ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

УПРАВЛЕНИЕ ЦИФРОВЫХ ОБРАЗОВАТЕЛЬНЫХ ТЕХНОЛОГИЙ

Кафедра «Прикладная математика»

Учебное пособие

«Лекции по теории алгоритмов»
по дисциплине

«Теория алгоритмов»

Авторы
Баранов И. В.,
Гусева И. А.

Ростов-на-Дону, 2019

Аннотация

Предлагаемое учебное пособие предназначено для студентов очной, заочной форм обучения, изучающих курс теории алгоритмов. Написано в соответствии с государственным образовательным стандартом высшего профессионального образования. Пособие может быть использовано для самостоятельного изучения дисциплины «Теория алгоритмов».

Авторы

к.ф.-м.н., доцент кафедры «Прикладная математика» Баранов И. В.,
к.ф.-м.н., доцент кафедры «Прикладная математика» Гусева И. А.

Содержание

Введение

Теория алгоритмов – раздел математической логики, в котором изучаются теоретические возможности эффективных процедур вычисления (алгоритмов) и их приложения.

Основное понятие этой теории – *алгоритм* – в интуитивном (наивном) понимании существует в математике очень давно (заведомо раньше, чем с IX века), однако, точные математические формулировки, которые в том или ином смысле формализуют интуитивное понятие алгоритма, были предложены только в середине 30-х годов XX-го века, и связаны прежде всего, с работами Алонзо Чёрча, Стивена Клини, Дэвида Гильберта, Курта Гёделя, Алана Тьюринга, Андрея Андреевича Маркова (младшего). Необходимость такой формализации была обусловлена как вопросами обоснования основ математики, так и вопросами доказательства алгоритмической разрешимости или же неразрешимости тех или иных математических задач, накопившихся к этому времени. Очевидно, что в математике доказываемый объект должен быть точно определен.

В настоящее время теорию алгоритмов разделяют на:
1) дескриптивную (описательную, абстрактную) и
2) метрическую (количественную).

Первая (*дескриптивная*) часть теории алгоритмов описывает и исследует алгоритмы с точки зрения устанавливаемого ими соответствия между исходными данными и результатами; к ней относятся, в частности, проблемы построения алгоритма, обладающего теми или иными свойствами, -- алгоритмические (массовые) проблемы (т.е. нахождение единого метода решения бесконечной серии однотипных единичных задач).

Поиск теоретических моделей алгоритмов происходит в трех направлениях, которые и определяют три основных класса

этих моделей: 1) арифметизации алгоритмов, 2) концепции абстрактной вычислительной машины и 3) принципа нормализации (т.е. преобразование слов в произвольных алфавитах с помощью подстановок). В этой части теории алгоритмов можно выделить, соответственно, три описательных подхода: теорию рекурсивных функций (А.Чёрч, С. Клини), концепцию абстрактной машины (А.Тьюринг, Э.Пост, Шенфилд Дж. и др.) и нормальные алгоритмы Маркова (А.А. Марков).

Вторая (*метрическая*) часть теории алгоритмов исследует алгоритмы с точки зрения *сложности* как *самих алгоритмов*, так *и задаваемых ими вычислений*, т.е. процессов последовательного преобразования конструктивных объектов. Важно подчеркнуть, что как сложность алгоритмов, так и сложность вычислений могут определяться различными способами. Разработка методов оценки сложности алгоритмов и вычислений имеет важное теоретическое и практическое значение. Необходимо также отметить, что метрическая теория полностью не построена. В ней имеются нерешённые проблемы. В частности, на сегодняшний момент, не решена знаменитая проблема равенства алгоритмических классов сложности *P* и *NP*. Это одна из центральных открытых проблем теории алгоритмов. Если на неё будет дан утвердительный ответ, то это будет означать, что теоретически, возможно решать многие сложные задачи существенно быстрее, чем сейчас. Проблема равенства классов *P* и *NP* является одной из семи задач тысячелетия, за решение которой Математический институт Клэя назначил премию в миллион долларов США.

§1. Понятие алгоритма и необходимость его уточнения.

1.1. Интуитивное представление о термине «алгоритм».

Понятие алгоритма является одним из основных понятий современной математики. Еще на самых ранних ступенях развития математики (Древний Египет, Вавилон, Греция) в ней стали возникать различные вычислительные процессы чисто механического характера. С их помощью искомые величины ряда задач вычислялись последовательно из исходных величин по определенным правилам и инструкциям. Со временем все такие процессы в математике получили название *алгоритмов* (*алгорифмов*).

Термин *алгоритм* происходит от имени средневекового узбекского математика Абу Абдуллах Мухаммеда ибн Муса аль-Хорезми (на рис. 1 – марка, выпущенная в его честь к 1200 летию), который еще в IX в. (825 г.) дал правила выполнения четырех арифметических действий в десятичной системе счисления в своей работе «Algoritmi de numero Indorum» (“Индийское искусство счёта”). Процесс выполнения арифметических действий был назван *алгоризмом.*

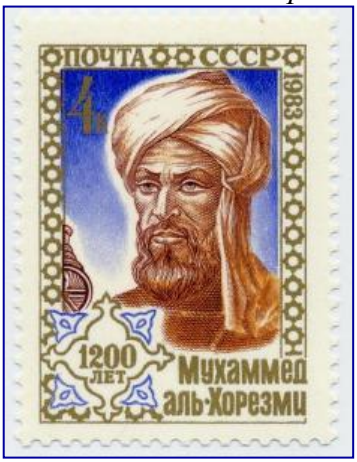


Рис. 1. Мухаммед аль-Хорезми

Вплоть до 30-х годов XX века понятие алгоритма имело скорее методологическое, чем математическое значение. *Под алгоритмом понимали конечную последовательность предписаний, точное исполнение которых приводит к решению*

поставленной задачи.

Заметим, что приведенное наивное (интуитивное) описание алгоритма не является точным математическим определением, а лишь объясняет смысл слова «алгоритм», в котором это слово используется в математике.

Один из основателей современной теории алгоритмов Марков А.А. в 1957 году писал:

« Алгоритм – это точное предписание которое задает вычислительный процесс, начинающийся с произвольного (но выбранного из фиксированной для данного алгоритма совокупности) исходного данного и направленный на получение полностью определяемого этим исходным данным результата».

Другой широко известный математик Колмогоров А.Н. в своей статье 1958 года «К определению алгоритма» отмечал:

«Алгоритмом принято называть систему вычислений, которая для некоторого класса математических задач из записи ***A*** «условий» задачи позволяет при помощи однозначно определенной последовательности операций, совершаемых «механически», без вмешательства творческих способностей человека, получить запись ***B*** «решения» задачи» .

Отличительными признаками алгоритма являются:

1. *Дискретность:* алгоритм представляет собой систему пошаговых предписаний, причем каждый шаг должен быть выполнен, или не выполнен полностью.
2. *Детерминированность*: после выполнения очередного шага однозначно определено, что (какой шаг) делать дальше. Следующий шаг определяется текущим шагом и предыдущими шагами алгоритма.
3. *Элементарность шагов:* алгоритм разбивается на шаги, каждый из которых должен быть по возможности наиболее простым и понятным для исполнителя алгоритма.
4. *Массовость:* алгоритм должен решать не одну

конкретную задачу, а целый класс подобных задач со сходными входными данными.

5. *Завершаемость:* при корректных входных данных алгоритм должен завершать свою работу за конечное число шагов.

6. *Результативность*: алгоритм должен заканчиваться вполне определённым результатом.

Некоторые примеры алгоритмов:

1. Процедура сложения двух чисел “столбиком”.
2. Нахождение наибольшего общего делителя двух натуральных чисел.
3. Вычисление определителя квадратной матрицы.

1.2. Необходимость уточнения понятия «алгоритм».

