## §7. ФОРМАЛИЗАЦИЯ ПОНЯТИЯ «АЛГОРИТМ»

### 7.1. ПОСТАНОВКА ПРОБЛЕМЫ

Понятие алгоритма, введенное в предыдущем параграфе, можно назвать понятием алгоритма в интуитивном смысле. Оно имеет нечеткий, неформальный характер, ссылается на некоторые точно не определенные, но интуитивно понятные вещи. Например, при определении и обсуждении свойств алгоритма мы исходили из возможностей некоторого исполнителя алгоритма. Его наличие предполагалось, но ничего определенного о нем не было известно. Говоря языком математики, ни аксиоматического, ни исчерпывающего конструктивного определения исполнителя мы так и не дали.

Установленные в предыдущем параграфе свойства алгоритмов следует называть эмпирическими. Они выявлены на основе обобщения свойств алгоритмов различной природы и имеют прикладной характер. Этих свойств достаточно для практического программирования, для создания обширного круга программ для компьютеров, станков с ЧПУ, промышленных роботов и т.д. Однако, как фундаментальное научное понятие алгоритм требует более обстоятельного изучения. Оно невозможно без уточнения понятия «алгоритм», более строгого его описания или, как еще говорят, без его **формализации.**

Известно несколько подходов к формализации понятия «алгоритм»:

• теория конечных и бесконечных автоматов;

• теория вычислимых (рекурсивных) функций;

• λ-исчисление Черча.

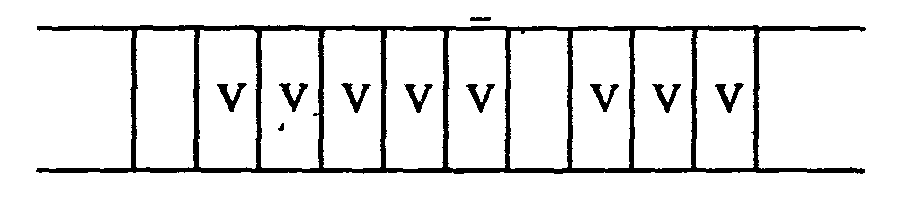
Все эти возникшие исторически независимо друг от друга подходы оказались впоследствии эквивалентными. Главная цель формализации понятия алгоритма такова: подойти к решению проблемы алгоритмической разрешимости различных математических задач, т.е. ответить на вопрос, может ли быть построен алгоритм, приводящий к решению задачи. Мы рассмотрим постановку этой проблемы и некоторые результаты теории алгоритмической разрешимости задач, но вначале обсудим формализацию понятия алгоритма в теории автоматов на примере машин Поста, Тьюринга, а также нормальных алгоритмов Маркова, а затем - основы теории рекурсивных функций. Идеи λ-исчислений Черча реализованы в языке программирования LISP (глава 3).

Вместе с тем, формально определенный любым из известных способов алгоритм не может в практическом программировании заменить то, что мы называли алгоритмами в предыдущем параграфе. Основная причина состоит в том, что формальное определение резко сужает круг рассматриваемых задач, делая многие практически важные задачи недоступными для рассмотрения.

### 7.2. МАШИНА ПОСТА

Абстрактные (т.е. существующие не реально, а лишь в воображении) машины Поста и Тьюринга, предназначенные для доказательств различных утверждений о свойствах программ для них, были предложены независимо друг от друга (и практически одновременно) в 1936 г. американским математиком Эмилем Постом и английским математиком Алланом Тьюрингом. Эти машины представляют собой универсальных исполнителей, являющихся полностью детерминированными, позволяющих «вводить» начальные данные, и после выполнения программ «читать» результат. Машина Поста менее популярна, хотя она значительно проще машины Тьюринга. С ее помощью можно вести обучение первым навыкам составления программ для ЭВМ.

Абстрактная машина Поста представляет собой бесконечную ленту, разделенную на одинаковые клетки, каждая из которых может быть либо пустой, либо заполненной меткой «V», и головки, которая может перемещаться вдоль ленты на одну клетку вправо или влево, наносить в клетку ленты метку, если этой метки там ранее не было, стирать метку, если она была, или проверять наличие в клетке метки. Информация о заполненных метками клетках ленты характеризует состояние ленты, которое может меняться в процессе работы машины. В каждый момент времени головка («-») находится над одной из клеток ленты и, как говорят, обозревает ее. Информация о местоположения головки вместе с состоянием ленты характеризует состояние машины Поста, рис. 1.16.



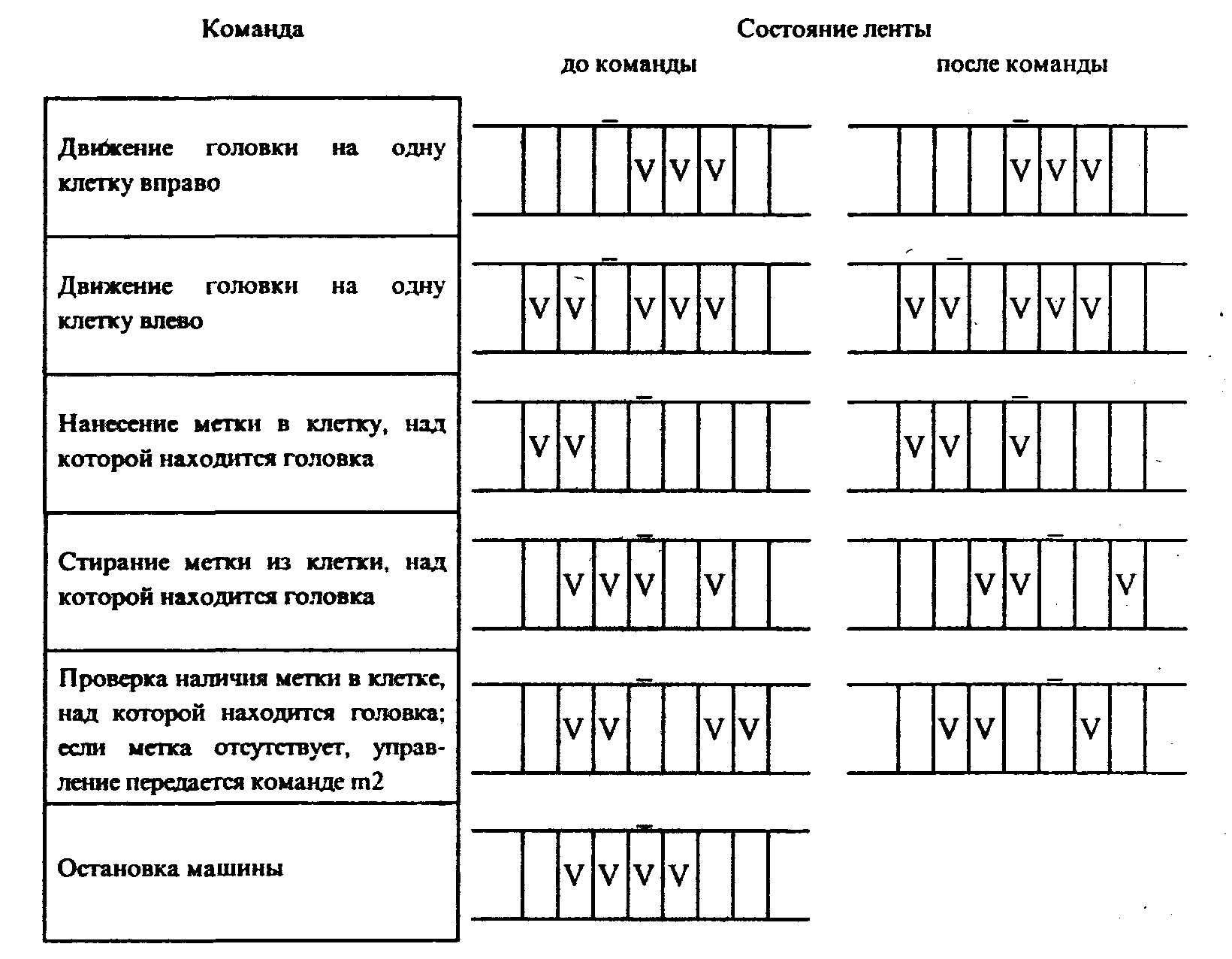
*Рис. 1.16.* Абстрактная машина Поста

Команда машины Поста имеет следующую структуру:

*п Km,*

где *п -* порядковый номер команды, *K*-действие, выполняемое головкой, *т -* номер следующей команды, подлежащей выполнению.

