Московский государственный университет
им. М. В. Ломоносова
Факультет вычислительной математики и кибернетики

А. А. Белеванцев, С. С. Гайсарян, Л. С. Корухова, Е. А. Кузьменкова

Элементы теории алгоритмов

учебно-методическое пособие для студентов 1 курса

Москва
2019

УДК 004.43(075.8)
ББК 32.973-018.1я73

Печатается по решению Редакционно-издательского совета факультета вычислительной математики и кибернетики Московского государственного университета имени М. В. Ломоносова

Рецензенты:
С. Ю. Соловьёв, профессор;
А. Н. Терёхин, доцент

Белеванцев А. А., Гайсарян С. С., Корухова Л. С., Кузьменкова Е. А. // Andrey Belevantsev, Sergey Gaissaryan, Ludmila Korukhova, and Evgeniya Kuzmenkova.

Элементы теории алгоритмов. // Elements of Algorithm Theory.

Учебно-методическое пособие для студентов 1 курса. – М. Издательский отдел факультета ВМиК МГУ (лицензия ИД №05899 от 24.09.2001 г.);

МАКС Пресс, 2019. – 35 с.

ISBN 978-5-89407-600-3

В учебном пособии представлены материалы в поддержку первой части курса лекций «Алгоритмы и алгоритмические языки» для студентов 1 курса факультета ВМК МГУ.

В пособии приводятся необходимые сведения по теории алгоритмов и предлагаются упражнения по соответствующим разделам курса.

Пособие предназначено для студентов, изучающих основной курс программирования, а также для преподавателей и аспирантов.

Ключевые слова: алгоритм, алфавит, машины Тьюринга, алгоритмы Маркова.

This textbook supports the first part of the programming introductory course taught at Faculty of Computational Mathematics and Cybernetics of Lomonosov Moscow State University.

The book contains materials about theory of algorithms and corresponding exercises. It may be useful for students taking the above course as well as for assistant teachers and postgraduates.

Keywords: algorithm, alphabet, Turing machines, Markov algorithms.

УДК 004.43(075.8)
ББК 32.973-018.1я73

ISBN 978-5-89407-600-3

Введение

Это пособие создано в поддержку курса «Алгоритмы и алгоритмические языки», читаемого студентам первого года обучения факультета ВМК МГУ имени М. В. Ломоносова в качестве вводного курса по программированию. Оно посвящено первой части курса, которая предлагает студентам введение в теорию алгоритмов, закладывающее основу для дальнейшего изложения.

Пособие вводит неформальное определение алгоритма, после чего описывает популярные формализации — машину Тьюринга и алгоритмы Маркова. Строится универсальная машина Тьюринга, доказываются теоремы об алгоритмической неразрешимости проблем останова и самоприменимости. Для упрощения построения сложных машин Тьюринга предлагается использовать диаграммы. Также пособие содержит ряд упражнений, которые могут быть полезны при проведении семинаров в помощь лекционному курсу.

Авторы выражают искреннюю признательность К. А. Батузову, сделавшему множество ценных замечаний по тексту, много способствовавших его улучшению, И. А. Дудиной, прочитавшей и улучшившей черновик пособия, и А. В. Монакову, оказавшему помощь с вёрсткой и вычиткой текста.

1 Неформальное (интуитивное) определение алгоритма

Интуитивное определение алгоритма. Основные свойства алгоритмов. Пример: алгоритм Евклида нахождения НОД(a, b). Упражнения.

Само понятие алгоритма в математике долгое время определялось лишь неформально, на интуитивном уровне, однако этого оказывалось вполне достаточно для построения алгоритмов решения различных математических задач.

1.1 Интуитивное определение алгоритма

Под *алгоритмом* в математике понимают точное предписание, задающее вычислительный процесс, ведущий от начальных данных, которые могут варьироваться, к искомому результату. Синоним алгоритма — вычислительная (эффективная) процедура, которая после какого-либо числа шагов (вычислений) приводит к решению поставленной задачи. При этом в интуитивном определении алгоритма слова «вычисления», «вычислительный процесс» понимаются в широком смысле как любой процесс обработки информации: вычисление некоторой величины, поиск решения некоторой (математической) задачи, четкое и ясное предписание по обработке информации.

Заметим, что в рамках данного курса понятие информации считается интуитивно ясным и, следовательно, не дается определения этого понятия. При этом необходимо уточнить, что, имея в виду обработку информации на компьютере, рассматривается только такая информация, которую можно задать в дискретном виде, т.е. конечным числом отдельных знаков (символов). В рамках такого ограничения, предполагается, что в дискретном виде можно задать любую информацию. Более точно: любую информацию в задачах обработки информации можно «подменить» дискретной информацией. Например, цветную картину можно изобразить в виде описания цветов и фактуры пусть большого, но конечного числа отдельных точек (растровое изображение). Следует отметить, что предположение о возможности дискретного представления информации подкреплено тысячелетиями проверенной практикой использования человечеством письменности.

1.2 Основные свойства интуитивно определенного алгоритма

Каждый алгоритм должен обладать следующими свойствами:

1. **Конечность** (результативность). Алгоритм должен заканчиваться за конечное (хотя и не ограниченное сверху) число шагов и

получать результат, обеспечивая решение тех задач, для которых он и создавался.

2. **Определенность** (детерминированность). Каждый шаг алгоритма и переход от шага к шагу должны быть точно определены и каждое применение алгоритма к одним и тем же исходным данным должно приводить к одинаковому результату.

3. **Простота и понятность**. Каждый шаг алгоритма должен быть четко и ясно определен, чтобы выполнение алгоритма можно было «поручить» любому исполнителю (человеку или механическому устройству).

4. **Массовость**. Алгоритм задает процесс вычисления для множества исходных данных (чисел, строк букв и т.п.), он представляет общий метод решения класса задач. Не имеет смысла строить, например, алгоритм нахождения НОД только для чисел 10 и 15.

В приведенном выше определении алгоритма и в формулировке его свойств используется целый ряд понятий, определяемых неформально, лишь на интуитивном уровне. Это и составляет основной недостаток интуитивного определения — в нём не определены базовые понятия, такие как: *конечность* (допустимы ли ситуации бесконечного зацикливания), *исходные данные* (какие исходные данные допустимы, как они представляются), *результат*, *исполнитель* (что значит «можно поручить любому исполнителю»), *класс задач* и др.

1.3 Пример интуитивно определенного алгоритма

В качестве примера интуитивно определенного алгоритма приведем алгоритм Евклида нахождения наибольшего общего делителя двух целых положительных чисел a и b НОД(a, b) (в геометрической форме это алгоритм нахождения общей меры двух отрезков). Легко видеть, что для алгоритма Евклида все введённые нами выше свойства алгоритмов верны.

Даны два целых числа a и b, найти НОД(a, b). Алгоритм Евклида состоит в выполнении следующих шагов:

1. Если $a < b$, то поменять их местами.

2. Разделить нацело a на b; получить остаток r.

3. Если $r = 0$, то НОД$(a, b) = b$.

4. Если $r \neq 0$, заменить: a на b, b на r и вернуться к шагу 2.

1.4 Упражнения

1. Объясните, почему для алгоритма Евклида выполняется свойство конечности.

2. Приведите словесное описание алгоритмов для решения следующих задач:

   (a) Дана последовательность из n ($n > 0$) чисел. Найти максимальное (минимальное) число в последовательности.

   (b) Дано целое неотрицательное число n. Найти сумму цифр в десятичной записи этого числа.

   (c) Даны число t и коэффициенты $a_0, a_1, \ldots, a_n$ полинома $a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x + a_n$. Вычислить значение полинома в точке t по схеме Горнера.

   (d) Найти приближенное решение уравнения $F(x) = 0$ на отрезке $[a, b]$ с точностью $\varepsilon > 0$ методом деления отрезка пополам. Считать, что на заданном отрезке выполнены достаточные условия применимости метода.

2 Формализация понятия алгоритма

Почему необходимо формальное определение алгоритма. Алфавиты и отображения. Кодирование. Обработка информации. Алгоритмы и частично вычислимые функции.

2.1 Необходимость в формализации определения алгоритма

До тех пор, пока математики верили в возможность построения алгоритмов для всех поставленных задач, не было повода уточнять интуитивное понятие алгоритма. Однако после того, как в 30-е годы прошлого века была доказана неразрешимость некоторых задач в разных разделах математики, возникла совершенно новая ситуация. Раз появились задачи, решение для которых не может быть найдено, то существуют *алгоритмически неразрешимые* задачи: для них алгоритм не просто не найден, а его не будет найдено никогда. В отличие от конкретных алгоритмов, доказательство алгоритмической неразрешимости, т.е. доказательство невозможности, в котором содержалось бы высказывание *обо всех мыслимых алгоритмах*, потребовало формального уточнения понятия «алгоритм». Не имея такого формального определения, невозможно говорить обо всех мыслимых алгоритмах. Следовательно, невозможно доказать, что задача алгоритмически неразрешима и алгоритм её решения никогда не удастся построить.

Если вспомнить наше интуитивное определение алгоритма, становится понятно, что для формального определения необходим способ задания начальных данных алгоритма, описания результата, а также задания самого вычислительного процесса обработки этих данных. Для этого требуется предварительно рассмотреть некоторые общие понятия, часто используемые в математике.

2.2 Алфавиты и отображения

Алфавит — это конечное множество A_p элементов a_i: $A_p = \{a_1, a_2, \ldots, a_p\}$. Элементы алфавита A_p называются *символами*. Последовательность из m символов алфавита A_p называется *словом* длины m над алфавитом A_p. Слово длины 0 называется *пустым* словом и обозначается символом ε. В дальнейшем, если это специально не оговаривается, мы будем рассматривать только непустые алфавиты.