В течение долгого времени, вплоть до конца XIX века, математики довольствовались интуитивным понятием термина «алгоритм», поскольку общей теории алгоритмов фактически не существовало. Однако, практически не было случаев, когда математики разошлись бы во мнениях относительно того, является ли алгоритмом тот или иной конкретно заданный процесс.

Положение существенно изменилось, когда на первый план выдвинулись такие алгоритмические проблемы, положительное решение которых было сомнительным. Действительно, одно дело доказать *существование* алгоритма, и, совсем другое – доказать *его отсутствие*. Первое можно сделать путем фактического описания процесса, решающего задачу. В этом случае достаточно и интуитивного представления об алгоритме, чтобы удостовериться в том, что описанный процесс есть алгоритм. Доказать же

несуществование алгоритма таким путем невозможно. Для этого надо точно знать, что такое алгоритм. В двадцатых годах прошлого века задача строгого определения термина «алгоритм» стала одной из центральных математических задач. Решение её было получено в середине 30-х годов в работах известных математиков Д. Гильберта, К. Гёделя, А. Черча, С. Клини, Э. Поста и А. Тьюринга в двух формах.

Первое решение было основано на понятии особого класса арифметических функций, получивших название *рекурсивных* функций, второе – на описании точно очерченного класса процессов, связанных с моделированием работы ***простейшего*** *и* ***универсального*** *механического вычислителя*.

Несколько позже, в работах А.А. Маркова (младшего), появилось другое толкование алгоритма, поставившее в основу определения понятия «алгоритм» особое соответствие между словами (наборами *символов*) в некотором абстрактном алфавите.

Таким образом, можно выделить три основных направления в уточнении и формализации понятия алгоритма:

1. Связано с понятием эффективно вычислимой функции. Это было сделано впервые А. Чёрчем, К. Гёделем, С. Клини в 1935-1936 годах. Был выделен класс *рекурсивных* функций, который, как впоследствии оказалось, совпадает с классом эффективно-вычислимых функций.
2. Связано с машинной математикой. Сущность алгоритма рассматривается путем рассмотрения процессов, осуществляемых в вычислительных машинах. Впервые это было сделано Э. Постом (1936 г.) и А. Тьюрингом (1937 г.).
3. Связано с понятием нормальных алгоритмов, введенных и разработанных российским математиком А. А. Марковым (конец 40-х - начало 50-х годов XX века).

1.3. Способы описания алгоритмов. Изображение алгоритма в виде блок-схемы.

К основным способам описания алгоритмов можно отнести следующие:

– **текстовый** или словесно-формульный (описание на естественном языке, специальном алгоритмическом языке или на «псевдокоде», который является промежуточным между естественными и алгоритмическими языками);

– **графический:** описание при помощи некоторой наглядной картинки (структурный или блок-схемный, граф-схемный (граф – совокупность точек и линий, в которой каждая линия соединяет две точки. Точки называются вершинами, линии – рёбрами).

Рассмотрим подробнее эти способы.

Словесно-формульный способ.

При словесно-формульном способе алгоритм записывается в виде текста с формулами по пунктам, определяющим последовательность действий.

Пример: Пусть, например, необходимо найти значение следующего выражения: $y = 2a - (x + 6)$.

Словесно-формульным способом алгоритм решения этой задачи может быть записан в следующем виде:

1.Ввести значения a и x.

2.Сложить x и 6.

3.Умножить a на 2.

4.Вычесть из $2a$ сумму $(x + 6)$.

5.Вывести y как результат вычисления выражения.

Псевдокод (пример: сортировка массива).

**вход: массив a[i], i=1,…,N*

**выход: отсортированный по возрастанию массив a[i]*

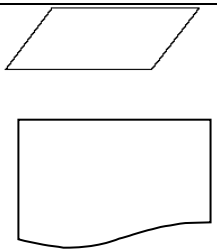
**сортировка массива методом пузырька:*

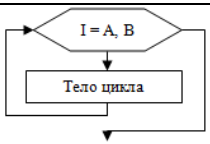
Для *i* начиная с 2 до *N* делать

Для *j* начиная с *N* с шагом -1 до *i* делать
Если *a*[*j*]<*a*[*j-1*] то
a[*j*]←→*a*[*j-1*] (поменять местами соседние элементы)
конец (если)
конец (для *j*)
конец (для *i*)

Блок-схемы.

Блок-схемой называется наглядное графическое изображение алгоритма, когда отдельные его этапы изображаются при помощи различных геометрических фигур – блоков, а связи между этапами (последовательность выполнения этапов) указываются при помощи стрелок, соединяющих эти фигуры. Блоки сопровождаются надписями. Типичные действия алгоритма изображаются следующими геометрическими фигурами:

№	Фигура	Название блока	Запись внутри блока	Количество входов и выходов
1.		Блок начала (конца)	Начало Конец	1 вход 1 выход
2.		Ввод (вывод) данных	Описание переменных, либо переменные, которые необходимо ввести (вывести)	1 вход 1 выход
3.		Арифмети-ческий блок или блок	Операция или группа операций	1 вход 1 выход

		действия		
4.		Условие	Условие «=», «<», «>», «<>», «>=», «<=»	1 вход 2 выход
5.		Цикл со счетчиком	Начальное значение, конечное значение	1 вход 1 выход 1циклический круг

Данный способ по сравнению с другими способами записи алгоритма имеет ряд преимуществ. Он наиболее нагляден: каждая операция вычислительного процесса изображается отдельной геометрической фигурой. Кроме того, графическое изображение алгоритма наглядно показывает разветвления путей решения задачи в зависимости от различных условий, повторение отдельных этапов вычислительного процесса и другие детали.

Из каких же основных (базовых) конструкций (структур) состоит любой алгоритм?

Базовые структуры алгоритмов – это определенный набор блоков и стандартных способов их соединения для выполнения типичных последовательностей действий.

К основным (базовым) структурам относятся следующие:

1. Линейные (рис. 2).

Определение. Линейными называются алгоритмы, в которых действия осуществляются последовательно друг за другом.

Блок-схема линейного алгоритма приводится ниже:

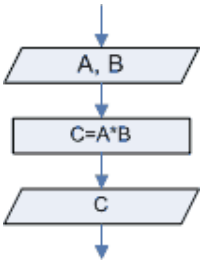


Рис. 2. Блок-схема линейного алгоритма

2. Разветвляющиеся (рис. 3).

Определение. Разветвляющимся называется алгоритм, в котором действие выполняется по одной из возможных ветвей решения задачи, в зависимости от выполнения условий.

В отличие от линейных алгоритмов, в которых команды выполняются последовательно одна за другой, в разветвляющиеся алгоритмы входит условие, в зависимости от выполнения или невыполнения которого выполняется та или иная последовательность команд (действий).

В качестве условия в разветвляющемся алгоритме может быть использовано любое понятное исполнителю утверждение, которое может соблюдаться (быть истиннным) или не соблюдаться (быть ложным). Такое утверждение может быть выражено как словами, так и формулой. Таким образом, алгоритм ветвления состоит из условия и двух последовательностей команд.

В зависимости от того, в обоих ветвях решения задачи находится последовательность команд или только в одной, разветвляющиеся алгоритмы делятся на полные и не полные (сокращенные).

Блок-схемы разветвляющегося алгоритма приведены ниже:

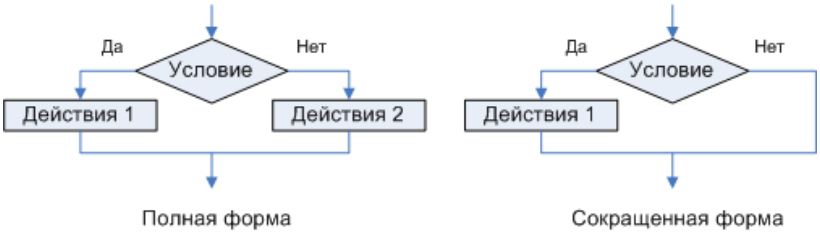


Рис. 3. Блок-схемы разветвляющегося алгоритма

4. Циклические (с оговоркой, что цикл можно построить из линейной и разветвляющейся структуры).