Существует всего шесть команд машины Поста, рис. 1.17:



*Рис. 1.17.* Команды машины Поста

Ситуации, в которых головка должна наносить метку там, где она уже имеется, или, наоборот, стирать метку там, где ее нет, являются аварийными (недопустимыми).

Программой для машины Поста будем называть непустой список команд, такой что 1) на *п-*м месте команда с номером *n*; 2) номер *т* каждой команды совпадает с номером какой-либо команды списка.

С точки зрения свойств алгоритмов, изучаемых с помощью машины Поста, наибольший интерес представляют причины останова машины при выполнении программы:

1) останов по команде «стоп»; такой останов называется результативным и указывает на корректность алгоритма (программы);

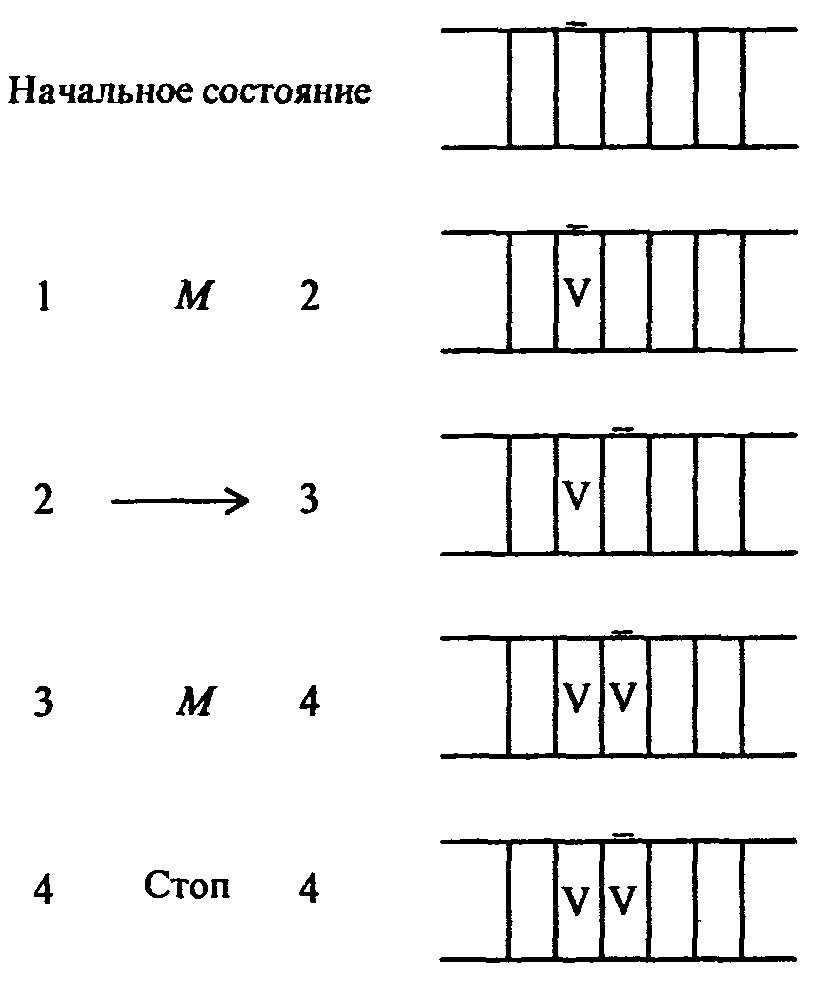
2) останов при выполнении недопустимой команды; в этом случае останов называется безрезультативным;

3) машина не останавливается никогда; в этом и в предыдущем случае мы имеем дело с некорректным алгоритмом (программой).

Будем понимать под начальным состояние головки против пустой клетки левее самой левой метки на ленте.

Рассмотрим реализацию некоторых типичных элементов программ машины Поста.

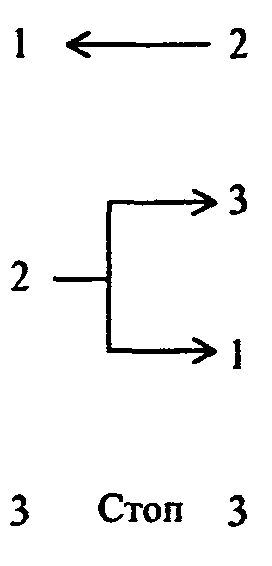
1. Пусть задано исходное состояние головки и требуется на пустой ленте написать две метки: одну в секцию под головкой, вторую справа от нее. Это можно сделать по следующей программе (справа от команды показан результат ее выполнения):



*Рис. 1.18.* Пример элемента программы машины Поста

2. Покажем, как можно воспользоваться командой условного перехода для организации циклического процесса. Пусть на ленте имеется запись из нескольких меток подряд и головка находится над самой крайней меткой справа. Требуется перевести головку влево до первой пустой позиции.

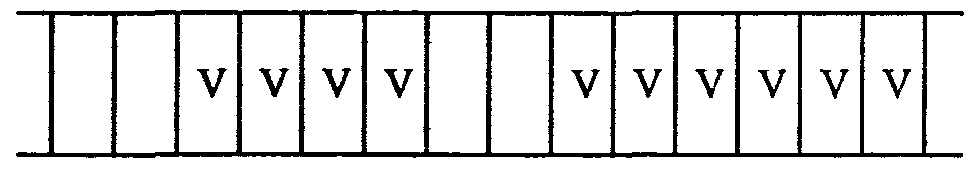
Программа будет иметь следующий вид:



Команда условного перехода является одним из основных средств организации циклических процессов, например, для нахождения первой метки справа (или слева) от головки, расположенной над пустой клеткой; нахождение слева (или справа) от головки пустой клетки, если она расположена над меткой и т.д.

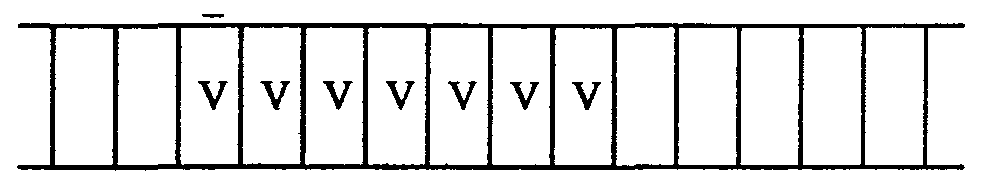
3. Остановимся на представлении чисел на ленте машины Поста и выполнении операций над ними.

