несколько примеров.

$$f(n) = n - 1. \quad (3.1)$$

Для этой функции область определенности: $M = [1, \infty)$, область значений: N. Таким образом, функция строит соответствие $\{1, 2, \ldots\} \to N$.

$$f(n) = 1 - n. \quad (3.2)$$

Для функции (3.2) область определенности: $M = [0,1]$, область значений: $[0,1]$. Таким образом, функция строит соответствие $\{0, 1\} \to \{0, 1\}$.

Можно выделить два крайних случая множества частичных арифметических функций $f(n)$: $M \to N$.

Всюду определенные функции ($M = N^*$), например:

$$f(n) = n + 1. \quad (3.3)$$

Множество всюду определенных частичных арифметических функций совпадает с множеством арифметических функций. Все остальные частичные арифметические функции имеют точки неопределенности.

Нигде неопределенные функции ($M = \varnothing$), например:

$$f(n) = 0 - (n + 1). \quad (3.4)$$

Нигде не определенные функции также являются подмножеством множества частичных арифметических функций.

Вычислимой арифметической функцией (ВАФ) называется арифметическая функция, для которой существует алгоритм вычисления значения в любой точке.

В силу тезиса Тьюринга это означает, что функция вычислима, если существует машина Тьюринга, ее вычисляющая.

Теорема 3.3. Множество вычислимых арифметических функций счетно.

Вычислимой частичной арифметической функцией (ВЧАФ) называется частичная арифметическая функция, для которой существует алгоритм вычисления значения в любой точке области определенности.

Теорема 3.4. Множество вычислимых частичных арифметических функций счетно.

Диаграмма вхождения рассмотренных множеств друг в друга представлена на рис.3.2.

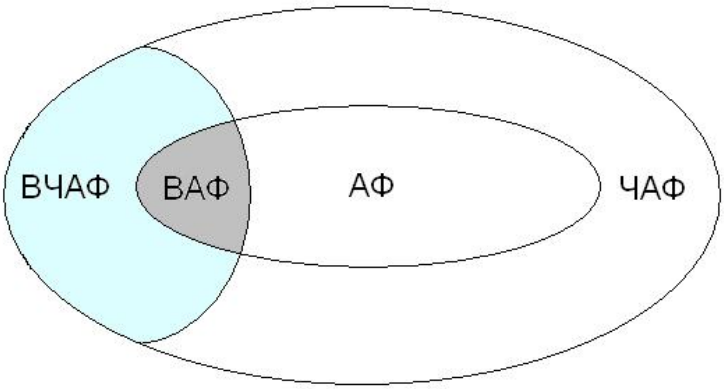

Рис. 3.2. Диаграмма вхождения классов АФ, ЧАФ, ВАФ и ВЧАФ

Теорема 3.5. Множество арифметических функций, описываемых конечным числом слов, счетно и эффективно перечислимо.

Теорема 3.6. Множество вычислимых частичных арифметических функций счетно и эффективно перечислимо.

Из того факта, что множество ВЧАФ счетно, а множество ЧАФ – несчетно, следует, что существуют невычислимые частичные арифметические функции, причем их множество несчетно.

Теорема 3.7. Множество невычислимых арифметических функций несчетно.

Например, невычислимой является функция, заданная следующим образом:

$$f(x) = \begin{cases} 1, \text{ если } T_x \text{ остановится на чистой ленте,} \\ 0, \text{ в противном случае,} \end{cases} \quad (3.5)$$

где T_0 ,T_1 ,T_2 … ,T_x – машины Тьюринга при их эффективном перечислении.

Функция невычислима, так как задача об остановке произвольной машины Тьюринга на чистой ленте алгоритмически неразрешима. Вместо указания на факт остановки машины Тьюринга можно использовать любую алгоритмически неразрешимую проблему, например проблему определения факт печати ровно одного символа «ноль».

Стоит быть внимательным при определении того, является ли функция, заданная похожим способом, действительно невычислимой.

Например, оказывается вычислимой функция, заданная следующим образом:

$$f(x) = \begin{cases} 1, \text{ если } T_x \text{ остановится на чистой ленте} \\ \quad \text{за первые пять шагов,} \\ 0, \text{ в противном случае,} \end{cases} \quad (3.6)$$

где T_0, T_1, T_2, …, T_x – машины Тьюринга при их эффективном перечислении. Алгоритм вычисления следующий. Возьмем точку x=0. Запустим процесс эффективного перечисления множества машин Тьюринга, найдем код программы машины T_0, запустим её, подождем пять тактов её работы и увидим, окажется ли машина в заключительном состоянии за эти пять шагов, если да – значит, машина остановилась, и f (0) = 1, если нет – значит, машина не остановилась, и f (0) = 0. Аналогично определяется значение функции в точках 1, 2 и т.д. Таким образом, для каждой точки расширенного натурального ряда существует алгоритм вычисления значения функции, а значит, по определению, функция вычислима.

3.2. Эффективная перечислимость и распознаваемость арифметических функций

Теорема 3.8 (теорема Тьюринга). Множество вычислимых арифметических функций n переменных не поддается эффективному перечислению.

Эффективным распознаванием функций называется процедура, позволяющая при помощи некоторого алгоритма определить, относится ли данная функция к рассматриваемому классу.

Теорема 3.9. Невозможно эффективно распознать функции-константы среди вычислимых арифметических функций.

Эффективным сравнением арифметических функций называется процедура, позволяющая при помощи некоторого алгоритма определить, совпадают ли значения функций во всех точках.

Теорема 3.10. Вычислимые арифметические функции не поддаются эффективному сравнению.

Теорема 3.11. Невозможно эффективно распознать функции-тождества среди вычислимых арифметических функций.

Теорема 3.12. Невозможно эффективно распознать вычислимые арифметические функции среди вычислимых частичных арифметических функций.

Теорема 3.13. (терема Черча). Невозможно эффективно распознать точки неопределенности вычислимой частичной арифметической функции.

Для задания области определенности или множества значений арифметических функций удобно использовать способ задания подмножества A множества N^* через характеристическую функцию.

Характеристической функцией χ_A какого-нибудь подмножества A множества натуральных чисел N^* называется функция от одной переменной, равная 1 в точках множества A и равная 0 в точках, не принадлежащих A.

Например, для пустого множества характеристическая функция всюду равна 0:

$$\chi_{\varnothing} = 0. \qquad (3.7)$$

Для расширенного множества натуральных чисел характеристическая функция всюду равна 1:

$$\chi_N = 1. \tag{3.8}$$

Для множества $A=\{a_1,a_2,..,a_n\}$: $a_1<a_2<,..,<a_n$ характеристическая функция определяется с помощью функции *unsg*:

$$\chi_A = unsg(|x - a_1|\cdot|x - a_2|\cdot\ldots\cdot|x - a_n|). \tag{3.9}$$

При составлении характеристических функций часто используются специальные функции типа (3.10)−(3.16):

$$sg(x) = \begin{cases} 1, \text{ если } x > 0, \\ 0, \text{ в противном случае;} \end{cases} \tag{3.10}$$

$$unsg(x) = \begin{cases} 0, \text{ если } x > 0, \\ 1, \text{ в противном случае;} \end{cases} \tag{3.11}$$

$$eql(x,y) = \begin{cases} 1, \text{ если } x=y, \\ 0, \text{ в противном случае;} \end{cases} \tag{3.12}$$

$$uneql(x,y) = \begin{cases} 0, \text{ если } x=y, \\ 1, \text{ в противном случае;} \end{cases} \tag{3.13}$$

$$more(x,y) = \begin{cases} 1, \text{ если } x>y, \\ 0, \text{ в противном случае;} \end{cases} \tag{3.14}$$

$$less(x,y) = \begin{cases} 1, \text{ если } x<y, \\ 0, \text{ в противном случае;} \end{cases} \tag{3.15}$$

$$rest(x,y) = \begin{cases} 1, \text{ если остаток от деления } x \text{ на } y \text{ больше нуля,} \\ 0, \text{ в противном случае;} \end{cases} \tag{3.16}$$

Например, зададим характеристические функции области определения и области значений для $f(x) = (10–x)/2$. Рассмотрим интервалы, которым принадлежат область определения A = {0, 2, 4, 6, 8, 10} и область значения B = {0, 1, 2, 3, 4, 5}. Теперь необходимо подобрать специальные функции, которые будут принимать значение 1 в точках из множества A и B соответственно и 0 в остальных точках. Так как в данном случае множество конечное, то можно воспользоваться перечислением значений (3.17)–(3.18) и функцией $eql(x,y)$:

$$\chi_A = eql(x,0)+eql(x,2)+eql(x,4)+eql(x,6)+eql(x,8)+eql(x,10); \quad (3.17)$$

$$\chi_B = eql(x,0)+eql(x,1)+eql(x,2)+eql(x,3)+eql(x,4)+eql(x,5). \quad (3.18)$$

Рассмотрим другой пример. Зададим характеристические функции области определения и области значений для $f(x) = (x - 10)*2$. Определим интервалы, которым принадлежат область определения A = {10, 11, 12, 13 и т.д.}, т.е. все числа больше 10 и область значения B = {0, 2, 4, 6, 8 и т.д.}, все чётные числа. Теперь необходимо подобрать специальные функции, которые будут принимать значение 1 в точках из множества A и B соответственно и 0 в остальных точках. Так как множества A и B являются бесконечными, то для области определения можно использовать функцию $more(x, 9)$ (19), а для области значений $rest(x, 2)$ и $unsg(x)$, поскольку характеристическая функция должна принимать значение 1, если остаток от деления равен нулю (3.20):

$$\chi_A = more(x, 9); \quad (3.19)$$

$$\chi_B = unsg(rest(x, 2)). \quad (3.20)$$

Рассмотрим пример восстановления вида функции по характеристическим функциям области определения и области значений. Даны следующие характеристические функции:

$$\chi_A = sg(x \dot{-} 5) \cdot sg(rest(x, 2)); \quad (3.21)$$

$$\chi_B = 1. \quad (3.22)$$

Область определения включает все нечётные числа больше 5, а область значений – всё множество натуральных чисел. Поскольку в характеристической функции для множества A есть ограничение, что числа должны быть нечётными, значит, в самой функции присутствует деление на 2. В прошлом примере наличие разности повлекло ограничение, что числа должны быть больше 9, значит, в данном случае так же есть разность (ограничение, что числа

больше 5, т.е. вычитается число 6 или 7). Вместе с ограничением, при котором числа должны быть нечётными, получается, что вычитается также нечётное число, т.е. семь. Учитывая это можно предположить, что функция выглядит следующим образом:

$$f(x) = (x-7)/2. \quad (3.23)$$

Проверим найденную функцию – рассмотрим её область значений. Для данной функции это множество натуральных чисел, т.е. функция восстановлена верно.

Проверочные задания и вопросы

Задача П.3.1. Для проверки и систематизации полученных знаний заполните табл. 3.1. В столбцах «Счетность» и «Эффективная перечислимость» напишите «да» или «нет», в столбце «Мощность» укажите конкретное кардинальное число.

Таблица 3.1

Счетность и эффективная перечислимость основных классов функций

№	Множество	Счетность	Эффективная перечислимость	Мощность
1	АФ			
2	ВАФ			
3	ЧАФ			
4	ВЧАФ			

Задача П.3.2. Для проверки и систематизации полученных знаний заполните табл. 3.2. В столбцах «Счетность» и «Эффективная перечислимость» напишите «+» или «−», в столбце «Мощность» укажите конкретное кардинальное число.

Таблица 3.2

Счетность и эффективная перечислимость основных классов функций

№	Множество	Счетность	Эффективная перечислимость	Мощность
1	ЧАФ \ ВЧАФ			
2	АФ \ ВАФ			
3	ЧАФ ∩ ВЧАФ			
4	АФ ∩ ВЧАФ			
5	ВАФ \ ЧАФ			
6	АФ ∪ ЧАФ			
7	ВАФ ∪ ВЧАФ			
8	ВЧАФ \ АФ			
9	Множество арифметических функций, описываемых конечным числом слов			

Задача П.3.3. Опишите словами указанные в табл. 3.2 классы функций. Приведите по три примера функций, принадлежащих указанным в табл. 3.2 классам.

Задача П.3.4. Для данных функций $y = f(x)$ задать область значений и область определенности в интервальном виде и в виде характеристических функций $\chi(x)$ и $\chi(y)$.

а) $f(x) = x/(x - 5)$;
б) $f(x) = x - 15$;
в) $f(x) = 3(x - 10)$;
г) $f(x) = (x - 1)/(x - 5)$;
д) $f(x) = x + 10$;
е) $f(x) = 15/(x - 10)$;

Задача П.3.5. Построить арифметические функции по заданным характеристическим функциям области определения и области значений.