Определение. Циклическим называется алгоритм, в котором некоторая часть операций (тело цикла – последовательность команд) выполняется многократно.

Однако слово «многократно» не значит «до бесконечности». Организация циклов, никогда не приводящая к остановке в выполнении алгоритма, является нарушением требования его результативности – получения результата за конечное число шагов. Заметим, что всякий циклический алгоритм может быть организован “вручную” с помощью линейной и разветвляющейся структур.

В этом смысле, *основными являются лишь две алгоритмические структуры – линейная и разветвляющаяся*.

Перед операцией цикла осуществляются операции присвоения начальных значений тем объектам, которые используются в теле цикла. В цикл входят в качестве базовых следующие структуры:

– блок проверки условия;

– блок, называемый телом цикла.

Существуют три типа циклов:

– *цикл с предусловием*. Если тело цикла расположено после проверки условий, то может случиться, что при определенных условиях тело цикла не выполнится ни разу. Такой вариант организации цикла, управляемый предусловием,

называется *циклом с предусловием* (рис. 4).

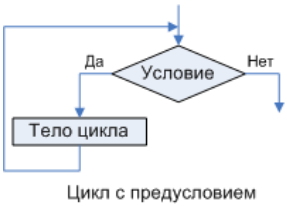


Рис. 4. Блок-схема цикла с предусловием

– *цикл с постусловием*. Тело цикла выполняется, по крайней мере, один раз и будет повторяться до тех пор, пока не станет ложным условие. Такая организация цикла, когда его тело расположено перед проверкой условия, носит название *цикла с постусловием* (рис. 5).

Рис. 5. Блок-схема цикла с постусловием

– *цикл с параметром (или со счётчиком)* (разновидность цикла с предусловием). *Цикл с параметром* (рис. 6) является разновидностью цикла с предусловием. Особенностью данного типа цикла является то, что в нем имеется параметр, начальное значение которого задается в заголовке цикла, там же задается условие продолжения цикла и закон изменения параметра цикла. Механизм работы полностью соответствует циклу с предусловием, за исключением того, что после выполнения тела цикла происходит изменение параметра по указанному закону и только потом переход на проверку условия.

Рис. 6. Блок-схема цикла с параметром

Пример: Дано натуральное число N. Определить, является ли оно простым. Решение представить в виде блок-схемы.

Решение. Натуральное число $N>1$ является простым, если оно делится нацело без остатка только на единицу и N.

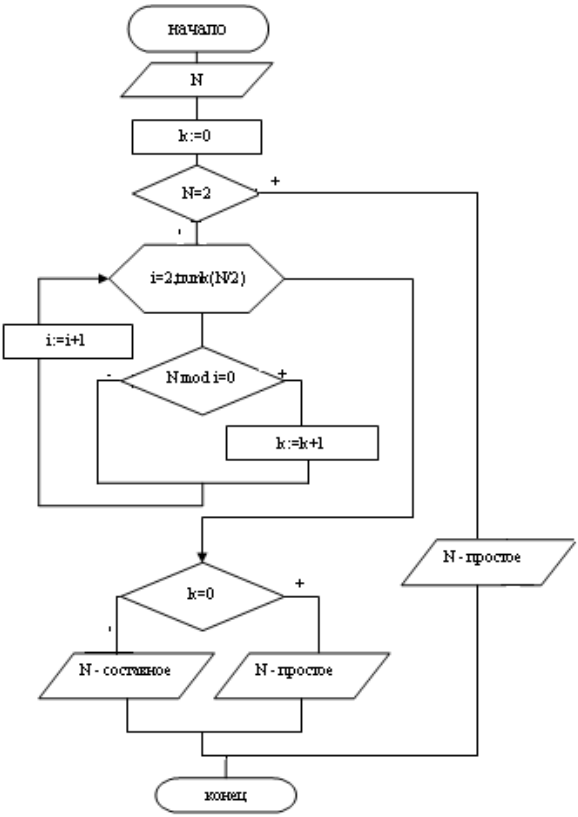
Входные данные: N – натуральное число.

Выходные данные: сообщение.

Промежуточные переменные: i – параметр цикла, возможные делители числа N, k – количество делителей числа N.

Алгоритм решения этой задачи заключается в том, что в переменную k, предназначенную для подсчёта количества делителей заданного числа, помещается значение, которое не влияло бы на результат, то есть нуль. Число N делится на параметр цикла i, изменяющийся в диапазоне от 2 до целой части от $\frac{N}{2}$. Если заданное число делится на i нацело, то к числу делителей прибавляем 1, т.о. i является делителем числа N. После выхода из цикла проверяем, изменилось ли число делителей числа N. Если k осталось равным 0, то число N – простое, иначе – составное.

Итак, реализуемый алгоритм в виде блок-схемы выглядит следующим образом:



§2. Формальная теория вычислимых функций

2.1. Числовые функции

Как правило, отыскание того или иного алгоритма в математике можно свести к нахождению алгоритма,

вычисляющего некоторую частичную числовую функцию или хотя бы к доказательству принципиальной вычислимости этой функции. Таким образом, понятие алгоритма тесно связано с понятием вычислимой функции, поэтому класс вычислимых функций был *формализован* или *аксиоматизирован* (в результате получен класс частично рекурсивных функции - *ЧРФ*). При этом были выделены некоторые *простейшие функции*, которые, очевидным образом, являются вычислимыми. Затем ведены три правила получения из имеющихся функций новых. Эти правила, примененные к вычислимым функциям, дают в результате функции вычислимые. Такие правила названы *основными вычислимыми операторами*.

Будем рассматривать только *числовые функции*, т. е. функции, аргументы и значения которых принадлежат множеству натуральных чисел с нулем N_0 ($N_0 = \{0,1,2,\ldots\}$).

Определение. Если область определения функции совпадает с множеством $N_0{}^n (f : N_0{}^n \to N_0)$, то функция называется *всюду определенной*, иначе – *частично определенной (частичной)*.

Определение. Частичная числовая функция называется *вычислимой*, если существует алгоритм, позволяющий вычислять ее значения для, тех наборов значений аргументов, для которых она определена, и работающий вечно на наборах значений аргументов, для которых эта функция не определена.

2.2. Простейшие функции

ЧРФ строятся на основе трех примитивных (заведомо однозначно вычислимых) функций. Их также называют *простейшими*.

- $\lambda(x) = x + 1$ – функция следования ($x \in N_0$)
- $O(x) = 0$ – нуль-функция($x \in N_0$)

- $I_n^m(x_1, x_2, \ldots x_n) = x_m$ $(m = 1,2,\ldots,n)$ – функция проектирования (выбора аргумента).

Ясно, что все три простейшие функции всюду определены и интуитивно вычислимы.

2.3. Вычислимые операторы

1. Оператор S суперпозиции (подстановки).

Рассмотрим функции $g_1(x_1, x_2,\ldots, x_n)$, $g_2(x_1, x_2,\ldots, x_n)$, …, $g_m(x_1, x_2,\ldots, x_n)$ и функцию $f(x_1, x_2,\ldots, x_m)$. Оператор суперпозиции S ставит в соответствие функциям f и $g_1,\ldots, g_m$ n-местную функцию h по следующему правилу:

$$h(x_1,\ldots, x_n) = f(g_1((x_1,\ldots, x_n),\ldots, g_m(x_1,\ldots, x_n)). \qquad (1)$$

Функция h является частичной функцией от n переменных. Ее значение $h(x_1, x_2,\ldots, x_n)$ определено тогда и только тогда, когда определены все функции в правой части (1). Если функции f и $g_1,\ldots, g_m$ вычислимы, то и функция h вычислима.

Если функция h получена с помощью оператора суперпозиции из функций f и $g_1,\ldots, g_m$, то это записывается так: $h = S(f, g_1, g_2,\ldots g_n)$.