Число *k* представляется на ленте машины Поста идущими подряд *k* + 1 метками (одна метка означает число «О»). Между двумя числами делается интервал как минимум из одной пустой секции на ленте. Например, запись чисел 3 и 5 на ленте машины Поста будет выглядеть так:



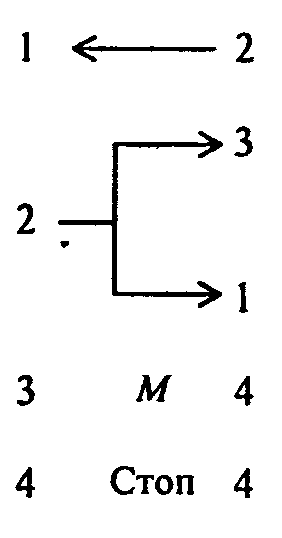
Обратим внимание, что используемая в машине Поста система записи чисел является непозиционной.

Составим программу для прибавления к произвольному числу единицы. Предположим, что на ленте записано только одно число и головка находится над одной из клеток, в которой находится метка, принадлежащая этому числу:

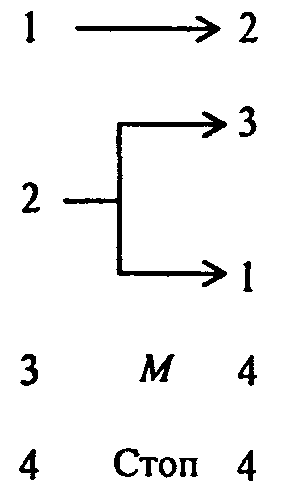


Для решения задачи можно переместить головку влево (или вправо) до первой пустой клетки, а затем нанести метку.

Программа, добавляющая к числу метку слева, имеет вид:

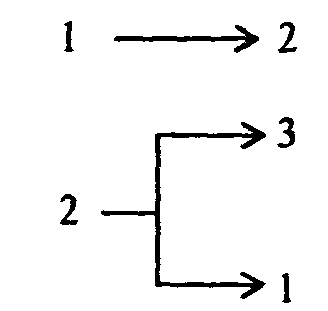


Программа, добавляющая к числу метку справа, имеет вид:

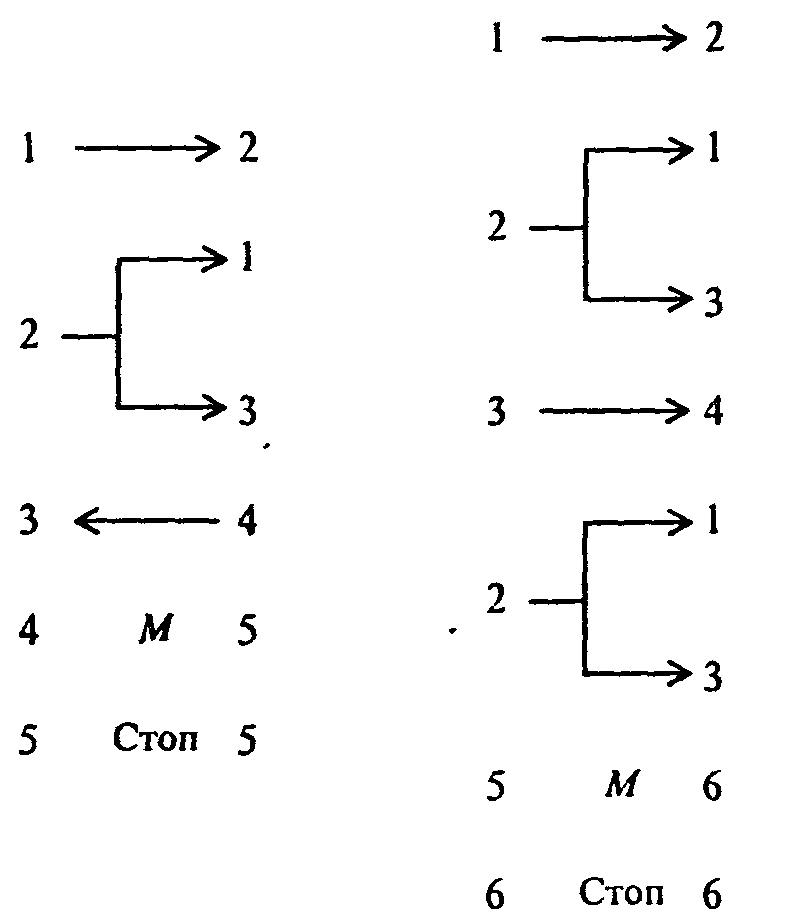


(отличие только в направлении движения головки в первой команде. Проверьте работоспособность этих программ на каких-либо частных примерах).

Предположим, что головка расположена на расстоянии нескольких клеток слева от числа, к которому нужно прибавить единицу. В этом случае программа усложняется. Появится «блок поиска числа» - две команды, приводящие головку в состояние, рассмотренное в предыдущем примере:

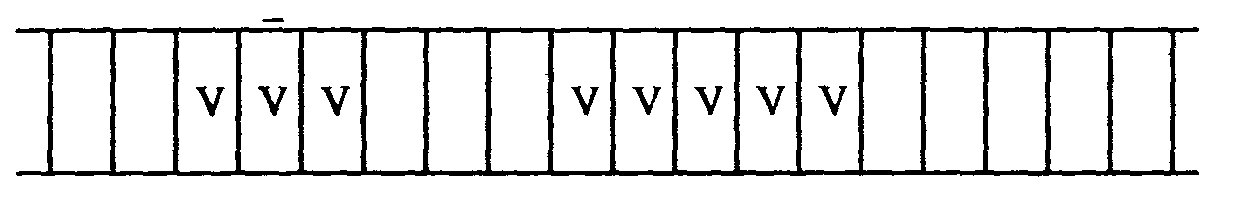


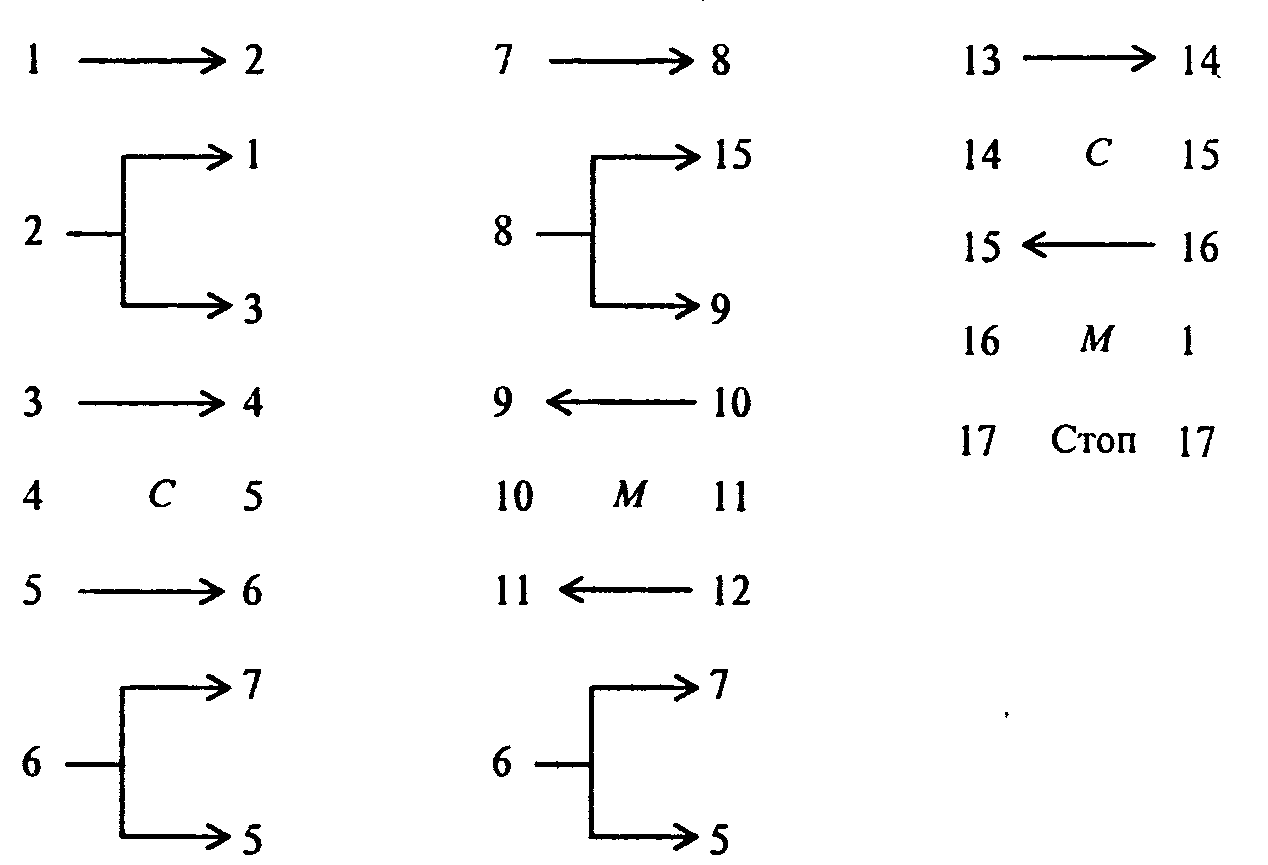
Ниже - полные тексты программ, добавляющие единицу слева и справа, соответственно:



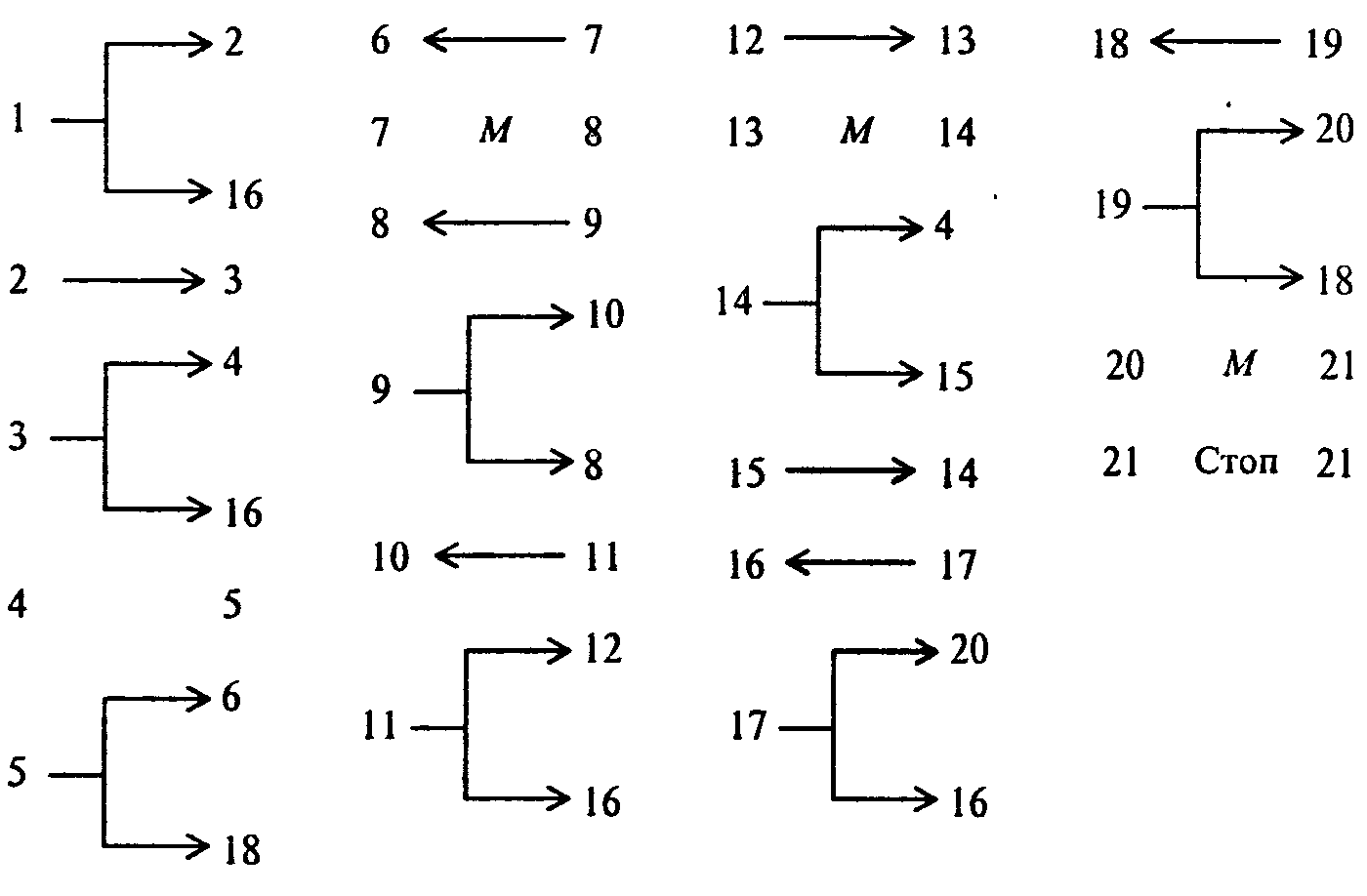
В первом случае не нужно перемещать головку к крайней левой метке числа

4. Приведем программу для сложения целых неотрицательных чисел а и и на машине Поста, когда головка находится над числом *а,* а число *b* находится правее числа *а* на некоторое число клеток. Эта программа реализует следующий алгоритм: первое число постепенно придвигается ко второму до их слияния, а потом стирается одна метка (иначе результат оказался бы на единицу больше правильного).





В случае более сложных начальных условий, когда неизвестно, справа или слева от головки (и на какое число клеток) находится число, можно применить такой принцип поиска числа: двигая головку вправо и влево и отмечая метками степень удаления головки от исходного положения, найти число, а потом уже применить известную программу сложения. При этом проверяется, находится ли головка над одной из меток числа и если да, то задача решена. Иначе проверяется, пуста ли секция справа от головки и следующая за ней; если обе пусты, то делается возврат головки на один шаг и ставится метка, а затем такая же операция выполняется слева и по отмеченной дорожке головка возвращается вправо и т.д. до тех пор, пока головка не натолкнется на число, после чего можно применить ранее рассмотренные выше программы:



Машину Поста можно рассматривать как упрощенную модель ЭВМ. В самом деле, как ЭВМ, так и машина Поста имеют:

• неделимые носители информации (клетки - биты), которые могут быть заполненными или незаполненными;

• ограниченный набор элементарных действий - команд, каждая из которых

выполняется за один такт (шаг).