Таблица 3.3

Характеристические функции области определения и области значений

№	Область определения	Область значений
1	$\chi(x) = unsg(rest(x, 2))$	$\chi(y) = sg(y \dot{-} 3)$
2	$\chi(x) = sg(x \dot{-} 9) \cdot sg(15 \dot{-} x)$	$\chi(y) = unsg\,(y \cdot eql\,(y, 3) \cdot eql\,(y, 4))$
3	$\chi(x) = sg(x \dot{-} 5) \cdot sg(rest(x,2))$	$\chi(y) = 1$
4	$\chi(x) = eql(x,1) + eql(x,2) +$ $+ eql(x, 5) + eql(x, 10)$	$\chi(y) = eql(y, 2) + eql(y, 3) +$ $+ eql(y, 6) + eql(y, 11)$
5	$\chi(x) = sg(x \dot{-} 2) \cdot sg(8 \dot{-} x)$	$\chi(y) = unsg\,(y \cdot eql\,(y, 3) \cdot eql(y, 4))$
6	$\chi(x) =$ $sg(x \dot{-} 10) \cdot unsg(rest(x,2))$	$\chi(y) = 1$
7	$\chi(x) = rest(x, 2)$	$\chi(y) = sg(y \dot{-} 7)$
8	$\chi(x) = eql(x, 1) + eql(x, 2) +$ $+ eql(x, 4) + eql(x, 8)$	$\chi(y) = eql(y, 3) + eql(y, 4) +$ $+ eql(y, 6) + eql(y, 10)$

Задача П.3.6. Определите, к каким указанным классам принадлежат приведенные ниже функции. Заполните табл. 3.4, поставив значок «+» в соответствующей ячейке, если функция относится к данному классу и «-» в противоположном случае. Во всех заданиях T_0, T_1, T_2, …, T_x – машины Тьюринга при их эффективном перечислении.

а)
$$f(x) = \begin{cases} 1, \text{ если } T_x \text{ не остановится на чистой ленте} \\ \text{за первые } x+1 \text{ шагов;} \\ \\ 0, \text{ в противном случае.} \end{cases}$$

б)
$$f(x) = \begin{cases} 1, \text{ если } T_x \text{ не остановится на чистой ленте;} \\ 0, \text{ в противном случае.} \end{cases}$$

в)
$$f(x) = \begin{cases} 2, \text{ если за } x \text{ шагов } T_x \text{ напечатает на пустой ленте символ ноль точно один раз;} \\ 5, \text{ в противном случае.} \end{cases}$$

г)
$$f(x) = \begin{cases} 0, \text{ если } T_x \text{ напечатает ровно пять нулей на пустой ленте не более чем за } x \text{ шагов;} \\ 1, \text{ в противном случае.} \end{cases}$$

д)
$$f(x) = \begin{cases} 1, \text{ если машина } T_x \text{ не остановится на чистой ленте за первые 13 тактов;} \\ x + 2, \text{ если машина } T_x \text{ остановится на чистой ленте за первые 13 тактов.} \end{cases}$$

е)
$$f(x) = \begin{cases} -x-5, \text{ если машина } T_x \text{ в процессе своей работы напишет ровно пять нулей;} \\ -x-10, \text{ если машина } T_x \text{ в процессе своей работы напишет ровно десять нулей;} \\ x - \text{иначе.} \end{cases}$$

ж)
$$f(x) = \begin{cases} 1, \text{ если машина } T_x \text{ в процессе своей работы остановится за 10 шагов;} \\ 2, \text{ если машина } T_x \text{ в процессе своей работы остановится за более чем 10;} \\ 3, \text{ иначе.} \end{cases}$$

Таблица 3.4

Принадлежность функций к заданным классам

Функция	ВАФ	ВЧАФ	АФ	ЧАФ	ЧАФ\ВЧАФ	АФ\ВАФ
а						
б						
в						
г						
д						
е						
ж						

Задача П.3.7. Определите, к каким указанным классам принадлежат приведенные ниже функции. Заполните табл. 3.5, поставив значок «+» в соответствующей ячейке, если функция относится к данному классу и «–» в противоположном случае. Во всех заданиях $x, y \subset N^*$:

а) $f(x, y) = -x^2y + xy - 2$.

б) $f(x) = \begin{cases} p(x), \text{ если } x \text{ нечетное, где } p(x) - \text{произвольная ВАФ;} \\ x/2, \text{ в противном случае.} \end{cases}$

в) $f(x) = x^2+3$.

г) $f(x) = x(x+1)/2$.

д) $f(x) = x^5-1$.

е) $f(x) = x^2-2x+1$.

ж) $f(x) = -x-3$.

з) $f(x) = \begin{cases} x-2, \ x \geq 3; \\ x^2, \ x<3. \end{cases}$

и)
$$f(x) = \begin{cases} x-10, & x \le 10; \\ x+10, & x>10. \end{cases}$$

к) $f(x) = (x+4)(x+5)(x+6)/6$.

л) $f(x) = x$.

м) $f(x) = -x$.

Таблица 3.5

Принадлежность функций к заданным классам

Функция	ВАФ	ВЧАФ	АФ	ЧАФ	АФ∩ВЧАФ	ВЧАФ\ВАФ
а						
б						
в						
г						
д						
е						
ж						
з						
и						
к						
л						
м						

ГЛАВА 4. РЕКУРСИВНЫЕ ФУНКЦИИ

4.1. Примитивно-рекурсивные функции

4.1.1. Теоретическая основа

Класс примитивно-рекурсивных функций определяется путем указания конкретных исходных функций (они называются базисными) и фиксированного множества операций над ними, применяемых для получения новых функций из ранее заданных.

В качестве базисных функций обычно берутся три функции:

1) следования;
2) тождества (проекции или выбора аргументов);
3) константы.

Допустимыми операциями над функциями являются операции суперпозиции (подстановки) и примитивной рекурсии.

Частичная арифметическая функция f называется ***примитивно-рекурсивной***, если она может быть получена из простейших функций C_q^n, S, U_m^n конечным числом операций подстановки и примитивной рекурсии (т.е. задана в базисе Клини).

Таким образом, базис Клини состоит из трех простейших функций и двух разрешенных операций. Рассмотрим простейшие функции:

1) функция-***следование***: $S(x) = x' = x + 1$, где x’ – число, непосредственно следующее за натуральным числом x. Например, 0’ = 1, 1’ = 2, … и т.д.

2) функция-***тождество***: $U_i^n(x_1,...,x_n) = x_i$, где n – количество переменных, а i – номер переменной, по которой берется тождество. Функция тождества – функция, равная одному из своих аргументов. Например, $U_2^3(x,y,z) = y$.

3) функция-***константа***: $C_q^n(x_1,...,x_n) = q$, где n – число переменных, q – значение, которое принимает функция. Функция

константа принимает всюду одно значение. Например, $C_2^3(16,9,10) = 2$.

Рассмотрим разрешенные операции:

1) операция ***примитивной рекурсии*** R. Если мы имеем функцию одной переменной $f(x)$, то схема рекурсии называется «рекурсия без параметров» и задается системой уравнений

$$f(0) = q;$$
$$f(x') = \chi(f(x), x).$$

Функция, заданная такими уравнениями, кратко задается схемой вида $R_q(\chi)$. Поскольку вид системы уравнений (и способ задания трех разрешенных функций) строго определен, то схема является однозначной.

Если мы имеем функцию нескольких переменных $f(x_1, x_2, \ldots, x_n)$, то схема рекурсии называется «рекурсия с параметрами» и задается системой уравнений

$$f(0, x_2, \ldots, x_n) = \Psi(x_2, \ldots, x_n);$$
$$f(x'_1, x_2, \ldots, x_n) = \chi(f(x_1, \ldots, x_n), x_1, \ldots, x_n).$$

Функция, заданная такими уравнениями, кратко может быть представлена схемой вида $R^n(\Psi, \chi)$

2) операция ***подстановки (суперпозиции)*** S. Пусть заданы m каких-либо частичных арифметических функций $f_1, f_2, \ldots f_m$ от одного и того же числа n переменных, определенных на множестве A со значениями в множестве B. Пусть на множестве B задана частичная функция Ф от n переменных, значения которой принадлежат множеству C. Введем теперь частичную функцию g от n переменных, определенную на A со значениями в C, полагая по определению для произвольных $x_1, \ldots, x_n$:

$$g(x_1, \ldots, x_n) = \Phi(f_1(x_1, \ldots, x_n), f_2(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n)).$$

Говорят, что функция g получается операцией суперпозиции или подстановки из функций $f_1, f_2, \ldots, f_m$. Обозначается эта операция буквой S с двумя индексами: верхний (n) показывает от скольких переменных зависят внутренние функции $f_i(x_1, \ldots, x_n)$, а нижний (m) – количество самих функций $f_1, f_2, \ldots, f_m$:

$$g(x_1, \ldots, x_n) = S^n_m(\Phi, f_1, f_2, \ldots, f_m),$$

При этом функция Ф при подсчете внутренних функций не учитывается.

4.1.2. Методология составления схем примитивной рекурсии

Для составления схемы примитивной рекурсии первоначально необходимо определить вид нужной схемы.

Для этого следует ответить на вопрос: «От скольких переменных зависит данная функция?» Допустим, переменных n, и это число равно или больше двух. Значит, используется схема рекурсии с параметрами. Следующие шаги:

1) уточнить задание: указано ли, по какой переменной составлять схему. Если нет, то следует выбрать наиболее простую переменную (наименьшее количество операций с этой переменной или наиболее простые операции).

Если при ответе на вопрос: «От скольких переменных зависит данная функция?» получено, что такая переменная одна-единственная, значит, используется схема рекурсии без параметров;

2) определить, чему равно значение функции в точке ноль? (верхний индекс в функциях и операциях для $n>1$ будет равен $n–1$);

3) определить, чему равно значение функции в следующей точке (переменная по которой берется рекурсия плюс единица). Постарайтесь преобразовать получившееся значение к самой функции, верхний индекс в функциях и операциях для любого значения n будет равен $n + 1$;

4) записать функцию через операцию примитивной рекурсии.

Приведем несколько примеров.

Задача 4.1. Доказать примитивную рекурсивность суммы $f(x,y)=x+y$.

Решение. По какой переменной составлять схему не указано, но из условия задачи следует, что обе переменные имеют идентичную сложность. Будем строить рекурсию по первой переменной:

$f(0, y) = 0 + y = y = U_1^1$;

$f(x', y) = (x + 1) + y = x + y + 1 = f(x, y) + 1 = S(f(x, y)) = S(U_1^3) = S_1^3(S, U_1^3)$.

Ответ: $f(x, y) = R^2[U_1^1, S_1^3(S, U_1^3)]$.

Далее признаком доказательства того, что функция примитивно-рекурсивная, будет являться выражение функции

через другие функции (примитивная рекурсивность которых доказана ранее) и разрешенные базисом Клини операции.

Задача 4.2. Доказать примитивную рекурсивность произведения $f(x,y) = xy$.

Решение. По какой переменной составлять схему не указано, но из условия задачи следует, что обе переменные имеют идентичную сложность. Будем строить рекурсию по первой переменной:

$f(0, y) = 0y = 0 = C_0^1(y) = C_0^1$;

$f(x', y) = (x + 1)y = xy + y = f(x, y) + y = \sum(f(x, y), y) = \sum(U_1^3, U_3^3) =$
$= S_2^3(\sum, U_1^3, U_3^3)$.

Ответ: $f(x,y) = R^2[C_0^1, S_2^3(\sum, U_1^3, U_3^3)]$.

Задача 4.3. Доказать примитивную рекурсивность факториала $\boldsymbol{f(x) = x!}$

Решение:

$f(0) = 1$;

$f(x') = (x + 1)! = x!\cdot(x + 1) = f(x)\cdot S(x) = \Pi(f(x), S(x)) =$
$= \Pi(U_1^2, S(U_2^2)) = S_2^2(\Pi, U_1^2, S(U_2^2)) = S_2^2(\Pi, U_1^2, S_1^2(S, U_2^2))$.

Ответ: $f(x) = R_1[S_2^2(\Pi, U_1^2, S_1^2(S, U_2^2))]$.

Задача 4.4. Доказать примитивную рекурсивность псевдоразности.

$$f(x,y) = x \dot{-} y = \begin{cases} x - y, \text{ если } x \geq y; \\ 0, \text{ иначе.} \end{cases}$$

Решение. Введем вспомогательную функцию $\sigma(x) = x \dot{-} 1$:

$\sigma(0) = 0$;

$\sigma(x') = (x + 1) \dot{-} 1 = x = U_2^2$.

Получили схему примитивной рекурсии для введенной дополнительной функции: $\sigma(x) = R_0(U_2^2)$.

Теперь вернемся к основной функции:

$f(0, y) = 0 \dot{-} y = 0$;

$f(x', y) = (x + 1) \dot{-} y = x \dot{-} (y \dot{-} 1)$;

$f(x, 0) = x \dot{-} 0 = x = U_1^1$;

$f(x, y') = x \dot{-} (y + 1) = (x \dot{-} y) \dot{-} 1 = f(x, y) \dot{-} 1 = \sigma(f(x, y)) = \sigma(U_1^3) =$
$= S_1^3(\sigma, U_1^3)$.

Ответ: $f(x,\ y) = R^2[U_1^1,\ S_1^3(R_0(U_2^2),\ U_1^3)]$.

Задача 4.5. Доказать примитивную рекурсивность знаковой функции

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

Решение. Данную функцию можно представить через псевдоразность: $f(x) = 1 \dot{-} (1 \dot{-} x)$.

Запишем схему рекурсии без параметров:

$f(0) = 0$;

$f(x') = 1 \dot{-} (1 \dot{-} (x + 1)) = 1 = C_1^2$.

Ответ: $f(x) = R_0(C_1^2)$.

Также примитивно-рекурсивной будет противоположная функция ***f* (*x*) = *unsg*(*x*)**, функция, возвращающая 1, если аргумент равен 0, и 0 – иначе. Докажите это самостоятельно.