Примеры:

1) Используя оператор суперпозиции, можно получить любую константу:

$\lambda(O(x)) = 0 + 1 = 1,$

$\lambda(\lambda(O(x))) = 0 + 1 + 1 = 0 + 2 = 2,$

$\lambda(\lambda\ldots(O(x))\ldots) = 0 + n = n,$ где n – число вложений функций следования.

2) Используя оператор суперпозиции, можно выполнить сдвиг на константу n, здесь также n число вложений функций следования.

$\lambda(x) = x + 1,$

$\lambda(\lambda(x)) = x+1+1 = x+2$,

$\lambda(\lambda...\lambda(x))...) = x+n$.

2. Оператор примитивной рекурсии R.

Оператор примитивной рекурсии R каждой $(n+2)$-местной функции h и n-местной функции g ставит в соответствие $(n+1)$-местную функцию f, удовлетворяющую следующей схеме:

$$\begin{aligned} f(x_1,...,x_n,0) &= g(x_1,...,x_n), \\ f(x_1,...,x_n,y+1) &= h(x_1,...,x_n,y,f(x_1,...,x_n,y)). \end{aligned} \quad (2)$$

Если все значения в правых частях равенств (2) существуют, то получим значение функции $f(x_1,...,x_n,y+1)$. Если какое-то значение не определено, то $f(x_1,...,x_n,y+1)$ не существует. Поэтому в общем случае мы получаем частичную функцию от $(n+1)$ переменной.

Для $n=0$ схема примитивной рекурсии имеет вид:

$f(0) = a$, где a – константа,

$$f(y+1) = h(y,f(y)) \quad (3)$$

Если функция f получена оператором примитивной рекурсии из функций g и h (т.е. задана в виде (2) и (3)), то записываем: $f = R(g,h)$.

Как и в случае оператора суперпозиции, вычислимость исходных функций h и g влечет вычислимость построенной из них функции f.

Пример: Покажем, что функция $s(x;y) = x+y$ может быть получена из простейших при помощи суперпозиции и примитивной рекурсии. Положим:

$g(x) = I_1^1(x)$,

$h(x,y,z) = \lambda(I_3^3(x,y,z))$.

Тогда:

$s(x,0) = g(x)$,

$s(x, y+1) = h(x, y, s(x, y))$.

Схема примитивной рекурсии образует процесс построения функции s, при котором на нулевом шаге используется функция g, а на каждом последующем шаге значение функции h от аргументов $x_1,...,x_n$, номера y предыдущего шага и значения функции f, вычисленное на предыдущем шаге.

3. Оператор минимизации (μ-оператор).

Пусть задана некоторая функция $f(x, y)$. Ставится задача отыскания для данной функции $f(x, y)$ и фиксированного x *наименьшего* из тех значений y, при которых функция $f(x, y) = 0$. Так как результат решения задачи зависит от x, то наименьшее значение y, при котором функция $f(x, y) = 0$ есть функция от x. Принято обозначение $\varphi(x) = \mu_y[f(x, y) = 0]$ (Читается: «наименьшее y такое, что $f(x, y) = 0$»). Аналогично определяется функция многих переменных:

$$\varphi(x_1, x_2,..., x_n) = \mu_y[f(x_1, x_2,..., x_n, y) = 0].$$

Переход от функции $f(x_1, x_2,..., x_n, y)$ к функции $\varphi(x_1, x_2,..., x_n)$ принято называть *применением μ-оператора.*

Для вычисления функции φ можно предложить следующий алгоритм:

1. Вычислим $f(x_1, x_2,..., x_n, 0)$. Если это значение f равно нулю, то полагаем $\varphi(x_1, x_2,..., x_n) = 0$. Если $f(x_1, x_2,..., x_n, 0) \neq 0$, то переходим к следующему шагу.

2. Вычислим $f(x_1, x_2,..., x_n, 1)$. Если $f(x_1, x_2,..., x_n, 1) = 0$, то полагаем $\varphi(x_1, x_2,..., x_n) = 1$. Если же $f(x_1, x_2,..., x_n, 1) \neq 0$, то переходит к следующему шагу. И так далее.

Если окажется, что для всех y функция

$f(x_1, x_2, ..., x_n, y) \neq 0$, то функцию $\varphi(x_1, x_2, ..., x_n)$ в этом случае считают неопределенной. Но возможно, что существует такое y_0, что $f(x_1, x_2, ..., x_n, y_0) = 0$ и, значит, есть и наименьшее $y = y_0$, при котором $f(x_1, x_2, ..., x_n, y) = 0$. Однако, может случиться, что при некотором z $(0 < z < y_0)$ значение функции $f(x_1, x_2, ..., x_n, z)$ не определено. Очевидно, что в этом случае процесс вычисления наименьшего y, при котором $f(x_1, x_2, ..., x_n, y) = 0$ не дойдет до y_0. И здесь функцию $\varphi(x_1, x_2, ..., x_n)$ считают неопределенной.

Определение. Пусть f – функция от $(n+1)$ переменной. Будем говорить, что функция φ от n переменных получена из функции f *с помощью оператора минимизации*, если равенство $\varphi(x_1, x_2, ..., x_n) = y$ выполняется тогда и только тогда, когда $f(x_1, x_2, ..., x_n, y) = 0$, а значения $f(x_1, x_2, ..., x_n, 0)$, …, $f(x_1, x_2, ..., x_n, y-1)$ определены и не равны нулю.

Данное определение можно выразить записью:

$$\varphi(x_1, x_2, ..., x_n) = y \Leftrightarrow \begin{cases} f(x_1, x_2, ..., x_n, 0) \neq 0, \\ ..... \\ f(x_1, x_2, ..., x_n, y-1) \neq 0, \\ f(x_1, x_2, ..., x_n, y) = 0, \end{cases} \qquad (4)$$

причем значения $f(x_1, x_2, ..., x_n, 0)$, …, $f(x_1, x_2, ..., x_n, y-1)$ существуют.

Если функция φ получается из функции f с помощью оператора минимизации, то будем также записывать $\varphi = \mu_y(f)$.

Очевидно, что правила (4) для вычисления функции f представляют собой алгоритм для ее вычисления, поэтому, если функция f вычислима, то и функция φ также вычислима.

Пример. Проиллюстрируем применение оператора минимизации по переменной z к функции трех переменных $f(x,y,z)=y+z-x$. В результате получим функцию двух переменных $\varphi(x,y)=x-y$. Формально:

$$\varphi(x,y)=\mu_z[y+z-x=0].$$

Действительно, пусть например $x=5$, $y=2$. Вычислим $\varphi(5,2)$, т.е. применим оператор минимизации μ_z к $f(5,2,z)=2+z-5$:

$z=0$: $f(5,2,0)=2+0-5\neq 0$,

$z=1$: $f(5,2,1)=2+1-5\neq 0$,

$z=2$: $f(5,2,2)=2+2-5\neq 0$,

$z=3$: $f(5,2,3)=2+3-5=0$.

Следовательно $\varphi(5,2)$=3.

2.4. Примитивно рекурсивные функции (ПРФ)

Определение. Функция называется *примитивно-рекурсивной (ПРФ),* если она может быть получена из простейших функций с помощью конечного числа применений операторов суперпозиции и примитивной рекурсии.

Очевидно, что

- простейшие функции примитивно рекурсивны;
- если функция получена из примитивно рекурсивных функций с помощью оператора суперпозиции *или* с помощью оператора примитивной рекурсии, то она примитивно рекурсивна.

Рассмотрим примеры ПРФ:

1. Функция $f(x)=x$ – ПРФ, т.к. $f(x)=x=I_1^1(x)$, а проектирующая функция является ПРФ.

2. Функция $f(x)=x+2$ – ПРФ, т.к. $f(x)=x+2=(x+1)+1=\lambda(\lambda(x))$, а значит, получена из функции следования (которая примитивно рекурсивна) с помощью оператора суперпозиции.