Обе машины работают на основе программы. Однако, в машине Поста информация располагается линейно и читается подряд, а в ЭВМ можно читать информацию по адресу; набор команд ЭВМ значительно шире и выразительнее, чем команды машины Поста и т.д.

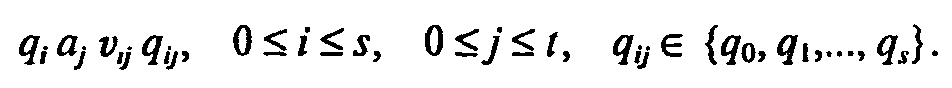
### 73. МАШИНА ТЬЮРИНГА

Машина Тьюринга подобна машине Поста, но функционирует несколько иначе.

Машина Тьюринга (МТ) состоит из счетной ленты (разделенной на ячейки и ограниченной слева, но не справа), читающей и пишущей головки, лентопротяжного механизма и операционного исполнительного устройства, которое может находиться в одном из дискретных состояний *qo, q1, ..., qs,* принадлежащих некоторой конечной совокупности (алфавиту внутренних состояний). При этом *q*о называется начальным состоянием.

Читающая и пишущая головка может читать буквы рабочего алфавита *А = [а0, a1*, ..., *аt},* стирать их и печатать. Каждая ячейка ленты в каждый момент времени занята буквой из множества *А.* Чаще всего встречается буква *a*0 - «пробел». Головка находится в каждый момент времени над некоторой ячейкой ленты -текущей рабочей ячейкой. Лентопротяжный механизм может перемещать ленту так, что головка оказывается над соседней ячейкой ленты. При этом возможна ситуация выхода за левый край ленты (ЛК), которая является аварийной (недопустимой), или машинного останова (МО), когда машина выполняет предписание об остановке.

Порядок работы МТ (с рабочим алфавитом *a0,* *a1*,..., *аt* и состояниями *q0*, *q1,..., qs)* описывается таблицей машины Тьюринга. Эта таблица является матрицей с четырьмя столбцами и (*s* + 1) (*t* + 1) строками. Каждая строка имеет вид



Здесь через *v*ij обозначен элемент объединения алфавита *{а0, а1,..., аt}* и множества предписаний для лентопротяжного механизма: *l* - переместить ленту влево, *r -*переместить ленту вправо, *s -* остановить машину; *v*ij - действие МТ, состоящее либо в занесении в ячейку ленты символа алфавита *a0*, *а1, ..., аt,* либо в движении головки, либо в останове машины; *qij* является последующим состоянием.

МТ работает согласно следующим правилам: если МТ находится в состоянии *qi,* головка прочитывает символ 0 в рабочей ячейке. Пусть строка *qi* *аj* *vij* *qij,* начинающаяся с символов *qi*, *aj*, встречается только один раз в таблице. Если *v*ij - буква рабочего алфавита, то головка стирает содержимое рабочей ячейки и заносит туда эту букву. Если *vij -* команда *r* или *l* для лентопротяжного механизма, то лента сдвигается на одну ячейку вправо или влево (если не происходит выход за левый край ленты) соответственно. Если *vij =s,* то происходит машинный останов.

Машина Тьюринга начинает работу из некоторой начальной конфигурации, включающей в себя начальное состояние (обычно *q0)* и положение считывающе-записывающей головки над определенной ячейкой ленты, содержащей один из символов рабочего алфавита A.

Отметим, что наличие разнообразных символов в рабочем алфавите МТ позволяет представлять на ленте произвольную текстовую и числовую информацию, а переходы управляющего центра МТ в различные состояния моделируют запоминание машиной Тьюринга промежуточных результатов работы. Таблица, определяющая порядок работы МТ, не является в прямом смысле слова программой (ее предписания выполняются не последовательно, одно за другим, а описывают преобразования символов некоторого текста, находящегося на ленте). Таблицу МТ часто называют схемой машины Тьюринга или попросту отождествляют с самой машиной Тьюринга, коль скоро ее устройство и принцип функционирования известны.

Рассмотрим примеры нескольких схем машины Тьюринга.

1. Алгоритм прибавления единицы к числу *п* в десятичной системе счисления. Дана десятичная запись числа *п* (т.е. представление натурального числа *п* в десятичной системе счисления); требуется получить десятичную запись числа *п* + 1.

Очевидно, что внешний алфавит МТ должен состоять из десяти цифр 0,1,2,3,4,5,6,7,8,9 и символа пробела \_. Эти цифры записывают по одной в ячейке (подряд, без пропусков).

Оказывается достаточным иметь два внутренних состояния машины: *q1* и *q2.*

Предположим, что в начальный момент головка находится над одной из цифр числа, а машина находится в состоянии *q1.* Тогда задача может быть решена в два этапа: движения головки к цифре единиц числа (во внутреннем состоянии *q1)* и замене этой цифры на единицу большую (с учетом переноса 1 в следующий разряд, если это 9, во внутреннем состоянии *q2.* Соответствующая схема МТ может иметь вид

|  |  |  |
| --- | --- | --- |
| *аi* | *qi* | |
| *q1* | *q2* |
| 0 | 0П*q1* | 1C*q2* |
| 1 | 1П*q1* | 2C*q2* |
| 2 | 2П*q1* | 3C*q2* |
| 3 | 3П*q1* | 4C*q2* |
| 4 | 4П*q1* | 5C*q2* |
| 5 | 5П*q1* | 6C*q2* |
| 6 | 6П*q1* | 7C*q2* |
| 7 | 7П*q1* | 8C*q2* |
| 8 | 8П*q1* | 9C*q2* |
| 9 | 9П*q1* | 0C*q2* |
| - | -Л*q1* | 1C*q2* |

2. Алгоритм записи числа в десятичной системе счисления.

Дана конечная последовательность меток, записанных в клетки ленты подряд, без пропусков. Требуется записать в десятичной системе число этих меток пересчитать метки).

Суть алгоритма может состоять в том, что к числу 0, записанному на ленте в начале работы машины, машина добавляет 1, стирая метку за меткой, так что вместо нуля возникает число 0 + *k.*

Легко могут быть построены алгоритмы сложения чисел, их перемножения, нападения наибольшего общего делителя и т.д. Однако, главная цель введения машин Поста и Тьюринга не программирование для них, а изучение свойств алгоритмов и проблемы алгоритмической разрешимости задач.

В зависимости от числа используемых лент, их назначения и числа состояний устройства управления можно рассматривать различные модификации машин Тьюринга.

Предположим, мы расширили определение МТ, добавив определенное состояние *q.* устройства управления машины. Будем говорить, что если устройство управления переходит в состояние *q0* для заданного входного слова *х,* то машина допускает *х;* если устройство переходит в состояние *qx*, то машина запрещает *х.* Такую машину будем называть машиной Тьюринга с двумя выходами. Могут быть рассмотрены многочисленные варианты машины Тьюринга, имеющие некоторое конечное число лент. В каждой клетке этих лент может находиться один из символов внешнего алфавита *А* = {*a0, a1*, ..., *аn}.* Устройство управления машиной в каждый момент времени находится в одном из конечного множества состояний *Q* = *{q0, q1, ..., qm}.* Для *K*-ленточной машины конфигурация ее в *i*-й момент времени описывается системой *k*-слов вида:

*ail1 … aillqi aill+1 … si1t;*

*aik1 … aiklqi aikl+1 …aikv;*

первый индекс соответствует моменту времени, второй - номеру ленты, третий -номеру клетки, считая слева направо. Говорят, что машина выполняет команду

*qiaa1 … aak → qj ab1 k1 … abk kk*,

К = {Л, С, П}.