Слово длины 2 над алфавитом A_p можно рассматривать как символ декартова произведения алфавита A_p на себя: если $a_i \in A_p$ и $a_j \in A_p$, то $a_i a_j \in A_p \times A_p = A_p^2$, т.е. A_p^2 — множество всех двухсимвольных слов над алфавитом A_p. Легко видеть, что $A_p^3 = A_p^2 \times A_p$ — множество всех трёхсимвольных слов над алфавитом A_p, а $A_p^m = A_p^{m-1} \times A_p$ — множество всех m-символьных слов над алфавитом A_p. Следовательно, если положить по определению $\{\varepsilon\} = A_p^0$, то множество A_p^* всех слов над алфавитом A_p можно представить в виде объединения множеств:

$$A_p^* = \{\varepsilon\} \cup A_p \cup A_p^2 \cup \ldots \cup A_p^m \cup \ldots = \bigcup_{m=0}^{\infty} A_p^m.$$

В объединении $\bigcup_{m=0}^{\infty} A_p^m$ бесконечно много членов, но бесконечность здесь не *актуальная* (как в математическом анализе), а *потенциальная* в том смысле, что допустимы слова произвольно большой длины (длину слова $w \in A_p^*$ будем обозначать $|w|$, в частности, для пустого слова $|\varepsilon| = 0$).

Если в алфавит A включить не только буквы (например, латинские и русские, строчные и прописные), но и цифры, знаки препинания, а также символы «пробел», «конец строки», «конец страницы», «знак абзаца», «знак табуляции» и т.п., то любой текст (например, страницу любой книги или даже всю книгу) можно представить в виде слова над этим алфавитом[1].

Рассмотрим понятие кодирования. *Кодирование* — это представление символов алфавита A словами над алфавитом B. Справедливо следующее утверждение:

[1]Здесь не указано количество символов алфавита A, это означает, что рассматривается алфавит с произвольным количеством символов либо алфавит, точное количество символов которого не представляет интереса.

Утверждение. Для любой пары алфавитов A и B можно выполнить кодирование алфавита A с помощью алфавита B и обратно, возможно, с применением дополнительно служебного символа $\imath$ («конец кода символа»).

Доказательство. В самом деле, пусть для определенности $|A| > |B|$. Будем представлять символы алфавита A словами над алфавитом B (первые $|B|$ символов — однобуквенными словами, следующие $|B|$ символов — двухбуквенными и т.д.). При этом кодирующие слова необходимо разделять символом $\imath$. Для кодирования символов алфавита B можно выбрать какие-либо $|B|$ символов алфавита A. □

Из данного утверждения следует, что любой алфавит, в том числе двухсимвольный алфавит A_2 (символы `0` и `1`) и даже односимвольный алфавит A_1 (обычно в качестве единственного символа используется `|`, «палочка»), пригоден для представления любого текста. Служебные символы, которые используются как разделители, как правило, не считаются частью исходного алфавита.

Отметим, что *кодирование позволяет ограничиться одним алфавитом.* В теории алгоритмов в качестве такого алфавита обычно рассматриваются A_1 или A_2.

2.3 Обработка информации

В предыдущем разделе было показано, что любой текст может быть представлен в виде слова над некоторым конечным алфавитом A. Следовательно, и любая информация, заданная некоторым текстом, также может быть представлена словом над алфавитом A. Задачей обработки информации является построение *частичного отображения* (функции) $F : A^* \to A^*$ (частичное отображение определено только для *части* элементов множества A^*). Любой алгоритм можно рассматривать как набор правил, позволяющих реализовать требуемое частичное отображение на множестве слов конечного алфавита A.

Введём понятие *нумерации.* Каждому элементу алфавита можно присвоить его номер — неотрицательное целое число (номер 0 обычно присваивают $\varepsilon \in A^*$). Справедливо следующее утверждение:

Утверждение. Существует взаимно-однозначное отображение $\# : A^* \leftrightarrow \mathbb{N}_0$, где $\mathbb{N}_0$ — множество целых неотрицательных чисел, которое любому слову $w \in A^*$ ставит в соответствие его номер $n \in \mathbb{N}_0$. Это отображение $\#$ и называется нумерацией.

Доказательство. Для доказательства утверждения построим данное отображение $\#$. Пустому слову ε присваивается номер 0 ($\#(\varepsilon) = 0$). Каждому односимвольному слову $w = a_i \in A_p^*$ присваивается номер i ($i = 1, 2, \ldots, p$). Произвольному слову $w = a_{i_0}a_{i_1}\ldots a_{i_m}$ длины $m + 1$

присваивается номер

$$\#(w) = i_0 + p \cdot i_1 + p^2 \cdot i_2 + \ldots + p^m \cdot i_m = \sum_{s=0}^{m} p^s \cdot i_s,$$

где i_s — номер односимвольного слова $a_{i_s} \in A_p^*$. Построенное отображение по сути записывает число (номер слова) в позиционной p-ичной системе счисления, и известно, что такое представление числа единственно. Т.е. отображение присваивает разные номера для разных слов, и для каждого номера найдётся соответствующее ему слово. Таким образом, наше отображение взаимно-однозначно. □

Построенное взаимно-однозначное соответствие дает возможность по любому слову $w \in A^*$ определить его номер $\#(w)$ и, наоборот, по номеру $n \in \mathbb{N}_0$ определить соответствующее ему слово $\#^{-1}(n)$. Нумерация позволяет каждой частичной функции $F : A^* \to A^*$ единственным образом поставить в соответствие частичную числовую функцию[2] $f : \mathbb{N}_0 \to \mathbb{N}_0$. Действительно, если $v = F(w)$, то

$$\#(v) = \#(F(w)) = \#(F(\#^{-1}(\#(w)))) = (\#^{-1} \circ F \circ \#)(\#(w)) = f(\#(w)),$$

следовательно, $\#(v) = f(\#(w))$, где $f = \#^{-1} \circ F \circ \#$. Аналогичным образом можно установить, что $F = \# \circ f \circ \#^{-1}$.

Связь частичных функций F и f можно представить в виде коммутативной диаграммы:

$$\begin{array}{ccc} A^* & \xrightarrow{\quad F \quad} & A^* \\ {\scriptstyle \#}\downarrow & & \uparrow{\scriptstyle \#^{-1}} \\ \mathbb{N}_0 & \xrightarrow{\quad f \quad} & \mathbb{N}_0 \end{array}$$

Таким образом, показано, что:

1. каждый алгоритм $F : A^* \to A^*$ определяет частично вычислимую функцию $f : \mathbb{N}_0 \to \mathbb{N}_0$;
2. каждая частично вычислимая функция $f : \mathbb{N}_0 \to \mathbb{N}_0$ определяет алгоритм $F : A^* \to A^*$.

3 Машина Тьюринга

Вычислимость по Тьюрингу. Понятие машины Тьюринга (МТ). Пример МТ. Нормальные МТ. Упражнения.

[2] То есть определённую не для любых $n \in \mathbb{N}_0$.

Одним из формальных определений понятия алгоритма является предложенная английским математиком Аланом Тьюрингом в 1936 году машина Тьюринга.

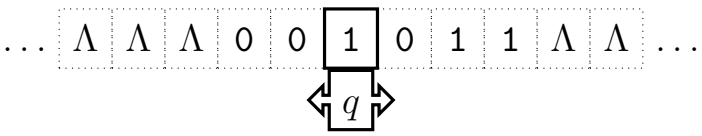
3.1 Вычислимость по Тьюрингу

Основная идея предложенного Тьюрингом подхода к уточнению понятия алгоритма заключается в том, что алгоритмами должны называться те и только те отображения $F : A^* \to A^*$, *которые могли бы осуществлять достаточно простые машины-автоматы.* Выполнение отображения понимается при этом следующим образом: машине-автомату предъявляется любое исходное слово $w \in A^*$, а она в результате обработки этого слова «выдаёт» слово $v = F(w)$. Таким образом, каждая машина Тьюринга (МТ) T_F строит отображение $F : A^* \to A^*$ для соответствующей функции F. При этом каждая частичная функция F, для которой можно построить МТ, называется *вычислимой по Тьюрингу.* Рассмотрим подробнее, что нужно задать для функционирования машины.

3.2 Понятие машины Тьюринга

Для задания МТ необходимо определить: *алфавит состояний* $Q = \{q_0, q_1, q_2, \ldots, q_n\}$ и *рабочий алфавит* $S = A \cup A'$, где A — алфавит *входных* символов, A' — алфавит *вспомогательных* символов (*маркеров*), который, в частности, может быть пустым.

У МТ есть *память* — лента, размеченная на ячейки, каждая из которых может быть пустой (пустая ячейка представляется символом Λ) либо содержать *один* символ рабочего алфавита S. МТ связана с лентой посредством *управляющей головки* (УГ), которая в каждый данный момент располагается под некоторой ячейкой ленты (эта ячейка называется *рабочей ячейкой* (РЯ)). Среди состояний МТ выделяются *начальное состояние* q_0 и *состояние останова* q_s. В начале работы МТ на ленту записываются начальные данные — слова над входным алфавитом A. Лента МТ с записанными на ней данными (например, словом `001011`) и текущим положением УГ может быть изображена следующим образом (Λ представляет пустую ячейку ленты, остальные символы представляют самих себя, РЯ обозначена рамочкой вокруг обозреваемого символа, под рамочкой указано текущее состояние УГ):



На рисунке изображена *конфигурация* МТ, которая определяется текущей записью на ленте, текущим положением головки и текущим

состоянием МТ. Конфигурация, соответствующая начальному состоянию МТ, начальным данным на ленте и исходному положению головки, называется *начальной*; конфигурация, соответствующая состоянию останова МТ, — *конечной*.