Задача 4.6. Доказать примитивную рекурсивность равенства.

$$f(x,\ y) = eql(x, y) = \begin{cases} 1, \text{ если } x = y; \\ 0, \text{ иначе.} \end{cases}$$

Решение. Данную функцию можно представить через псевдоразность: $f(x, y) = 1 \dot{-} [(x \dot{-} y) + (y \dot{-} x)]$.

Запишем схему рекурсии с параметрами:

$f(0, y) = 1 \dot{-} [(0 \dot{-} y) + (y \dot{-} 0)] = 1 \dot{-} y = \dot{-} (1, y) = \dot{-} (C_1^1,\ U_1^1) =$

$= S_2^1(\dot{-},\ C_1^1,\ U_1^1)$;

$f(x', y) = 1 \dot{-} [((x + 1) \dot{-} y) + (y \dot{-} (x + 1))] = 1 \dot{-} [(S(x) \dot{-} y) + (y \dot{-} S(x))] =$

$= 1 \dot{-} \sum[\dot{-}(S(x), y), \dot{-}(y, S(x))] =$

$= 1 \dot{-} \sum[(S_2^3(\dot{-},\ S(U_2^3),\ U_3^3),\ S_2^3(\dot{-},\ U_3^3,\ S(U_2^3))] =$

$= \dot{-}[C_1^3,\ S_2^3(\sum,\ S_2^3(\dot{-},\ S(U_2^3),\ U_3^3),\ S_2^3(\dot{-},\ U_3^3,\ S(U_2^3)))] =$

$= S_2^3[\dot{-},\ C_1^3,\ S_2^3(\sum,\ S_2^3(\dot{-},\ S(U_2^3),\ U_3^3),\ S_2^3(\dot{-},\ U_3^3,\ S_1^3(S,\ U_2^3)))]$.

Ответ: $f(x, y) = R^2[S_2^1(\dot{-},\ C_1^1,\ U_1^1),\ S_2^3(\dot{-},\ C_1^3,\ S_2^3(\sum,\ S_2^3(\dot{-},\ S(U_2^3), U_3^3),\ S_2^3(\dot{-},\ U_3^3,\ S(U_2^3))))]$.

Задача 4.7. Доказать примитивную рекурсивность степени:
$f(x, y) = st(x, y) = x^y$.

Решение. Запишем схему рекурсии без параметров:

$f(x, 0) = x^0 = 1 = C_1^1$;

$f(x, y') = x^{y+1} = f(x, y)x = \Pi(U_1^3, U_2^3) = S_2^3[\Pi, U_1^3, U_2^3]$.

Ответ: $f(x, y) = R^2[C_1^1, S_2^3(\Pi, U_1^3, U_2^3)]$.

Задача 4.8. Доказать примитивную рекурсивность модуля разности:

$$f(x, y) = mod(x, y) = \begin{cases} x \dot{-} y, \text{ если } x \geq y; \\ y \dot{-} x, \text{ иначе.} \end{cases}$$

Решение. Данную функцию можно представить через псевдоразность: $f(x, y) = (x \dot{-} y) + (y \dot{-} x)$.

Запишем схему рекурсии с параметрами:

$f(0, y) = (0 \dot{-} y) + (y \dot{-} 0) = y = U_1^1$;

$f(x+1, y) = ((x+1) \dot{-} y) + (y \dot{-} (1+x)) = (S(x) \dot{-} y) + (y \dot{-} S(x)) =$

$= \sum((S(x) \dot{-} y), (y \dot{-} S(x))) = S_2^3(\sum, S_2^2(\dot{-}, S(x), U_3^3), S_2^3(\dot{-}, U_3^3, S(x))) =$

$= S_2^3(\sum, S_2^3(\dot{-}, S_1^3(S, U_2^3), U_3^3), S_2^3(\dot{-}, U_3^3, S_1^3(S, U_2^3)))$.

Ответ:

$f(x, y) = R^2(U_1^1, S_2^3(\sum, S_2^3(\dot{-}, S_1^3(S, U_2^3), U_3^3), S_2^3(\dot{-}, U_3^3, S_1^3(S, U_2^3))))$.

Задача 4.9. Доказать примитивную рекурсивность функции «больше»:

$$f(x, y) = more(x, y) = \begin{cases} 1, \text{ если } x > y; \\ 0, \text{ иначе.} \end{cases}$$

Решение. Данную функцию можно представить через псевдоразность: $f(x, y) = 1 \dot{-} ((y + 1) \dot{-} x)$.

Запишем схему рекурсии с параметрами:

$f(0, y) = 1 \dot{-} (y+1) = 1 \dot{-} y \dot{-} 1 = 0 \dot{-} y = 0 = C_0^1$;

$f(x+1, y) = 1 \dot{-} ((y+1) \dot{-} (x+1)) = 1 \dot{-} (S(y) \dot{-} S(x)) =$

$= 1 \dot{-} (\dot{-}, (S(U_3^3), S(U_2^3))) = 1 \dot{-} (S_2^3(\dot{-}, S(U_3^3), S(U_2^3))) =$

$= \dot{-}(C_1^3, S_2^3(\dot{-}, S(U_3^3), S(U_2^3))) =$

$= S_2^3(\dot{-}, C_1^3, S_2^3(\dot{-}, S_1^3(S, U_3^3), S_1^3(S, U_2^3)))$.

Ответ: $f(x, y) = R^2 (C_0^1, S_2^3(\dot{-}, C_1^3, S_2^3 (\dot{-}, S_1^3(S, U_3^3), S_1^3(S, U_2^3))))$.

Задача 4.10. Доказать примитивную рекурсивность функции «больше или равно»:

$$f(x, y) = moreql(x, y) = \begin{cases} 1, \text{ если } x \geq y; \\ 0, \text{ иначе.} \end{cases}$$

Решение. Данную функцию можно представить через псевдоразность: $f(x, y) = 1 \dot{-} (y \dot{-} x)$.

Запишем схему рекурсии с параметрами:

$f(0, y) = 1 \dot{-} y = S_2^1(\dot{-}, C_1^1, U_1^1)$;

$f(x + 1, y) = 1 \dot{-} (y \dot{-} (x + 1)) = 1 \dot{-} (y \dot{-} S(U_2^3)) = 1 \dot{-} (\dot{-}(U_3^3, S_1^3 (S, U_2^3))) = 1 \dot{-} S_2^3(\dot{-}, U_3^3, S_1^3 (S, U_2^3)) = S_2^3 (\dot{-}, C_1^3, S_2^3(\dot{-}, U_3^3, S_1^3 (S, U_2^3))$.

Ответ: $f(x, y) = R^2 (S_2^1(\dot{-}, C_1^1, U_1^1), S_2^3(\dot{-}, C_1^3, S_2^3 (\dot{-}, U_3^3, S_1^3(S, U_2^3))))$.

Задача 4.11. Доказать примитивную рекурсивность остатка от деления аргумента x на 2.

$f(x) = rest_2(x)$.

Решение. Запишем схему рекурсии без параметров:

$rest_2(0) = 0$

$$rest_2(x') = \begin{cases} 1, \text{ если } rest_2(x) = 0; \\ 0, \text{ если } rest_2(x) = 1; \end{cases}$$

$rest_2(x') = unsg\ (rest_2\ (x)) = unsg\ (U^2_1) = S^2_1(unsg, U^2_1)$.

Ответ: $f(x) = R_0\ [S^2_1(unsg, U^2_1)]$.

Задача 4.12. Доказать примитивную рекурсивность целой части от деления аргумента x на 2:

$f(x) = div2(x)$.

Решение. Запишем схему рекурсии без параметров:

$div_2(0) = 0$;

$div_2(x') = div_2(x) + mod_2(x)$;

$div_2(x') = div_2(U_2^2) + mod_2\ U_2^2) = U_2^1 + S^2_1(mod_2, U_2^2) =$

$= S_2^2\ [\Sigma, U_2^1, S_2^1(mod_2, U_2^2)]$.

Ответ: $f(x, y) = R_0 (S^2_2 [\Sigma, U^2_1, S^2_1(mod_2, U^2_2)])$.

Задача 4.13. Доказать примитивную рекурсивность целой части от деления аргумента x на y:

$f(x, y) = div(x, y)$.

Решение. Чтобы иметь дело со всюду определенными функциями, дополнительно положим $div(x, 0) = 0$.

Согласно определению, при $y>0$ число $div(x, y) = n$ удовлетворяет соотношениям

$$ny \le x < (n + 1)y.$$

Отсюда видно, что n равно числу нулей в последовательности

$$1y \dot{-} x, 2y \dot{-} x, \ldots, ny \dot{-} x, \ldots, xy \dot{-} x.$$

Поэтому для $y>0$ имеем формулу

$$div(x, y) = \sum (unsg(iy \dot{-} x)) \qquad (i = 1, ..., x).$$

Непосредственная проверка показывает, что формула верна и при $y = 0$. Так как функция $(unsg(iy \dot{-} x))$, стоящая под знаком суммы, примитивно рекурсивная, то на основании теоремы о суммировании заключаем, что функция $div(x, y)$ примитивно рекурсивна.

Задача 4.14. Доказать примитивную рекурсивность остатка от деления аргумента x на y:

$f(x, y) = rest(x, y)$.

Решение. Чтобы иметь дело с всюду определенными функциями, дополнительно положим, что $rest(x, 0) = x$ для всех x. Ясно, что так определенная функция связана с только что доказанной функцией $div(x, y)$ тождеством

$$rest(x, y) = x \dot{-} (y \cdot div(x, y)),$$

и, значит, из примитивной рекурсивности функции $div(x, y)$ напрямую вытекает и примитивная рекурсивность функции $rest(x, y)$.

Задача 4.15. Доказать примитивную рекурсивность числа различных делителей x, включая число 1:

$f(x) = nd(x)$.

Решение. Говорят, что число x делится (без остатка) на число y или что y делит x, если $rest(x, y) = 0$.

Тогда верно соотношение

$$nd(x) = \sum unsg(rest(x, i))) \quad (i =0,\ldots, x).$$

Если $x \neq 0$, то делители числа x не превосходят x и потому для положительных значений x число $nd\ (x)$ совпадает с числом различных делителей x (включая число 1). Из формулы видно, что функция $nd\ (x)$ примитивно рекурсивна.

Задача 4.16. Доказать примитивную рекурсивность нумерации пар чисел $f\ (x, y) = c(x, y)$.

Решение. Все пары натуральных чисел можно расположить в простую последовательность, и притом многими способами.

Для определенности рассмотрим следующее расположение этих пар, которое будем называть канторовским:

(0, 0); (0, 1), (1, 0); (0, 2), (1, 1), (2, 0); (0, 3), …

В этой последовательности пары идут в порядке возрастания суммы их членов, а из пар с одинаковой суммой членов ранее идет пара с меньшим первым членом.

Обозначим через $c(x, y)$ номер пары (x, y) в последовательности, причем нумерацию начинаем с нуля.

Таким образом, $c(0, 0) = 0$, $c(0, 1) = 1$, $c(1, 0) = 2$ и т.д. Получим (попробуйте самостоятельно разобраться почему):

$c(x, y) = ½ \cdot (x + y)(x + y + 1) + x = ½ \cdot ((x + y)^2 + 3x + y) =$
$= div_2((x + y)^2 + 3x + y).$

Функция div_2 примитивно-рекурсивна, а значит и функция с (x, y) – тоже примитивно рекурсивна.

Задача 4.17. Доказать примитивную рекурсивность $f\ (x, y, z) = x^3 + y^3 + 2xy^2 + yz^{12}$. Рекурсию взять по первой переменной.

Решение:

$f(0, y, z) = y^3+yz^{12} = S_3^2(\prod, U_1^2, U_1^2, U_1^2)+\prod(y, S_{12}^2(\prod, U_2^2,\ldots, U_2^2)) =$

$=S_2^2(\sum, S_3^2(\prod, U_1^2, U_1^2, U_1^2), S_2^2(\prod, U_1^2, S_{12}^2(\prod, U_2^2,..., U_2^2)))$

$f(x+1, y, z) = (x+1)^3 +y^3 +2(x+1)y^2+yz^{12} =$

$= \prod(S(x), S(x), S(x))+S_3^4(\prod, U_3^4, U_3^4, U_3^4)+S_3^4(\prod, C_2^4, S(U_2^4),$
$S_2^4(\prod, U_3^4, U_3^4))+S_2^4(\prod, U_3^4, S_{12}^4(\prod, U_4^4,\ldots, U_4^4)) =$

$= S_5^4(\sum, S_3^4(\prod, S(U_2^4), S(U_2^4), S(U_2^4)), S_3^4(\prod, U_3^4, U_3^4, U_3^4),$
$S_3^4(\prod, C_2^4, S(U_2^4), S_2^4(\prod, U_3^4, U_3^4)), S_2^4(\prod, U_3^4, S_{12}^4(\prod, U_4^4,\ldots, U_4^4))).$

Ответ: $f\ (x, y) = R^3(S_2^2(\sum, S_3^2(\prod, U_1^2, U_1^2, U_1^2), S_2^2(\prod, U_1^2, S_{12}^2$
$(\prod, U_2^2,..., U_2^2))),S_5^4(\sum,S_3^4(\prod,S(U_2^4),S(U_2^4),S(U_2^4)), S_3^4(\prod, U_3^4, U_3^4,$
$U_3^4), S_3^4(\prod, C_2^4, S(U_2^4), S_2^4(\prod, U_3^4, U_3^4)),$
$S_2^4(\prod, U_3^4, S_{12}^4(\prod, U_4^4,\ldots, U_4^4)))).$

4.1.3. Методология восстановления примитивно-рекурсивных функций из схем примитивной рекурсии

Для восстановления функции из схемы примитивной рекурсии рекомендуется:

1) определить, от скольких переменных зависит данная функция (индекс при операции примитивной рекурсии)? Допустим, переменных n;

2) восстановить вид функции в точке ноль;

3) восстановить вид функции в точке $x + 1$ (x – переменная, по которой берется рекурсия);

4) подсчитать значение функции в точках $x = 1$, $x = 2$ (x – переменная, по которой берется рекурсия);

5) попытаться представить общий вид функции, основываясь на том, как растёт функция при увеличении значения переменной на единицу. Возможно, для этого потребуется подсчитать значение функции в нескольких дополнительных точках;

6) проверить правильность выбора функции, записав схему примитивной рекурсии для восстановленной функции и сравнить её с условием задачи.