Если, находясь в состоянии *qi* и обозревая ячейки с символами *aa1* — *aаk,* машина переходит в состояние *qj,* заменяя содержимое ячеек соответственно символами *аb1 — аbк,* то после этого ленты соответственно сдвигаются в направлениях *k1 ... kk.*

До сих пор принималось, что различные алгоритмы осуществляются на различных машинах Тьюринга, отличающихся набором команд, внутренним и внешним алфавитами. Однако, можно построить универсальную машину Тьюринга, способную выполнять любой алгоритм любой машины Тьюринга. Это достигается путем кодирования конфигурации и программы любой данной машины Тьюринга в символах внешнего алфавита универсальной машины. Само кодирование должно выполняться следующим образом:

1) различные символы должны заменяться различными кодовыми группами, но один и тот же символ должен заменяться всюду, где бы он не встретился, одной и той же кодовой группой;

2) строки кодовых записей должны однозначно разбиваться на отдельные кодовые группы;

3) должна иметься возможность распознать кодовые группы, соответствующие командам *Л, П, С,* различать кодовые группы, соответствующие символам внешнего алфавита и внутренним состояниям.

Для сравнения структур различных машин и оценки их сложности необходимо иметь соответствующую меру сложности машин. К.Шеннон предложил рассматривать в качестве такой меры произведение числа символов внешнего алфавита и числа внутренних состояний. Большой интерес вызывает задача построения универсальной машин Тьюринга наименьшей сложности.

Может быть рассмотрено еще одно обобщение машин Тьюринга: их композиции. Операции композиции, выполняемые над алгоритмами, позволяют образовывать новые, более сложные алгоритмы из ранее известных простых алгоритмов. Поскольку машина Тьюринга - алгоритм, то операции композиции применимы и к машинам Тьюринга. Рассмотрим основные из них: произведение, возведение в степень, итерацию.

Пусть заданы машины Тьюринга *T1* и *T2*, имеющие общий внешний алфавит *А* = {*a0*, *a1*, ..., *am*} и внутренние состояния *Q1 =* {*q0, q1,… qn*} и *Q2* = *{q0, q1,* ..., *qt}* соответственно. Композицией или произведением машины *T1* и машины *T2* будем называть машину *Т* с тем же внешним алфавитом *А = {a0, а1, ...,* *am*} и набором внутренних состояний *Q* = *{q0, q1,..., q2,,* *qn+1*,..., *qn+1*} и программой, эквивалентной последовательному выполнению программ машин *Т1* и *Т2:*

*Т =* *T1* \* *T2..*

Таким же образом определяется операция возведения в степень: *n*-й степенью машины *Т* называется произведение *T*...*Т* c *n* сомножителями.

Операция итерации применима к одной машине и определяется следующим образом. Пусть машина *T1* имеет несколько заключительных состояний. Выберем ее *r-е* заключительное состояние и отождествим его в схеме машины с ее начальным состоянием. Полученная машина *T* является результатом итерации машины *Т1 : Т = T1.*

Прежде чем остановиться на проблеме алгоритмической разрешимости задач обратимся к другим способам формализации понятия алгоритма.

### 7.4. НОРМАЛЬНЫЕ АЛГОРИТМЫ МАРКОВА

Для формализации понятия алгоритма российский математик А.А.Марков предложил использовать ассоциативные исчисления.

Рассмотрим некоторые понятия ассоциативного исчисления. Пусть имеется алфавит (конечный набор различных символов). Составляющие его символы будем называть буквами. Любая конечная последовательность букв алфавита (линейный их ряд) называется словом в этом алфавите.

Рассмотрим два слова *N* и *М* в некотором алфавите *А.* Если *N* является частью *М,* то говорят, что *N* входит в *М.*

Зададим в некотором алфавите конечную систему подстановок *N - М, S - Т,...,* где *N, М, S, Т,... -* слова в этом алфавите. Любую подстановку *N-M* можно применить к некоторому слову *К* следующим способом: если в *К* имеется одно или несколько вхождений слова *N,* то любое из них может быть заменено словом *М,* и, наоборот, если имеется вхождение *М,* то его можно заменить словом *N.*

Например, в алфавите *А = {а, b, с}* имеются слова *N* = *ab, М = bcb, К = abcbcbab,* Заменив в слове *К* слово *N* на *М,* получим *bcbcbcbab* или *abcbcbbcb,* и, наоборот, заменив *М* на *N,* получим *aabcbab* или *аbсаbаb.*

Подстановка *ab - bcb* недопустима к слову *bacb,* так как ни *ab,* ни *bcb* не входит в это слово. К полученным с помощью допустимых подстановок словам можно снова применить допустимые подстановки и т.д. Совокупность всех слов в данном алфавите вместе с системой допустимых подстановок называют **ассоциативным исчислением.** Чтобы задать ассоциативное исчисление, достаточно задать алфавит и систему подстановок.

Слова *P1* и *Р2* в некотором ассоциативном исчислении называются смежными, если одно из них может быть преобразовано в другое однократным применением допустимой подстановки.

Последовательность слов *Р, P1, Р2, ..., М* называется дедуктивной цепочкой, ведущей от слова *Р к* слову *М,* если каждое из двух рядом стоящих слов этой цепочки - смежное.

Слова *Р* и *М* называют эквивалентными, если существует цепочка от *Р* к *М* и обратно.

Пример

Алфавит Подстановки

{*а, b, с, d, е*} *ас - сa, abac - abace*

*ad - da; eca - ae*

*bc - cb; eda - be*

*bd - db; edb - be*

Слова *abcde* и *acbde* - смежные (подстановка *bc* - *cb*). Слова *abcde* - *cadbe* эквивалентны.

Может быть рассмотрен специальный вид ассоциативного исчисления, в котором подстановки являются ориентированными: N → М (стрелка означает, что подстановку разрешается производить лишь слева направо). Для каждого ассоциативного исчисления существует задача: для любых двух слов определить, являются ли они эквивалентными или нет.

Любой процесс вывода формул, математические выкладки и преобразования также являются дедуктивными цепочками в некотором ассоциативном исчислении. Построение ассоциативных исчислений является универсальным методом детерминированной переработки информации и позволяет формализовать понятие алгоритма.

Введем понятие алгоритма на основе ассоциативного исчисления: алгоритмом в алфавите А называется понятное точное предписание, определяющее процесс над словами из А и допускающее любое слово в качестве исходного. Алгоритм в алфавите А задается в виде системы допустимых подстановок, дополненной точным предписанием о том, в каком порядке нужно применять допустимые подстановки и когда наступает остановка.

Пример

Алфавит: Система подстановок *В:*

А = {*а, b, с*} *cb - cc*

*сса - аb*

*ab – bса*

Предписание о применении подстановок: в произвольном слове *Р* надо сделать возможные подстановки, заменив левую часть подстановок на правую; повторить процесс с вновь полученным словом.

Так, применяя систему подстановок *В* из рассмотренного примера к словам *babaac* и *bсaсаbс* получаем:

*babaac* → *bbcaaac* → остановка

*bcacabc → bcacbcac* → *bcacccac* → *bсасаbс* → бесконечные процесс (остановки нет), так как мы получили исходное слово.