Для того, чтобы задавать конфигурацию, удобно перенумеровать ячейки ленты МТ: выделить ячейку с номером `0` (нулевую ячейку), все ячейки справа от нулевой занумеровать положительными целыми числами, а все ячейки слева от нулевой — отрицательными целыми числами. Тогда для задания текущей записи на ленте можно определить функцию $F : \mathbb{Z} \to S$, которая каждой ячейке с целочисленным номером j ставит в соответствие символ рабочего алфавита s_j, записанный в этой ячейке, а конфигурация будет задаваться тройкой $\langle n, F, q \rangle$, где n — номер текущей рабочей ячейки, F — текущая запись на ленте, q — текущее состояние. Пару $\langle n, q \rangle$ будем называть *позицией* МТ.

На каждом такте работы МТ УГ считывает символ из РЯ, и в зависимости от этого символа и состояния УГ может записать в РЯ новый символ и передвинуться на следующую слева или справа ячейку ленты, либо остаться на месте; кроме того, МТ принимает новое состояние (которое также может совпасть с предыдущим). В результате МТ переходит из текущей конфигурации в следующую, и именно такой переход называется *тактом* МТ. Действия МТ при выполнении каждого такта работы определяются *инструкциями* вида:

$$\langle \textit{состояние}, \textit{символ} \rangle \to \langle \textit{состояние}, \textit{символ}, \textit{направление} \rangle.$$

Работа МТ может быть описана как последовательность конфигураций от начальной конфигурации до конечной, т.е. как последовательность тактов ее работы.

3.3 Пример МТ

В качестве примера построим МТ, осуществляющую проверку правильности скобочных выражений, т.е. последовательностей открывающих и закрывающих скобок.

Правильным называется скобочное выражение[3], удовлетворяющее следующим двум условиям: 1) число открывающих скобок равно числу закрывающих, 2) все скобки можно разбить на непересекающиеся пары так, что каждая открывающая скобка предшествует парной закрывающей скобке. Например, `(())()` — правильное скобочное выражение, а `)(` или `(()` — неправильные скобочные выражения.

Если скобочное выражение правильное, МТ должна записать на ленту результат `1` и остановиться, если нет, — записать `0` и остановиться. Таким образом, после работы МТ на ленте должен быть записан

[3]Для простоты будем рассматривать выражения, содержащие единственный тип скобок.

только результат: `0` или `1`, и УГ должна быть установлена на ячейке, в которой записан результат.

Рабочий алфавит: $S = \{(,), 0, 1\} \cup \{\Lambda, X\}$ (Λ — обозначение пустой ячейки, X — маркер). Алфавит состояний $Q = \{q_0, q_1, q_2, q_3, q_s\}$:

- q_0 — начальное состояние МТ: поиск ближайшей справа закрывающей скобки, замена ее на маркер X и переход в состояние q_1. Если до конца слова закрывающей скобки не нашлось — переход в состояние q_2 и движение влево;
- q_s — состояние останова;
- q_1 — поиск парной открывающей скобки при движении влево, замена её на X и переход в q_0;
- q_2 — достигнут конец выражения и при этом не найдено закрывающей скобки: движение влево, стирая все маркеры X; при достижении левого конца выражения (Λ) — запись результата `1`; при обнаружении открывающей скобки — переход в состояние q_3, так как выражение неправильное;
- q_3 — выражение неправильное: движение влево, стираем символы `(` и X, запись результата `0` в первую пустую ячейку и переход в состояние q_s.

Считаем, что в начальном состоянии УГ обозревает самый левый символ входного слова.

Описание данной МТ может быть выполнено с помощью «пятёрок» вида:

$$\langle \textit{состояние}, \textit{символ} \rangle \to \langle \textit{состояние}, \textit{символ}, \textit{направление} \rangle,$$

где направление: L — влево, R — вправо, H — на месте. Каждая такая «пятёрка» задает очередную инструкцию МТ. В этом случае описание МТ (её *программа*) имеет вид:

$q_0, (\to q_0, (, R$	$q_0,) \to q_1, X, L$	$q_0, X \to q_0, X, R$	$q_0, \Lambda \to q_2, \Lambda, L$
$q_1, (\to q_0, X, R$	$q_1,) \to q_1,), L$	$q_1, X \to q_1, X, L$	$q_1, \Lambda \to q_3, \Lambda, R$
$q_2, (\to q_3, \Lambda, H$	$q_2,)$ невозможно	$q_2, X \to q_2, \Lambda, L$	$q_2, \Lambda \to q_s, 1, H$
$q_3, (\to q_3, \Lambda, L$	$q_3,)$ невозможно	$q_3, X \to q_3, \Lambda, L$	$q_3, \Lambda \to q_s, 0, H$

Для описания МТ можно использовать и другой способ записи. Те же самые «пятёрки» можно наглядно представить с помощью *таблицы переходов*, в которой строки соответствуют состоянию МТ, столбцы

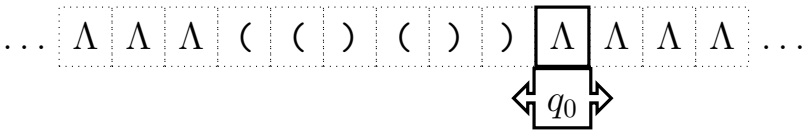
— обозреваемому символу, а в ячейке таблицы записываются выполняемые ею действия в данном состоянии с данным обозреваемым символом. Тогда для рассматриваемой МТ получим следующую таблицу:

$q_i \downarrow \setminus s_j \rightarrow$	(	)	X	Λ
q_0	$q_0, (, R$	q_1, X, L	q_0, X, R	q_2, Λ, L
q_1	q_0, X, R	$q_1,), L$	q_1, X, L	q_3, Λ, R
q_2	q_3, Λ, H	—	q_2, Λ, L	$q_s, 1, H$
q_3	q_3, Λ, L	—	q_3, Λ, L	$q_s, 0, H$

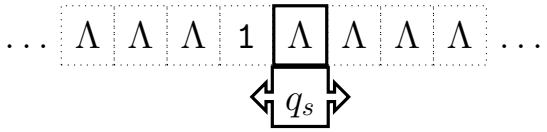
3.4 Нормальные МТ

При построении сложных МТ возникает естественное желание использовать для этого композицию более простых МТ, при этом под *композицией* понимают последовательное применение двух (или нескольких) МТ. Чтобы композиция не вызывала сложностей, введем понятие *нормальной* МТ. Можно показать, что любую МТ можно перестроить таким образом, что она, вычисляя ту же функцию, будет удовлетворять следующим двум условиям (соглашениям или ограничениям):

1. в начальном состоянии q_0 УГ установлена напротив пустой ячейки, которая следует за всеми символами исходного слова w:

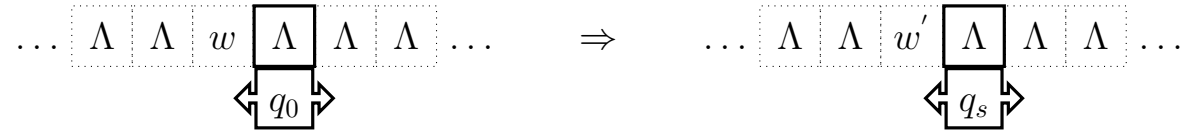


2. в состоянии останова q_s УГ установлена напротив пустой ячейки, которая следует за всеми символами слова-результата $F(w)$:



МТ, удовлетворяющая условиям (1) и (2), называется *нормальной* МТ.

Таким образом, любая нормальная МТ T_F выполняет следующее преобразование исходной конфигурации в конечную:



где w — исходное слово, $w^{'}$ — слово-результат, $w^{'} = F(w)$.

3.5 Упражнения

1. Рассмотрим пример МТ для проверки правильности скобочных выражений из раздела 3.3.

   а) Объясните, почему в состояниях q_2 и q_3 машине не может встретиться закрывающая скобка.

   б) Остаются ли на ленте символы исходного скобочного выражения после окончания работы МТ, если оно было правильным? А если неправильным?

   в) Измените программу МТ так, чтобы после окончания работы на ленте оставался только символ 0 или 1. Понадобятся ли дополнительные состояния?

2. Пусть $A = \{a, b\}$. Используя табличный способ описания МТ, опишите нормальные МТ для решения следующих задач:

   а) Если исходное слово имеет нечётную длину, оставить на ленте символ a, иначе — пустое слово.

   б) Перенести все символы a исходного слова на его левый конец, все символы b — на правый конец, т.е. привести исходное слово к виду $a \dots ab \dots b$.

   в) Оставить на ленте тот символ исходного слова, который встречается в слове чаще. Если символы входят в слово равное количество раз, оставить на ленте пустое слово.

   г) Удвоить каждый символ исходного слова (например: $aba \Rightarrow aabbaa$).

   д) Перевернуть исходное слово, т.е. для слова вида $w_1 w_2 \dots w_k$ выполнить преобразование $w_1 w_2 \dots w_k \Rightarrow w_k w_{k-1} \dots w_1$.

4 Диаграммы Тьюринга

Диаграммы Тьюринга (ДТ). Перестройка МТ к виду, более удобному для ДТ: МТ с лентой, ограниченной слева; МТ с укороченными инструкциями. Построение ДТ. Элементарные МТ. Правила построения ДТ. Примеры. Упражнения.

Диаграммы Тьюринга (ДТ) позволяют удобным и наглядным способом описывать построение МТ из более простых МТ, используя их композицию. Например, МТ, выполняющая копирование слова на ленте, может быть представлена через следующие МТ:

1. поиск начала слова;

2. поиск конца слова;

3. копирование символа и т.п.

Это существенно упрощает построение МТ, что особенно важно при конструировании сложных МТ.

4.1 Перестройка МТ к виду, более удобному для ДТ

Перед тем, как перейти непосредственно к описанию диаграмм Тьюринга, выполним некоторую перестройку МТ, чтобы привести её к виду, более удобному для построения ДТ.

