МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики

Л.С. Корухова, М.Р. Шура-Бура

ВВЕДЕНИЕ В АЛГОРИТМЫ

(учебное пособие для студентов I курса)

Москва

2010

УДК 519.6+510.6

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

Рецензенты:
профессор В.П. Иванников, доцент В.Н. Пильщиков

Корухова Л.С., Шура-Бура М.Р. Введение в алгоритмы. Учебное пособие для студентов I курса, 2-е исправленное издание. — М. Издательский отдел факультета ВМиК МГУ (лицензия ИД № 05899 от 24.09.2001 г.); МАКС Пресс, 2010, - 26 с.
ISBN 978-5-89407-399-6
ISBN

Учебное пособие представляет собой введение к основному курсу лекций для студентов факультета ВМК МГУ "Алгоритмы и алгоритмические языки". Обсуждается роль компьютера в решении проблемы накопления и сохранения знаний, детализируется представление о задаче обработки информации. Вводятся понятия процесса обработки и алгоритма. Подчеркивается эквивалентность задачи обработки слов и задачи вычисления целочисленных функций, формулируется основная гипотеза теории алгоритмов. Рассматриваются различные формальные уточнения понятия алгоритма: машина Тьюринга, нормальный алгоритм Маркова, и показывается их эквивалентность. Демонстрируется возможность реализации алгоритма реальным автоматом.

Для студентов факультета ВМК в поддержку основного лекционного курса “Алгоритмы и алгоритмические языки” и для преподавателей, ведущих практические занятия по этому курсу.

УДК
ББК

Учебное издание

КОРУХОВА Людмила Сергеевна, ШУРА-БУРА Михаил Романович
ВВЕДЕНИЕ В АЛГОРИТМЫ

Учебное пособие

Издательский отдел факультета вычислительной математики и кибернетики
МГУ им. М.В. Ломоносова
Лицензия ИД N 05899 от 24.09.01 г.

119992, ГСП-2, Москва, Ленинские горы, МГУ имени М.В. Ломоносова, 2-й учебный корпус

Напечатано с готового оригинал-макета
в издательстве ООО "Макс Пресс"
Лицензия ИД N 00510 от 01.12.99
Подписано к печати 21.12.2009 г. Формат 60x90 1/16 Усл.печ.л. 1,5 Тираж 400 экз. Заказ

119992, ГСП-2, Москва, Ленинские горы, МГУ им. М.В. Ломоносова, 2-й учебный корпус, 627 к..
Тел. 939-3890, 939-3891. Тел./Факс 939-3891

1. О ЗАДАЧЕ ОБРАБОТКИ ИНФОРМАЦИИ

Прежде всего, буквально несколько слов о предмете изучения курса лекций "Алгоритмы и алгоритмические языки" и его роли в образовании.

Машины, о которых идет речь в курсах лекций по информатике, по традиции называли электронно-вычислительными (ЭВМ), несмотря на то, что их роль далеко выходит за рамки понятия вычислений. В настоящее время за такими машинами прочно закрепилось название – компьютеры.

Роль вычислений по мере развития человеческого общества постоянно возрастала. На протяжении многих лет люди, в основном, удовлетворялись так называемыми ручными вычислениями и весьма примитивными средствами. Но все же интерес к совершенствованию способов изображения числовой информации, методов и средств вычислений постоянно сохранялся. Отметим изобретение в XVII веке известным ученым Блезом Паскалем арифмометра, являющегося прототипом одной из главных компонент ЭВМ - арифметического устройства. Наряду с появлением новых задач, решение которых требовало все более сложных вычислений, существенно увеличивались и объемы традиционных достаточно простых вычислений. Любопытно отметить, что первые серьезные шаги в развитии современных средств вычислений связаны с обработкой результатов переписи населения в США в конце позапрошлого века. Как средства такой обработки были использованы перфокарты и появились так называемые счетно-аналитические машины - табуляторы. С появлением прикладной атомной физики возникла острая потребность в сложных вычислениях, способных заменить эксперимент. Несомненно, что появление первых ЭВМ в 40-х годах 20-го века в значительной степени стимулировалось этой потребностью. Принципы работы, сформулированные известным ученым Джоном фон - Нейманом и заложенные в первые ЭВМ, в классических архитектурах сохранились с 40-х годов по сей день. Однако реальные возможности и разнообразие архитектур компьютеров выросли настолько, что в современном обществе они превратились в один из основных рычагов технического прогресса.

Благодаря компьютерам открылись совершенно новые перспективы в решении проблемы накопления и сохранения знаний, что без преувеличения позволяет рассматривать появление ЭВМ в ряду таких крупных событий в развитии человечества, как появление письменности и изобретение книгопечатания.

Создание ЭВМ инициировало появление совершенно новой области человеческой деятельности - программирования. Развитие ЭВМ и программирования превратило компьютер в незаменимый инструмент для целей хранения, передачи и обработки информации. Нет сомнения, что ЭВМ способствовали существенному росту интереса к обработке информации и к ее роли в жизни человечества. Роль какой-либо сферы деятельности характеризуется числом людей, занятых в этой области. В конце XIX века только 5% людей работали в области обработки информации. В 40-е годы XX века этот процент возрос до 30%, а сейчас уже и превысил 50%.

В рамках настоящего курса мы будем считать понятие информации интуитивно ясным и не будем давать определение этого понятия. Однако уточним, что, имея в виду обработку информации на компьютере, мы ограничиваемся информацией, которую можно задать в дискретном виде, т.е. конечным числом отдельных знаков. Принимая такое ограничение, мы исходим из предположения, что любую информацию можно задать в таком виде. Вернее, что любую информацию в задачах обработки информации можно "подменить" дискретной информацией. Например, картину можно изобразить в виде описания цветов пусть большого, но конечного числа отдельных точек. Наше

предположение о возможности дискретного изображения информации подкреплено тысячелетиями проверенной практикой использования человечеством письменности.

В общем виде обработку информации можно представить себе как извлечение ИСКОМОЙ информации из ИСХОДНОЙ:

< ИСХОДНАЯ информация > ➔ **ЗАДАЧА** ➔ < ИСКОМАЯ информация >

В этом схематическом представлении описание задачи и способа ее решения можно включить в понятие <ИСХОДНАЯ информация> и иметь дело со схемой :

< ИСХОДНАЯ информация > ===> < ИСКОМАЯ информация >

Если мы внимательно проанализируем эту общую схему, то вынуждены будем признать, что <ИСКОМАЯ информация> должна содержаться в исходной и, в принципе, никакой новой информации в результате обработки мы не получаем. В каком-то смысле такой довод не лишен основания. Например, при переводе Московского времени во время по Гринвичу достаточно вычесть 3 часа и, так сказать, представить ту же информацию в другой форме.

Однако нельзя не согласиться с тем, что форма информации имеет, вообще говоря, существенное значение. Ведь информация нужна лишь в том случае, когда она может быть "понята" (воспринята, проинтерпретирована…). Понятность же информации, безусловно, зависит от ее формы. Так, например, приказывая солдату остановиться словами "Прошу остановиться" (вместо команды "Стой!"), командир рискует быть непонятым.

В системе уравнений:

$$\begin{cases} X + Y = 5 \\ X - Y = 7 \end{cases}$$

конечно, содержится информация о значениях X и Y. Однако эта информация в форме

$$\begin{cases} X = 6 \\ Y = -1 \end{cases}$$

может оказаться куда более понятной и пригодной.

Рассмотрим еще одну задачу – определение приговора судом:

< суть дела и закон > ====> < приговор >

Здесь нет вычислений в традиционном смысле этого слова, но огромна исходная информация и необычайно важна форма искомой информации.

В связи с информацией и задачами её обработки и использования следует отметить наличие двух важных понятий:

1) *КОДИРОВАНИЕ*

< информация > ===> < форма информации >

2) *ИНТЕРПРЕТАЦИЯ*

< форма информации > ===> < понимание информации >

В этой общей постановке оба понятия обозначены лишь на интуитивно содержательном уровне.

Термин "интерпретация" для обозначения понимания подчеркивает условность понимания, о котором идет речь. При обработке информации, особенно когда речь идет об автоматизации обработки, всегда появляются различные уровни понимания

обрабатываемой информации, представленной в той или иной форме. С интерпретацией связано понятие исполнителя, способного по-своему понять заданную форму.

Компьютер предъявляет особые требования к форме задаваемой информации, в частности, к форме той части информации, которая определяет процесс обработки. В связи с этим возникают три уровня общения с ЭВМ:

1. Обращение за услугой
2. Обучение компьютера и проверка ее "умения"
3. Создание компьютера

Для реализации второго и третьего уровней нужны специалисты особой квалификации. Эту квалификацию вы должны получить, обучаясь на факультете.

Цель настоящего курса - заложить основы вашего умения учить машины и участвовать в их создании.

2. ПРОЦЕССЫ ОБРАБОТКИ ИНФОРМАЦИИ И АЛГОРИТМЫ

Детализируя представление о задаче обработки информации

<*ИСХОДНАЯ* информация > ==> < *ИСКОМАЯ* информация >,

мы приходим к понятию процесса обработки:

$I_1 ==> I_2 ==> ... ==> I_{n-1} ==> I_n$,

где I_1 - ИСХОДНАЯ информация, I_n - искомая информация, а I_j, j = 2, 3, ..., n-1 - получаемая в процессе обработки информация, которую можно назвать промежуточной.