Задача 4.18. Восстановить общий вид функции $f(x, y) = R_{60}[S_2^2(\dot{-}, U_1^2, C_3^2)]$.

Решение. Это функция от одной переменной, в точке ноль она принимает значение 60:

$$f(0) = 60.$$

Определим значение в следующей точке:

$$f(x') = U_1^2 \dot{-} 3 = f(x) \dot{-} 3.$$

То есть при увеличении значения аргумента значение функции уменьшается на 3, но не становится меньше 0. Сделаем предположение, что функция выглядит следующим образом:

$$f(x) = 60 \dot{-} 3 \cdot x.$$

Проверим для точки 0. Значение функции равно 60. Теперь запишем схему примитивной рекурсии для этой функции:

$$f(x, y) = R_{60}[S_2^2(\dot{-}, U_1^2, C_3^2))].$$

Данная схема совпала со схемой в задаче, т.е. функция определена верно.

Ответ: $f(x) = 60 \div 3x$.

Задача 4.19. Восстановите общий вид функции $f(x, y) = R^2[C_{10}^1, S_2^3(\Sigma, U_1^3, S_2^3(\Pi, U_3^3, C_8^3))]$. Рекурсия взята по первой переменной.

Решение. Это функция от двух переменных, в точке ноль она принимает значение 10:

$$f(0, y) = 10.$$

Определим значение в следующей точке:

$$f(x', y) = U_1^3 + S_2^3(\Pi, U_3^3, C_8^3) = f(x, y) + U_3^3 \cdot C_8^3 = f(x, y) + 8y.$$

То есть при увеличении значения аргумента значение функции увеличивается на $8y$. Сделаем предположение, что функция выглядит так:

$$f(x, y) = 10 + 8xy.$$

Проверим для точки 0. Значение функции равно 10. Теперь запишем схему примитивной рекурсии для этой функции:

$$f(x, y) = R^2[C_{10}^1, S_2^3(\Sigma, U_1^3, S_2^3(\Pi, U_3^3, C_8^3))].$$

Данная схема совпала со схемой в задаче, т.е. функция определена верно.

Ответ: $f(x, y) = 10 + 8xy$.

4.2. Общерекурсивные и частично-рекурсивные функции

4.2.1. Теоретическая основа

Класс частично-рекурсивных функций определяется путем указания конкретных исходных функций и фиксированного множества операций получения новых функций из ранее заданных.

В качестве базисных обычно берутся следующие функции:

- следования;
- тождества (проекции или выбора аргументов);
- константы.

Допустимыми операциями над функциями являются операции суперпозиции (подстановки), примитивной рекурсии и минимизации.

Функция называется ***частично-рекурсивной***, если она может быть получена из простейших функций C_q^n, S, U_m^n конечным

числом операций подстановки, примитивной рекурсии и минимизации (т.е. задана в расширенном базисе Клини).

Таким образом, расширенный базис Клини состоит из трех простых функций (аналогичны стандартному базису Клини) и трех разрешенных операций (две из них аналогичны стандартному базису Клини, третья называется операцией минимизации).

Оператор минимизации. Пусть имеется функция $f(x_1, \ldots, x_n)$, принадлежащая множеству частичных арифметических функций (ЧАФ), и существует какой-то механизм ее вычисления, причем значение функции не определено тогда и только тогда, когда этот механизм работает бесконечно, не выдавая никакого определенного результата.

Фиксируем какие-нибудь значения $x_1, \ldots, x_n$ для первых $(n-1)$ аргументов функции f и рассмотрим уравнение

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

Чтобы найти решение y (натуральное) этого уравнения, будем вычислять при помощи указанного выше «механизма» последовательно значения $f(x_1, \ldots, x_{n-1}, y)$ для $y = 0, 1, 2, \ldots$ Наименьшее значение a, для которого получится $f(x_1, \ldots x_{n-1}, a) = x_n$, обозначим через

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

Описанный процесс нахождения выражения $\mu_y(f(x_1, \ldots, x_{n-1}, y) = x_n)$ будет продолжаться бесконечно в следующих случаях:

- значение $f(x_1, \ldots, x_{n-1}, 0)$ не определено;
- значения $f(x_1, \ldots, x_{n-1}, y)$ для $y = 0, 1, \ldots, a-1$ определены, но отличны от x_n, а значение $f(x_1, \ldots, x_{n-1}, a)$ – не определено;
- значения $f(x_1, \ldots, x_{n-1}, y)$ определены для всех $y = 0, 1, 2, \ldots$ и отличны от x_n.

Во всех этих случаях значение выражения $\mu_y(f(x_1, \ldots, x_{n-1}, y) = x_n)$ считается неопределенным. В остальных случаях описанный процесс обрывается и дает наименьшее решение $y = a$ уравнения $f(x_1, \ldots, x_{n-1}, y) = x_n$. Это решение, как сказано, и будет значением выражения $\mu_y(f(x_1, \ldots, x_{n-1}, y) = x_n)$.

Например, для функции разности $f(x, y) = x - y$ в соответствии с указанным смыслом символа μ, для всех x, y имеем

$$f(x, y) = x - y = \mu_z(y + z = x).$$

Подсчет значений функции $f(x, y) = x - y$ в этом случае будет проходить по двум сценариям.

Пример 1. Пусть $x = 5$, $y = 3$. Последовательно, начиная с $z = 0$, перебираем возможные значения для нахождения $\mu_z(y + z = x)$:

- при $z = 0$ получим выражение $3 + 0 = 5$, его значение – «ложь»;
- при $z = 1$ получим выражение $3 + 1 = 5$, его значение – «ложь»;
- при $z = 2$ получим выражение $3 + 2 = 5$, его значение – «истина», следовательно, процесс обрывается, и получаем результат: $\mu_z(3 + z = 5) = 2$.

Таким образом, не умея производить операцию вычитания (которая отсутствует в базисе Клини), мы смогли посчитать значение частичной функции $f(x, y) = x-y$ для двух заданных значений аргументов.

Пример 2. Пусть $x = 3$, $y = 5$. Последовательно, начиная с $z = 0$, перебираем возможные значения для нахождения $\mu_z(y + z = x)$:

- при $z = 0$ получим выражение $5 + 0 = 3$, его значение – «ложь»;
- при $z = 1$ получим выражение $5 + 1 = 3$, его значение – «ложь»;
- при $z = 2$ получим выражение $5 + 2 = 3$, его значение – «ложь» и т.д.

То есть при увеличении z ситуация, при которой выражение $5 + z = 3$ будет иметь значение «истина», так и не наступит, а значит, значение функции $\mu_z(5 + z = 3)$ так и не будет найдено.

Пример 2 – иллюстрация случая, при котором оператор минимизации не дает результата именно тогда, когда такой результат не может быть получен в принципе, по той причине, что в заданной точке функция действительно не определена.

Напротив, значение выражения $\mu_y((y + 1)(y-(x + 1)) = 0)$ не определено, так как уже при $y = 0$ значение терма $1\cdot(0-(x + 1))$ для любого x не определено. В то же время уравнение $(y + 1)(y-(x + 1)) = 0$ имеет решение $y = x + 1$, но оно не совпадает со значением выражения $\mu_y((y + 1)(y-(x + 1)) = 0)$.

Этот пример показывает, что для частичных функций $f(x_1, \ldots, x_{n-1}, y)$ выражение $\mu_y(f(x_1, \ldots, x_{n-1}, y) = x_n)$, строго говоря, не

есть наименьшее решение уравнения $f(x_1, \ldots, x_{n-1}, y) = x_n$. Если же функция $f(x_1, \ldots, x_{n-1}, y)$ всюду определенная и уравнение $f(x_1, \ldots, x_{n-1}, y) = x_n$ имеет решения, то $\mu_y(f(x_1, \ldots, x_{n-1}, y) = x_n)$ есть наименьшее решение для этого уравнения. Отсюда возникает закономерное желание использовать под оператором минимизации только всюду определенные функции. Наилучший способ убедиться в том, что функция всюду определена, – использовать заведомо примитивно рекурсивные функции (или функции, примитивная рекурсивность которых уже доказана ранее или вытекает из способа их задания). В этом случае значения $f(x_1, \ldots, x_n)$ будут неопределенными только в одном из трех случаев, а именно: если значения $f(x_1, \ldots, x_{n-1}, y)$ определены для всех $y = 0, 1, 2, \ldots$ и отличны от x_n. Для этого имеющуюся функцию нужно преобразовать таким образом (путем переноса слагаемых, умножения обеих частей, и т.д.), чтобы по обе стороны равенства стояли примитивно рекурсивные (а значит, всюду определенные) выражения. В общем виде под оператором минимизации получим некое рекурсивное уравнение, части которого зависят от переменных $(x_1, \ldots, x_{n-1}, x_n, y)$. Это уравнение принимает значение «истина» или «ложь» при последовательно выбираемых значениях параметра y.

Например, для нигде не определенной функции $f(x) = -x - 1$ рекурсивное уравнение будет выглядеть так: $f(x) = \mu_z(z + x + 1 = 0)$. При этом результат операции минимизации не определен даже для точки $x = 0$.

Для функции, определенной в одной точке, например $f(x) = -x$, рекурсивное уравнение будет выглядеть так: $f(x) = \mu_Z(z + x = 0)$. Результат операции минимизации определен только для точки $x = 0$, при остальных значениях x вычисление (подбор нужного значения z) никогда не будет закончено.

Значение операции минимизации, примененное к функции $f(x, y)$, можно записать как $M_f(x, y)$.

Тезис Черча. Класс алгоритмически (или машинно) вычислимых частичных арифметических функций совпадает с классом всех частично рекурсивных функций (рис. 4.1).

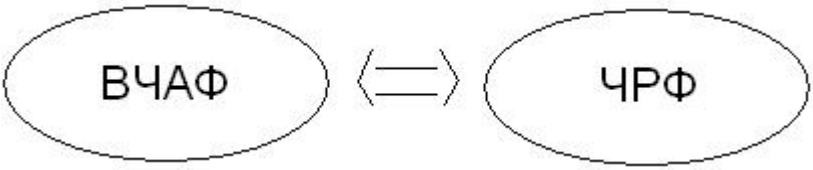

Рис. 4.1. Эквивалентность классов функций ВЧАФ и ЧРФ

Операция минимизации, введенная ранее, ставит в соответствие любой заданной функции f определенную частичную функцию M_f. Введем еще одну операцию, которую будем обозначать символом M^l_f и называть ***слабой минимизацией***.

По определению положим $M^l_f = M_f$, если функция M_f всюду определена.

Частичная арифметическая функция f называется ***общерекурсивной***, если она может быть получена из простейших функций C_q^n, S, U_m^n конечным числом операций подстановки, примитивной рекурсии и *слабой минимизации*.

Согласно определению, каждая примитивно-рекурсивная функция является общерекурсивной. Классы рекурсивных функций отображены на рис. 4.2.

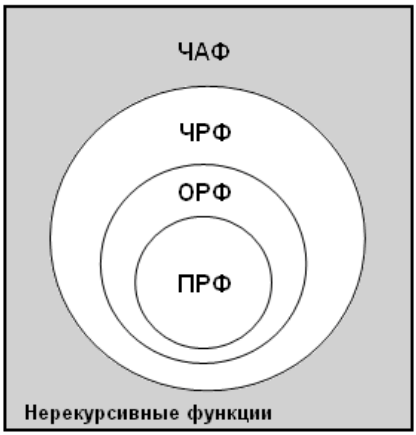

Рис. 4.2. Диаграмма вхождения классов ПРФ, ЧРФ, ОРФ, нерекурсивных функций в класс ЧАФ

Согласно тезису Черча, класс всюду определенно вычислимых числовых функций также будет совпадать с классом общерекурсивных функций (рис. 4.3).

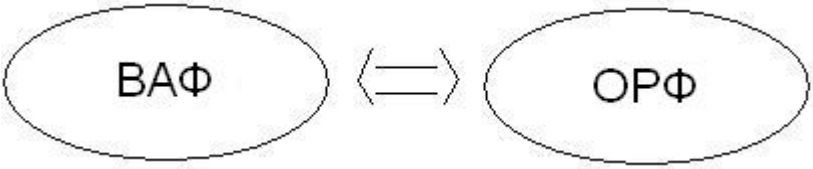

Рис. 4.3. Эквивалентность классов функций ВАФ и ОРФ

4.2.2. Обращение функций

Представляет интерес частный случай применения операции минимизации – функции одной переменной. Тогда вместо $M_f(x)$ используется запись $f^{-1}(x)$. В этом случае $f^{-1}(x) = \mu_y(f(y) = x)$ называется обращением функции $f(x)$.