МТ с лентой, ограниченной с левого конца

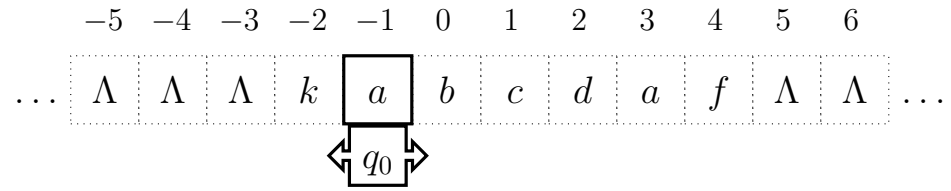
Рассмотрим произвольную МТ T с неограниченной лентой. Построим МТ T' с лентой, ограниченной с левого конца, которая работает так же, как и МТ T. Для этого:

а) перегнём ленту по ячейке с номером 0;

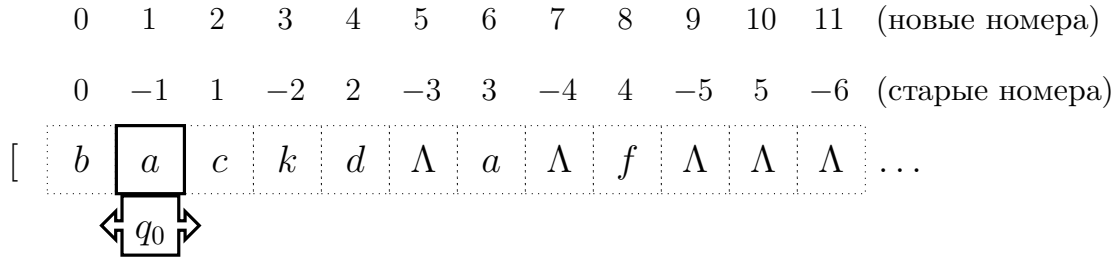
б) все ячейки с положительными номерами отобразим на ячейки с чётными номерами: то, что было записано в ячейке с номером $n > 0$, перепишем в ячейку с номером $2 \times n$;

в) в освободившиеся ячейки с нечётными номерами перенесём содержимое ячеек с отрицательными номерами: то, что было записано в ячейке с номером $n < 0$, перепишем в ячейку с номером $2 \times |n| - 1$.

В результате конфигурация МТ T:



перейдёт в конфигурацию МТ T':



В МТ T' для ячеек с чётным номером сдвигу вправо (влево) МТ T будет соответствовать сдвиг на две ячейки вправо (влево). Аналогично для ячеек с нечётным номером сдвигу вправо (влево) МТ T будет соответствовать сдвиг на две ячейки влево (вправо). Для ячейки с номером 0 сдвигу влево МТ T будет соответствовать сдвиг на одну ячейку вправо, а сдвигу вправо МТ T — сдвиг на две ячейки вправо. Для ячейки

с номером 1 сдвигу влево МТ T будет соответствовать сдвиг на две ячейки вправо, а сдвигу вправо МТ T — сдвиг на одну ячейку влево. Край ленты обозначим символом [.

Рассуждая более строго, зададимся вопросом о том, как именно должна быть преобразована программа исходной МТ для машины T' с укороченной лентой — для реализации описанных выше правил машина должна «знать» чётность номера текущей ячейки (или, что то же самое, знак номера ячейки исходной ленты). Для этого можно использовать приём, называемый *размножением состояний*: если в машину необходимо внести вариативность некоторых действий в зависимости от какого-либо условия, то состояния, ответственные за выполнение этих действий, размножаются в количестве, соответствующем числу вариантов условий, и каждое новое отдельное состояние независимо меняет свои действия нужным образом.

В нашем случае для каждого состояния q машины T заводятся два состояния q^+ и q^-, соответствующие положительному и отрицательному номеру ячейки исходной ленты. В этих состояниях выполняется запись того же символа, что и в исходном состоянии q, а сдвиг выполняется по-разному в соответствии с описанными правилами. Сложность возникает только в случае, когда возможен переход через нулевую ячейку. Так как проверить достижение края ленты в нашей формулировке МТ нельзя, отведём самую левую ячейку ленты для специального маркера нулевой ячейки. Если при выполнении сдвига УГ обозревает этот маркер, то необходимо, во-первых, изменить направление движения, и, во-вторых, сменить «знак» состояния (с положительного на отрицательный или наоборот).

МТ с укороченными инструкциями

Рассмотрим произвольную инструкцию МТ T: $q, a \to\ q', b, R$. Разобьём её на две инструкции:

$q, a \to\ q'', b, S$ (только записывает символ в РЯ);

$q'', b \to\ q', b, R$ (только сдвигает головку).

Можно доказать, что для любой МТ T можно построить МТ T', каждая инструкция которой либо только сдвигает головку, либо только записывает символ в РЯ. МТ T' и есть МТ с укороченными инструкциями.

Далее будем рассматривать класс МТ, который содержит только МТ с укороченными инструкциями и лентой, ограниченной слева. Кроме того, будем считать, что МТ, принадлежащие рассматриваемому классу, выполняют нормальные вычисления по Тьюрингу. Все эти предположения не являются ограничением общности, так как по произвольной МТ нетрудно построить МТ рассматриваемого класса.

Основным преимуществом рассматриваемого класса МТ является возможность ввести понятие действия. В зависимости от состояния МТ и обозреваемого в РЯ символа МТ может выполнить *действие*

$$v = \{L, R, H, s_i \in S\},$$

где L означает сдвиг УГ на одну ячейку влево, R — на одну ячейку вправо, H — на месте, s_i — запись в РЯ символа $s_i \in S$. Запись символа в РЯ или сдвиг УГ вправо или влево называются *элементарными действиями*. Машины, выполняющие только одно элементарное действие, будут являться базовыми «кирпичиками» для построения сложных МТ с помощью композиции простых.

4.2 Построение диаграмм Тьюринга

Над алфавитом $A_p = \{a_1, a_2, \ldots, a_p\} \cup \{\Lambda\}$ определим следующие элементарные МТ (каждая элементарная МТ определяет одно элементарное действие):

1. МТ $\boldsymbol{l}$ «сдвиг УГ на одну ячейку влево» (для края ленты **[** действие не определено),
2. МТ $\boldsymbol{r}$ «сдвиг УГ на одну ячейку вправо»,
3. МТ $\boldsymbol{a_i}$ «запись символа a_i в РЯ».

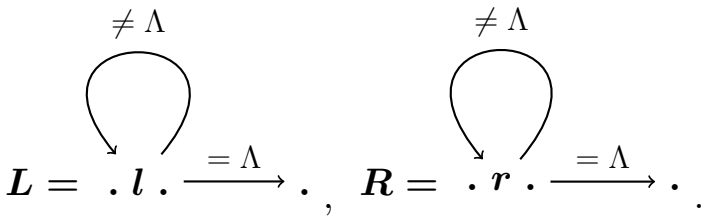
Для каждой из этих элементарных МТ q_0 — начальное состояние, q_1 — состояние останова. Программы и диаграммы Тьюринга для соответствующих МТ приведены ниже.

Элементарная МТ	Программа	Диаграмма
$\boldsymbol{l}$	$q_0\Lambda \to Lq_1$, $q_0a_1 \to Lq_1$, …, $q_0a_p \to Lq_1$	$\boldsymbol{\cdot l \cdot}$
$\boldsymbol{r}$	$q_0\Lambda \to Rq_1$, $q_0a_1 \to Rq_1$, …, $q_0a_p \to Rq_1$	$\boldsymbol{\cdot r \cdot}$
$\boldsymbol{a_i}$	$q_0\Lambda \to a_iq_1$, $q_0a_1 \to a_iq_1$, …, $q_0a_p \to a_iq_1$	$\boldsymbol{\cdot a_i \cdot}$

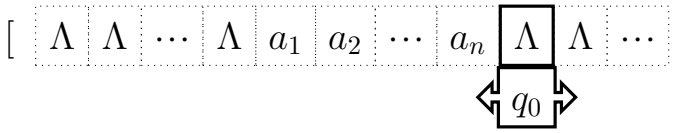
В приведенных выше диаграммах левая точка соответствует состоянию q_0, правая — состоянию останова q_1. Заметим, что иногда программы, описывающие данные элементарные МТ, дополняют инструкциями вида $q_1a_i \to Hq_s$, где $a_i \in A_p$.

Пользуясь элементарными МТ, можно построить более сложные МТ. Например, рассмотрим МТ, осуществляющую сдвиг УГ на одно слово влево/вправо над тем же алфавитом A_p. МТ «сдвиг УГ на одно слово влево» (обозначим ее $\boldsymbol{L}$) должна перемещать УГ влево до тех пор, пока не «увидит» на ленте символ Λ, т.е. эта МТ должна многократно применять элементарную МТ $\boldsymbol{l}$. Аналогично МТ $\boldsymbol{R}$ «сдвиг

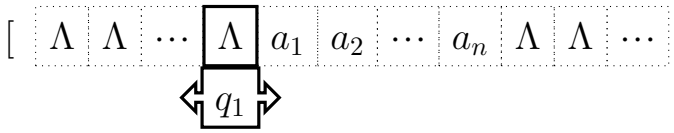
УГ на одно слово вправо» должна многократно применять элементарную МТ $\boldsymbol{r}$. Эти МТ можно представить в виде следующих диаграмм Тьюринга (ДТ):



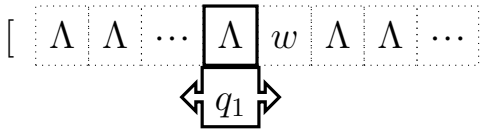
МТ $\boldsymbol{L}$ переводит конфигурацию



в конфигурацию



Договоримся в дальнейшем слово $a_1a_2a_3\ldots a_n$ на ленте обозначать w, тогда конечная конфигурация для МТ $\boldsymbol{L}$ принимает вид



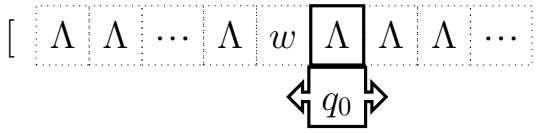
На примере приведённых выше диаграмм сформулируем *правила построения* ДТ. Состояниям МТ соответствуют точки на ДТ — перед символом $\boldsymbol{l}$ (состояние q_0) и после него (состояние q_1): **. $\boldsymbol{l}$.** Каждая стрелка в ДТ соединяет точку, стоящую после символа МТ и точку, стоящую перед символом следующей МТ. Стрелка, проведённая от точки к точке, означает переход между соответствующими состояниями. Символы над стрелками показывают условия перехода, определяемого соответствующей стрелкой. Поскольку в данных МТ переход в конечное состояние q_1 происходит при достижении УГ символа Λ, над стрелкой (обозначающей возврат в состояние q_0) должны быть надписаны все символы рабочего алфавита A_p, кроме Λ.