Конечно, здесь точнее было бы говорить о пошаговом изменении формы информации и получении ИСКОМОЙ информации, так сказать, в "явном" виде, который можно интерпретировать как требуемый результат обработки. Переход от I_j к I_{j+1}(получение I_{j+1} из I_j) можно назвать шагами процесса обработки. Легко понять, что обработка информации $I_1 ==> I_n$ может быть реализована различными процессами. До тех пор, пока мы никак не ограничиваем того, что мы называем шагами процесса, мы мало продвигаемся в понимании последнего. Интуитивно хотелось бы, чтобы переходы $I_j ==> I_{j+1}$ были проще обработки $I_1 ==> I_n$. Естественно желать, чтобы шаги были достаточно просты и, более того, чтобы каждый из них, в свою очередь, сводился к последовательности промежуточных щагов перехода (подшагов), принадлежащих заранее четко определенной совокупности. С другой стороны, поскольку, как было замечено выше, процесс не определяется однозначно задачей получения I_n из I_1, появляется нужда в его описании. Заманчиво пользоваться таким описанием, которое не содержит в явном виде перечень всех шагов процесса, но дает возможность определить очередной шаг $I_j ==> I_{j+1}$, коль скоро выполнены все предыдущие. Математика давно и широко пользуется процессами такого рода, получившими название ***алгоритмов*** .

Классическим примером математического алгоритма является алгоритм Евклида нахождения наибольшего общего делителя (НОД) двух натуральных чисел. Этот алгоритм можно описать следующим образом.

2.1. АЛГОРИТМ ЕВКЛИДА

Пусть даны два натуральных числа $N_1 >= N_2$

Выполняем :

1) Делим нацело N_1 на N_2, получая остаток N_3, удовлетворяющий неравенствам $0 <= N_3 < N_2$

2) Если $N_3 = 0$, то N_2 есть искомый НОД, иначе делим N_2 нацело на N_3, получая остаток $N_4 >= 0$, с которым поступаем так же, как с N_3, и т.д.

Таким образом получаем убывающую последовательность

$N_1 > N_2 > N_3 > ... > N_k > N_{k+1} = 0$,

где N_k есть наибольший общий делитель чисел N_1 и N_2 .

2.2. СВОЙСТВА АЛГОРИТМОВ

В математике, пользуясь понятием алгоритма, долгое время удовлетворялись лишь интуитивным уровнем определения самого понятия алгоритма. Считалось, что алгоритм - это процесс обработки исходных данных, который описан конечным числом правил. При этом каждое правило точно определяет некоторое достаточно простое действие (шаг алгоритма). После выполнения каждого шага точно определено "что делать дальше", т.е. получен ли окончательный результат или же оказывается известным, какое из правил определяет следующий шаг процесса. Результат достигается конечным числом шагов или вообще не достигается.

Очевиден интуитивный уровень приведенного определения, содержащего такие понятия, как " ***исходные данные*** ", " ***точно*** ", " ***достаточно просто*** ", " ***действие*** " и ***" конечный результат*** ".

Особого внимания заслуживает понятие " исходных данных ". Предполагается, что алгоритм предназначен для работы с различными исходными данными и, более того, совокупность различных " исходных данных ", вообще говоря, потенциально бесконечна. Так, например, для алгоритма Евклида " исходными данными " является любая пара натуральных чисел.

В литературе указанные свойства алгоритмов часто формулируются следующими терминами :

1) Однозначность (детерминированность)
2) Результативность (конечность)
3) Простота и понятность
4) Массовость

До тех пор, пока математики ограничивались лишь поисками и построением (описанием) алгоритмов, интуитивный уровень определения алгоритма оказывался достаточным. Однако ситуация изменилась, как только математика столкнулась с задачами, для решения которых никак не удавалось ни найти алгоритмы, ни подтвердить возникшие предположения, что таких алгоритмов попросту не существует. Строгому же обоснованию таких предположений препятствовало отсутствие формального понятия алгоритма. Одной из таких задач оказался вопрос о разрешимости алгебраических уравнений в целых числах - проблема, включенная в начале XX века в число наиболее важных и трудных задач, стоящих перед математикой - так называемых проблем Гильберта.

Было предложено несколько способов определения понятия алгоритма, основанных на различных подходах. Замечательно, что все эти определения оказались, по существу, эквивалентными.

Обратим внимание на то, что в требуемом описании алгоритма, вообще говоря, не содержится указаний на общее число шагов процесса. Более того, описанный процесс может и вовсе не завершиться или, как говорят, зациклится.

Требовать в определении алгоритма завершения процесса нецелесообразно. Это требование, во-первых, существенно обеднит понятие алгоритма и, во-вторых, вообще говоря, оно не проверяемо. Как это будет видно из дальнейших рассмотрений, можно считать, что алгоритм неприменим на тех исходных данных, при которых он зацикливается, и применим на тех, при которых процесс заканчивается.

3. АЛГОРИТМЫ И ОТОБРАЖЕНИЯ

Для формального определения алгоритма нам потребуются некоторые широко используемые в математике понятия и обозначения.

Прежде всего, мы будем иметь дело с понятием множества, как совокупности объектов, называемых элементами этого множества. Будем считать, что существует особое " пустое " множество $\varnothing$, в котором нет ни одного элемента. Через $a \in A$ будем обозначать, что объект a является элементом множества **A**. $c \notin A$ будет означать, что объект c не является элементом **A**. Формула $A = \{ a_i \}$ означает, что **A** является множеством объектов a_i в предположении, что i принадлежит опреде ленной совокупности значений. В этом определении мы не накладываем никаких ограничений на характер объектов. Не предполагается и какая-то их однородность или "похожесть". В качестве объекта, входящего в множество, может выступать и любое множество. Запись $A = \{ a , b , c \}$ означает, что множество **A** состоит из элементов a , b , c .

Множество **B** будем называть подмножеством множества **A**, если каждый элемент множества **B** является элементом множества **A**. $B \subset A$ ($A \supset B$) означает, что **B** - подмножество множества **A**. Заметим, что пустое множество является подмножеством любого множества. Если $B \subset A$ и существует элемент $a \in A$ и $a \notin B$, то **B** называется собственным подмножеством множества **A**.

Объединением множеств **A** и **C** называется множество, состоящее из всех объектов (элементов), входящих по крайней мере в одно из множеств **A** или **C**. Объединение (сумма) двух множеств **A** и **C** обозначается $A \cup C$. Множество, состоящее из всех элементов, входящих как в множество **A**, так и в множество **C,** называется пересечением множеств **A** и **C** и обозначается $A \cap C$.

Множество **C**, состоящее из всех тех элементов множества **A**, которые не входят в множество **B**, обозначается $A \setminus B$ (вычитание множеств). $C = A \setminus B$ означает, что, если $c \in C$, то $c \in A$ и $c \notin B$.

Говорят, что задано отображение **F** множества **A** в множество **C**, если каждому $a \in A$ поставлено в соответствие $c \in C$. Иногда пишут $c = F(a)$ и говорят, что c является образом элемента a (c - значение отображения **F** на a) и что a - прообраз c. Наличие отображения **F** изображают диаграммой

$$A \xrightarrow{F} C$$

Иногда мы будем опускать в этой диаграмме обозначение отображения и писать просто :

$$A \longrightarrow C$$

Если при таком отображении **F** каждый элемент из C оказывается образом некоторого элемента из **A**, то **F** называется отображением множества **A** *на* множество **C**.

Если $B \subset A$ и

$$A \xrightarrow{G} C$$

то говорят, что **G** является частичным отображением множества **A** *в* **C** или что **G** - частичная функция. Считается, что на **A \ B** отображение **G** не определено.

Отображение **F** множества **A** на множество **C** называется взаимно-однозначным , если из $a \in A$, $b \in A$ и $a \neq b$ следует, что $F(a) \neq F(b)$.

Вводя понятие отображения (функции), мы не накладываем никаких ограничений на способы задания отображения. В ряде случаев нас может интересовать лишь факт существования соответствующего отображения и вытекающие из этого факта следствия. Нет сомнения, однако, что отображения (функции), для которых при заданном a мы можем найти $F(a)$, представляют особый и несомненный интерес. Отображения такого рода тесно связаны с понятием алгоритмов.

Пусть даны два множества **A** и **B**. Декартовым произведением множеств **A** и **B** называется множество **C** всех пар (a, b), где a - любой элемент из **A**, b - любой элемент из **B**.

$$C = A \times B$$

$c \in C$ тогда и только тогда, когда $c = (a, b)$, где $a \in A$ и $b \in B$.

Понятие декартова произведения естественно распространяется на произведение любого конечного числа множеств. По определению:

$A_1 \times A_2 \times A_3 = (A_1 \times A_2) \times A_3 = A_1 \times (A_2 \times A_3)$

$A_1 \times A_2 \times \ldots \times A_{k-1} \times A_к = (A_1 \times A_2 \times \ldots \times A_{k-1}) \times A_к$

Легко дать "прямое" эквивалентное определение декартова произведения

$A_1 \times A_2 \times \ldots \times A_к$

как множества всех "кортежей" $a_1, a_2, \ldots, a_k$ длины **k**, где $a_i \in A_i$, $i = 1,2,\ldots,k$.

В случае, когда $A_1 = A_2 = \ldots = A_k = A$, обозначаем декартово произведение $A_1 \times A_2 \times \ldots \times A_k$ как A^k ($A \times A = A^2$, $A \times A \times A = A^3$ и т.д.) Особый интерес для нас представляет случай, когда множество **A** конечно, т.е. состоит из конечного числа элементов:

$$A = \{ a_1, a_2, \ldots, a_p \}$$

Мы будем называть такое множество алфавитом (абстрактным), а его элементы - символами. Множество, образованное объединением всех A^k, где k = 1, 2, ..., и одного идеального элемента, называемого "пустым словом", будем обозначать через A^* . Элементы из A^* - конечные слова (цепочки символов) в алфавите **A**. Для каждого слова $w \in A^*$ естественно определяется длина слова - число символов, образующих это слово. Длина пустого слова считается равной нулю. Длина слова **w** обозначается $|w|$. Если представить себе, что алфавит $A = \{ a_1, a_2, \ldots, a_p \}$ содержит не только все буквы естественного языка, но и знаки препинания, пробела, переноса, цифры, специальные знаки, используемые в текстах, а также знаки, обозначающие концы строк, страниц и т.д., то среди элементов множества A^* окажутся все мыслимые конечные тексты. Любой конечный текст будет "словом" в алфавите **A** и, следовательно, любая информация, описанная конечным текстом, может быть изображена "словом" в алфавите **A**.