Функция $f(x)=a$ – ПРФ, т.к. $f(x)=a=\underbrace{\lambda(\lambda(...\lambda}_{a\ раз}(0(x))))=a$, а значит получена из простейших функций (нуль функции и функции следования) с помощью конечного числа применений оператора суперпозиции

Заметим, что всякой примитивной рекурсивной функции можно поставить в соответствие наименьший номер n шага, на котором она может быть получена. При этом простейшие функции $\lambda(x)=x+1$, $O(x)=0$ и $I_n^m(x_1,x_2,...x_n)=x_m$ очевидно являются ПРФ шага 0. Следовательно, можно проводить доказательство различных утверждений для примитивно рекурсивных функций индукцией по шагу, на котором они получены. В связи со сказанным дадим другое, эквивалентное предыдущему, определение ПРФ.

Определение. Функция называется *примитивно рекурсивной,* если она может быть получена за конечное число шагов из простейших функций при помощи операций суперпозиции и примитивной рекурсии.

Свойства ПРФ:

1. Любая n-местная примитивно рекурсивная функция (ПРФ) всюду *определена* на множестве N_0^n.
2. Всякая *ПРФ* является алгоритмически вычислимой.
3. Следующие функции являются примитивно рекурсивными:

а) $f(x)=x+n$; б) $f(x)=n$;

в) $f(x)=nx$; г) $f(x)=x^2$;

д) $f(x,y)=x+y$; е) $f(x,y)=xy$;

ж) $f(x,y)=x^y$; з) $f(x,y)=x!$

и) $sg(x)=\begin{cases}1, & если \quad x>0\\ 0, & если \quad x=0\end{cases}$;

к) $f(x)=x\div 1=\begin{cases}x-1, если\ x>0,\\ 0, если\ x=0.\end{cases}$;

л) $f(x,y)=x\div y=\begin{cases}x-y, если\ x>y,\\ 0, если\ x\le y.\end{cases}$;

м) $f(x,y)=|x-y|=\begin{cases}x-y, если\ x\ge y,\\ y-x, если\ x<y.\end{cases}$

Докажем, что $f(x,y)=|x-y|=\begin{cases}x-y, если\ x\ge y,\\ y-x, если\ x<y.\end{cases}$ является примитивно рекурсивной функцией.

Для этого покажем, что $|x-y|=(x\div y)+(y\div x)$ (*).

1) $x>y$:

$|x-y|=x-y$, по определению модуля;

$(x\div y)+(y\div x)=x-y+0=x-y$, по определению усечённой разности. Видим, что левая часть равна правой части, т.о. для $x>y$ формула (*) верна.

2) $x=y$:

$|x-y|=x-x=0$, по определению модуля;

$(x\div y)+(y\div x)=x\div x+x\div x=0+0=0$, по определению усечённой разности. Т.о., при $x=y$ формула (*) верна.

3) $x<y$:

$|x-y|=y-x$, по определению модуля;

$(x\div y)+(y\div x)=0+(y-x)=y-x$, по определению усечённой разности. Т.к. левая часть равна правой, то формула (*) справедлива при $x<y$.

Из 1) – 3) следует, что $|x-y|=(x\div y)+(y\div x)$

Т. к. функции $f(x,y)=x+y$ и

$f(x,y)=x\div y=\begin{cases}x-y, если\ x>y,\\ 0, если\ x\le y.\end{cases}$ примитивно рекурсивные, следовательно, функция $f(x,y)=|x-y|$, являясь их суперпозицией, также является примитивно рекурсивной функцией.

2.5. Частично рекурсивные функции. Тезис Чёрча.

Определение. Функция называется *частично рекурсивной* (ЧРФ), если она может быть получена из простейших функций с помощью конечного числа применений операторов суперпозиции, примитивной рекурсии и минимизации.

Очевидно, что

- простейшие функции частично рекурсивны,
- если функция получена из частично рекурсивных функций с помощью операторов суперпозиции, примитивной рекурсии *или* минимизации, то она частично рекурсивна.

Очевидно, что *каждая примитивно рекурсивная функция является частично рекурсивной, но обратное неверно.* Действительно, с помощью оператора минимизации могут получаться не всюду определенные функции, а примитивно рекурсивные функции всюду определены, поэтому существуют частично рекурсивные функции, которые не являются примитивно рекурсивными.

Как следует из определения ЧРФ, всякой частично рекурсивной функции можно поставить в соответствие n - наименьший номер шага, на котором она может быть получена.

Пример: Рассмотрим функцию: $g(x,y)=x-sg(y)$, где $sg(y)=\begin{cases}0, если\ y=0,\\ 1, если\ y>0.\end{cases}$. $g(x,y)$ примитивно рекурсивна, т. к.

$sg(x)=\begin{cases}1, & если \quad x>0\\ 0, & если \quad x=0\end{cases}$ и $f(x,y)=x\div y=\begin{cases}x-y, если\ x>y,\\ 0, если\ x\le y.\end{cases}$ примитивно рекурсивные функции. Рассмотрим функцию $f(x)=\mu_y[x-sg(y)=0]$. Значения $f(x)=\mu_y[x-sg(y)=0]$ не определены при $x>1$. Наименьшее среди $y\in N\cup\{0\}$, удовлетворяющих $0-sg(y)=0$, будет равно 0, а наименьшее среди y, при которых $1-sg(y)=0$, равно 1. Следовательно, $f(0)=0$, $f(1)=1$, и *f(x)* не определена при $x>1$. Функция $g(x,y)=x-sg(y)$ примитивно рекурсивна, значит, $f(x)$ – частично рекурсивная.

Теорема. Всякая ЧРФ f является алгоритмически вычислимой.

Доказательство проведем индукцией по шагу n, на котором получена функция f.

Пусть $n=0$. Тогда f совпадает с одной из простейших функций, которые вычислимы.

Предположим, что для функций шага n утверждение верно. Пусть f – функция шага $n+1$. Тогда существует частично рекурсивные функции $g_1, g_2, ..., g_k$ шага n, из которых получена функция f одним из трех способов: с помощью оператора суперпозиции, примитивной рекурсии или минимизации. По предположению индукции частично рекурсивные функции $g_1, g_2, ..., g_k$ шага n вычислимы. Тогда и вычислима функция f.

Действительно, как отмечалось при определении операторов, каждый из них, будучи применен к вычислимым функциям, дает только вычислимые функции. Поэтому f – вычислимая функция. Что и требовалось доказать.

Фразу «функция f частично рекурсивна» будем заменять на более краткое словосочетание « f рекурсивна».

Определение. Всюду определенная частично-рекурсивная функция называется *общерекурсивной (ОРФ).*

Общерекурсивными являются все простейшие функции, а также функции $f(x,y)=x+y$; $f(x,y)=xy$; $f(x)=x+n$ и др.

Итак, класс формализованных указанным выше способом функций состоит из вычислимых функций. Возникает вопрос, а всякая ли вычислимая функция попадает в этот класс? Примера интуитивно вычислимой функции, не попавшей в указанный класс, не построено. И, более того, дальнейшие исследования в этом направлении позволили выдвинуть гипотезу о том, что таких примеров не существует.

Тезис Чёрча. Всякая вычислимая частично числовая функция является частично-рекурсивной функцией.

В формулировку тезиса Чёрча входит понятие эффективной вычислимости, поэтому его нельзя доказать в математическом смысле. Однако, тезис Чёрча является естественнонаучным фактом, который подтверждается опытом, накопленным математикой за весь период ее развития. Никто не сумел построить вычислимую функцию, рекурсивность которой нельзя было бы доказать, или хотя бы указать правдоподобный метод построения такой функции.