Таким образом, ДТ состоит из символов (имен) входящих в нее МТ, точек для изображения состояний и стрелок для изображения переходов между состояниями с надписанными над ними условиями переходов.

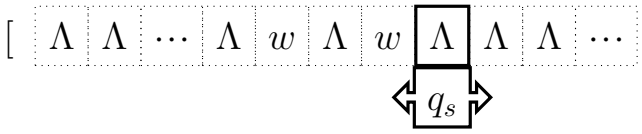
Пользуясь указанными правилами, легко построить диаграмму по таблице МТ с укороченными инструкциями: каждой ячейке таблицы, описывающей действие, будет соответствовать одна из элементарных МТ, а переходы между ними будут определять стрелки между точками на ДТ, которые строятся по паре <*состояние, символ*>. Покажем далее, как строить более простой вид таких «полных» диаграмм.

4.3 Пример: построение ДТ для МТ копирования слова

Рассмотрим построение МТ $\boldsymbol{K}$, которая выполняет копирование заданного на ленте входного слова w, размещая копию слова через символ Λ от исходного слова. Поскольку в нормальных МТ лента ограничена слева, копию будем строить справа от входного слова. Таким образом, искомая МТ $\boldsymbol{K}$ переводит конфигурацию



в конфигурацию



Пусть построение МТ осуществляется над алфавитом $\{0, 1\} \cup \{\Lambda\}$. ДТ может быть построена следующим образом:

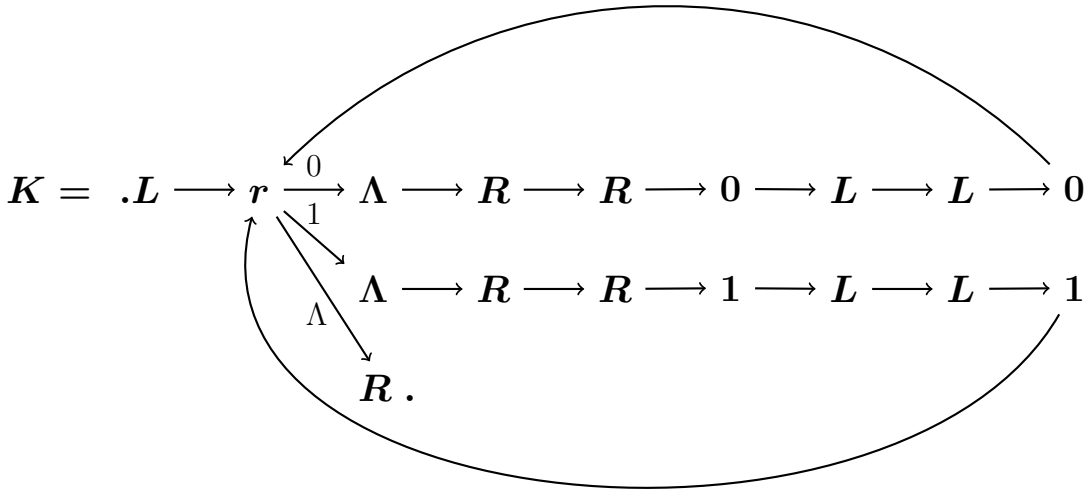
1. Установить УГ на ячейку, предшествующую началу копируемого слова w, с помощью МТ $\boldsymbol{L}$. Соответствующая часть ДТ имеет вид $\boldsymbol{.\ L\ .}$, где левая точка соответствует состоянию q_0, а правая — состоянию q_1.

2. Теперь необходимо поместить УГ напротив первого символа слова w, применив элементарную МТ $\boldsymbol{r}$, и затем задействовать одну из МТ копирования символа[4] (для используемого алфавита таких МТ две — по одной для каждого символа). Для этого на диаграмме точка, соответствующая конечному состоянию МТ $\boldsymbol{L}$, должна быть соединена с точкой, отвечающей начальному состоянию элементарной МТ $\boldsymbol{r}$. Фрагмент ДТ, устанавливающий УГ на первом символе входного слова, имеет вид $\boldsymbol{.\ L \to r\ .}$. Для упрощения диаграммы договоримся опускать точки, соответствующие промежуточным состояниям.

3. Выполнить копирование символа. Первым символом копируемого слова может быть либо `0`, либо `1`, (либо Λ, если слово пустое); пусть для определенности первым символом является `0`. В этом случае УГ переходит в состояние, с которого начинается копирование символа `0`. Следовательно, на диаграмме точка после символа $\boldsymbol{r}$ должна быть соединена с точкой, соответствующей начальному состоянию МТ, копирующей символ `0` в составе слова.

[4] Создание нескольких МТ копирования конкретного символа из одной МТ копирования символа является еще одним примером применения техники размножения состояний.

В итоге получим следующую ДТ (левая точка соответствует состоянию q_0, правая — состоянию q_s):



Обратите внимание, что на диаграмме обе МТ копирования символа в свою очередь представлены как композиция более простых МТ.

Приведённый пример наглядно демонстрирует основное преимущество ДТ: они позволяют описывать новые МТ, используя уже построенные, т.е. строить МТ в виде композиции более простых МТ. В частности, построенную выше МТ $\boldsymbol{K}$ в дальнейшем можно использовать для построения новых МТ.

Примем следующие соглашения по упрощению записи ДТ:

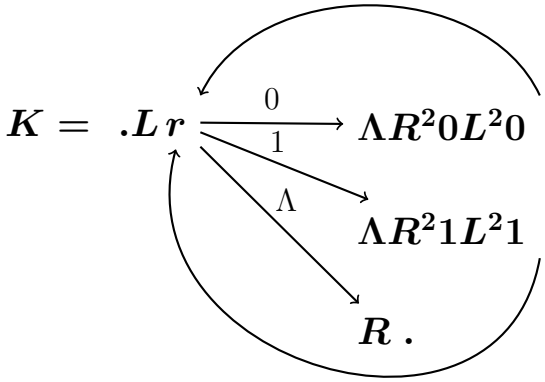
1. Если над стрелкой в ДТ не указано никаких символов, это означает, что над ней нужно надписать все символы рабочего алфавита. Такую стрелку можно опустить, т.е. договориться, что если в ДТ два символа МТ записаны непосредственно один за другим, это значит, что между ними была стрелка, которая была надписана всеми символами рабочего алфавита рассматриваемой МТ. Следовательно, в ДТ K строку

   $\boldsymbol{k_0 = \Lambda \to R \to R \to 0 \to L \to L \to 0}$,

   описывающую МТ $\boldsymbol{k_0}$ копирования символа `0`, можно более компактно записать в виде $\boldsymbol{k_0 = \Lambda RR0LL0}$.

2. Если на ДТ расположено подряд n символов одной и той же машины $\boldsymbol{M}$, то их можно заменить одним символом $\boldsymbol{M^n}$. Тогда получим еще более короткое определение МТ $\boldsymbol{k_0}$: $\boldsymbol{k_0 = \Lambda R^2 0 L^2 0}$.

С учётом указанных соглашений получим *упрощённую диаграмму* МТ $\boldsymbol{K}$:



4.4 Упражнения

1. Выпишите явно состояния МТ с укороченной лентой, отвечающие за сдвиг УГ этой МТ, аналогичный сдвигу УГ машины с неограниченной лентой (можно использовать описание из раздела 4.1 или предложить свой способ).

2. Как изменится машина $\boldsymbol{K}$ в случае алфавита из четырех символов? шести символов? n символов? Что можно сказать о зависимости количества состояний машины от числа символов алфавита?

3. Пусть $A = \{0, 1\}$. Постройте диаграммы Тьюринга (ДТ) для решения следующих задач:

   а) Пусть исходное слово представляет собой запись числа в двоичной системе счисления. Увеличить это число на 1.

   б) Если исходное слово начинается с 1, инвертировать это слово, т.е. заменить в нем все 0 на 1 и наоборот, в противном случае оставить слово без изменения.

   в) Удалить из исходного слова все вхождения символа 0.

   г) Приписать справа к слову такое количество символов 1, сколько символов 0 встречается в этом слове.

   д) Если исходное слово является палиндромом (равноудаленные от концов слова символы совпадают), то оставить на ленте 1, иначе 0.

   Каждая ДТ должна описывать нормальную МТ и может использовать элементарные машины $\boldsymbol{l}$, $\boldsymbol{r}$, $\boldsymbol{a_i}$, а также машины сдвигов $\boldsymbol{L}$ и $\boldsymbol{R}$.