Дело, однако, не в том, что нами выбран такой обширный и удобный алфавит. Легко показать, что при помощи даже так называемого побуквенного кодирования (т.е. замены каждой буквы алфавита **A** определенным словом в алфавите B), любое слово в алфавите **A** может быть представлено словом в алфавите B, если последний содержит не менее двух символов. Из дальнейшего изложения будет следовать существование простого алгоритма, определяющего взаимно-однозначное отображение

$$A^* \longrightarrow B^*$$

для любой пары алфавитов **A** и **B**.

Следовательно оказывается, что любой алфавит, в том числе и однобуквенный, в принципе, пригоден для изображения информации соответствующими словами.

Таким образом задача обработки информации сводится к нахождению значений некоторого частичного отображения

A* $\longrightarrow$ **A***,

где **A** = { $\boldsymbol{a}_1$, $\boldsymbol{a}_2$, ... , $\boldsymbol{a}_p$ } - конечный алфавит, а **A*** - множество конечных слов в этом алфавите.

Среди частичных отображений **A*** $\longrightarrow$ **A*** нас будут интересовать, в первую очередь, отображения, значения которых могут быть получены процессами преобразований, определяемыми алгоритмами. Одной из задач, связанных с исследованием частичных отображений, оказывается задача установления принадлежности того или иного частичного отображения **A*** $\longrightarrow$ **A*** к интересующему нас классу.

Аналогичная задача давно возникла в математике в связи с заданием целочисленных функций целочисленных аргументов, являющихся частичными отображениями N $\longrightarrow$ N или N × N × .... × N ---> N, где N - множество неотрицательных целых чисел { 0, 1, 2,... }, и возможной их вычислимостью, т.е. с возможностью получать значения функций в результате некоторой цепочки вычислений.

Интуитивно понятие вычислений тесно связано с понятием алгоритма, прежде всего потому, что вычисления являются частным случаем процессов преобразования информации - информации, представленной в форме чисел.

Однако оказывается, что к этому частному случаю может быть сведен любой алгоритм.

Прежде всего установим связь частичного отображения A* ---> A* с частичными функциями N $\longrightarrow$ N. Установим следующее взаимно-однозначное соответствие, которое можно назвать нумерацией (#) :

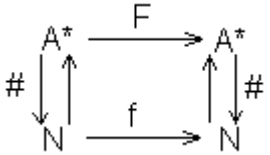
$$A^* \xrightarrow{\#} N$$

так, что по слову w ∈ A* можно было бы определить соответствующий ему номер #(w) и наоборот, по номеру можно было бы определить получившее этот номер слово. Пустому слову λ присвоим номер ноль (число). Однобуквенному слову a_i присвоим номер i для i = 1, 2,....,p. Слову $w = a_{i0}a_{i1}a_{i2}....a_{is}$ присвоим номер

$$N_w = i_0 + i_1 \times p + i_2 \times p^2 + ...+ i_s \times p^s .$$

Нетрудно доказать, что выбранная нумерация обладает требуемыми свойствами, а ее наличие подтверждает существование вычислимого взаимно-однозначного отображения A* --> B* для любой пары алфавитов A и B.

Для каждого отображения F : A*-- F --> A* отображение # индуцирует (порождает) отображение: f : N --- f---> N и для каждого отображения N -- f-->N отображение # индуцирует отображение F из A* в A* в соответствии с диаграммой:



Ограничиваясь отображениями, порождаемыми алгоритмами и вычислимыми функциями, этот результат можно сформулировать следующим образом:

1) Каждый алгоритм, определяя частичное отображение $A^* \xrightarrow{F} A^*$ ***, определяет вычислимую функцию*** $N \xrightarrow{f} N$ ***, где f(n) = #(F(w)) для n = #(w).***

2) Каждая вычислимая функция f(n) : N $\xrightarrow{f}$ N определяет алгоритм (частичное отображение на множестве слов алфавита А) A* $\xrightarrow{F}$ A* , для которого # (F(w)) = f(#(w)).

4. ВЫЧИСЛИМЫЕ ФУНКЦИИ И ТЕЗИС ЧЕРЧА

Вопрос о вычислимости целочисленных функций, целочисленного аргумента, если говорить совсем кратко, решен следующим образом. Признано (***тезис Черча***), что совокупность всех вычислимых частичных функций совпадает с совокупностью так называемых частично-рекурсивных функций, т.е. функций, которые можно построить из набора совсем простых функций путем использования трех четко определенных операций: суперпозиции, примитивной рекурсии и минимизации.

В качестве исходных простых, считающихся вычислимыми по определению, могут быть взяты :

1) Тождественно равная нулю функция **O**(x) = O

2) Набор функций $\mathbf{I}_n^m(x_1, x_2, ..., x_n) = x_m$.

3) Функция **S**(x) = x + 1.

Три операции определяются следующим образом:

1) **Суперпозиция**

Из вычислимой функции $h(z_1, z_2, ..., z_n)$ и n вычислимых функций

$g_1(x_1, x_2, ..., x_m)$, $g_2(x_1, x_2, ..., x_m)$, ..., $g_n(x_1, x_2, ..., x_m)$ получаем вычислимую функцию $f(x_1, x_2, ..., x_m)$, определяя ее следующим образом :

$f(x_1, x_2, ..., x_m)=h(g_1(x_1, x_2, .., x_m),g_2(x_1, x_2, ..., x_m),..,g_n(x_1, x_2, .., x_m))$

2) **Примитивная рекурсия**

По двум вычислимым функциям $g(x_1, x_2, ..., x_n)$ и $h(x_1, x_2, ..., x_n, x_{n+1}, x_{n+2})$ определяем вычислимую функцию $f(x_1, x_2,..., x_n, x_{n+1})$ следующим образом :

$f(x_1, x_2, ..., x_n, 0) = g(x_1, x_2,..., x_n)$

$f(x_1, x_2, ..., x_n, y+1) = h(x_1, x_2,..., x_n,y,f(x_1, x_2, ...,x_n, y))$

3) **Операция минимизации**

По вычислимой функции $g(x_1, x_2, ..., x_n)$ определяем вычислимую функцию

$z = f(x_1, x_2, ..., x_n)$ из уравнения

$$g(x_1, x_2, ..., x_{n-1}, z) = x_n \quad (*)$$

Значение $z=f(x_1, x_2, ..., x_n)$ для данного набора значений аргументов $x_1, x_2, ..., x_n$ находим, подставляя в уравнение (*) конкретные значения аргументов $x_1, x_2, ..., x_{n-1}$,

а вместо z последовательно подставляем числа 0, 1, 2, .. и т.д. Эта последовательность подстановок прекращается, как только набор конкретных значений аргументов $x_1, x_2, ..., x_{n-1}$ и очередное подставляемое значение z оказываются набором значений аргументов, на котором не определена функция $g(x_1, x_2, ..., x_n)$, или в случае, когда уравнение (*) оказывается удовлетворенным. В первом случае функция $f(x_1, x_2, ..., x_n)$ считается неопределенной на данном наборе значений аргументов $x_1, x_2, ..., x_n$. Во втором - значением функции $f(x_1, x_2, ..., x_n)$ на наборе значений аргументов $x_1, x_2, ..., x_n$ считается последнее, подставленное вместо z, число. Если последовательность подстановок не прекращается, функция $f(x_1, x_2, ..., x_n)$ считается не определенной на данном наборе значений аргументов. Ясно, что, если функция $f(x_1, x_2, ..., x_n)$ определена для данного набора значений аргументов $x_1, x_2, ..., x_n$, ее значение получается после конечного числа шагов, и мы вправе считать ее вычислимой частичной функцией.

Замечания

1. Хотя исходные функции **O**(x), $\mathbf{I}_n^m(x_1, x_2, ..., x_n)$ и **S**(x) определены всюду, операция минимизации приводит к частичным функциям (определенным не всюду), и, следовательно, в определенных выше операциях суперпозиции и примитивной рекурсии могут участвовать частичные функции. Считается, что функции, определяемые этими операциями, определены только для тех наборов значений аргументов, для которых определены все функции, входящие в правые части соответствующих равенств.

2. Тезис Черча постулирует возможность получения значений любой вычислимой частичной функции процессом, использующим лишь операции суперпозиции, примитивной рекурсии, минимизации и вычисления функций **0**(x), **S**(x), $\mathbf{I}_n^m(x)$. Конечно, тезис Черча не утверждает, что такой же результат не может быть получен другими вычислениями.

Так, например, вычисление суммы двух целых x и y можно свести к вычислению рекурсивной функции **SS**(x,y), определяемой примитивной рекурсией

$$\mathbf{SS}(x, 0)=x,\ \mathbf{SS}(x, y+1))= \mathbf{SS}(x, \mathbf{S}(y))=\mathbf{S}(\ \mathbf{SS}(x, y))$$

Конечно же, искомая сумма может быть получена и обычным сложением чисел x и y.

Несмотря на простоту используемых исходных функций и скромный набор операций, в отсутствии (по крайней мере, явном) средств разделения области значения независимых переменных, а также средств выделения области определения функции, в частично - рекурсивной форме представимы все известные вычисления, и есть серьезные основания считать, что и все возможные.