Например, для знаковой функции $sg(x)$ обращение будет выглядеть следующим образом:

$$sg^{-1}(x) = \mu_y(f(y) = x) = \begin{cases} x, \text{ при } x = 0, 1; \\ \text{не определена, при } x>1. \end{cases}$$

Для рассмотренной ранее функции следования $s(x)$ обращение будет выглядеть следующим образом:

$$s^{-1}(x) = \mu_y(f(y) = x) = \begin{cases} x-1, \text{ при } x>0; \\ \text{не определена, при } x = 0. \end{cases}$$

Задача 4.20. Запишите через базисные операторы (схема рекурсии не нужна) обращение функции $f(x) = 2x$ и найдите значение $f^{-1}(6)$.

Решение:

$f^{-1}(x) = \mu_y(f(y) = x) = \mu_y(2y = x)$.

$f^{-1}(6) = \mu_y(2y = 6) = 3$.

Задача 4.21. ***З***апишите через базисные операторы (схема рекурсии не нужна) обращение функции $f(x) = x^3$ и найдите значение $f^{-1}(27)$.

Решение:

$f^{-1}(x) = \mu_y(f(y) = x) = \mu_y(y^3 = x)$.

$f^{-1}(27) = \mu_y(y^3 = 27) = 3$.

Задача 4.22. Запишите через базисные операторы (схема рекурсии не нужна) обращение функции $f(x) = x+5$ и найдите значение $f^{-1}(0), f^{1}(3), f^{-1}(5), f^{-1}(10)$.

Решение:

$f^{-1}(x) = \mu_y(f(y) = x)$;

$f^{-1}(0) = \mu_y(f(y) = 0) = \mu_y(y + 5 = 0)$ – не опр.;

$f^{-1}(3) = \mu_y(f(y) = 3) = \mu_y(y + 5 = 3)$ – не опр.;

$f^{-1}(5) = \mu_y(f(y) = 5) = \mu_y(y + 5 = 5) = 0$;

$f^{-1}(10) = \mu_y(f(y) = 10) = \mu_y(y + 5 = 20) = 5$.

4.2.3. Дополнительные способы задания примитивно-рекурсивных функций

Произвольная система рекурсивных уравнений задает частично-рекурсивную функцию. Если данная функция определена на всем множестве натуральных чисел, то она будет общерекурсивной. Также верно другое утверждение: функция является общерекурсивной, если она определена посредством произвольной системы рекурсивных уравнений.

Система уравнений может быть задана через базисные операции в виде схемы рекурсии. Например, рассмотрим систему уравнений, задающую функцию $F(x)$:

$$\begin{cases} E(x) = R_0(U^2_1); \\ F(x) = E^{-1}(x). \end{cases}$$

Если раскрыть операции рекурсии и обращения функций, получим

$$\begin{cases} E(0) = 0; \\ E(x + 1) = E(x); \\ F(x) = \mu_y[E(y) = x]. \end{cases}$$

Для $x = 0$ значение $F(x)$ определено и равно 0. Но для $x = 1$ наше определение «не работает», так как *выражение* $F(1) = \mu_y[E(y) = 1]$, означает:

- сначала посмотри, истинно ли $E(0) = 1$;
- если нет, посмотри, истинно ли $E(1) = 1$;
- если нет, посмотри, истинно ли $E(2) = 1$;

- если нет, то ... и т.д.

Вычисление никогда не заканчивается, потому что $E(x)$ всегда равна нулю, и для $F(1)$ не будет найдено никакого значения.

Вследствие этого при рассмотрении функций, заданных системами рекурсивных уравнений, будем обычно говорить о *частично-рекурсивной функции*. Когда говорится о частично-рекурсивной функции $F(x)$, то надо понимать, что может не существовать значения, определенного для некоторого (или даже любого!) значения x. Если $F(x)$ *о*казывается определенной для всех значений x, то будем называть ее *общерекурсивной функцией*. Конечно, любая общерекурсивная функция также является частично-рекурсивной.

Расширим класс операций, которые дают в качестве результата примитивно-рекурсивные функции.

Теорема 4.1. Пусть n-местная функция $g(x_1,..., x_{n-1}, x_n)$ примитивно-рекурсивная. Тогда n-местная функция $f(x_1,....., x_{n-1}, x_n)$, определенная равенством $f(x_1,\ldots, x_{n-1}, x_n) = \sum g(x_1,\ldots, x_{n-1}, i)$, где i изменяется от 0 до x_n, также примитивно-рекурсивная.

Теорема 4.2. Пусть n-местная функция $g(x_1,\ldots, x_{n-1}, x_n)$ примитивно-рекурсивна. Тогда n-местная функция $f(x_1,\ldots, x_{n-1}, x_n)$ определенная равенством $f(x_1,\ldots, x_{n-1}, x_n) = \prod_i g(x_1,\ldots, x_{n-1}, i)$, где i изменяется от 0 до x_n, также примитивно-рекурсивна.

Пусть заданы некоторые функции $f_i(x_1,..., x_n)$, $I = 1,\ldots, s+1$ и указаны какие-то условия $P_j(x_1,..., x_n)$, $j=1,\ldots, s$, которые для любого набора чисел $x_1,..., x_n$ могут быть истинными или ложными. Допустим, что для любого набора чисел $x_1,..., x_n$ никакие два из упомянутых условий не могут быть одновременно истинными. Функция $f(x_1,..., x_n)$, заданная схемой

$$f(x_1,..., x_n) = \begin{cases} f_1(x_1,\ldots, x_n), \text{ если } P_1(x_1,\ldots, x_n) \text{ истинно;} \\ f_2(x_1,\ldots, x_n), \text{ если } P_2(x_1,\ldots, x_n) \text{ истинно;} \\ \ldots\ldots \\ f_s(x_1,\ldots, x_n), \text{ если } P_s(x_1,\ldots, x_n) \text{ истинно;} \\ f_{s+1}(x_1,\ldots, x_n), \text{ для остальных } x_1,\ldots, x_n, \end{cases}$$

называется ***кусочно-заданной***.

Вопрос о том, будет ли функция f примитивно-рекурсивной, зависит от природы функций f_i и условий P_i. Зато для простейшего случая можно доказать очень полезную теорему.

Теорема 4.3. Пусть заданы n-местные примитивно-рекурсивные функции $f_1(x_1,\ldots, x_n)$, $f_2(x_1,\ldots, x_n)$, …, $f_{s+1}(x_1,\ldots, x_n)$, $a_1(x_1,\ldots, x_n)$, $a_2(x_1,\ldots, x_n)$, …, $a_s(x_1,\ldots, x_n)$, причем ни при каких значениях переменных никакие две из функций a_i одновременно в 0 не обращаются. Тогда функция, определенная кусочной схемой

$$f(x_1,\ldots,x_n) = \begin{cases} f_1(x_1,\ldots,x_n), \text{ если } a_1(x_1,\ldots,x_n)=0; \\ f_2(x_1,\ldots,x_n), \text{ если } a_2(x_1,\ldots,x_n)=0; \\ \\ f_s(x_1,\ldots,x_n), \text{ если } a_s(x_1,\ldots,x_n)=0; \\ f_{s+1}(x_1,\ldots,x_n), \text{ для остальных } x_1,\ldots,x_n, \end{cases}$$

будет примитивно-рекурсивной.

Задача 4.23. Определить, является ли примитивно-рекурсивной функция $f(x)$, заданная кусочным образом:

$$f(x) = \begin{cases} x, \text{ если } x>50; \\ 2x, \text{ если } x<30; \\ q(x), \text{ иначе,} \end{cases}$$

где $q(x)$ – рекурсивная, но не примитивно-рекурсивная функция.

Решение. Рассмотрим значения данной функции на всех интервалах. При x больше 50 она является ПРФ, при x меньше 30 тоже. Теперь рассмотрим интервал от 30 до 50. Поскольку на этом интервале функция равна рекурсивной, не примитивно-рекурсивной функции, то она не примитивно-рекурсивна, но необходимо рассмотреть, является ли интервал бесконечным. Интервал – конечный, т.е. на этом интервале можно задать функцию перечислением значений, и поэтому она является ПРФ. Если бы интервал был бесконечным, то функция не была бы примитивно-рекурсивной.

Ответ: функция $f(x)$ примитивно-рекурсивная, так как может быть выражена через функции, примитивная рекурсивность которых доказана ранее: $f(x) = x\cdot sg(x\dot{-}50) + 2x\cdot sg(30\dot{-}x) + q(30)\cdot eql(x,30) + q(31)\cdot eql(x,31) + \ldots + q(50)\cdot eql(x,50)$.

Обратим внимание на очень важный момент. Большой ошибкой было бы думать, что функция, выраженная при помощи оператора минимизации, однозначно является непримитивно-рекурсивной. В

ряде случаев использование оператора минимизации есть не более чем удобный способ задания процедуры вычисления, позволяющий обойти определенные формальные трудности. Как будет показано ниже, иногда пусть и более сложным образом, но все же удается заменить процедуру поиска минимально возможного решения для уравнения, задающего всюду определенную функцию, на последовательность примитивно-рекурсивных операций.

Рассмотрим уравнение $g(x_1,\ldots, x_n, y) = 0$, левая часть которого есть всюду определенная функция. Допустим, для каждого $x_1, \ldots, x_n$ уравнение имеет единственное решение y. Тогда это решение будет однозначной всюду определенной функцией от $x_1,\ldots, x_n$. Актуален вопрос: будет ли функция $y = f(x_1,\ldots, x_n)$ примитивно-рекурсивной, если левая часть уравнения, а именно: $g(x_1,\ldots, x_n, y)$, есть примитивно рекурсивная функция?

В общем случае это не так. Зато в некоторых отдельных случаях ответ будет положительным.

Теорема 4.4 (теорема о мажорируемых неявных функциях). Пусть $g(x_1,\ldots, x_n, y)$, $a(x_1,\ldots, x_n)$ – такие примитивно-рекурсивные функции, что уравнение $g(x_1,\ldots, x_n, y) = 0$ для каждых $x_1,\ldots, x_n$ имеет по меньшей мере одно решение и $\mu_y(g(x_1,\ldots, x_n, y) = 0) \le a(x_1,\ldots, x_n)$ для любых $x_1,\ldots, x_n$. Тогда функция $f(x_1,\ldots, x_n) = \mu_y(g(x_1,\ldots, x_n, y)=0)$ также примитивно-рекурсивная.

Задача 4.24. Доказать примитивную рекурсивность функции, вычисляющей $(n + 1)$-е простое число в натуральном ряде чисел

$f(n) = p_n$.

Решение. В арифметике натуральное число x называется простым, если оно имеет точно 2 различных делителя. Число 0 имеет бесконечно много делителей, а число 1 – лишь один. Поэтому 0 и 1 – не простые числа. Для простых чисел (2, 3, 5, 7, …) введем стандартные обозначения $p_0 = 2$, $p_1 = 3,\ldots$ Таким образом, p_n есть $(n + 1)$-е простое число в натуральном ряде чисел.

Обозначим через $\chi_p(n)$ характеристическую функцию свойства быть простым числом. Иначе говоря, положим $\chi_p(n) = 0$ для n простого и $\chi_p(n) = 1$ для n непростого. Так как простые числа и только они имеют ровно 2 делителя, то

$$\chi_p(x) = sg(nd\, x\text{-}2).$$

и, следовательно, функция $\chi_p(x)$ примитивно-рекурсивная.

Одной из наиболее известных арифметических функций являются функция $\pi(x)$, равная числу простых чисел, не превосходящих x.

Формула

$$\pi(x) = \sum_i \; unsg\, \chi_p(i)\ ,\ i = 0,\ldots,\ x$$

доказывает, что функция $\pi(x)$ примитивно рекурсивна.

Из определения функции $\pi(x)$ непосредственно следует, что

$$\pi(p_n) = n + 1;$$

$$\pi\,(x) < n + 1, \text{ если } x < p_n.$$

Отсюда видно, что $x = p_n$ есть минимальное решение уравнения $\pi(x) = n + 1$. Поэтому

$$p_n = p(n) = \mu_x(|\pi\,(x) - (n + 1)| = 0).$$

Стоящая под знаком μ-операции функция $|\pi(x) - (n + 1)|$ примитивно рекурсивна. В силу теоремы о мажорируемых неявных функциях для доказательства примитивной рекурсивности функции остается лишь найти такую примитивно-рекурсивную функцию $\alpha(n)$, чтобы для всех n было $p_n \le \alpha(n)$. Из теории чисел хорошо известно, что в качестве функции $\alpha(n)$ можно взять 2^{2^n}.

В самом деле, требующееся нам неравенство

$$p_n \le 2^{2^n}$$

заведомо истинно при n=0. По индукции далее предполагаем, что неравенство истинно для всех значений n, меньших некоторого числа s+1. Докажем, что оно истинно и для n=s+1.

По предположению имеем

$$p_0 p_1 \ldots p_s + 1 \le 2^{2^0 + 2^1 + \ldots + 2^s} + 1 < 2^{2^{s+1}}.$$

Число $a = p_0 p_1 \ldots p_s + 1$ больше 1 и потому должно иметь какой-то простой делитель p_r. Этот делитель не может совпадать ни с одним из простых чисел $p_0, p_1, \ldots, p_s$, так как при делении числа a на любое из чисел $p_0, p_1, \ldots, p_s$ получаем остаток 1. Но все простые числа, не превосходящие p_s, содержатся в последовательности p_0, p_1, … ,p_s. Число p_r в нее не входит и потому $p_{s+1} \le p_r$. Так как $p_r \le a$, то

$$p_{s+1} \le a \le 2^{2^{s+1}},$$

что и требовалось доказать.