Предложенный А.А.Марковым способ уточнения понятия алгоритма основан на понятии нормального алгоритма, который определяется следующим образом. Пусть задан алфавит *А* и система подстановок *В.* Для произвольного слова *Р* подстановки из *В* подбираются в том же порядке, в каком они следуют в *В.* .Если подходящей подстановки нет, то процесс останавливается. В противном случае берется первая из подходящих подстановок и производится замена ее правой частью первого вхождения ее левой части в *Р.* Затем все действия повторяются для получившегося слова *P1*. Если применяется последняя подстановка из системы *В,* процесс останавливается.

Такой набор предписаний вместе с алфавитом *А* и набором подстановок *В* определяют нормальный алгоритм. Процесс останавливается только в двух случаях: 1) когда подходящая подстановка не найдена; 2) когда применена последняя подстановка из их набора. Различные нормальные алгоритмы отличаются друг от друга алфавитами и системами подстановок.

Приведем пример нормального алгоритма, описывающего сложение -натуральных чисел (представленных наборами единиц).

Пример

Алфавит: Система подстановок В:

А = (+, 1) 1 + → + 1

+ 1 → 1

1 → 1

Слово Р: 11+11+111

Последовательная переработка слова *Р* с помощью нормального алгоритма Маркова проходит через следующие этапы:

Р = 11 + 11 + 111 Р5 = + 1 + 111111

Р1 = 1 + 111 + 111 Р6 = ++ 1111111

Р2 = + 1111 + 111 Р7 = + 1111111

Р3 = + 111 + 1111 Р8 = 1111111

Р4 = + 11 + 11111 Р9 = 1111111

Нормальный алгоритм Маркова можно рассматривать как универсальную форму задания любого алгоритма. Универсальность нормальных алгоритмов декларируется принципом нормализации: для любого алгоритма в произвольном конечном алфавите *А* можно построить эквивалентный ему нормальный алгоритм над алфавитом *А,*

Разъясним последнее утверждение. В некоторых случаях не удается построить нормальный алгоритм, эквивалентный данному в алфавите *А,* если использовать в подстановках алгоритма только буквы этого алфавита. Однако, можно построить требуемый нормальный алгоритм, производя расширение алфавита *А* (добавляя к нему некоторое число новых букв). В этом случае говорят, что построенный алгоритм является алгоритмом над алфавитом *А,* хотя он будет применяться лишь к словам в исходном алфавите *A*.

Если алгоритм *N* задан в некотором расширении алфавита *А,* то говорят, что *N* есть нормальный алгоритм над алфавитом *А.*

Условимся называть тот или иной алгоритм нормализуемым, если можно построить эквивалентный ему нормальный алгоритм, и ненормализуемым в противном случае. Принцип нормализации теперь может быть высказан в видоизмененной форме: все алгоритмы нормализуемы.

| Данный принцип не может быть строго доказан, поскольку понятие произвольного алгоритма не является строго определенным и основывается на том, что все Известные в настоящее время алгоритмы являются нормализуемыми, а способы ромпозиции алгоритмов, позволяющие строить новые алгоритмы из уже известных, не выводят за пределы класса нормализуемых алгоритмов. Ниже перечислены способы композиции нормальных алгоритмов.

I. *Суперпозиция алгоритмов.* При суперпозиции двух алгоритмов *А* и *В* выходное слово первого алгоритма рассматривается как входное слово второго алгоритма *В.* Результат суперпозиции *С* может быть представлен в виде *С(р)* = *В(А(р)),*

II. *Объединение алгоритмов.* Объединением алгоритмов *А* и *В* в одном и том же алфавите называется алгоритм С в том же алфавите, преобразующий любое слово *р,* содержащееся в пересечении областей определения алгоритмов *А* и *В,* в записанные рядом слова *А(р)* и *В(р).*

III. *Разветвление алгоритмов.* Разветвление алгоритмов представляет собой композицию *D* трех алгоритмов *А, В* и *С,* причем область определения алгоритма *D* является пересечением областей определения всех трех алгоритмов *А, В* и С, а для любого слова *р* из этого пересечения *D(p)* = *А(р),* если *С(р)* = *е, D(p)* = *B(p),* если *С(р) = е,* где *е -* пустая строка.

IV. *Итерация алгоритмов.* Итерация (повторение) представляет собой такую композицию С двух алгоритмов *А* и *В,* что для любого входного слова *р* соответствующее слово *С(р)* получается в результате последовательного многократного применения алгоритма *А* до тех пор, пока не получится слово, преобразуемое алгоритмом *В.*

Нормальные алгоритмы Маркова являются не только средством теоретических построений, но и основой специализированного языка программирования, применяемого как язык символьных преобразований при разработке систем искусственного интеллекта. Это один из немногих языков, разработанных в России и получивших известность во всем мире.

Существует строгое доказательство того, что по возможностям преобразования нормальные алгоритмы Маркова эквивалентны машинам Тьюринга.

### 7.5. РЕКУРСИВНЫЕ ФУНКЦИИ

Еще одним подходом к проблеме формализации понятия алгоритма являются, так называемые, рекурсивные функции. Исторически этот подход возник первым, поэтому в математических исследованиях, посвященных алгоритмам, он имеет наибольшее распространение.

Рекурсией называется способ задания функции, при котором значение функции при определенном значении аргументов выражается через уже заданные значения функции при других значениях аргументов. Применение рекурсивных функций в теории алгоритмов основано на идее нумерации слов в произвольном алфавите последовательными натуральными числами. Таким образом любой алгоритм можно свести к вычислению значений некоторой целочисленной функции при целочисленных значениях аргументов.

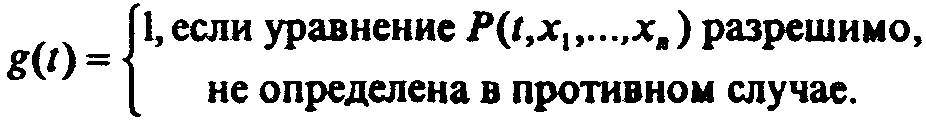
Введем несколько основных понятий. Пусть *X, Y -* два множества. Частичной функцией (или отображением) из *Х* в *Y* будем называть пару *<D(f), f>*, состоящую из подмножества *D(f)* *⊂ X* (называемого областью определения *f)* и отображения *f*: *D(f)* → *Y.* Если *D(f)* пусто, то *f* нигде не определена. Будем считать, что существует единственная нигде не определенная частичная функция.

Через *N* будем обозначать множество натуральных чисел. Через *(N)n* (при *п ≥* 1) будем обозначать *n*-кратное декартово произведение *N* на себя, т.е. множество упорядоченных *n*-ок (*х1* ..., *xn*), *хi* ⊂ *N.* Основным объектом дальнейших построений будут частичные функции из *(N)m* в *(N)n* для различных *т* и *п.*

Частичная функция *f* из *(N)m* в *(N)n* называется вычислимой, если можно указать такой алгоритм («программу»), который для входного набора *х* ⊂ *(N)m* дает на выходе *f(x),* если *х* ⊂ *D(f)* и нуль, если *х* ⊄ *D(f).* В этом определении неформальное понятие алгоритма (программы) оказывается связанным (отождествленным) с понятием вычислимости функции. Вместо алгоритмов далее будут изучаться свойства вычислимых функций. Вместо вычислимых функций оказывается необходимым использовать более широкий класс функций (и более слабое определение) - полувычислимые функции. Частичная функция из *(N)"* в *(N)"* полувычислима, если можно указать такой алгоритм (программу), который для входного набора *х* с *(N)"* дает на выходе *х* е *D(f),* или алгоритм работает неопределенно долго, если *х* е *D(f).* Очевидно, что вычислимые функции полувычислимы, а всюду определенные полувычислимые функции вычислимы.