В качестве первого примера, иллюстрирующего возможности выбранных средств, рассмотрим функцию f(x,y), задаваемую формулами f(x, y) = x - y, если x >= y, и f(x, y)=0, если x < y. Легко проверить, что f(x, y) может быть задана примитивной рекурсией :

$$f(x, 0)=x,\ f(x, y+1)=f(x, \mathbf{S}(y))=g(f(x, y)),$$

где g(z) определяется примитивной рекурсией : g(0) = 0, g(**S**(z)) = z.

Вторым примером рассмотрим вычитание z = x-y, определенное только при

x >= y. Эта частичная функция z является результатом операции минимизации для соотношения **SS**(x,z) = y, где **SS**(x,z) определенная выше функция сложения. Легко видеть, что значение z определено операцией минимизации только при x >= y и поиск значений z при x < y не завершится. Последний пример иллюстрирует общее правило, заключающееся в том, что частично - рекурсивная функция определена для тех и только тех значений аргументов, для которых ее значение может быть вычислено по общим правилам в соответствии с ее представлением.

3. Устанавливая связь отображений на множестве слов из **A*** и функциями целочисленного аргумента, мы рассматривали функции лишь одного аргумента. Сформулированное же решение вопроса вычислимости касается также и функций многих переменных. Рассмотрение функций многих переменных вполне естественно в математике, хотя их привлечение, в принципе, не расширяет понятие вычислимости. Действительно, можно задать алгоритмы взаимно-однозначных отображений

$$N \times N \times ... \times N = N^k \xrightarrow{t_k} N$$

и, следовательно, воспользоваться диаграммой

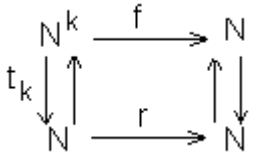


Диаграмма показывает, что при установлении взаимно-однозначного отображения между N^k и N каждая вычислимая функция $f(x_1, x_2, ..., x_n)$ порождается вычислимой функцией одного аргумента r(x), и наоборот.

5. ФОРМАЛИЗАЦИЯ АЛГОРИТМА

Установленная выше связь между процессами обработки слов и вычислениями значений целочисленных функций вместе с тезисом Черча, утверждающим, что всякая вычислимая функция частично-рекурсивна, позволяет подвести под понятие алгоритма строгий математический базис. Для этого достаточно считать, что алгоритмы порождают те и только те частичные отображения на множестве слов, которым соответствуют одноместные частично-рекурсивные функции на "номерах" слов. Номера слов могут определяться, например, описанной выше нумерацией (функцией #).

Формально такой подход открывает возможность доказательства строгих математических утверждений и, в частности, утверждений о том, что те или иные отображения не могут быть заданы алгоритмами, т.е. об алгоритмической неразрешимости некоторых проблем. Однако такой способ исследования алгоритмов привлекает довольно сложные понятия и представляется несколько искусственным.

В математике были предложены другие, так сказать, "прямые" способы формализации понятия алгоритма. На двух из них мы остановимся подробнее. Несмотря на различия, каждый из этих двух способов основан на идее исключения из формального описания алгоритма какой-либо содержательной информации, касающейся целей и смысла как самого алгоритма, так и операций, выполняемых на его отдельных шагах.

Реализация этой идеи, на первый взгляд, входит в противоречие с интуитивным представлением об алгоритмах. Разные алгоритмы предназначены для отыскания решений разнообразных задач, в то же время каждый конкретный алгоритм может быть предназначен лишь для обработки исходных данных лишь определенного типа. Так, алгоритм Евклида, обрабатывает пару натуральных чисел и исходное слово для этого алгоритма должно изображать эту пару. Для исходных данных, не задающих пару натуральных чисел, алгоритм Евклида теряет смысл. Ориентация конкретного алгоритма на определенный тип исходных данных может выражаться явными требованиями к ним и в ограничениях, вытекающих из описания операций, предусмотренных в шагах алгоритма. Обнаружение нарушения этих требований и ограничений естественно расценивать, как неприменимость алгоритма. Однако такой подход можно признать приемлемым только в том случае, когда обнаружение нарушений требований и ограничений, о которых говорится выше, достигалось бы, так сказать, алгоритмически. То есть, в результате предварительного алгоритма распознавания допустимости исходных данных и далее в процессе выполнения шагов алгоритма при помощи каких-то простых операций, включенных в эти шаги. Все эти дополнения разумно считать неотъемлемой частью алгоритма.

В такой ситуации формально исчезает отличие успешного окончания процесса выполнения алгоритма от его неудачного завершения. Отличие между такими завершениями обнаруживается только при содержательном рассмотрении результата. На

первый взгляд, заманчиво ввести в понятие алгоритма два типа окончаний - "нормальное" (результативное) и "аварийное" (нерезультативное). Однако на этом пути возникают два существенных замечания. Во-первых, нет оснований формально ограничиваться лишь двумя типами окончаний. Во-вторых, расширяя таким образом понятие алгоритма, мы остаемся с тем же классом частичных отображений на множестве слов (**A***) конечного алфавита (**A**), что и при исходном понятии алгоритма. Это так, потому что каждому алгоритму F, реализующему частичное отображение:

$$\mathbf{A}^* \xrightarrow{F} \mathbf{A}^*$$

можно сопоставить эквивалентный алгоритм ***f***, который не завершается ни на одном слове (***v***), не принадлежащем области его определения (***v*** $\in$ **A***, но ***v*** $\notin$ **B** $\subset$ **A***, где **B** - область определения F). В самом деле, каждый алгоритм с "аварийными" завершениями может быть превращен в эквивалентный ему алгоритм без "аварийных" завершений. Для этого достаточно каждое предписание об "аварийном" окончании заменить предписанием о переходе к выполнению бесконечного цикла, предварительно включив такой цикл в описание алгоритма как один из его шагов.

Таким образом, мы можем избавиться от "аварийных" (нерезультативных), "нормальных" (результативных) и других вариантов окончаний алгоритма. И без потери общности можем ограничиться лишь рассмотрением и исследованием алгоритмов, для каждого из которых область определения (применимости) совпадает с множеством слов, исходя из которых он завершается после выполнения конечного числа шагов. А областью неприменимости оказывается множество всех слов, исходя из которых алгоритм не завершается (зацикливается). Рассматривая только такие алгоритмы, мы избавляемся от трудностей, связанных с содержательным рассмотрением исходных данных и результатов алгоритма, а также открываем путь для максимального упрощения промежуточных шагов и отказа от содержательных рассмотрений.

Другими словами, абстрагируясь от конкретного содержания решаемых с помощью алгоритмов задач, мы сводим все такие задачи к нахождению слова - "ответа" по заданному слову - "исходным данным". И, в соответствии с этим, рассматриваем алгоритмы, как процессы преобразования конечных слов в конечном алфавите. Такие достаточно формализованные процессы мы вправе считать эквивалентными общему понятию алгоритма.

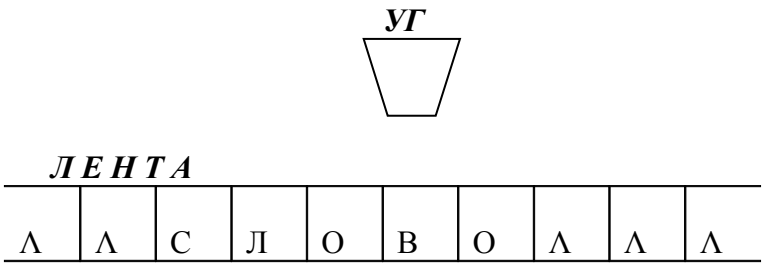
Перейдем к рассмотрению двух способов формализации понятия алгоритма. Первым рассмотрим подход Тьюринга - Поста.

5.1. МАШИНА ТЬЮРИНГА

Английский математик Алан Тьюринг в 1936 году предложил формальное определение понятия алгоритма, получившее впоследствии название "машина Тьюринга". Основная идея предложенного Тьюрингом подхода к уточнению понятия алгоритма, заключается в том, что алгоритмами должны называться те и только те отображения на множестве слов, которые могли бы осуществлять достаточно простые машины - автоматы. Выполнение отображения понимается при этом следующим образом. Машине - автомату предъявляется любое исходное слово, а она "выдает" слово, являющееся образом исходного слова при данном отображении.

Если предъявление исходного слова должно предшествовать полностью автономной работе машины, то последняя должна обладать устройством, на котором можно было бы зафиксировать любое сколь угодно длинное слово. Неограниченная лента, пожалуй, самое простое из таких воображаемых устройств.

Машина Тьюринга работает с лентой, разбитой на равновеликие ячейки, и имеет в своем "устройстве" управляющую головку (УГ), мимо которой может протягиваться лента.



При остановке ленты перед управляющей головкой оказывается одна из ячеек. Лента протягивается шагами на одну ячейку в любую сторону. Конечно, можно считать, что от ячейки к соседней ячейке продвигается управляющая головка, а лента остается неподвижной. Мы будем предполагать, что лента неограниченна в обе стороны, хотя эквивалентные результаты получаются, если считать, что лента ограничена с одной из сторон.

Можно считать, что лента конечна, но дополняема, т.е. что каждый раз, когда возникает потребность в протяжке ленты на шаг, выводящий за один из концов ленты, к последней "приклеивается" дополнительный кусок. Вместо "приклеивания" можно представлять себе, что в этом случае в лентопротяжный механизм ставится другая лента, которая становится продолжением снятой ленты. Лента машины в процессе работы превращается в последовательность лент, число которых может расти. Такая работа должна поддерживаться дополнительными возможностями машины. В частности, машина должна реагировать на ситуацию "выход за конец ленты" и выдавать соответствующий сигнал, приостанавливая протяжку и дожидаясь сигнала для ее продолжения.