Итак, неравенство доказано, а вместе с ним доказана и примитивная рекурсивность функции $f(n) = p_n$.

Задача 4.25. Доказать примитивную рекурсивность экспоненты числа p_x в числе y:

$f(x, y) = ex_x y$.

Решение. Экспонента числа p_x в числе y вычисляется при помощи функции $ex_x y$. По определению, при $y \neq 0$ $ex_x y$ полагаем равным показателю наивысшей степени простого числа p_x, на которую делится y. Для $y = 0$ полагаем, по определению $ex_x 0 = 0$ для всех значений x. Например:

$$ex_0 8 = 3,\ ex_1 8 = 0,\ ex_0 0 = 0,$$

так как $p_0 = 2$ (первое простое число), $p_1 = 3$ (второе).

Каждое натуральное число $n>1$ можно однозначно представить в виде произведения

$$n = p_{i_0}^{a_0} \cdot p_{i_1}^{a_1} \cdot \ldots \cdot p_{i_s}^{a_s},\ a_j>0,\ i_0<i_1<\ldots<i_s$$

положительных степеней различных простых чисел. Для удобства визуального восприятия, чтобы избавиться от нижнего индекса, будем обозначать функцию $ex_x y$ как $ex(x, y)$. Отсюда в силу определения функции $ex_x y$ получаем

$$ex(i_0, n) = a_0, \ldots, ex(i_s, n) = a_s.$$

При этом $ex(i, n) = 0$ для всех значений i, отличных от $i_0, \ldots, i_s$.

Чтобы доказать примитивную рекурсивность функции $ex_x y$, снова воспользуемся теоремой о мажорируемых неявных функциях. По определению, $ex(x, y + 1)$ есть наибольшее значение u, для которого p_x^u есть делитель $y + 1$. Поэтому можно утверждать также, что $ex(x, y + 1)$ есть наименьшее значение u, для которого p_x^{u+1} не делит $y + 1$, т.е.

$$ex(x, y + 1) = \mu_u(unsg(rest(y + 1, p_x^{u+1})) = 0).$$

Функция p_x^{u+1} примитивно рекурсивна и, кроме того,

$$ex(x, y + 1) \leq y + 1.$$

В силу упомянутой теоремы о мажорируемых неявных функциях, из вышесказанного следует, что функция $ex(x, y + 1)$, а с нею и $ex(x, y) = ex(x, (y \dot{-} 1) + 1)$ примитивно рекурсивны.

Задача 4.26. Доказать примитивную рекурсивность квадратичного остатка числа *x:*

$f(x) = q(x) = x \dot{-} [\sqrt{x}\,]^2$.

Решение. В функции

$$q(x) = x \dot{-} [\sqrt{x}\,]^2$$

символом $[z]$ обозначается целая часть вещественного числа z, равная наибольшему целому, не превосходящему z. Число $q(x)$ называется квадратичным остатком числа x. Оно равно расстоянию от x до ближайшего слева точного квадрата.

Равенство $n = [\sqrt{x}\,]$ равносильно соотношению

$$n^2 \le x < (n+1)^2,$$

где n – натуральное. Таким образом,

$$[\sqrt{x}\,] = \mu_t(sg((t+1)^2 \dot{-} x) = 1),$$

и вместе с тем $[\sqrt{x}\,] \le x$. По теореме о мажорируемой неявной функции отсюда следует, что функция $[\sqrt{x}\,]$ примитивно-рекурсивная. Вместе с нею примитивно-рекурсивной является и функция $q(x) = x \dot{-} [\sqrt{x}\,]^2$.

4.3. Методология нахождения значений частично-рекурсивных функций на основе систем рекурсивных уравнений

Задача 4.27. Найти общий вид функций $C(x)$, $E(x)$, задать их область определений и область значений с помощью характеристических функций. Найти значение $S(161)$, $S(460)$:

$$\begin{cases} C(x) = R_4(S_2^2(\sum, U_1^2, C_3^2); \\ E(x) = R_{100}(S_2^2(\dot{-}, U_1^2, C_2^2)); \\ S(x) = \mu_y(E(y) + C(y) = x). \end{cases}$$

Решение. Для первой функции получим после раскрытия схемы рекурсии:

$C(0) = 4$;

$C(x+1) = U_1^2 + C_3^2 = C(x) + 3$.

Общий вид функции: $C(x) = 4 + 3x$. Характеристическая функция для её области определений: $\Psi(x) = 1$. Характеристическая функция для области значений: $\Psi(y) = sg(y \dot{-} 3) \cdot (unsg(rest(y \dot{-} 4, 3)))$.

Для второй функции получим после раскрытия схемы рекурсии:

$E(0) = 100$;

$E(x+1) = U_1^2 \dot{-} C_2^2 = E(x) \dot{-} 2$.

Общий вид функции: $E(x) = 100 \dot{-} 2x$. Характеристическая функция для области определений $\Psi(x) = 1$. Характеристическая функция для области: $\Psi(y) = sg(101 \dot{-} y) \cdot rest(y+1, 2)$

Построим сводную таблицу (табл. 4.1).

Таблица 4.1

Значения функций $C(x)$, $E(x)$ и их суммы

x	$C(x)$	$E(x)$	$E(y) + C(y)$
0	4	100	104
1	7	98	105
2	10	96	106
3	13	94	107
…	…	…	…
49	151	2	153
50	154	0	154
51	157	0	157
52	160	0	160
53	163	0	163

Из табл. 4.1 видно, что значение функции $S(161)$ – не определено, так как сумма $E(y) + C(y)$ принимает значение 160, а затем уже 163, и далее после $x = 50$ растет по закону: $S(x) = 4 + 3x$ (поскольку вклад функции $E(x)$ теперь равен нулю). Отсюда следует, что $S(460) = 152$.

Ответ: $C(x) = 4 + 3x$, $E(x) = 100 \dot{-} 2x$, $S(161)$ – не определено, $S(460) = 152$. Характеристические функции области определений и области значений приведены в решении.

Задача 4.28. Найти общий вид $G(x)$, $E(x)$, задать их область определений и область значений с помощью характеристических функций. Найти значение $f(6), f(14), f(16)$.

$$\begin{cases} G(x) = R_{60}(S_2^2(\div, U_1^2, C_3^2)) \\ E(x) = R_0(S_2^2(\div, S_2^2(\Pi, U_2^2, C_{10}^2), C_{40}^2) \\ f(x) = \mu_y(G(y) + E(y) = 10x) \end{cases}$$

Решение. Для первой функции получим после раскрытия схемы рекурсии:

$G(0) = 60$;

$G(x+1) = G(x) \dot{-} 3$.

Общий вид функции: $G(x) = 60 \dot{-} 3x$. Характеристическая функция для её области определений: $\Psi(x) = 1$. Характеристическая функция для области значений: $\Psi(y) = sg(61 \dot{-} y) \cdot (unsg(rest(y, 3)))$.

Для второй функции получим после раскрытия схемы рекурсии:

$E(0) = 0$;

$E(x+1) = 10 \cdot x \dot{-} 40$;

$E(x) = 10 \cdot x \dot{-} 50$.

Общий вид функции: $E(x) = 10 \cdot x \dot{-} 50$. Характеристическая функция для области определений $\Psi(x) = 1$. Характеристическая функция для области: $\Psi(y) = unsg(rest(y, 10))$.

Построим сводную таблицу (табл. 4.2).

Таблица 4.2

Значения функций $G(x)$, $E(x)$ и их суммы

x	$G(x)=60 \dot{-} 3x$	$E(x)=10x \dot{-} 50$	$G(x)+E(x)$
0	60	0	60
1	57	0	57
2	54	0	54
3	51	0	51
4	48	0	48
5	45	0	45
6	42	10	52

Продолжение табл. 4.2

7	39	20	59
…	…	…	…
19	3	140	143
20	0	150	150
21	0	160	160

Вторая функция $E(x)$ начинает вносить свой вклад в общее поведение $f(x)$ только после значения $x = 6$. До этого момента значение функции $f(x)$ уменьшалось с шагом 3, а после этой точки начинает возрастать с шагом 7. Отсюда получим

$f(6) = \mu_y(G(y) + T(y) = 60) = 0$;

$f(14) = \mu_y(G(y) + T(y) = 140)$ – не определено;

$f(16) = \mu_y(G(y) + T(y) = 160) = 21$.

Важно обратить внимание, что после прохождения значения $x = 20$ значение функции $f(x)$ будет возрастать с шагом 10 и далее всегда оставаться кратным 10. По этой причине можно смело утверждать, что $f(x)$ будет определена на всём интервале от 16 до бесконечности.

Ответ: $G(x) = 60 \div 3x$, $E(x) = 10x \div 50$, $f(6) = 0$, $f(14)$ – не определено, $f(16) = 21$.

Характеристические функции области определений и области значений приведены в решении.

Задача 4.29. Найти общий вид $G(x)$, $E(x)$, задать их область определений и область значений с помощью характеристических функций. Найти значение $f(7), f(30), f(555)$.

$$\begin{cases} G(x) = R_6(S_2^2(\sum, U_1^2, C_6^2)) \\ E(x) = R_1(S_2^2(\Pi, U_1^2, S_1^2(\mathrm{S}, U_2^2))) \\ f(x) = \mu_y(G(y+1) + E(y+1) = x) \end{cases}$$

Решение. Для первой функции получим после раскрытия схемы рекурсии:

$G(0) = 6$;

$G(x+1) = G(x) + 6$.

Общий вид функции: $G(x) = 6x+6$. Характеристическая функция для её области определений: $\Psi(x) = 1$. Характеристическая функция для области значений: $\Psi(y) = sg(y \dot{-} 5) \cdot (unsg(rest(y, 6)))$.

Для второй функции получим после раскрытия схемы рекурсии:

$E(0) = 1$;

$E(x+1) = E(x)(x+1)$.

Общий вид функции: $E(x) = x!$ Характеристическая функция для её области определений: $\Psi(x) = 1$. Характеристическая функция для области значений: $\Psi(y) = sg(eql(y, 1))+ sg(eql(y, 2))+ sg(eql(y, 6))+ + sg(eql(y, 24))+…+ sg(eql(y, 1*2*…*n))+….$

Построим сводную таблицу (табл. 4.3). Из нее видно, что значение функции $f(7)$ не определено, поскольку $y + 1 = 0$, и, значит, y не существует. Зато можно легко найти $f(30)$: так как $G(x) + E(x) = 30$ при $x = 3$, значит, $y + 1 = 3$, и $y = 2$. Исходя из поведения функции, видно, что $f(555)$ не определено, поскольку 555 нечетное, а факториал числа, большего двух, всегда четный, равно как и функция $G(x) = 6x + 6$, а значит, их сумма тоже четное число.

Таблица 4.3

Значения функций $G(x)$, $E(x)$ и их суммы

x	$G(x)=6x+6$	$E(x)= x!$	$G(x)+E(x)$
0	6	1	7
1	12	1	13
2	18	2	20
3	24	6	30
4	30	24	54
5	36	120	156
6	42	720	762
7	48	5040	5088
8	54	40320	40374
9	60	362880	362940
10	66	3628800	3628866
11	72	39916800	39916872

Ответ: $G(x) = 6x + 6$, $E(x) = x!$, $f(7)$ – не определено, $f(30) = 2$, $f(555)$ – не определено.

Характеристические функции области определений и области значений приведены в решении.

4.4. Эффективная перечислимость и распознаваемость рекурсивных функций

Теорема 4.5. Примитивно-рекурсивные функции эффективно перечислимы.

Теорема 4.6. Множество частично-рекурсивных функций эффективно перечислимо.

Теорема 4.7. Общерекурсивные функции не поддаются эффективному перечислению.

Теорема 4.8. Примитивно-рекурсивные функции не поддаются эффективному распознаванию среди общерекурсивных функций.

Теорема 4.9. Общерекурсивные функции не поддаются эффективному распознаванию среди частично рекурсивных функций.

Проверочные задания и вопросы

Задача П.4.1. Для проверки и систематизации полученных знаний заполните табл. 4.4. В столбцах «Счетность» и «Эффективная перечислимость» напишите «+» или «–», в столбце «Мощность» укажите конкретное кардинальное число.

Таблица 4.4

Счетность и эффективная перечислимость основных классов функций

Множество	Счетность	Эффективная перечислимость	Мощность
ПРФ			
ОРФ			
ЧРФ			
НРФ			

Задача П.4.2. Определите, к каким указанным классам принадлежат приведенные ниже функции. Заполните табл. 4.5,

поставив значок «+» в соответствующей ячейке, если функция относится к данному классу и «−» в противоположном случае. Во всех заданиях $x, y \in N^*$.