§3. Машины Тьюринга

3.1. Устройство машины Тьюринга

Точное описание класса частично рекурсивных функций вместе с тезисом Чёрча дает одно из возможных решений задачи об уточнении понятия алгоритма. Однако это решение не вполне прямое, так как понятие вычислимой функции является вторичным по отношению к понятию алгоритма. Возникает вопрос, нельзя ли уточнить непосредственно само

понятие алгоритма, а затем при его помощи определить и класс вычислимых функций?

Это было сделано в 1936-1937 гг. Э. Постом и А. Тьюрингом независимо друг от друга и почти одновременно с работами А. Чёрча и С.К. Клини. Основная мысль Э. Поста и А. Тьюринга заключалась в том, что алгоритмические процессы – это процессы, которые может совершать подходяще устроенная «машина». Это положение было сформулировано как тезис Тьюринга. Постом и Тьюрингом с помощью точных математических терминов были описаны довольно узкие классы машин. На этих машинах оказалось возможным осуществить или имитировать *все алгоритмические процессы, которые фактически когда-либо описывались математиками*. В этом смысле, такие машины являются самыми универсальными вычислительными устройствами. Причем это простейшие из таких устройств.

Машины, введенные Э. Постом и А. Тьюрингом, отличались не очень существенно и в дальнейшем стали называться машинами Тьюринга. Понятие машины Тьюринга (МТ) является строгим уточнением понятия алгоритма. Переход от интуитивного понятия алгоритма к точному понятию машины Тьюринга позволяет решить вопрос алгоритмической (машинной) разрешимости той или иной проблемы.

Рассмотрим алгоритмические системы, представленные этими машинами.

Машина Тьюринга есть математическая (воображаемая) машина, а не машина физическая. Она есть такой же математический объект, как функция, производная, интеграл, группа и так далее. *Устройство машины Тьюринга* включает в себя:

1. *Внешний алфавит*, то есть конечное множество символов $A=\{a_0,a_1,a_2,...,a_n\}$. В этом алфавите в виде слова кодируется та информация, которая подается в машину. Машина перерабатывает информацию, поданную в виде слова,

в новое слово.

2. *Внутренний алфавит* Q машины, состоящий из символов $q_0, q_1, q_2, ..., q_m$, R, L, W. Символы $q_0, q_1, q_2, ..., q_m$ выражают конечное число состояний машины. Для любой машины число состояний фиксировано. Два состояния имеют особое значение: q_1 – начальное состояние машины, q_0 – заключительное состояние (стоп-состояние). Символы R, L, W – это символы сдвига (вправо, влево, на месте).

3. *Программа* машины – это совокупность выражений $T(i, j)\ (i = \overline{1,m};\ j = \overline{0,n})$, каждое из которых имеет один из следующих видов: $q_i a_j \to q_k a_l W$, $q_i a_j \to q_k a_l R$, $q_i a_j \to q_k a_l L$, где $0 \le k \le m$, $0 \le l \le n$. Выражения $T(i, j)$ называют *командами*. Программа представляется в виде двумерной таблицы и называется Тьюринговой функциональной схемой.

4. *Бесконечная в обе стороны лента* (внешняя память машины). Она разбита на клетки (ячейки). В каждую клетку может быть записана только одна буква из внешнего алфавита A. Пустую клетку будем обозначать символом a_0. К ленте могут присваиваться ячейки в пустом состоянии.

5. *Механическое устройство* пристраивает к ленте по соответствующим командам дополнительные ячейки и передвигает управляющую головку.

6. *Управляющая головка*. Она управляет процессом преобразования машинных слов в машине Тьюринга. В каждый конкретный момент времени управляющая головка воспринимает одну и только одну ячейку ленты. По соответствующим командам она может передвигаться влево или вправо, менять содержимое ячейки. Если управляющая головка попала в крайнее правое (левое) положение, то механическое устройство присваивает справа (слева) ячейку в пустом состоянии.

3.2. Конфигурация машин Тьюринга. Работа машины Тьюринга

Машинным словом (конфигурацией) называется всякое слово вида $M = Cq_i a_j B\,(0 \le i \le n, 0 \le j \le m)$, где C, B – некоторые слова в алфавите A (возможно пустые).

Для машинного слова M и машины Тьюринга T через M'_T обозначим слово, которое получается по следующим правилам.

Рассмотрим два случая:

1. $i = 0$. В данном случае $M'_T = M$, так как q_o – стоп-состояние.

2. $i > 0$. Тогда:

а) $T(i, j) = q_i a_j \to q_k a_l W$ и машинное слово M'_T имеет вид: $M'_T = Cq_k a_l B$

б) $T(i, j) = q_i a_j \to q_k a_l R$

если $B \neq \emptyset$ и $B = a_s B_1$, то $M'_T = Ca_l q_k a_s B_1$	если $B = \emptyset$, то $M'_T = Ca_l q_k a_0$

в) $T(i, j) = q_i a_j \to q_k a_l L$

если $C \neq \emptyset$ и $C = C_1 a_s$ то $M'_T = C_1 q_k a_s a_l B$,	если $C = \emptyset$, то $M'_T = q_k a_0 a_l B$

Замечание. Иногда для удобства в программе символ W (отсутствие сдвига) будем опускать, т.е. команды $T(i, j) = q_i a_j \to q_k a_l W$ и $T(i, j) = q_i a_j \to q_k a_l$ суть одно и тоже.

Состояние машины Тьюринга полностью определяется

машинным словом: $M = Cq_i a_j B$, где $Ca_j B$ – состояние ленты; q_i – внутреннее состояние машины.

Работа машины Тьюринга – это пошаговый переход от одного состояния к другому (от одного машинного слова к другому); и за один шаг мы переходим от M к M'_T.

МТ удобно представлять в виде автоматически работающего устройства. В каждый дискретный момент времени устройство, находясь в некотором состоянии, обозревает содержимое одной ячейки, протягиваемой через устройство ленты, и делает шаг, заключающийся в том, что устройство переходит в новое состояние, изменяет (или оставляет без изменения) содержимое обозреваемой ячейки и переходит к обозрению следующей ячейки – справа или слева. Причем шаг осуществляется на основании предписанной команды. Таким образом, работа машины Тьюринга полностью определяется ее программой. Иными словами, две машины Тьюринга с общей функциональной схемой неразличимы, и различные машины Тьюринга имеют различные программы.

Рассмотрим результат работы МТ. Заметим, что в качестве начальной информации A на ленту можно подать любую конечную систему знаков внешнего алфавита, расставленную произвольным образом по ячейкам. Но в зависимости от того, какая была подана начальная информация, возможны два случая:

1) После конечного числа тактов машина останавливается и при этом на ленте оказывается изображенной информация B. В таком случае говорят, что машина применима к начальной информации A и перерабатывает ее в результирующую информацию B.

2) Машина никогда не останавливается. В таком случае говорят, что машина не применима к начальной информации A.

Пример 1. Реализация в машине Тьюринга алгоритма перехода от n к $n+1$ (т.е. реализация функции $\lambda(x)$).

1) в десятичной системе счисления, если начальное слово записано в последовательных ячейках ленты, все другие ячейки пусты и машина обозревает крайнюю справа ячейку из тех, в которых записано начальное слово.

	a_0	0	1	2	3	4	5	6	7	8	9
q_1	$1\,q_0$	$1\,q_0$	$2\,q_0$	$3\,q_0$	$4\,q_0$	$5\,q_0$	$6\,q_0$	$7\,q_0$	$8\,q_0$	$9\,q_0$	$0\,Lq_1$

Пусть начальная конфигурация имеет вид $a_0 39 q_1 9 a_0$, тогда, используя указанную функциональную схему, получим:

$$a_0 39 q_1 9 a_0 \vdash a_0 3 q_1 90 a_0 \vdash a_0 q_1 300 a_0 \vdash a_0 q_0 400 a_0$$

2) в алфавите $A = \{0,1\}$

	0	1
q_1	$q_2 0R$	$q_1 1R$
q_2	$q_0 1$	$q_2 1R$

(*)

или

$q_1 0 \to q_2 0R$, $q_1 1 \to q_1 1R$,

$q_2 0 \to q_0 1$, $q_2 1 \to q_2 1R$.