В каждой ячейке ленты может быть записан любой символ из связанного с данной машиной Тьюринга конечного алфавита $\mathbf{A} = \{ a_1, a_2, \ldots, a_p \}$, называемого ***внешним алфавитом*** машины. К алфавиту **A** присоединяется специальный символ "пустоты", который для определенности мы будем обозначать Λ. Расширенный таким образом алфавит будем обозначать **Ã.** Считается, что в каждой ячейке ленты, в которую не был записан никакой символ из **Ã**, записан символ Λ. При записи любого символа из **Ã** в какую-либо ячейку ленты, ранее записанный в нее символ, исчезает. Запись символов из в **Ã** ячейки ленты может быть произведена как перед началом работы машины, так и в процессе ее работы. В каждый момент только в конечном числе ячеек ленты могут быть записаны символы, отличные от Λ. Говоря о записи на ленту машины слова **w** из **Ã***, мы будем понимать запись последовательности составляющих слово **w** символов в последовательно расположенные слева - направо ячейки ленты машины.

Управляющая головка является устройством, способным находиться в любом из конечного множества внутренних состояний. Эти состояния будем помечать некоторыми символами ($q_0, q_1, \ldots, q_s$), составляющими ***внутренний алфавит*** $\mathbf{Q} = \{ q_0, q_1, \ldots, q_s \}$ машины Тьюринга. Среди состояний управляющей головки имеются два особых состояния - состояние останова и состояние, называемое начальным. Для определенности будем считать, что останов помечен символом q_s, а начальное состояние - символом q_0. Управляющая головка, находясь в отличном от останова состоянии, способна воспринимать символ, записанный в расположенную перед ней ячейку, записывать в эту ячейку любой символ из **Ã**, изменять свое состояние и выдавать сигналы для протяжки ленты на одну ячейку. Оказавшись в состоянии останова, управляющая головка выдает сигнал конца работы. Перед началом работы машины Тьюринга на ее пустую ленту записывается слово, которое мы будем называть исходным.

Работа машины Тьюринга начинается по сигналу "пуск", инициируемому извне, например, нажатием кнопки, и распадается на последовательность тактов. Начало каждого такта определяется специальным сигналом, возникающим через определенные промежутки времени. Сигнал "пуск" переводит головку в начальное состояние q_0 и запускает генератор сигналов начала такта. По сигналу начала такта управляющая головка (если она не находится в состоянии останова) распознает символ, записанный в оказавшейся перед ней ячейке ленты, и конкретизирует свои действия в текущем такте работы. То есть, "решает" - какой символ из **Ã** следует записать в стоящую перед головкой ячейку, в какое новое состояние перейти и, наконец, какой сигнал выдать лентопротяжному механизму. Эти решения должны однозначно определяться парой (q_i, a_j) - текущим состоянием управляющей головки q_i и распознанным символом a_j в обозреваемой ячейке ленты. После выполнения выбранных действий и прошествии некоторого промежутка времени, достаточного для протяжки ленты, согласно выданному сигналу, снова возникает сигнал начала такта, и т.д. Такая работа продолжается до тех пор, пока управляющая головка не окажется в состоянии останова (q_s). Слово, оказавшееся при этом на ленте, называется выходным словом и считается результатом работы машины. Если такого не случится, то машина будет работать без конца, и считается, что машина не применима к соответствующему исходному слову. Заметим, что внутри исходного слова и внутри слова - результата не должен содержаться символ Λ.

В каждом такте "поведение" машины Тьюринга или ее работа определяется значениями трех функций: $\varphi(q_i, a_j)$, $\psi(q_i, a_j)$ и $\delta(q_i, a_j)$, заданных на конечном множестве **Q'** x **Ã** и принимающих, соответственно, значения из **Q**, из **Ã** и из **S** = {R, L}, где R - сигнал протяжки для подвода к головке следующей (правой) ячейки ленты, а L - сигнал для подвода предыдущей (левой) ячейки. Здесь **Q'** = **Q** \ { q_s}, т.е. **Q** без состояния останова.

Таким образом, все поведение автомата можно описать тремя функциями - **φ** , **ψ** и **δ**, которые удобно изобразить тремя таблицами вида:

Сост.\ симв.	Λ	a_1	…	a_j	…..	a_p
q_0		З	Н	А	**-**	
q_1	Ч	Е	Н	И	Я	
….	Ф	У	Н	К	**-**	
q_s	Ц	И	И	**φ**	**ψ**	**δ**

Таблицы для приведенных выше трех функций можно "слить" в одну, которую можно считать описанием соответствующей машины Тьюринга:

Сост.\ симв.	Λ	a_1	…	a_j	…	a_p
q_0						
q_1						
…						
q_i				**a_j',X,q_i'**		
….						
q_s						

где $a_j' \in$ **Ã,** $q_i' \in$ **Q,** X = { R, L}

Эта таблица обычно называется таблицей переходов. Строка таблицы, соответствующая состоянию q_s, фактически состоит из пустых клеток, и ее можно исключить. Однако при рассмотрении композиции алгоритмов наличие этой строки упростит понимание того, как строится таблица переходов для композиции. Работа машины Тьюринга оказывается процессом выполнения весьма простого алгоритма, использующего таблицу переходов в соответствии с описанными выше правилами. На результат работы машины Тьюринга может влиять начальное расположение управляющей головки относительно изображения исходного слова на ленте. Исключить эту неопределенность можно, введя дополнительное соглашение. Например, можно считать, что управляющая головка в начале работы обозревает клетку, расположенную левее исходного слова или клетку, в которой записан первый символ слова. В конце работы управляющая головка "указывает" на первый символом слова-результата и находится в состоянии останова q_s.

Свяжем с машиной Тьюринга отображение

$$\mathbf{A}^* \xrightarrow{T} \mathbf{A}^*$$

на множестве слов внешнего алфавита и будем говорить, что машина T реализует это отображение. Более того, мы будем считать, что эта машина реализует отображение

$$\mathbf{B}^* \xrightarrow{T'} \mathbf{A}^* ,$$

где $\mathbf{B} \subset \mathbf{A}$ и $\mathbf{T'(w)} = \mathbf{T(w)}$, если $\mathbf{w} \in \mathbf{B}^*$. Такая ситуация возникает, если машина T переводит слова из $\mathbf{B}^*$ только в слова из $\mathbf{B}^*$, хотя в процессе работы машины на ленту могли записываться при этом и символы из $\mathbf{A} \setminus \mathbf{B}$.

Если забыть о неограниченности ленты машины Тьюринга, то такая машина выглядит в наше время вполне реальной. Все мы знаем о существовании машин - автоматов, способных выполнять куда более сложные последовательности операций, чем те, которые должны выполнять машины Тьюринга. Более того, чтобы понять, как можно сконструировать машину Тьюринга, достаточно знакомства с элементарными сведениями из электротехники. Однако препятствием к полной реализации машины Тьюринга является предположение о неограниченности ленты. Напомним, однако, что, если мы желаем использовать машину Тьюринга для реализации алгоритмов, то предположение о неограниченности ленты весьма существенно из-за бесконечности множества $\mathbf{A}^*$ всех слов в алфавите **A** и необходимости задания для начала работы машины сколь угодно длинных слов изображающих исходные данные. Так, например, в случае алгоритма Евклида надо обеспечить задание пары сколь угодно больших чисел.

Оставляя в стороне вопросы материальной реализации машин Тьюринга, под построением такой машины мы будем понимать выбор ее внешнего и внутреннего алфавитов и задание таблицы переходов.

Тьюрингом была высказана гипотеза о возможности реализации любого порожденного алгоритмом отображения на множестве слов конечного алфавита при помощи подходящего предложенного им автомата, т. е. машины Тьюринга. Эта гипотеза недоказуема, поскольку употребляемое в ней понятие алгоритма четко не определено. Но ее можно обосновать построением машин, реализующих конкретные алгоритмы.

Говоря о гипотезе Тьюринга, как о возможности реализации любого алгоритма при помощи машины, подразумевают существование машины, реализующей алгоритм, эквивалентный данному.

ПРИМЕР

Построим машину Тьюринга (МТ), которая к числу, записанному на ленте, прибавляет 1. Число задается последовательностью десятичных цифр, т.е. словом в алфавите **A** = {0, 1, ..., 9 }. Пусть Λ обозначает пустую ячейку ленты.

q $_0$ - начальное состояние МТ - поиск первой (левой) цифры числа;
q $_s$ - состояние останова;
q $_1$ - "ищем" последнюю цифру числа;
q $_2$ - прибавляем 1 к цифре, записанной в обозреваемой клетке
q $_3$ - возвращаемся к первому (самому левому) символу слова-результата

Сост.\ симв.	Λ	0	1	2	…	8	9
q $_0$	Λ,R,q_0	$0,R,q_1$	$1,R,q_1$	$2,R,q_1$		$8,R,q_1$	$9,R,q_1$
q $_1$	Λ,L,q_2	$0,R,q_1$	$1,R,q_1$	$2,R,q_1$		$8,R,q_1$	$9,R,q_1$
q $_2$	$1,L,q_s$	$1,L,q_3$	$2,L,q_3$	$3,L,q_3$		$9,L,q_3$	$0,L,q_2$
q $_3$	Λ,R,q_s	$0,L,q_3$	$1,L,q_3$	$2,L,q_3$		$8,L,q_3$	$9,L,q_3$

Для того, чтобы эта машина выполняла прибавление 1, управляющая головка в начале работы должна находиться перед клеткой ленты, в которой записана цифра числа, либо перед клеткой левее заданного числа.