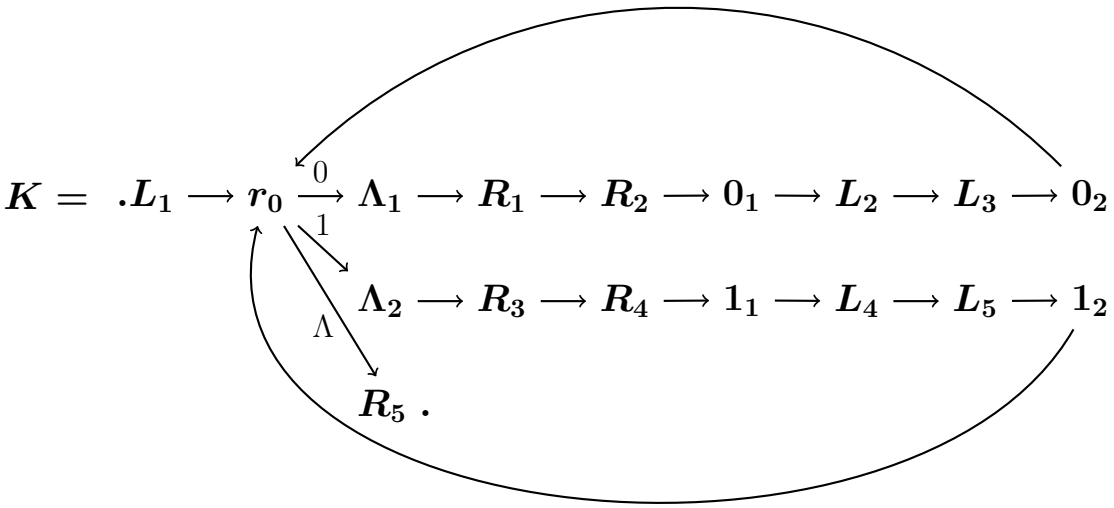
5 Эквивалентность МТ и ДТ

Построение таблиц МТ по диаграммам Тьюринга. Эквивалентность МТ, заданных таблицами, и МТ, заданных диаграммами.

На примере ДТ для МТ копирования слова $\boldsymbol{K}$ рассмотрим построение таблицы этой МТ по построенной в предыдущем разделе диаграмме Тьюринга. Для этого выполним следующие преобразования:

1. Заменим упрощённую диаграмму полной.

2. С помощью индексации добиваемся того, чтобы каждый символ МТ входил в диаграмму только один раз. В итоге получим следующую диаграмму МТ $\boldsymbol{K}$.



3. Сопоставим каждому символу МТ её таблицу (таблицу запишем в виде набора соответствующих инструкций). Например, МТ $\boldsymbol{r_0}$ сопоставим таблицу

   $q_{00}\Lambda \to Rq_{01}$, $q_{00}0 \to Rq_{01}$, $q_{00}1 \to Rq_{01}$;

   $q_{01}\Lambda \to Hq_{0s}$, $q_{01}0 \to Hq_{0s}$, $q_{01}1 \to Hq_{0s}$.

4. Перепишем все таблицы одну за другой (в любой последовательности). Заметим, что благодаря введенной на шаге 2 индексации множество состояний для каждой таблицы уникально.

   Диаграмма каждой МТ начинается и заканчивается точкой (начальное состояние и состояние останова). При композиции диаграмм конечная точка диаграммы сливается с начальной точкой следующей диаграммы и тем самым исключается. Следовательно, у каждой диаграммы остается только одна точка (начальная).

5. Добавим в таблицу следующие строки:

   а) для каждого символа a входного алфавита, которому соответствует стрелка, ведущая из точки снова к ней же, добавим строку $q_0a \to aq_0$;

   б) для каждого символа a входного алфавита, которому соответствует стрелка, ведущая из точки к символу МТ $\boldsymbol{M}$, добавим строку $q_0a \to aq_{M0}$;

в) для каждого символа a входного алфавита, которому не соответствует никакая стрелка, ведущая из точки, добавим строку $q_0 a \to H q_s$;

г) если два символа МТ $\boldsymbol{M}$ и $\boldsymbol{M}'$ соединены стрелкой, над которой надписан символ a, то для состояния останова q_{Ms} из части таблицы, соответствующей $\boldsymbol{M}$, добавляем строку $q_{Ms} a \to a q_{M'0}$ (аналогично для стрелки в состояние останова).

В результате преобразований 1–5 получится таблица МТ, которая выполняет те же действия, что и МТ, заданная диаграммой. Тем самым мы всегда можем построить таблицу МТ по диаграмме, а построение диаграммы по таблице МТ уже рассматривалось в разделе 4.2. Следовательно, МТ, задаваемые диаграммами, эквивалентны МТ, задаваемым таблицами.

6 Универсальная машина Тьюринга

Понятие моделирования МТ. Универсальная машина Тьюринга (УМТ). Этапы построения УМТ.

6.1 Моделирование МТ

Рассмотрим две МТ $\boldsymbol{M}$ и $\boldsymbol{M}'$ и введем понятие моделирования МТ.

Определение. Будем говорить, что МТ $\boldsymbol{M}$ моделирует МТ $\boldsymbol{M}'$, если выполнены следующие условия:

1. Данная начальная конфигурация вызывает машинный останов МТ $\boldsymbol{M}$ после конечного числа шагов тогда и только тогда, когда указанная начальная конфигурация вызывает машинный останов МТ $\boldsymbol{M}'$ после конечного числа шагов.

2. Данная начальная конфигурация вызывает переход за край ленты у МТ $\boldsymbol{M}$ после конечного числа шагов тогда и только тогда, когда указанная начальная конфигурация вызывает переход за край ленты у МТ $\boldsymbol{M}'$ после конечного числа шагов.

3. Для последовательности (c_n') текущих конфигураций МТ $\boldsymbol{M}'$ для данной начальной конфигурации можно указать *моделирующую* подпоследовательность (c_n) последовательности текущих конфигураций МТ $\boldsymbol{M}$ для той же начальной конфигурации: для каждой конфигурации c_i' машины $\boldsymbol{M}'$ её лента будет «частью» ленты конфигурации c_i машины $\boldsymbol{M}$, УГ машины $\boldsymbol{M}$ будет находиться на ячейке, соответствующей положению рабочей ячейки машины $\boldsymbol{M}'$, и по конфигурации c_i можно указать состояние машины $\boldsymbol{M}'$ в конфигурации c_i'.

Легко видеть, что если МТ $\boldsymbol{M}$ моделирует (в смысле вышеприведенного определения) МТ $\boldsymbol{M}'$, то обе эти МТ выполняют один и тот же алгоритм, но, возможно, машине $\boldsymbol{M}$ для этого требуется пройти больше конфигураций. Кроме того, если алфавиты машин различны, необходимо указать способ кодирования алфавита A' машины $\boldsymbol{M}'$ с помощью символов алфавита A машины $\boldsymbol{M}$. Мы уже знаем, что такой способ всегда существует.

Через понятие моделирования можно определить универсальную машину Тьюринга.

6.2 Универсальная машина Тьюринга

Определение. Универсальной машиной Тьюринга (УМТ) для алфавита A называется такая машина $\boldsymbol{U}$, на которой может быть промоделирована любая МТ над алфавитом A.

Доказательство существования УМТ — один из важнейших результатов Тьюринга. УМТ показывает, что можно создать универсальный вычислитель – одно устройство, на котором выполняются алгоритмы решения самых разных задач. Современные компьютеры являются такими вычислителями.

Существование УМТ доказывается *конструктивно*, т.е. явно описывается способ ее построения. Идея УМТ заключается в записи как таблицы моделируемой МТ (программы), так и входных данных моделируемой МТ на ленту УМТ — все это составит входные данные УМТ. Моделирование МТ состоит в последовательной интерпретации её тактов: УМТ хранит текущее состояние моделируемой МТ и позицию её управляющей головки, по этим данным находит на своей ленте в записи программы МТ соответствующую команду и выполняет необходимое действие. УМТ была описана Тьюрингом в 30-х годах XX века и в дальнейшем, как считается, послужила источником вдохновения при создании Джоном фон Нейманом компьютерной архитектуры, носящей сейчас его имя.

Итак, для построения УМТ нужно решить следующие задачи:

- Как представить программу моделируемой МТ на ленте УМТ?
- Как выглядит лента в исходном состоянии УМТ?
- Как происходит интерпретация моделируемой МТ?
- Как происходит останов УМТ?

Для записи программы нужно придумать формат записи одного правила. Пусть моделируемая МТ использует алфавит A_p как рабочий, q_0, q_1, ..., q_s — её состояния, а правило выглядит как $q_i a_j \to v_{ij} q_k$, где $i, k = 0, \ldots, n$; $j = 1, \ldots, p$; $v_{ij} \in \{a_1, a_2, \ldots, a_p, l, r, h\}$. Тогда введём

алфавит $B_p = \{b_1, b_2, \ldots, b_p\}$, дополнительные символы $\{l, r, h, +, -, O\}$ и запишем правило следующим образом:

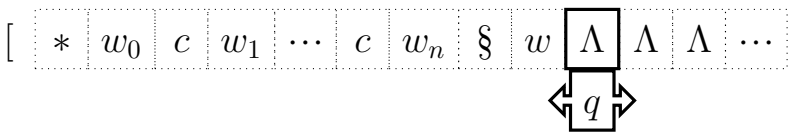
$$\begin{cases} b_j v_{ij} +^{k-i}, \text{если } k > i; \\ b_j v_{ij} O, \text{если } k = i; \\ b_j v_{ij} -^{i-k}, \text{если } k < i. \end{cases}$$

Здесь $+^{k-i}$ означает символ $+$, повторенный $k - i$ раз.

При записи правила нужно указать, для какого обозреваемого символа записано правило, что является действием и какое состояние является следующим после выполнения действия. Чтобы не путать запись обозреваемого символа и запись действия, удобно для записи символа ввести «зеркальный» алфавит B_p, в котором для каждого символа a_j есть свой соответствующий символ b_j. Действие записывается как обычно, т.е. одним из символов a_i или символами сдвига $\{l, r, h\}$. Наконец, следующее состояние кодируется разницей в номерах текущего и следующего состояний: так, если МТ находится в состоянии q_2, а нужно перейти в состояние q_5, то на ленту записывается три символа $+$: $+ + +$.