Пусть начальная конфигурация имеет вид $q_1 110$, тогда используя функциональную схему (*) мы придем к следующим конфигурациям:

$$q_1 110 \vdash 1\,q_1 10 \vdash 11\,q_1 0 \vdash 110\,q_2 0 \vdash 110\,q_0 1.$$

Если начальная конфигурация имеет вид $q_1 0110$, тогда получим следующие конфигурации:

$$q_1 0110 \vdash 0\,q_2 110 \vdash 01\,q_2 10 \vdash 011\,q_2 0 \vdash 011\,q_0 1.$$

Пример 2. Задается машина Тьюринга с внешним алфавитом $A = \{a,b,c\}$, алфавитом внутренних состояний $Q = \{q_1, q_2, q_3, q_4\}$ и программой:

$$q_1 a \to q_1 aL, q_2 a \to q_3 bR, q_3 a \to q_1 aL, q_1 b \to q_2 aL, q_2 b \to q_2 bL,$$

$$q_3 b \to q_3 bR, q_1 c \to q_0 a, q_2 c \to q_2 cL, q_2 a_0 \to q_2 a, q_3 c \to q_3 cR$$

Заметим, что программа этой машины может быть

записана в виде следующей таблицы:

A \ Q	q_1	q_2	q_3
a_0		q_2a	
a	q_1aL	q_3bR	q_1aL
b	q_2aL	q_2bL	q_3bR
c	q_0a	q_2cL	q_3cR

Для того чтобы определить по таблице, что будет делать машина, находясь, например, в состоянии q_2 и наблюдая в обозреваемой ячейке символ b, нужно найти в таблице клетку, находящуюся на пересечении столбца q_2 и строки b. В этой клетке записано q_2bL. Это означает, что на следующем шаге машина останется в прежнем состоянии q_2, сохранит содержимое обозреваемой ячейки b и перейдет к обозрению следующей левой ячейки на ленте.

Предположим, что в начальной конфигурации головка находится над крайней правой клеткой. Применим эту машину к слову $bbcbb$. Последовательность конфигураций, возникающих в процессе работы машины (исходная конфигурация – стандартная начальная):

$$bbcbq_1b \vdash bbcq_2ba \vdash bbq_2cba \vdash bq_2bcba \vdash q_2bbcba \vdash q_2abbcba \vdash bq_3bbcba \vdash bbq_3bcba \vdash bbbq_3cba \vdash bbbcq_3ba \vdash bbbcbq_3a \vdash bbbcq_1ba \vdash bbbq_2caa \vdash bbq_2bca \vdash bq_2bbca \vdash q_2bbbca \vdash q_2abbbca \vdash bq_3bbbca \vdash bbq_3bbca \vdash bbbq_3bca \vdash bbbbq_3ca \vdash bbbbcq_3a \vdash bbbbq_1ca \vdash bbbbq_0aa.$$

Нетрудно заметить, что данная МТ реализует операцию сложения: в результате ее работы на ленте записано подряд столько букв b, сколько их было всего записано по обе стороны от буквы c перед началом работы машины.

Пример 3. Машина Тьюринга с внешним алфавитом $A=\{0,1\}$ определятся следующей программой:

A \ Q	q_1	q_2	q_3
a_0	q_2a_0R	q_2a_0R	q_0a_0
1	q_11R	q_31R	q_31R

Остановится ли когда-нибудь эта машина, если она начнёт перерабатывать слово 11 (в начальный момент машина обозревает ячейку, в которой записана самая левая буква перерабатываемого слова)?

$$q_111 \vdash 1q_11 \vdash 11q_10 \vdash 110q_20 \vdash 1100q_20 \vdash 11000q_20 \vdash \ldots$$

Видим, что машина никогда не остановится.

3.3. Универсальная машина Тьюринга

До сих пор предполагалось, что различные алгоритмы осуществляются на различных машинах Тьюринга, отличающихся набором команд, внутренним и внешним алфавитами. Однако можно построить универсальную машину Тьюринга, способную в известном смысле выполнять любой алгоритм, а значит, способную выполнять работу любой машины Тьюринга.

В универсальной машине Тьюринга, как и во всякой машине Тьюринга, информация изображается символами, расположенными одновременно на магнитной ленте. При этом универсальная машина Тьюринга может располагать лишь *фиксированным конечным внешним алфавитом*. Между тем она должна быть приспособлена к приему в качестве исходной информации всевозможных состояний устройства управления и конфигураций, в которых могут встречаться буквы из разнообразных алфавитов со сколь угодно большим числом различных букв.

Это достигается путем кодирования конфигурации и программы любой данной машины Тьюринга в символах входного (внешнего) алфавита универсальной машины. Само кодирование должно выполняться следующим образом:

1) различные буквы должны заменяться различными кодовыми группами, но одна и та же буква должна заменяться всюду, где бы она ни встречалась, одной и той же кодовой группой;

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

3) должна иметь место возможность распознать, какие кодовые группы соответствуют различным сдвигам, то есть каждой из букв R, L, W в отдельности, и различать кодовые группы, соответствующие буквам внутреннего алфавита и буквам внешнего алфавита.

Рассмотрим пример такого кодирования для машины Тьюринга T, имеющей внешний алфавит $A=\{a_0, a_1, ..., a_k\}$ и внутренний алфавит $Q=\{q_0, q_1, ..., q_m\}$.

Если внешний алфавит универсальной машины Тьюринга состоит из символов $A=\{0,1\}$, то эти условия будут наверняка соблюдены при следующем способе кодирования.

1. В качестве кодовых групп берутся $3+k+m$ различных слов вида 100...01 (между единицами – сплошь нули), где k – число символов внешнего алфавита; m – число состояний устройства управления.

Тогда разбивка строк на кодовые группы производится однозначным образом путем выделения последовательностей нулей, заключенных между двумя единицами.

2. Сопоставление кодовых групп исходным символам внешнего и внутреннего алфавитов осуществляется согласно следующей таблице кодирования:

	Буква	Кодовая группа	
	L W R	101 1001 10001	
Внешний алфавит	a_0	100001 - 4 нуля 10000001 – 6 нулей	Четное число нулей,

	a_1 ... a_k	…… 10…01 – $2k+4$ нулей	большее 2
Внутренний алфавит (состояния)	q_0 q_1 .... q_m	1000001 - 5 нулей 100000001 – 7 нулей ……. 10…01 – $2m+5$ нулей	Нечетное число нулей, большее 3

Так, например, для машины Тьюринга, перерабатывающей слова $bcadc$ в слово $bcdcc$, входное слово в универсальной машине Тьюринга с данным кодом будет представлено следующей строкой:

10000001 1000000001 100001 100000000001 1000000001,

где 100001 – a, 10000001 – b, 1000000001 – c, 100000000001 – d.

Программа же будет представлена следующими строками:

1000001 10000001 1000001 10000001 101 ($q_0b \to bLq_0$);

1000001 1000000001 1000001 1000000001 101 ($q_0c \to cLq_0$);

1000001 100001 100000001 100001 101 ($q_0a \to aLq_1$);

100000001 100000000001 10000000001 1000000001 10001 ($q_1d \to cRq_2$);

10000000001 100001 100000000000001 100000000001 10001 ($q_2a \to dRq_5$).

При кодировании программ, конфигураций и входных слов вместо нулей и единиц можно брать любые другие два символа, например a и b.

Таким образом, если какая-либо машина Тьюринга T решает некоторую задачу, то и универсальная машина Тьюринга способна решить эту задачу при условии, что кроме кодов исходных данных этой задачи на ее ленту будет подан код программы машины T. Задавая универсальной машине

Тьюринга T_u изображение программы любой данной машины Тьюринга T_n и изображение любого ее входного слова x_n, получим изображение выходного слова y_n, в которое машина T_n переводит слово x_n.