5.1.1. ОБОСНОВАНИЕ ГИПОТЕЗЫ ТЬЮРИНГА

Правдоподобность гипотезы Тьюринга подтверждается не только возможностью построения машин, реализующих конкретные алгоритмы, но и доказательством возможности построения машин, реализующих различные композиции алгоритмов, при наличии машин, реализующих алгоритмы - компоненты рассматриваемых композиций.

Прежде всего отметим возможность рассмотрения лишь какого-либо одного конкретного внешнего алфавита **A**, вспомнив об эквивалентности различных алфавитов. Кроме того, полезно ограничиться машинами Тьюринга, управляющая головка которых в состоянии останова всегда оказывается перед ячейкой ленты, в которой записан первый символ слова-результата (как в примере выше).

Пусть алгоритм ***D*** является произведением алгоритмов ***B*** и ***C*** (обозначается как ***D=B*C*** или ***D=C(B)***). Иначе говоря, слово-результат алгоритма ***D*** для исходного слова ***w*** является результатом применения алгоритма ***C*** к слову-результату алгоритма ***B*** для исходного слова ***w***, т. е.

$$D(w)=C(B(w))$$

В предположении существования машин Тьюринга T^{B} и T^{C}, реализующих, соответственно, алгоритмы ***B*** и ***C,*** построим машину Тьюринга T^{D}, реализующую алгоритм ***D***.

Пусть $\{ q_0^B, q_1^B, ..., q_r^B, q_s^B \}$ - внутренний алфавит машины T^{B} и $\{q_0^C, q_1^C, ..., q_t^C, q_s^C\}$ - внутренний алфавит машины T^{C}. Внутренним алфавитом машины T^{D} выберем $\{ q_0^B, q_1^B, ..., q_r^B, q_0^C, q_1^C, ..., q_t^C, q_s^C \}$, т.е. объединение внутренних алфавитов машин T^{B} и T^{C} с исключением заключительного состояния машины T^{B} (q_s^B). Таблицу переходов машины T^{D} составим из всех строк таблиц переходов машин T^{B} и T^{C}, в клетках которых всюду q_s^B заменим на q_0^C, удалив, кроме того, строку, соответствующую состоянию q_s^B . Впрочем это удаление носит лишь"косметический" характер, так как после проделанных замен q_s^B на q_0^C в состояние q_s^B управляющая головка построенной машины T^{D} никогда не переходит. Легко убедиться, что построенная машина реализует алгоритм ***D = B*C.***

Покажем, как построить машину Тьюринга, реализующую альтернативу выполнения алгоритмов ***B*** или ***C*** в зависимости от истинности вычислимого предиката **P**, если построены машины Тьюринга T^B, T^C, T^P, реализующие соответствующие алгоритмы. Предположим, что внешние алфавиты всех трех машин совпадают, а относительно машины T^P предположим, что результатом ее работы для исходного слова ***w*** будет приписывание к этому слову ***w*** в его начало некоторого, выделенного для обозначения истинности предиката, символа внешнего алфавита или любого другого символа, если значение предиката ложь.

Сохранив внешний алфавит, внутренним алфавитом реализующей альтернативу машины Тьюринга T^{AL} возьмем объединение внутренних алфавитов машин T^B, T^C и T^P. Символом начального состояния машины T^{AL} объявим символ начального состояния машины T^P, а символы конечных состояний машин T^B и T^C отождествим, объявив их символом конечного состояния машины T^{AL}. Таблицу переходов машины T^{AL} составим из всех строк таблиц переходов машин T^B, T^C и T^P, внеся в эти строки следующие изменения: в строке, соответствующей символу q_s^P, т.е. конечному состоянию машины T^P, во все клетки, за исключением клетки из столбца, соответствующего символу, выделенному для обозначения истинности предиката **P**, впишем тройку (Λ,R, q_0^C), а в исключенную клетку для символа "ложь"запишем тройку (Λ,R, q_0^B). Построенная машина T^{AL}, как нетрудно убедиться, реализует требуемую альтернативу.

Существование машин Тьюринга, реализующих различные композиции алгоритмов, является серьезным доводом в пользу истинности гипотезы Тьюринга, хотя и не приводит к формальному доказательству ее истинности. Другим подтверждением гипотезы Тьюринга можно считать строгие доказательства ее эквивалентности гипотезам, положенным в основу других подходов к формализации понятия алгоритма, - как тезису Черча о вычислимых функциях, так и принципу нормализации в рассмотренных ниже нормальных алгоритмах Маркова.

5.2. НОРМАЛЬНЫЕ АЛГОРИТМЫ МАРКОВА

В 1954 году русский математик А.А.Марков предложил новый подход к формализации понятия алгоритма. В отличие от подхода Тьюринга, доводящего простоту описываемых операций до замены символов, А.А.Марков предложил использовать замену слов, включив в шаг алгоритма процедуру выбора реализуемой замены.

Для дальнейшего изложения нам потребуется знакомство с некоторыми терминами и понятиями, играющими заметную роль не только в теории алгоритмов.

Пусть **A** = { $\boldsymbol{a}_1$, $\boldsymbol{a}_2$, ... , $\boldsymbol{a}_p$ } - конечный алфавит.

Пусть ***u*** = a_{i1} ...a_{ik} и ***v*** = a_{j1}...a_{jl} - слова из **A***. Говорят, что слово ***w*** = a_{i1}... a_{ik} a_{j1}...a_{jl} получено конкатенацией (сцеплением) слов ***u*** и ***v*** (обозначается ***uv***),а также, что слово ***w*** является конкатенацией этих слов. Очевидным образом конкатенация обобщается на случай, когда ***u*** или ***v*** оказываются пустыми словами. Операция конкатенации ассоциативна. Следовательно, однозначно определяется конкатенация

uvw = (uv)w = u(vw) и конкатенация $\boldsymbol{u}_1\boldsymbol{u}_2...\boldsymbol{u}_m$.

Говорят, что имеет место вхождение слова ***v*** в слово ***w***, если ***w*** = $\boldsymbol{u}_1\boldsymbol{v}\boldsymbol{u}_2$. При этом слова $\boldsymbol{u}_1$ и $\boldsymbol{u}_2$ называются, соответственно, левым и правым контекстом вхождения. Вхождения слова ***v*** в ***w*** могут различаться контекстами. Среди таких вхождений вхождение с минимальной длиной левого контекста называется левым или первым вхождением ***v*** в ***w***.

Замена в слове ***w*** = $\boldsymbol{u}_1\boldsymbol{v}\boldsymbol{u}_2$ слова ***v*** на слово ***v′***, т.е. получение слова ***w′*** = $u_1\boldsymbol{v}'\boldsymbol{u}_2$, называется подстановкой, определяемой парой (***v, v′***). Пара (***v, v″***), определяющая

операцию подстановки называется формулой или правилом подстановки и для наглядности часто изображается в виде:

$$v \rightarrow v'$$

Как v, так и v', могут быть пустыми словами. Формула подстановки $v \rightarrow v'$ применима к слову w, если имеет место вхождение слова v в w. Подстановка с пустым левым словом всегда применима.

Алгоритм, названный А.А.Марковым ***нормальным***, сводится к ряду подстановок, определяемых заданием конечной последовательности определяющих формул подстановки

$$v_1 \rightarrow v'_1$$
$$\ldots\ldots$$
$$v_m \rightarrow v'_m ,$$

некоторые из них заранее выделены, как играющие особую роль в завершении алгоритма. Выделенные формулы подстановки называются заключительными и обычно отмечаются дополнительным значком при стрелке $\rightarrow$, например: $\mid\!\rightarrow$ или $^{*}\!\rightarrow$.

В нормальных алгоритмах всегда идет речь о подстановках, связанных с заменой ***первого вхождения*** левого слова формулы подстановки. В процессе выполнения нормального алгоритма исходное слово преобразуется применением заданных формул подстановки. На каждом шаге исполнения нормального алгоритма либо осуществляется одна из заданных подстановок (а именно - первая применимая подстановка), либо алгоритм завершается, если ни одна из этих подстановок неприменима. Алгоритм завершается и в случае выполнения на очередном шаге одной из заключительных подстановок. Слово, полученное в результате последней из выполненных подстановок, считается результатом работы алгоритма.

Описанные алгоритмы, в которых, как исходные слова, так и слова в подстановках, принадлежат **A***, называются нормальными алгоритмами ***в*** алфавите **A**. А. А. Марков предложил рассматривать более широкий класс алгоритмов, допустив для задания пар, определяющих подстановки, использование алфавитов, являющихся расширением алфавита **A**, употребляемого для задания исходных слов. Такие алгоритмы А. А. Марков назвал нормальными алгоритмами ***над*** алфавитом ***A***. Слова, получаемые на отдельных шагах таких алгоритмов, могут и не принадлежать **A***. А. А. Марковым была сформулирована гипотеза, названная им принципом нормализации, заменяющая гипотезу Тьюринга:

ПРИМЕР

Построим нормальный алгоритм, который "инвертирует" любое слово, заданное в алфавите **A** = { ***a, b*** }, т.е. заменяет в заданном слове все символы ***a*** на символы ***b*** и все ***b*** на ***a***.

$$+a \rightarrow b+$$
$$+b \rightarrow a+$$
$$+ \mid\!\rightarrow$$
$$\rightarrow +$$

Работа алгоритма начинается с приписывания к исходному слову слева дополнительного символа +. Такое приписывание обеспечивается правилом подстановки, которое применимо к любому слову. Но поскольку оно стоит последним, то применяется только один раз, когда другие правила еще не применимы. Далее символ + "перегоняется" в конец слова с заменой стоящей за ним буквы на "противоположную". Когда +

оказывается в конце слова, применимо третье правило подстановки, которое "стирает" символ + и прекращает выполнение алгоритма.