а) $E(x) = \begin{cases} E(0) = 5; \\ E(x+1) = E(x) \cdot x; \end{cases}$

б) $F(x) = \begin{cases} E(0) = 0; \\ E(x+1) = E(x)+x; \\ F(x) = \mu_y[E(y) = x]; \end{cases}$

в) $F(m, n) = \begin{cases} n + 1, m=0; \\ F(m - 1, 1), m > 0, n = 0; \\ F(m - 1, F(m, n - 1)), m > 0, n > 0; \end{cases}$

г) $F(x)=\mu_y[y + x = 5];$

д) $F(x,y) = \begin{cases} x-y, \text{ если } x > y + 2; \\ 2y-x, \text{ если } x < y; \\ (x+y)/2, \text{ если } x = y; \end{cases}$

е) $F(x) = \mu_y[E(y) = 3]$, где $E(x) = \mu_t[t + x = 5];$

ж) $F(x) = \begin{cases} x, \text{ если машина Тьюринга } T_x \text{ не остановится на} \\ \quad \text{чистой ленте;} \\ 1, \text{ иначе;} \end{cases}$

з) $F(x) = \begin{cases} 8x, \text{ если машина Тьюринга } T_x \text{ напечатает символ} \\ \text{«Т» на чистой ленте хотя бы один раз} \\ \text{за первые 67 тактов;} \\ 7x + 11, \text{ иначе;} \end{cases}$

и) $F(x) = \begin{cases} x^3, \text{ если машина Тьюринга } T_x \text{ напечатает символ} \\ \text{«Т» на чистой ленте хотя бы один раз за первые } x \\ \text{шагов;} \\ 2\cdot q(x), \text{ иначе, где } q(x) \text{ – рекурсивная, но} \\ \text{не примитивно-рекурсивная функция;} \end{cases}$

к) $F(x) = \begin{cases} x^2, \text{ если машина Тьюринга } T_x \text{ остановится} \\ \text{на чистой ленте за первые 10 шагов;} \\ x / 2, \text{ иначе.} \end{cases}$

Таблица 4.5

Принадлежность функций к заданным классам

Функция	ПРФ	ОРФ	ЧРФ	НРФ
а)				
б)				
в)				
г)				
д)				
е)				
ж)				
з)				
и)				
к)				

Задача П.4.3. Найти общий вид $G(x)$, $E(x)$, задать их область определений и область значений с помощью характеристических функций. Найти значение $f(0), f(35), f(60)$.

$$\begin{cases} G(x) = R_{100}(S_2^2(\div, U_1^2, C_4^2)); \\ E(x) = R_{50}(S_2^2(\div, U_1^2, C_2^2)); \\ f(x) = \mu_y(G(y) + E(y) = 2x). \end{cases}$$

Задача П.4.4. Найти общий вид $G(x)$, $E(x)$, задать их область определений и область значений с помощью характеристических функций. Найти значение $f(12), f(404), f(199)$.

$$\begin{cases} G(x) = R_2(S_2^2(\sum, U_1^2, C_4^2)); \\ E(x) = R_0(S_2^2(\prod, C_8^2, S_2^2(\div, U_2^2, C_4^2))); \\ f(x) = \mu_y(G(y) + E(y) = x + 6). \end{cases}$$

ГЛАВА 5. СЛОЖНОСТЬ АЛГОРИТМОВ

5.1. Теоретическая основа

Временной сложностью алгоритма $T(x)$ называется число его шагов, необходимых для решения данной задачи размера x в худшем случае.

Пространственная сложность $S(x)$ – это число единиц памяти, необходимых для решения задачи размера x в худшем случае.

Введем понятие верхнего и нижнего порядка одной функции относительно другой функции.

Назовем арифметическую функцию $f(x)$ ***функцией одного верхнего порядка*** с функцией $g(x)$ и запишем $f(x) = O(g(x))$, если существует такое натуральное число $c > 0$, что, в конце концов (т.е. начиная с некоторого x) получим $f(x) \le c \cdot g(x)$.

Назовем арифметическую функцию $f(x)$ функцией ***одного нижнего порядка*** с функцией $g(x)$ и запишем $f(x) = o(g(x))$, если существует такое $c > 0$, что в конце концов (начиная с некоторого x) получим $f(x) \ge c \cdot g(x)$.

Назовем арифметическую функцию $f(x)$ ***функцией одного порядка*** с функцией $g(x)$, если она одного верхнего и одного нижнего порядка с функцией $g(x)$, $f(x) = o(g(x))$ & $f(x) = O(g(x))$.

Функции одного верхнего порядка с многочленами называются ***полиномиальными.***

Это не только все полиномы, но и некоторые трансцендентные функции. Все остальные функции – экспоненциальные, в широком смысле этого слова.

Строго говоря, ***экспоненциальными*** называются функции одного нижнего порядка с экспонентой.

Тогда функции между экспоненциальными и полиномиальными называются ***субэкспоненциальными.*** Обычно их не рассматривают. Экспоненциальные функции разделяются по скорости еще на несколько классов.

Арифметические функции $f(x)$ и $g(x)$ называются ***полиномиально-связанными*** или ***полиномиально-***

эквивалентнтными, если существуют такие многочлены $P_1(x)$ и $P_2(x)$, что, в конце концов (начиная с некоторого числа), $f(x) \le P_1 \cdot g(x)$ и $g(x) \le P_2 \cdot f(x)$.

5.2. Методология оценки сложности

Временная сложность алгоритма — это, в худшем случае, функция размера входных данных, которая показывает максимальное количество элементарных операций, затрачиваемых алгоритмом для решения экземпляра задачи указанного размера.

В наилучшем случае – функция размера входных данных, которая показывает минимальное количество элементарных операций, затрачиваемых алгоритмом для решения экземпляра задачи указанного размера.

Пространственная сложность алгоритма — это, в худшем случае, функция размера входных данных, которая показывает максимальное количество памяти, затрачиваемое алгоритмом для решения экземпляра задачи указанного размера.

В наилучшем случае – функция размера входных данных, которая показывает минимальное количество памяти, затрачиваемое алгоритмом для решения экземпляра задачи указанного размера.

В конечном счете функции пространственной и временной сложности должны определяться для конкретных машин (в операциях). Но теория сложности алгоритмов определяет общие типы алгоритмов, дифференцируя их по трудоемкости на два класса:

1) полиномиальные алгоритмы;
2) экспоненциальные алгоритмы.

Алгоритм, обе функции сложности которого полиномиальные, называется ***полиномиальным***.

Такой алгоритм считается хорошим, быстрым, практичным.

Алгоритм, у которого хотя бы одна из двух функций сложности экспоненциальная, называется ***экспоненциальным***.

Задача, решаемая полиномиальным алгоритмом, называется ***легкоразрешимой***.

Задача, которую нельзя решить полиномиальным алгоритмом, называется ***трудноразрешимой***.

В число трудноразрешимых задач входят алгоритмически неразрешимые задачи. Неразрешимость есть крайний случай экспоненциальности.

Для наглядности рассмотрим таблицу возможностей алгоритмов, отражающую объем вычислений (табл. 5.1). При этом объем вычислений, характеризующийся словом «сверхбольшой», считается на практике недоступным.

Таблица 5.1

Таблица роста объема вычислений

Размер задачи	Тип алгоритма	
	полиномиальный	экспоненциальный
Малый	Малый	Малый
Большой	Большой	Сверхбольшой
Сверхбольшой	Сверхбольшой	Сверхбольшой

Обычно не нужно точное определение функции сложности, а надо найти ее верхний или нижний порядок.

Для одноленточных машин Тьюринга временная сложность оценивается в тактах работы, а пространственная – в количестве клеток, пройденных машиной в ходе работы. $S(n) \leq T(n)$ (пространственная сложность всегда меньше или равна временной сложности алгоритма).

Для многоленточной машины Тьюринга временная сложность также определяется в тактах работы машины, но за один такт машина обрабатывает все свои ленты, поэтому $S(n) = O(T(n))$. Пространственная сложность имеет один верхний порядок с временной сложностью.

Задача 5.1. Вычислить точно сложность машины Тьюринга для вычисления функции $f(n) = 2n + 3$, а также произвести быструю приблизительную оценку сложности этой машины. Вычисления производятся в так называемом унарном коде. Натуральные числа представляются палочками: 0 – одной палочкой, 1 – двумя; 2 – тремя и т.д. Значение аргумента n записывается на машинной ленте $n+1$ палочками после начальной метки; значение функции $f(n) = 2n + 3$ записывается $2n+4$ палочками через клетку от значения аргумента. Закончив запись значения функции, машина возвращается в начальную клетку и останавливается. Значения аргумента и функции остаются на ленте.

Решение. Размер задачи – значение аргумента n. Выделим два этапа: копирование $n + 1$ палочки и добавление еще двух недостающих палочек. Копирование i–n палочки занимает $n + 1 – (i–1) + 1 + 2i + 1 = n + i + 4$ единиц времени (тактов), возвращение к ней $2i–1 + 1 + n + 1–i = n + i + 1$, а весь i–n цикл $2n + 2i + 5$. Следовательно, $n + 1$ циклов выполняется за время

$$2n(n + 1) + 2\sum i + 5(n + 1) = 3n^2 + 10n + 7,$$

а весь первый этап – за время $3n^2 + 10n + 8$, включая первый такт машины.

Прибавление двух палочек на втором этапе требует $2 + 2(n + 1) + 2 = 2n + 6$ единиц времени, возвращение в начальную клетку $2(n + 1) + 1 + n + 1 + 1 = 3n + 5$.

Следовательно, второй этап выполняется за время $5n + 11$, а все решение задачи – за время $3n^2 + 15n + 19$.

В общем случае задача может иметь много входных слов данного размера. Временная сложность $T(n)$ машины есть наибольшее время её работы при входных словах размера n. В нашем случае существует только одно входное слово данного размера, поэтому $T(n) = 3n^2 + 15n + 19$.

Пространственная, сложность $S(n)$ машины есть наибольший путь (наибольше число клеток), проходимой ею при входных словах размера n. Нетрудно видеть, что в нашем случае $S(n) = 1 + n + 1 + 1 + 2(n + 1) + 2 = 3n + 7$. При $n = 2$ получим: $T(2) = 61$, $S(2) = 13$. Функции $T(n)$ и $S(n)$ здесь суть многочлены (полиномы). Следовательно, машина имеет полиномиальную сложность и реализует хороший, быстрый алгоритм. Вычисление

функции $f(n) = 2n + 3$ – легко разрешимая задача, с точки зрения объема вычислений.

Можно также произвести приближенный расчет сложности. Так, i-й цикл первого этапа требует $O(n)$ единиц времени, а весь первый этап $(n + 1)O(n) + 1 = O(n \cdot n)$. Второй этап требует $O(n)$ единиц, откуда $T(n) = O(n \cdot n) + O(n) = O(n \cdot n)$. Ясно также, что $S(n) = O(n)$. Функции $T(n)$ и $S(n)$ суть полиномиальные функции. Задача легко разрешима.

Ответ. Точные значения функций сложности: $T(n) = 3n \cdot n + 15n + 19$, $S(n) = 3n + 7$. Приблизительная оценка: $T(n) = O(n \cdot n)$, $S(n) = O(n)$. Задача легко разрешима.

Проверочные задания и вопросы

Задача П.5.1. Найти пространственную и временную сложность алгоритма машины Тьюринга, вычисляющей в унарном коде функцию $f(x) = x^2$.

Задача П.5.2. Найти пространственную и временную сложность алгоритма машины Тьюринга, вычисляющей в унарном коде функцию $f(x) = x^y$.

Задача П.5.3. Найти пространственную и временную сложность алгоритма машины Тьюринга, вычисляющей в унарном коде функцию $f(x) = 2^x$.

ОТВЕТЫ НА ПРОВЕРОЧНЫЕ ЗАДАНИЯ И ВОПРОСЫ

Глава 1

П.1.1. {0, *Sa*}.

П.1.2. Последовательность «001» бесконечное число раз.

П.1.3. 2.

П.1.4. 168.

П.1.5. 7.

П.1.6. 246.

П.1.7. 6.

П.1.8. Программа заменяет все четные символы слова на противоположные.

П.1.9. *B* является эффективно распознаваемым в *A*, так как множества *A*, *B* и дополнение *B* до *A* эффективно перечислимы.

П.1.10. $0 \to 1$.

П.1.11. В исходном слове все символы «*a*» меняются на «*b*» и наоборот.

П.1.12. б.

П.1.13. В исходном слове все символы «*a*» меняются на «*b*» и наоборот.

П.1.14. Последовательность «σ*abaa*λ».

Глава 2

П.2.1.:

Множество	Счетность	Эффективная перечислимость	Мощность	Вычислимость
N	Да	Да	χ_0	+
*N**	Да	Да	χ_0	+
Z	Да	Да	χ_0	+
Q	Да	Да	χ_0	+
A	Да	Да	χ_0	+
R	Нет	Нет	χ	+/–

Продолжение табл.

Множество	Счетность	Эффективная перечислимость	Мощность	Вычислимость
T	Нет	Нет	χ	+/–
ВДЧ	Да	Нет	χ_0	+
I	Нет	Нет	χ	+/–
C	Нет	Нет	χ	+/–

П.2.2.:

Множество	Счетность	Эффективная перечислимость	Мощность
$R \setminus$ ВДЧ	Нет	Нет	χ
$T \setminus$ ВДЧ	Нет	Нет	χ
$A \cup$ ВДЧ	Да	Нет	χ_0
$Q \setminus I$	Да	Да	χ_0
$N \cup Z$	Да	Да	χ_0
$A \cap N$	Да	Да	χ_0
$T \cap A$	Да	Да	0
$Z \setminus N$	Да	Да	χ_0
$Q \cup T$	Нет	Нет	χ
$R \cap N$	Да	Да	χ_0

П.2.3.:

Множество	Счетность	Эффективная перечислимость	Мощность
$N \times N$	Да	Да	χ_0
$T \times T$	Нет	Нет	χ
$Q \times R$	Нет	Нет	χ
$N \times N \times N$	Да	Да	χ_0
N^N	Нет	Нет	χ
R^N	Нет	Нет	χ

Продолжение табл.