Правила программы МТ записываются подряд: сначала все правила для состояния q_0, потом для состояния q_1 и т.д. Записи правил разных состояний отделяются друг от друга вспомогательным маркером c, а вся программа заканчивается маркером §. Таким образом, вся программа кодируется словом $cw_0cw_1 \ldots cw_n\S$, где w_i — слово с записью подряд всех правил состояния q_i.

Для начала работы универсальной машины, помимо программы моделируемой МТ, нужно записать на ленту исходные данные для этой МТ, а также правильным образом расположить управляющую головку и пометить начальное состояние в программе МТ. Исходные данные для МТ удобно записывать на ленте справа от её программы (после символа-разделителя §): можно считать, что с этого разделителя и начинается лента для моделируемой МТ, бесконечная только справа. Управляющая головка располагается на первой ячейке ленты после входного слова, как и установлено соглашением об использовании нормальных МТ. Наконец, для пометки начального состояния можно заменить маркер c, соответствующий этому состоянию, на другой маркер, например, $*$. Тем самым вся лента УМТ перед началом работы будет выглядеть так:



Теперь для интерпретации программы моделируемой МТ нужно научиться выполнять каждый такт этой машины. Разберём по шагам необходимые действия:

1. Поиск правила для выполнения. Правило определяется парой «состояние, символ», при этом запись правил для текущего состояния q_i моделируемой МТ начинается на ленте с маркера-звёздочки, а символ a_j обозревается УГ. Запись правила для a_j, как мы помним, помечено в программе его зеркальным символом b_j. Поэтому УМТ выполняет следующие шаги:

   а) «Запоминает» обозреваемый символ a_j размножением состояний (т.е. дальнейшие действия будут варьироваться в зависимости от этого символа).

   б) Заменяет символ a_j на его зеркальную пару b_j.

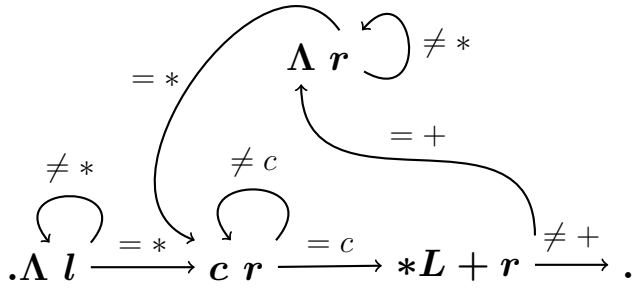
   в) Ищет слово w_i, содержащее запись правила. Для этого достаточно двигаться влево до маркера $*$.

   г) Ищет запись правила для символа a_j, что можно делать движением вправо до символа b_j.

2. Изменение текущего состояния моделируемой МТ. При выполнении такта моделируемой МТ нужно как выполнить необходимое действие, так и изменить текущее состояние. Оказывается удобным сначала изменить состояние, а потом выполнять действие — тогда после окончания выполнения можно сразу начинать интерпретацию следующего такта моделируемой МТ. УМТ выполняет следующие действия:

   а) Сдвигается на один символ вправо, пропуская v_{ij} — описание действия моделируемой МТ, до записи «смещения» до следующего состояния.

   б) Если обозреваемый символ — это символ O, то состояние моделируемой МТ не меняется, и ничего делать не нужно. Иначе требуется сдвинуть маркер $*$ на столько слов программы вправо или влево, сколько символов $+$ или $-$ находится на ленте, другими словами — пропустить соответствующее количество символов-маркеров c. Ниже приведена машина, выполняющая сдвиг вправо по символам $+$; машина для сдвигов влево строится аналогично.



   в) Возвращается на символ описания v_{ij} действия.

3. Выполнение действия моделируемой МТ. УМТ необходимо снова

найти ячейку ленты, на которой находится УГ моделируемой МТ. Для этого достаточно двигаться вправо до маркера §, а потом — до зеркального символа b_j. Далее, если действием являлась запись символа, то нужно записать символ, который обозревала УМТ перед началом выполнения этого шага; если сдвиг, то сначала вместо зеркального символа записать на ленту исходный символ a_j и далее выполнить соответствующий сдвиг УГ.

Важно, что если при выполнении сдвига влево УГ перешла на ячейку, содержащую маркер §, то моделируемая МТ зашла за левый край ленты.

4. Переход на выполнение нового такта. УМТ находится в готовности выполнять следующий такт моделируемой МТ, и снова можно переходить на шаг 1.

Осталось определить, когда происходит останов моделируемой машины. Если на шаге 2б) при сдвиге маркера текущего состояния с происходит переход на символ §, отделяющий программу моделируемой МТ от её ленты, то следующим состоянием будет являться состояние останова. В таком случае УМТ нужно выполнить действие моделируемой машины по шагу 3, а потом остановиться.

6.3 Упражнения

1. Объясните, почему в диаграмме МТ для шага 2б) первый из символов + обрабатывается отдельно от всех остальных (первая итерация цикла обработки отлична от последующих).

2. Измените диаграмму МТ шага 2б) так, чтобы обрабатывался случай перехода в состояние останова.

3. Постройте полную диаграмму УМТ, создав диаграммы для всех остальных пунктов шагов 1-4.

4. Придумайте собственную кодировку программы моделируемой МТ и измените диаграмму УМТ соответствующим образом. Например, откажитесь от зеркальных символов b_j. Как это решение изменит алгоритм интерпретации?

7 Алгоритмически неразрешимые задачи

Понятие алгоритмической неразрешимости. Проблема останова. Проблема самоприменимости.

Как говорилось в разделе 2, наличие формального определения алгоритма (т.е. машины Тьюринга) позволяет нам делать суждения обо

всех мыслимых алгоритмах, в частности, доказывать алгоритмическую неразрешимость некоторой задачи — принципиальную невозможность построить алгоритм её решения. Для этого нужно доказать, что не существует машины Тьюринга, решающей нашу задачу. Как правило, такие доказательства проводятся двумя способами: делается предположение о существовании искомой МТ, которое либо приводит к противоречию непосредственно, либо позволяет построить МТ для другой задачи, алгоритмическая неразрешимость которой известна изначально.

7.1 Проблема останова

Проблема останова формулируется следующим образом: существует ли алгоритм, определяющий, произойдет ли когда-либо останов машины $\boldsymbol{T}$, запущенной на входных данных w? Или иначе, остановится ли универсальная машина Тьюринга, моделирующая МТ $\boldsymbol{T}$ на входных данных w?

Существенно, что алгоритм, решающий проблему останова, должен давать ответ для всех возможных машин Тьюринга и любых входных данных (можно считать, что фиксирован входной алфавит МТ, что, как нам известно, не ограничивает общности). Понятно, что для некоторых классов машин можно построить алгоритм, определяющий, останавливаются ли они. Например, если на каждом такте МТ номер текущего состояния растёт, то количество тактов не превысит количества состояний.

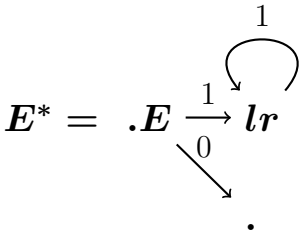
Теорема. Проблема останова алгоритмически неразрешима.

Доказательство. Пусть существует машина $\boldsymbol{D}$, решающая проблему останова для всех МТ $\boldsymbol{T}$ и входных данных w. Для этого машине $\boldsymbol{D}$ выдаётся некоторое описание машины $\boldsymbol{T}$ (например, подобное тому, как мы строили описание моделируемой МТ в универсальной машине) и входные данные w. Для удобства можно считать, что для входных данных и описания машины используется один и тот же алфавит (иначе легко добавить процедуру кодирования).

Построим машину $\boldsymbol{E}$, которая по данной МТ $\boldsymbol{T}$ запускает машину $\boldsymbol{D}$ для МТ $\boldsymbol{T}$ и записи (описания) $\boldsymbol{T}$ на ленте. Машине $\boldsymbol{E}$ уже не требуются никакие другие входные данные, помимо описания машины $\boldsymbol{T}$, и ей достаточно скопировать входное слово, описывающее машину $\boldsymbol{T}$, чтобы получить также и входные данные для этой машины, а далее запустить машину $\boldsymbol{D}$.

Не зная в деталях, как устроена машина $\boldsymbol{E}$, всё же можно сказать, что результатом её работы является запись на ленту числа `1` или `0` в зависимости от того, останавливается ли машина $\boldsymbol{T}$ или нет. Немного модифицируем машину $\boldsymbol{E}$, построив машину $\boldsymbol{E}^*$, которая останавливается в том случае, если машина $\boldsymbol{E}$ дала ответ `0`, и зацикливается

в противном случае (см. рисунок ниже).



Зададимся вопросом, останавливается ли машина $\boldsymbol{E}^*$, будучи применённой к описанию самой себя (т.е. описанию машины $\boldsymbol{E}^*$)? Если машина $\boldsymbol{E}^*$ останавливается, то машина $\boldsymbol{E}$ должна записать единицу на ленту в качестве ответа, однако в этом случае машина $\boldsymbol{E}^*$ зацикливается по построению. Если же машина $\boldsymbol{E}^*$ не останавливается, то машина $\boldsymbol{E}$ должна была записать на ленту ноль, однако в этом случае, опять же по построению, машина $\boldsymbol{E}^*$ должна остановиться. Полученное противоречие заставляет нас, как пишет М. Минский в своей замечательной книге «Вычисления и автоматы», с грустью заключить, что машина $\boldsymbol{E}^*$, а, следовательно, и машины $\boldsymbol{E}$ и $\boldsymbol{D}$ не существуют. □

7.2 Проблема самоприменимости

Машина Тьюринга $\boldsymbol{T}$ называется *самоприменимой*, если она останавливается, когда в качестве входного слова для неё используется описание самой машины $\boldsymbol{T}$ (как и ранее, будем считать, что с помощью кодирования описание задано во входном алфавите нашей машины). Проблемой самоприменимости, таким образом, является вопрос о существовании алгоритма, определяющего самоприменимость любой заданной машины $\boldsymbol{T}$.