Любой алгоритм, отображающий слова из **A***, где **A** - конечный алфавит, эквивалентен нормальному алгоритму ***над*** алфавитом **A**. Как и гипотеза Тьюринга, принцип нормализации А.А.Маркова недоказуем. Как гипотеза Тьюринга, так и принцип нормализации, являются формальными определениями алгоритма, и важным обстоятельством для их обоснования является эквивалентность этих определений. Для установления их эквивалентности достаточно доказать:

I) *Алгоритм, выполняемый каждой машиной Тьюринга, эквивалентен некоторому нормальному алгоритму над внешним алфавитом машины*.

II) *Каждый нормальный алгоритм над алфавитом **A** эквивалентен алгоритму, выполняемому некоторой машиной Тьюринга*

Доказательство утверждения I) оказывается совсем простым. Нормальный алгоритм, эквивалентный заданной машине Тьюринга с внешним алфавитом $\mathbf{A} = \{ \boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_p \}$ и внутренним алфавитом $\mathbf{Q} = \{q_0,\ldots q_n, q_s\}$, задается подстановками, определяемыми следующим образом. Для каждой клетки $(q_i, \boldsymbol{a}_j)$ таблицы переходов, в которой записана тройка $(\boldsymbol{a}_j', R, q_i')$, т.е. предписывается переход управляющей головки к рассмотрению следующей (правой) ячейки ленты, вводим подстановку $q_i \boldsymbol{a}_j \rightarrow \boldsymbol{a}_j' q_i'$, если $q_i' \neq q_s$, или заключительную подстановку $q_i \boldsymbol{a}_j \mid\rightarrow \boldsymbol{a}_j'$, если $q_i' = q_s$. Если же в клетке $(q_i, \boldsymbol{a}_j)$ записана тройка $(\boldsymbol{a}_j', L, q_i')$, то управляющая головка должна перейти к рассмотрению предыдущей (левой) ячейки ленты. В этом случае для всех $\boldsymbol{a}_k \in \mathbf{A}$ при $q_i' \neq q_s$ вводим группу подстановок $\boldsymbol{a}_k q_i \boldsymbol{a}_j \rightarrow q_i' \boldsymbol{a}_k \boldsymbol{a}_j'$, либо при $q_i' = q_s$ - группу заключительных подстановок $\boldsymbol{a}_k q_i \boldsymbol{a}_j \mid\rightarrow \boldsymbol{a}_k \boldsymbol{a}_j'$. К выписанным подстановкам добавим подстановку $\rightarrow q_0$, поместив ее в конце всех остальных, упорядочение которых произвольно.

Легко видеть, что нормальный алгоритм, определенный введенными подстановками, начинается с приписывания перед исходным словом символа q_0.Такое приписывание обеспечивается формулой подстановки $\rightarrow q_0$. Далее выполняется нормальный алгоритм, который полностью соответствует работе заданной машины Тьюринга и завершается тем же результатом.

Доказательство положения II требует преодоления некоторых технических трудностей. Это связано с тем, что операции подстановки, выполняемые в нормальном алгоритме, существенно сложнее элементарных шагов машины Тьюринга и каждый шаг нормального алгоритма требует порой сложной серии различных тактов машины Тьюринга. Для каждого нормального алгоритма, заданного конечной последовательностью подстановок

$$\boldsymbol{v}_i \rightarrow \boldsymbol{v}_i', \; i = 1,\ldots,n,$$

мы построим машину Тьюринга, выполняющую алгоритм, эквивалентный заданному. Через a(i,j) будем обозначать символы, входящие в слова $\boldsymbol{v}_i$ - левые части подстановок $\boldsymbol{v}_i \rightarrow \boldsymbol{v}_i'$, определяющих нормальный алгоритм. Пусть $\boldsymbol{v}_i = a(i,1)\, a(i,2) \ldots a(i,m_i)$,
где $i = 1,2,\ldots,n-1$ и $m_i = |\boldsymbol{v}_i|$. Заметим, что для $i < n$ можно считать, что $\boldsymbol{v}_i \neq \boldsymbol{\lambda}$, так как подстановки, следующие за подстановкой с пустым словом в ее левой части, не имеют никакого значения в выполнении нормального алгоритма и могут быть отброшены. Если $\boldsymbol{v}_i \neq \boldsymbol{\lambda}$, то считаем, что $\boldsymbol{v}_n = a(n,1)\ldots a(n,m_n)$. Через b(i,j) будем обозначать символы, входящие в слова $\boldsymbol{v}_i'$ - правые части подстановок, определяющих нормальный алгоритм. При этом $\boldsymbol{v}_i' = b(i,1)\, b(i,2) \ldots b(i,n_i)$, $i = 1,2,\ldots,n$. Если $n_i = 0$, то в правой части i-той формулы подстановки - пустая цепочка ($\boldsymbol{v}_i' = \boldsymbol{\lambda}$).

Соотношение длин цепочек $\boldsymbol{v}_i$ и $\boldsymbol{v}_i'$ в соответствующей подстановке задается формулой $n_i = m_i + d_i$, где d_i может принимать положительные, отрицательные, и нулевые значения. Рассмотрим далее вариант реализации подстановок, для которых $d_i >= 0$. При

реализации подстановок, правая часть которых "короче" левой ($d_i < 0$), не возникает новых принципиальных трудностей. Предлагаем рассмотреть этот вариант самостоятельно.

Для определения вхождения левых частей подстановок, определяющих заданный нормальный алгоритм, в управляющей головке соответствующей машины Тьюринга предусмотрим следующие состояния:

1). q(i,j) для j =1,..,m_i и i = 1,2,...,n-1, и i = n, если $m_n = 0$

Эти состояния будут соответствовать проверке наличия символа a(i,j) в обозреваемой головкой ячейке ленты машины Тьюринга. Состояние q(1,1) отождествим с начальным состоянием q_0.

2). q(i,j)' и q(i)' - состояния, обеспечивающие возврат управляющей головки для продолжения поиска вхождения или его завершения

3). $q(i,m_i+1) = p(i,d_i)$, i= 1,2,...,n - состояния, соответствующие установлению вхождения ν_i и переходу к реализации i-ой подстановки

4). $q(n+1,1) = q_s$ - заключительное состояние машины Тьюринга

Через I^z обозначим множество номеров подстановок, которые нумеруют заключительные подстановки нормального алгоритма, а q^z - соответствующие таким подстановкам состояния. Если машина оказывается в состоянии q^z, то нужно перевести управляющую головку в начало слова и закончить работу, т.е. перейти в состояние останова q_s.

Для описания таблицы переходов конструируемой машины Тьюринга T^N мы будем описывать клетки таблицы соотношениями вида :

[q, a] **:** [a', X, q'],

где слева в квадратных скобках обозначена клетка таблицы переходов символами ее строки q и столбца a, справа же -.содержимое этой клетки. Здесь a' - записываемый в обозреваемую ячейку символ, X - один из символов указателей R и L, перехода головки к рассмотрению следующей ячейки ленты, а q' - состояние, в которое переходит управляющая головка. Клетки таблицы переходов, обеспечивающие обнаружение соответствующих вхождений, зададим следующими формулами

1. [q(i,j), a(i,j)] : [a(i,j), R, q(i,j+1)]

Осуществляется поиск вхождения левой цепочки i-ой подстановки, j-ая и все предыдущие буквы цепочки ν_i найдены - отождествляем j+1 - ю букву

2. [q(i,1), a] : [a, R, q(i,1)], где a- буква, $a \neq \Lambda$ и $a \neq a(i,1)$

Поиск вхождения первой буквы цепочки ν_i, идем вправо, пока не встретим первую букву

3. [q(i,j), a] : [a, L, q(i,j-1)'], где j>1 и $a \neq \Lambda$ и $a \neq a(i,1)$

При поиске встретили букву, отличную от нужной a(i,j), - возвращаемся влево

4. [q(i,j), Λ] : [Λ, L, q(i)']

Если дошли до конца слова и не нашли вхождение левой цепочки i-ой подстановки, то идем влево по слову

5. [q(i,1)', a] : [a, R, q(i,1)]

Ищем начало вхождения ν_i, начиная со следующей буквы слова

6. [q(i,j)', a] : [a, L, q(i,j-1)']

Возврат влево при неудачном поиске

7. [q(i)', a] : [a, L, q(i)']

Дойдя до конца слова, не нашли ν_i - идем влево к первой букве слова

8. [q(i)', Λ] : [Λ, R, q(i+1,1)]

Ищем вхождение следующей (i+1)- ой подстановки.

После того, как найдено вхождение в заданное слово левой цепочки (.ν_i.) подстановки ($\nu_i \rightarrow \nu_i'$) из формул подстановок, задающих нормальный алгоритм, нужно реализовать соответствующую подстановку, т.е. заменить в исходном слове цепочку ν_i на цепочку ν_i'. Реализация такой замены происходит после того, как машина оказывается в состоянии $q(i,m_i+1) = p(i, d_i)$, и ее последующая работа осуществляется в соответствии со следующими действиями, записанными в соответствующие клетки таблицы переходов машины :

9. [p(i,j), a] : [Λ, R, p(i,j,a)] для всех i
10. [p(i,j,a), b] : [a, R, p(i,j,b)] для символов $b \neq \Lambda$
11. [p(i,j,a), Λ] : [a, L, p(i,j-1)']
12. [p(i,j)', a] : [a, L, p(i,j)']
13. [p(i,j)', Λ] : [Λ, R, p(i,j)']
14. [p(i,0), a] :[a, L, r(i,n_i)]
15. [r(i,j), a] : [b(i,j), L, r(i,j-1)] для $j > 0$

При этом $r(i,0) = q^z$, если $i \in I^z$ и
$r(i,0) = q$, если $i \notin I^z$

16. [$r(q^z)$, a] : [a, L, $r(q^z)$]
17. [$r(q^z)$, Λ] : [Λ, R, q_s]
18. [q, a] : [a, L, q]
19. [q, Λ] : [Λ, R, q_0] , где $q_0 = q(1,1)$

6. САМОПРИМЕНИМОСТЬ И ПРОБЛЕМА ПРИМЕНИМОСТИ

Таблицы переходов машин Тьюринга кроме символов внешнего алфавита содержат также символы протяжки ленты и символы состояний управляющей головки. Однако, не составляет большого труда представить таблицу переходов каждой машины некоторым словом внешнего алфавита этой машины, если алфавит содержит по крайней мере два символа. Тот или иной способ такого представления не имеет принципиального значения. Но, учитывая дальнейшее изложение, касающееся универсальной машины Тьюринга, мы рассмотрим конкретный способ изображения таблиц переходов. При этом двухмерную таблицу переходов представим в виде линейного слова, которое можно записать на ленту машины Тьюринга.