Множество	Счетность	Эффективная перечислимость	Мощность
N^R	Нет	Нет	χ_1
Точек на плоскости	Нет	Нет	χ
Точек в трехмерном пространстве	Нет	Нет	χ
Точек в χ_0-мерном пространстве	Нет	Нет	χ

П.2.4.:

Множество	Счетность	Эффективная перечислимость	Мощность
$P(N)$	Нет	Нет	χ
$P(T)$	Нет	Нет	χ_1
$P(N \times N)$	Нет	Нет	χ
$P(\varnothing)$	Да	Да	1
$P(P(\varnothing))$	Да	Да	2
$P(P(P(\varnothing)))$	Да	Да	4
Фигур на плоскости	Нет	Нет	χ_1
Тел в 3-х мерном пространстве	Нет	Нет	χ_1
Тел в n-мерном пространстве	Нет	Нет	χ_1
Тел в χ_0-мерном пространстве	Нет	Нет	χ_1

П.2.5.:

Множество	Счетность	Эффективная перечислимость	Мощность
1	Да	Да	χ_0
2	Да	Нет	χ_0
3	Да	Да	χ_0
4	Нет	Нет	χ
5	Да	Да	χ_0
6	Да	Да	χ_0
7	Нет	Нет	χ
8	Нет	Нет	χ
9	Нет	Нет	χ
10	Да	Да	χ_0

П.2.6.:

Операция	Результат
1	χ_1
2	χ
3	χ
4	χ
5	χ_1
6	χ
7	χ
8	χ_1
9	χ_0
10	χ_1

П.2.7.:

Элемент	$R\setminus Q$	$T\cap A$	$P((Z\setminus N^*)\cup N)$	$P(R^*R)$	$(R\setminus Z)\setminus T$	Q	$P((N^*\setminus N)\cap T)$
1	Нет	Нет	Нет	Нет	Да	Да	Нет
2	Нет	Нет	Нет	Нет	Нет	Да	Нет
3	Да	Нет	Нет	Нет	Да	Нет	Нет
4	Да	Нет	Нет	Нет	Нет	Нет	Нет
5	Нет	Нет	Нет	Нет	Нет	Нет	Нет
6	Нет	Да	Да	Да	Нет	Нет	Да
7	Нет	Нет	Да	Нет	Нет	Нет	Нет
8	Нет	Нет	Нет	Нет	Нет	Нет	Нет
9	Нет	Нет	Нет	Да	Нет	Нет	Нет
10	Нет	Нет	Да	Нет	Нет	Нет	Нет
11	Нет	Нет	Нет	Нет	Нет	Да	Нет
12	Нет	Нет	Нет	Нет	Нет	Да	Нет
13	Нет	Нет	Да	Нет	Нет	Нет	Нет
14	Да	Нет	Нет	Нет	Да	Нет	Нет
15	Да	Нет	Нет	Нет	Нет	Нет	Нет

Глава 3

П.3.1.:

Множество	Счётность	Эффективная перечислимость	Мощность
АФ	-	-	χ
ВАФ	+	-	χ_0
ЧАФ	-	-	χ
ВЧАФ	+	+	χ_0

П.3.2.:

Множество	Счётность	Эффективная перечислимость	Мощность
ЧАФ \ ВЧАФ	-	-	χ
АФ \ ВАФ	-	-	χ
ЧАФ ∩ ВЧАФ	+	+	χ_0
АФ ∩ ВЧАФ	+	+	χ_0
ВАФ \ ЧАФ	-	-	0
АФ ∪ ЧАФ	-	-	χ_0
ВАФ ∪ ВЧАФ	+	+	χ_0
ВЧАФ \ АФ	+	+	χ_0
Множество АФ, описываемых конечным числом слов	+	+	χ_0

П.3.3. Без ответа, это совсем несложно.

П.3.4:

а) $x \in \{6, 10\}$, $\chi(x) = unsg\,(eql\,(x, 6)\ eql\,(x, 10))$;
$y \in \{2, 6\}$, $\chi(y) = unsg\,(eql\,(y, 2) eql\,(y, 6))$;

б) $x \in (14, \infty)$, $\chi(x) = sg(x \dot{-} 14)$;
$y \in N$, $\chi(y) = 1$;

в) $x \in (9, \infty)$, $\chi(x) = sg(x \dot{-} 9)$;
$y \in \{0, 3, 6, 9, 12, 15\}$, $\chi(y) = unsg\,(rest\,(y, 3))$;
г) $x \in \{6, 7, 9\}$, $\chi(x) = unsg\,(eql\,(x, 6)\ eql\,(x, 7)\ eql\,(x, 9))$;
$y \in \{2, 3, 5\}$, $\chi(y) = unsg\,(eql\,(y, 2) eql\,(y, 3)\ eql\,(y, 5))$;
д) $x \in N$, $\chi(x) = 1$;
$y \in \{10, 11, 12, 13, 14\}$, $\chi(y) = more(x, 9)$;
е) $x \in \{11, 13, 15, 25\}$, $\chi(x) = unsg(eql(x, 11) eql(x, 13) eql(x, 15)\ eql\,(x, 25))$;
$y \in \{1, 3, 5, 15\}$, $\chi(y) = unsg(eql\,(y, 1) eql\,(y, 3)\ eql\,(y, 5)\ eql\,(y, 15))$.

П.3.5:

а) $f(x) = (x + 8)/2$;
б) $f(x) = (x\text{-}10)\cdot(14\text{-}x)$;
в) $f(x) = (x\text{-}7)/2$;
г) $f(x) = (x\text{+}10)/x$;
д) $f(x) = (x\text{-}3)\cdot(7\text{-}x)$;
е) $f(x) = (x\text{-}12)/2$;
ж) $f(x) = (15\text{+}x)/2$;
з) $f(x) = (2x\text{+}8)/x$.

П.3.6.:

Функция	ВАФ	ВЧАФ	АФ	ЧАФ	ЧАФ\ВЧАФ	АФ\ВАФ
а	+	+	+	+	-	-
б	-	-	+	+	+	+
в	+	+	+	+	-	-
г	+	+	+	+	-	-
д	+	+	+	+	-	-
е	-	-	-	+	+	-
ж	+	+	+	+	-	-

П.3.7.:

Функция	ВАФ	ВЧАФ	АФ	ЧАФ	ЧАФ\ВЧАФ	АФ\ВАФ
а	-	+	-	+	-	+
б	+	+	+	+	+	-
в	+	+	+	+	+	-
г	+	+	+	+	+	-
д	-	+	-	+	-	+
е	+	+	+	+	+	-
ж	-	+	-	+	-	+
з	+	+	+	+	+	-
и	-	+	-	+	-	+
к	+	+	+	+	+	-
л	+	+	+	+	+	-
м	-	+	-	+	-	+

Глава 4

П.4.1.:

Множество	Счетность	Эффективная перечислимость	Мощность
ПРФ	+	+	χ_0
ОРФ	+	-	χ_0
ЧРФ	+	+	χ_0
нерекурс. АФ	-	-	χ

П.4.2.:

Функция	ПРФ	ОРФ	ЧРФ	НРФ
а	+	+	+	-
б	-	-	+	-
в	-	+	+	-
г	-	-	+	-
д	+	+	+	-
е	+	+	+	-
ж	-	-	-	+
з	+	+	+	-
и	-	+	+	-
к	-	-	+	-

П.4.3. $G(x) = 100 \dot{-} 4x$; $E(x) = 50 \dot{-} 2x$.

x	***f*(*x*)**
0	25
60	5
35	Не определено (так как 70 не делится на 6)

П.4.4. $G(x) = 4x + 2$; $E(x) = 8 \cdot x \dot{-} 40$.

x	***f*(*x*)**
12	4
40	7
199	Не определено (так как 199+6=205 нечетное)

Глава 5

П.5.1. Как было определено ранее в задаче для $f(n) = 2n + 3$, копирование числа x один раз производится за время $3x \cdot x + 6x + 1$. Соответственно копирование x раз производится за $(3x \cdot x + 6x + 1)x$, т.е. $T(x) = (3x \cdot x + 6x + 1)x$.

Пространственная сложность $S(x) = 1 + x + 1 + xx + 1) = (x + 1)(x + 1) + 1$.

П.5.2. Как было определено в задаче 5.1, возведение числа x в квадрат производится за время $(3x \cdot x + 6x + 1)x$. Соответственно возведение x в степень y раз производится за $T(x,y) = ((3x \cdot x + 6x + 1)x)2^{(y-2)}$.

Пространственная сложность $S(x,y) = 1 + x + 1 + xx + 1)\ 2^{(y-2)} = ((x + 1)(x + 1) + 1)\ 2^{(y-2)}$.

П.5.3. Как было определено ранее в задаче 5.1, возведение числа 2 в квадрат производится за время $T(x) = (3 \cdot 2 \cdot 2 + 6 \cdot 2 + 1)2 = 50$. Соответственно возведение 2 в степень x раз производится за $T(x) = 50 \cdot 2^{(x-2)}$.

Пространственная сложность $S(x) = 9 \cdot 2^{(x-2)}$.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. *Тихомирова А.Н.* Теория алгоритмов: Учебное пособие. М.: МИФИ, 2008. – 176 с.
2. *Мальцев А.И.* Алгоритмы и рекурсивные функции. М.: Наука, 1965, 1986.
3. *Минский М.* Вычисления и автоматы. М.: Мир, 1971.
4. *Ахо А.* Построение и анализ вычислительных алгоритмов / Пер. с англ. М.: Мир, 1979.
5. *Ахо А. Хопкрофт Д.* Структуры данных и алгоритмы / Пер. с англ. М.: Издательский дом «Вильямс», 2000.
6. *Яблонский С.В.* Введение в дискретную математику. М.: Наука, 1979; 1996.
7. *Гашков С.Б.* Арифметика. Алгоритмы. Сложность вычислений: Учебное пособие для студентов вузов с углубленным изучением математики. М.: Дрофа, 2005. – 320 с.
8. *Верещагин Н.К., Шень А.* Лекции по математической логике и теории алгоритмов. Ч. 3. Вычислимые функции. – 3-е изд., стереотип. М.: МЦНМО, 2008. – 192 с.
9. *Пападимитриу Х., Стайглиц К.* Комбинаторная оптимизация. Алгоритмы и сложность / Пер. с англ. М.: Мир, 1985.
10. *Крупский В.Н., Плиско В.Е.* Теория алгоритмов: учебное пособие для студентов вузов. М.: Издательский центр «Академия», 2009. – 208 с.
11. *Максимова Л.Л.* Задачи по теории множеств, математической логике и теории алгоритмов. М.: Наука, 1975, 1984.
12. *Гэри М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982.
13. *Гашков С.Б., Чубариков В.Н.* Аримфетика. Алгоритмы. Сложность вычислений. М.: Высшая школа, 2000.
14. *Гуц А.К.* Кардинальные и трансфинитные числа: Учебное пособие. Омск: Омский государственный университет, 1995.

15. *Хаусдорф Ф.* Теория множеств / Пер. с нем. М.: КомКнига, 2006.
16. *Бурова И.Н.* Парадоксы теории множеств и диалектика. М.: Наука, 1976.
17. *Виленкин Н.Я.* Рассказы о множествах. М.: Наука, 1969.
18. *Коэн П. Дж.* Теория множеств и континуум-гипотеза / Пер. с англ. М.: Мир, 1969.
19. *Успенский В.А., Семенов А.Л.*. Теория алгоритмов: основные открытия и приложения. М.: Наука, 1987.
20. *Бурбаки Н.* Теория множеств / Пер. с франц. М.: Мир, 1965.
21. *Горбатов В.А.* Фундаментальные основы дискретной математики. М.: Наука, 2000.
22. *Горбатов В.А., Горбатов А.В., Горбатова М.В.* Дискретная математика: учебник для студентов втузов. М.: АСТ: Астрель, 2006.
23. *Кантор Г.* Труды по теории множеств. М.: Наука, 1985.
24. *Эшби У.Р.* Введение в кибернетику / Пер. с англ. М.: КомКнига, 2005.
25. *Фалевич Б.Я.* Теория алгоритмов: Учебное пособие. М.: Машиностроение, 2004.
26. *Пенроуз Р.* Новый ум короля: о компьютерах, мышлении и законах физики / Пер. с англ. М.: Едиториал УРСС, 2005.
27. *Пойя Д.* Математика и правдоподобные рассуждения / Пер. с англ. М.: Издательство иностранной литературы, 1957.
28. *Френкель А.А., Бар-Хиллел И.* Основания теории множеств / Пер. с англ. М.: КомКнига, 2006.
29. *Гельфонд А.О.* Трансцендентные и алгебраические числа. М.: КомКнига, 2006.
30. *Марков А.А., Нагорный Н.М.* Теория алгорифмов. М.: ФАЗИС, 1996.

Анна Николаевна Тихомирова
Надежда Викторовна Сафоненко

Практикум по теории алгоритмов

Учебное пособие

Редактор Е.Г. Станкевич

Подписано в печать 15.12.2010. Формат 60x84 1/16
Печ.л. 8,25. Уч.-изд.л. 8,25. Тираж 200 экз.
Изд. № 1/3/11. Заказ № 3

Национальный исследовательский ядерный университет «МИФИ».
115409, Москва, Каширское ш., 31

ООО «Полиграфический комплекс «Курчатовский»
144000, Московская область, г. Электросталь, ул. Красная, 42.