Частичная функция *f* называется невычислимой, если она не является ни вычислимой, ни полувычислимой.

Из вновь введенных понятий основным является полувычислимость, так как вычислимость сводится к нему. Существуют как невычислимые функции, так и функции, являющиеся полувычислимыми, но не вычислимые. Пример такой функции:



Можно показать, что существует такой многочлен с целыми коэффициентами *P(t, x1,...,xn*), что *g(t) -* невычислима. Однако, легко видеть, что *g(t) -* полувычислима.

Фундаментальным открытием теории вычислимости явился, так называемый, тезис Черча, который в слабейшей форме имеет следующий вид: можно явно указать а) семейство простейших полувычислимых функций; б) семейство элементарных операций, которые позволяют строить по одним полувычислимым функциям другие полувычислимые функции с тем свойством, что любая полувычислимая функция получается за конечное число шагов, каждый из которых состоит в применении одной из элементарных операций к ранее построенным или к простейшим функциям.

Простейшие функции:

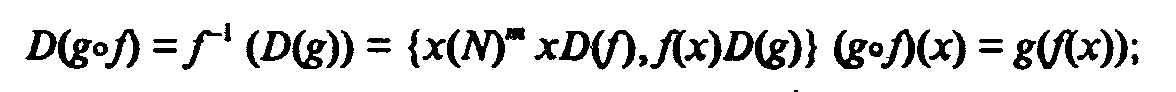
suc: *N → N;* suc*(x)* = *x*+1 - определение следующего за *х* числа;

*l(n)*: *(N)n* *→* *N; l(n) (x1,..., хn)* = 1, *п ≥* 0 - определение «размерности» области определения функции;

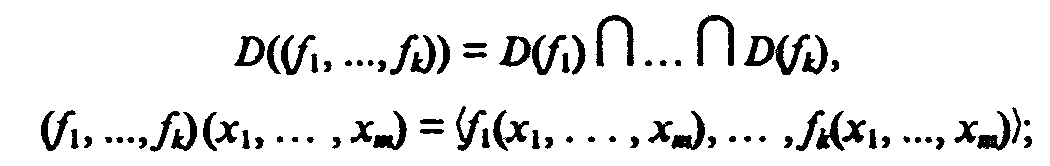
*рr: (N)n→* *N; pr(x1,..., хn)* = *хi, х* ≥ 1 - «проекция» области определения на одну из переменных.

Элементарные операции над частичными функциями:

а) *композиция* (или подстановка) ставит в соответствие паре функций *f* из *(N)m* в *(N)n* и *g* из *(N)n* в *(N)p* функцию *h = gof* из *(N)m* в *(N)p ,* которая определяется как



*б) соединение* ставит в соответствие частичным функциям *fi* из *(N)ni, i =* 1, ..., *k* функцию (*f*i, ..., *fk)* из *(N)m* в *(N)n1* х... х *(N)nk,* которая определяется как

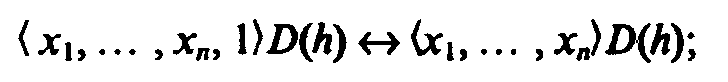


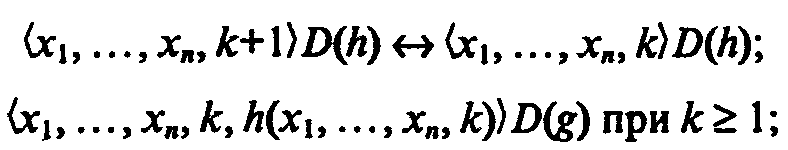
в) *рекурсия* ставит в соответствие паре функций *f* из *(N)n* в *N* и *g* из *(N)n+2* в *N* функцию *h* из *(N)n+2 в N,* которая определяется рекурсией по последнему аргументу

*h(x1,*... , *хn*, 1) = *f* (*x1*,*... ,xп)* (начальное условие),

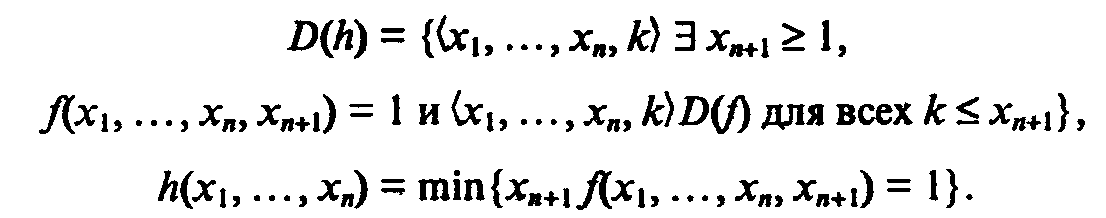
*h (x1,... ,хn, k+1) = g(x1,... ,xn, k, h(x1,... ,хn, k))* при *k*≥ 1 (рекурсивный шаг).

Область определения *D(h)* описывается также рекурсивно:





г) *операция т,* которая ставит в соответствие частичной функции *f* из *(N)n+1* в *N* частичную функцию *h* из *(N)n* в *N,* которая определяется как



Операция *т* позволяет вводить в вычисления перебор объектов для отыскания нужного в бесконечном семействе.

Теперь, когда введены простейшие функции и элементарные операции, можно дать следующие основные определения:

а) последовательность частичных функций *f*i, . . . *,fN* называют частично рекурсивным (соответственно примитивно рекурсивным) описанием функции *fN = f*, если *f*i - одна из простейших функций; *fi* для всех *i ≥* 2 либо является простейшей функцией, либо получается применением одной из элементарных операций к некоторым из функций *f*i,..., *fi-1* (соответственно одной из элементарных операций, кроме *т);*

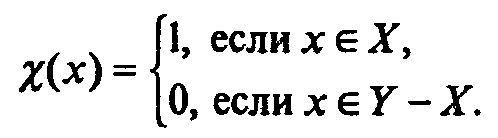
б) функция *f* называется частично рекурсивной (соответственно примитивно рекурсивной), если она допускает частично рекурсивное (соответственно примитивно рекурсивное) описание.

Теперь можно привести тезис Черча в обычной форме:

а) функция *f* полувычислима, если и только если она частично рекурсивна;

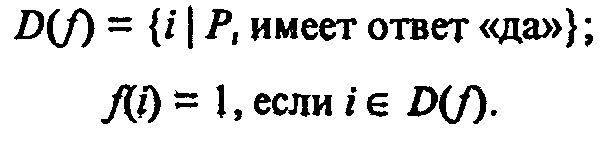
б) функция *f* вычислима, если и только если рекурсивны *f* и характеристическая функция *XD(f).*

Характеристическая функция подмножества *Х* в *Y(X ⊂ Y)* есть такая функция, что



Тезис Черча может использоваться как определение алгоритмической неразрешимости.

Пусть имеется счетная последовательность «задач» *P1*, *P2, ...,* которые имеют ответ «да» или «нет». Такая последовательность носит название «массовой проблемы». Свяжем с ней функцию *f* из *N* в *N:*



Массовая проблема *Р* называется алгоритмически разрешимой, если функции *f* и *XD(f)* частично рекурсивны. В противном случае *Р* называется алгоритмически неразрешимой.