Расширим внешний алфавит двумя символами, при помощи которых будем разделять цепочки символов, задающие строки и клетки таблиц переходов, и изображать заданные в клетках состояния управляющей головки. Обозначим эти два символа через & и @. Занумеруем символы состояний управляющей головки данной машины Тьюринга так, чтобы номером 1 оказался символ начального состояния, а последним - символ заключительного. Последовательностью @@...@ будем изображать символ состояния с номером, равным длине такой последовательности. Включив в изображение клетки таблицы переходов указание на столбец, которому принадлежит клетка, и признак, позволяющий легко обнаружить, является или нет состояние, к которому предписывает перейти клетка, заключительным, представим каждую клетку цепочкой

@@...@ & S Z $a_j a_i$ & ,

где @@...@ - изображение состояния, в которое переходит управляющая головка в такте, соответствующем изображаемой клетке;

S - & или @ - символы, кодирующие сигналы R и L протяжки ленты;

Z - & или @ - символы, кодирующие признак заключительного состояния;

a_j - символ, записываемый на ленту в соответствующем такте работы машины;

a_i - символ столбца таблицы переходов, которому принадлежит изображаемая клетка.

Изображение клетки завершается символом &, отмечающим конец "содержимого" клетки.

Каждую строку таблицы переходов представим конкатенацией символа & и последовательности цепочек, изображающих составляющие строку клетки. И, наконец, всю таблицу переходов представим конкатенацией всех цепочек, изображающих строки, и цепочки &&, отмечающей конец изображения. В этой конкатенации цепочки, изображающие строки, берутся в порядке возрастания номеров соответствующих состояний. Введенное представление (изображение) таблицы переходов машины Тьюринга можно использовать для решения следующей задачи.

Если машина Тьюринга применима к полученному таким образом слову, изображающему ее таблицу переходов, то мы будем говорить, что эта машина самоприменима или что самоприменим алгоритм, реализуемый этой машиной. Если же конкретная машина не применима к рассматриваемому слову, то мы будем говорить о несамоприменимости машины или о несамоприменимости алгоритма. Само по себе понятие самоприменимости выглядит весьма экзотически. Однако оно оказывается простым средством для строгого доказательства интересных и важных утверждений об алгоритмической неразрешимости некоторых проблем, касающихся алгоритмов.
Докажем прежде всего следующую теорему:

Теорема 1:

Не существует алгоритма, определяющего по изображению произвольного алгоритма, самоприменим он или несамоприменим.

Доказательство

Предположим, что такой алгоритм существует. Без потери общности можно считать его алгоритмом вычисления предиката, который по изображению любого алгоритма *A* вычисляет значение **истина** (да), если *A* самоприменим и **ложь** (нет), если *A* не самоприменим.

Рассмотрим алгоритм *B*, реализующий альтернативу :

если "алгоритм *A* самоприменим" **то** "бесконечный цикл" **иначе** СТОП. Здесь *A* - произвольный алгоритм. Вспомним, что выше было показано существование машины Тьюринга, реализующей альтернативу, при условии существования машины Тьюринга, вычисляющей значение предиката истинности условия.

Алгоритм *B* должен быть либо самоприменим, либо не самоприменим. Однако каждое из этих предположений оказывается ложным. Действительно, применение алгоритма *B* к своему собственному изображению завершается только в том случае, когда алгоритм *B* несамоприменим, что противоречит определению самоприменимости. Если же алгоритм *B* самоприменим, то применение его к собственному изображению приводит к бесконечному циклу, что противоречит предположению о самоприменимости. Полученное противоречие доказывает теорему, из которой следует Теорема 2.

Теорема 2:

Не существует алгоритма, определяющего применимость произвольного алгоритма к произвольному слову.

Доказательство

Существование такого алгоритма противоречило бы теореме 1, поскольку изображение алгоритма можно считать одним из возможных слов. И в соответствии с теоремой 1 не существует алгоритма, определяющего применимость любого алгоритма к его изображению в виде слова. Теорема 2 подтверждает невозможность в общем виде заменить зацикливание алгоритмов аварийными завершениями.

7. УНИВЕРСАЛЬНАЯ МАШИНА ТЬЮРИНГА

Как уже было отмечено выше, работа каждой машины Тьюринга сводится к процессу выполнения весьма простого алгоритма U, использующего исходное слово ***w*** и соответствующую таблицу переходов.

Рассматривая понятие самоприменимости, мы познакомились с конкретным способом представления таблицы переходов машины Тьюринга словом внешнего алфавита. Используя такое представление, мы можем свести исходные данные алгоритма U к паре слов - слову ***m***, представляющему таблицу переходов соответствующей машины Тьюринга, и слову ***w*** - исходному слову для работы этой машины. Дополнив внешний алфавит специальным разделяющим символом, например, *, мы можем считать, что исходным словом алгоритма U является слово ***m*w***, а словом - результатом : ***m****T(***w***), если машина T применима к слову ***w***. Если же машина T не применима к слову ***w***, то алгоритм U не завершается при исходном слове ***m*w***.

В силу гипотезы Тьюринга существует машина Тьюринга T^U, эквивалентная алгоритму U, т.е. машина, способная выполнить работу любой машины Тьюринга над любым заданным словом или, как говорят, способная моделировать любую машину Тьюринга. Машина T^U называется универсальной машиной Тьюринга. Существование такой машины вытекает из гипотезы Тьюринга. Конкретное же построение такой машины может рассматриваться как дополнительный аргумент в пользу принятия гипотезы Тьюринга.

Вместо полного описания таблицы переходов универсальной машины Тьюринга мы ограничимся здесь общей схемой работы такой машины.

Пусть на ленте машины записано слово @***m*w***, а управляющая головка обозревает первый символ этого слова, находясь в начальном состоянии. Далее выполняются следующие этапы:

1. Управляющая головка просматривает слово @ ***m*w*** и обнаруживает символ *, переходя в новое состояние к первому символу слова ***w***.

2. Обозрев ячейку, управляющая головка "запоминает" записанный в ячейке символ, переходя в соответствующее состояние, записывает в ячейку ленты символ * и "отправляется" налево до пустой ячейки, где переходит затем к правой соседней ячейке.

Возникшая ситуация будет повторяться в процессе работы универсальной машины Тьюринга и соответствует началу поиска алгоритмом U клетки в таблице переходов, соответствующей состоянию и обнаруженному символу. Напомним, что обнаруженный символ отражен в состоянии управляющей головки универсальной машины, а состояние управляющей головки моделируемой машины отражено цепочкой из символов @, предшествующей слову ***m***. На первый раз эта цепочка состоит из одного символа @, что соответствует начальному состоянию. В дальнейшем эта цепочка оказывается такой, какая возникает в результате выполнения этапа 4.

3. В слове ***m*** ищется соответствующая строка.

4. В найденной строке ищется клетка, помеченная запомненным на шаге 2 символом. Запоминаются символ, сигнал протяжки и признак заключительного состояния, указанные в этой клетке. А также переносится цепочка @@...@, изображающая новое состояние, так чтобы ее копия оказалась на ленте непосредственно перед словом ***m*** .

5. Управляющая головка отправляется к символу *, стоящему следом за словом ***m*** , а затем ищет следующую звездочку, на место которой записывает символ, запомненный из найденной на этапе 4 клетки таблицы переходов ***m***. Затем переходит к соседней клетке в соответствии с запомненным тогда же сигналом протяжки.

6. Если в ячейке, оказавшейся перед управляющей головкой, записан символ *, то цепочка @@...@ ***m*** * (последний символ этой цепочки как раз и оказался перед управляющей головкой) переписывается на одну ячейку левее. Управляющая головка возвращается к ячейке, перед которой она находилась перед переписью, и в эту ячейку записывается Λ (символ пустоты)

7. Если состояние, в которое перешла моделируемая машина отлично от ее заключительного состояния, то переходим к этапу 2.

8. Стоп.

Существование универсальной машины Тьюринга непосредственно вытекает из гипотезы Тьюринга. Однако, как видно из приведенной нами схемы работы, построение такой машины может быть осуществлено непосредственно - без привлечения этой гипотезы. Такое построение можно рассматривать как дополнительное обоснование истинности гипотезы Тьюринга.

ЛИТЕРАТУРА

1. А.А.Марков, Н.М.Нагорный. Теория алгорифмов. - М., ФАЗИС, 1996.
2. Б.А.Трахтенброт. Алгоритмы и вычислительные автоматы. - М., Сов. Радио, 1974.
3. Э.З.Любимский, В.В.Мартынюк, Н.П.Трифонов. Программирование. - М., Наука, 1980.
4. А.И.Мальцев. Алгоритмы и рекурсивные функции. - М., Наука, 1986.

СОДЕРЖАНИЕ