Алгоритмическая неразрешимость проблемы самоприменимости может быть доказана тем же способом, что и неразрешимость проблемы останова. Действительно, предположив существование машины, решающей проблему самоприменимости, легко заметить, что таковой является машина $\boldsymbol{E}$ из теоремы о проблеме останова. Тем самым, повторив остальные рассуждения этой теоремы, мы тем же способом придем к противоречию.

8 Нормальные алгоритмы Маркова

Определение нормальных алгоритмов Маркова (НАМ). Процедура интерпретации НАМ. Примеры НАМ.

Алгоритмы Маркова (точнее — алгорифмы) являются другим примером формализации понятия алгоритма, существенно отличающейся

от машин Тьюринга. Алгоритмы были предложены русским математиком А. А. Марковым в середине XX века. В качестве основной операции в формализации Маркова используется замена слов, а не отдельных символов, что позволяет проще решать задачи обработки текстов.

Как и в случае машины Тьюринга, для построения формализации нам необходимо договориться о способах задания как обрабатываемой информации, так и вычислительного процесса. Для первой части можно также использовать алфавиты. Пусть V — алфавит основных символов, а $V^{'}$ — алфавит символов-маркеров; выберем два слова σ и $\sigma^{'}$ над объединённым алфавитом $V \cup V^{'}$. Тогда назовём *подстановкой* $\sigma \to \sigma^{'}$ перевод слова $\tau = \alpha\sigma\beta$ в слово $\tau^{'} = \alpha\sigma^{'}\beta$, где $\tau, \tau^{'}, \alpha, \beta \in V \cup V^{'}$. Другими словами, при применении некоторой подстановки ко входному слову τ в нём ищется слово σ, стоящее в левой части подстановки. Если этого слова в слове τ не содержится, то подстановка считается *неприменимой* к данному слову, иначе выбирается самое левое вхождение в слово τ слова σ и заменяется на слово $\sigma^{'}$. Например, результатом применения подстановки *м* $\to$ *ми* к слову «вмкмгу» является слово «вмикмгу», а к слову «фкнвшэ» эта подстановка неприменима.

Нужно заметить, что как слова α и β, так и слова σ и $\sigma^{'}$ могут быть пустыми. В первом случае заменяется вхождение, находящееся в начале или конце слова соответственно (т.е. префикс или суффикс слова). Во втором случае, если слово σ пусто, считается, что вхождение пустого слова находится слева от первой буквы входного слова, и в результате ко входному слову приписывается слева слово $\sigma^{'}$. Если же слово $\sigma^{'}$ пусто, то из входного слова удаляется самое левое вхождение слова σ. Например, после применения подстановки $\lrcorner \to$ *вмк* к слову «мгу» получается слово «вмкмгу», а после применения подстановки *мат* $\to \lrcorner$ к слову «мехмат» — слово «мех» (значком $\lrcorner$ помечено пустое слово).

Нормальный алгоритм Маркова (НАМ) задаётся конечной последовательностью подстановок $p_1, p_2, \ldots, p_n$. «Такт» работы алгоритма состоит в поиске подстановки, применимой к текущему обрабатываемому слову, начиная с первой подстановки в последовательности. Если ни одна подстановка не оказалась применимой, алгоритм завершается, иначе подстановка применяется (напомним, что заменяется самое левое вхождение слова из левой части подстановки). Подстановка может быть помечена как терминальная (для этого справа от стрелки добавляется точка, т.е. используется обозначение $\to .$, или применяется стрелка с чёрточкой $\mapsto$). В этом случае после применения подстановки алгоритм также завершается, иначе начинается следующий такт интерпретации и подстановка, применимая к текущему слову, снова ищется, начиная с первого элемента последовательности.

Более формально процедуру интерпретации можно описать следующим образом. Пусть задано входное слово $\sigma_0 \in (V \cup V^{'})^*$ и набор

подстановок $p_1, p_2, \ldots, p_n$.

1. Положить $i = 0$.
2. Положить $j = 1$.
3. Если подстановка p_j применима к слову σ_i, перейти к шагу 5.
4. Положить $j = j + 1$. Если $j \leqslant n$, то перейти к шагу 3, иначе остановиться.
5. Применить подстановку p_j к слову σ_i и построить слово σ_{i+1}. Если p_j — терминальная подстановка, то остановиться. Иначе положить $i = i + 1$ и перейти к шагу 2.

Говорят, что НАМ *применим* к слову σ_0, если в результате выполнения описанной процедуры интерпретации произойдёт остановка.

В качестве примера нормального алгоритма Маркова рассмотрим задачу шифрования входного слова шифром Юлия Цезаря. В этом шифре каждая буква алфавита заменяется на зашифрованную, которая отстоит от исходной в алфавите на некоторое фиксированное количество позиций. Точнее, i-я буква латинского алфавита шифруется $(i + c)\ mod\ 26$-й буквой, где i — номер буквы (начиная с нуля), c — некоторая константа. По словам Светония, Цезарь использовал этот шифр для важной переписки со сдвигом $c = 3$.

Шифрование слова заключается в последовательной замене каждого символа слова на зашифрованный аналог. Замена может выполняться подстановкой, при этом нужно каким-либо образом обозначить текущий обрабатываемый символ. В алгоритмах Маркова с этой целью часто используется символ-маркер $*$. Перед началом шифрования маркер устанавливается в начало слова с помощью подстановки с пустой левой частью ($\varnothing \to *$). Само шифрование выполняется одной из 26 подстановок вида $*a_i \to a_{(i+3)\ mod\ 26)}*$, где $0 \leqslant i < 26$, $a_i \in A_{26} = \{a, b, c, \ldots, z\}$. Применение подстановки одновременно шифрует символ слова, находящийся справа от маркера, и сдвигает маркер на следующий символ. Наконец, последняя подстановка удаляет маркер из зашифрованного слова ($* \mapsto \varnothing$). Работа алгоритма на этом заканчивается, поэтому данная подстановка является терминальной.

Таким образом, полный НАМ выглядит следующим образом: $*a \to d*$, $*b \to e*$, $*c \to f*$, …, $*y \to b*$, $*z \to c*$, $* \mapsto \varnothing$, $\varnothing \to *$. Заметим, что порядок перестановок играет существенную роль. В начале работы все перестановки, содержащие маркер, очевидно, неприменимы ко входному слову, и срабатывает перестановка, добавляющая маркер перед началом слова. Эта перестановка должна быть последней в НАМ. Далее, пока в слове остались незашифрованные символы, применяются перестановки из первых двадцати шести. Наконец, когда справа от

маркера не осталось символов (т.е. маркер — это последний символ слова), необходимо его удалить, причем соответствующая подстановка должна идти после шифрующих подстановок и до подстановки, вбрасывающей маркер.

С помощью нормальных алгоритмов Маркова можно получить те же теоретические результаты, которые уже были обсуждены в разделах 6 и 7. Можно построить универсальный алгоритм Маркова, который интерпретирует любой другой НАМ над заданным алфавитом. Также можно доказать алгоритмическую неразрешимость проблем останова и самоприменимости в формализации НАМ, т.е. доказать, что не существует НАМ, решающих эти проблемы.

8.1 Упражнения

1. Как изменится НАМ для шифра Цезаря, если входной алфавит будет состоять из 5 символов? из ста символов?

2. Постройте НАМ для решения следующих задач:

   а) $A = \{0, 1, 2, 3, 4\}$. Считая входное слово записью числа в 5-ричной системе счисления, увеличить это число на 1.

   б) $A = \{0, 1, 2, 3\}$. Считая входное слово записью числа в 4-ричной системе счисления, перевести это число в двоичную систему счисления.

   в) $A = \{a, b\}$. Определить наиболее часто встречающийся символ входного слова и выдать его в качестве результата алгоритма. Если символы входят в слово равное количество раз, в качестве результата выдать пустое слово.

   г) $A = \{a, b\}$. Удалить из входного слова последнее вхождение символа a, если оно есть, в противном случае оставить слово без изменения.

   д) $A = \{a, b\}$. Проверить, является ли входное слово палиндромом. В случае положительного ответа выдать в качестве результата слово a, иначе — пустое слово.

9 Заключение

Рассмотренные нами формализации понятия алгоритма далеко не исчерпывают все исследования в этой области. Существуют и другие формальные описания алгоритмов, например, машина Поста, рекурсивные функции, λ-исчисление. Про все эти формализации, включая НАМ, доказана их *эквивалентность* машинам Тьюринга: если для некоторой задачи можно построить алгоритм в одной из указанных

формализаций, то по этой формализации можно построить и выполняющую тот же алгоритм машину Тьюринга, и обратно.

Использование нами различных формальных моделей для построения алгоритмов опирается на *тезис Тьюринга-Чёрча*: для любого интуитивного алгоритма может быть построена реализующая его машина Тьюринга (аналогичный тезис о НАМ называется тезисом Маркова). Тезис Тьюринга-Чёрча невозможно строго доказать или опровергнуть, поскольку он связывает неформальное определение алгоритма и строго формализованную машину Тьюринга. Пока нет также и никаких отдельных свидетельств, которые опровергали бы этот тезис.

Список литературы

1. М. Минский. Вычисления и автоматы. Москва: Издательство «Мир», 1971.
2. Г. Эббинхауз, К. Якобс, Ф. Манн и Г. Хермес. Машины Тьюринга и рекурсивные функции. Москва: Издательство «Мир», 1972.
3. М. Р. Шура-Бура и Л. С. Корухова. Введение в алгоритмы (учебное пособие для студентов I курса). 2-е исправленное издание. Москва: Изд. отдел ф-та ВМК МГУ имени М.В. Ломоносова; МАКС Пресс, 2010.
4. В. Е. Зайцев и С. С. Гайсарян. Курс информатики: учебное пособие. Москва: Издательство МАИ, 1993.

Содержание