Если же алгоритм, реализуемый машиной T_n, не применим к слову x_n, то алгоритм, реализуемый универсальной машиной T_u, также не применим к слову, образованному из изображения x_n и программы машины T_n.

Таким образом, машина Тьюринга T_n может рассматриваться как одна из программ для универсальной машины T_u.

В связи с существованием универсальной тьюринговой машины таблицы соответствия, описывающие различные состояния устройства управления машины, имеют двоякое назначение:

1) для описания состояний устройства управления специальной машины Тьюринга, реализующей соответствующий алгоритм;

2) для описания программы, подаваемой на ленту универсальной машины Тьюринга, при реализации соответствующего алгоритма.

Современные ЭВМ строятся как универсальные; в запоминающее устройство наряду с исходными данными поставленной задачи вводится также и программа ее решения. Однако в отличие от машины Тьюринга, в которой внешняя память (лента) бесконечна, в любой реальной вычислительной машине она конечна.

3.4. Сложность машины Тьюринга.

Для сравнения структур различных машин и оценки их сложности необходимо иметь соответствующую меру величины или сложности машин. Шеннон предложил рассматривать в качестве такой меры произведение числа символов и числа состояний. Таким образом, «сложность» машины Тьюринга равна произведению числа символов внешнего алфавита на число внутренних состояний, отличных от заключительного. Значительный интерес вызвала задача построения универсальных машин Тьюринга минимальной сложности.

К началу 1963 г. последними результатами в этом направлении были теорема Ватанабе о существовании универсальной машины с 5-ю символами и 6-ю состояниями, теорема Минского о существовании машины с 4-мя символами и 7-ю состояниями и теорема Триттера о существовании универсальной машины с 4-мя символами и 6-ю состояниями.

3.5. Композиция машин Тьюринга.

Определение. Пусть T_1 и T_2 – машины Тьюринга с внешними алфавитами A_1 и A_2, внутренними алфавитами Q_1 и Q_2 соответственно. Причем внутренние алфавиты не пересекаются, то есть $Q_1 \cap Q_2 = \varnothing$. Тогда композицией (произведением) машин $T_1 \circ T_2$ называется машина с внешним алфавитом $A_1 \cup A_2$, алфавитом внутренних состояний $(Q_1 \cup Q_2) \setminus q_0^1$ и работающая по правилу: $(T_1 \circ T_2)(U) = T_2(T_1(U))$, где U – некоторое слово и q_0^1 – стоп-состояние машины T_1.

Теорема. Композиция машин Тьюринга существует.

Доказательство.

Пусть заданы машины Тьюринга T_1 и T_2, имеющие внешние алфавиты A_1 и A_2 и внутренние алфавиты $Q_1 = \{q_0, q_1, ..., q_n\}$ и $Q_2 = \{q'_0, q'_1, ..., q'_t\}$.

В качестве композиции этих машин рассмотрим машину $T = T_2(T_1)$ с внешним алфавитом $A = A_1 \cup A_2 = \{a_0, a_1, ..., a_m\}$, внутренним алфавитом $Q = \{q_0, q_1, ..., q_n, q_{n+1}, ..., q_{n+t}\}$ и программой, получающейся следующим образом. Во всех командах T_1, содержащих заключительный символ q_0, заменяем его на символ q_{n+1}. Все остальные символы в командах T_1 оставляем неизменными. В командах T_2, напротив, символ q_0 оставляем неизменным, но зато каждый из остальных символов q'_j ($j = 1,2,...,t$) заменяем символом q_{n+j}. Совокупность всех команд T_1 и T_2, измененных указанным способом, и будет программой композита или произведения машин T_1 и T_2.

Что и требовалось доказать.

Заметим, что машина T есть произведение машин T_1 и T_2, если последовательная работа этих двух машин эквивалентна работе одной машины T.

Пример. Пусть T_1 машина, складывающая числа в унарной системе счисления, T_2 – машина Тьюринга, удваивающая числа, записанные в унарной системе счисления. Тогда $T_2 \circ T_1$ - машина, проводящая вычисления по формуле $2(a+b)$.

Пусть T_1 – МТ с внешним алфавитом $A_1 = \{|, +, a_0\}$, внутренним алфавитом $Q_1 = \{q_1, q_2, q_3\}$ и T_2 – МТ с внешним алфавитом $A_2 = \{|, \alpha, a_0\}$, внутренним алфавитом $Q_2 = \{q_4, q_5, q_6\}$.

Программа такой композиции машин может выглядеть так:

$(Q_1 \cup Q_2) \setminus q_0^1$ / $A_1 \cup A_2$	q_1	q_2	q_3	q_4	q_5	q_6
|	$q_1 \mid R$	$q_3 a_0 L$	$q_3 \mid L$	$q_4 \alpha R$	$q_5 \mid L$	$q_6 \mid R$
+	$q_1 \mid R$					
a_0	$q_2 a_0 L$		$q_4 a_0 R$	$q_5 a_0 L$	$q_0 a_0 R$	$q_5 \mid L$
α					$q_6 \mid R$	

3.6. Вычислимость по Тьюрингу. Тезис Тьюринга.

Определение. ЧЧФ называется *вычислимой по Тьюрингу*, если существует машина Тьюринга, вычисляющая эту функцию.

Будем говорить, что *машина Тьюринга* T *вычисляет* n-местную частичную числовую функцию f с областью определения $X_f \subseteq N_0^n$ и множеством значений $I_f \subseteq N_0$, если выполнены следующие условия:

1) если $(x_1, x_2, ..., x_n) \in X_f$, то в этом случае машина Тьюринга начинает работу со слова $M = q_1 01^{x_1} 01^{x_2} 0 ... 01^{x_n} 0$ и перерабатывает его в слово Cq_0B, где 0 – разделители; C и B – некоторые слова в алфавите $\{0,1\}$, причем слово Cq_0B содержит $f(x_1, x_2, ..., x_n)$ вхождений символа «1».

2) если $(x_1, x_2, ..., x_n) \notin X_f$, то машина Тьюринга начинает работу со слова $M = q_1 01^{x_1} 01^{x_2} 0 ... 01^{x_n} 0$. В этом случае машина Тьюринга работает вечно, то есть $(\forall n \in N)\ (q_0 \notin M_T^{(n)})$.

Тезис Тьюринга. Класс вычислимых частично числовых функций совпадает с классом функций, вычислимых по Тьюрингу.

Как и тезис Чёрча, данное утверждение невозможно доказать в математическом смысле.

Приведем без доказательства теорему, связывающую понятия частично рекурсивной функции и функции вычислимой по Тьюрингу.

Теорема: Частичная числовая функция является частично рекурсивной тогда и только тогда, когда она вычислима по Тьюрингу.

Данная теорема позволяет сделать вывод об эквивалентности определений понятия алгоритма в форме рекурсивной функции и в форме машины Тьюринга. Кроме того, она является косвенным подтверждением тезиса Черча и эквивалентного ему тезиса Тьюринга.

Пример. Доказать, что функция $f(x)=2x$ вычислима по Тьюрингу, для чего постройте машину Тьюринга, вычисляющую её.

Нужно от слова $01^{x-1}q_110$ перейти к слову $01^{2x}q_00$.

Построим программу:

	q_1	q_2	q_3	q_4
0	$q_3\alpha R$	$q_3\alpha R$	$q_3\alpha R$	q_00
1	$q_2\beta L$	q_21L	q_31R	
α	$q_1\alpha L$	$q_2\alpha L$	$q_3\alpha R$	q_41R
β			$q_1\beta L$	q_41R

Проверим: $01^{x-1}q_110 \vdash 01^{x-2}q_21\beta0 \vdash \ldots \vdash q_201^{x-1}\beta0 \vdash \alpha q_31^{x-1}\beta0 \vdash \ldots \vdash \alpha1^{x-1}q_3\beta0 \vdash \alpha1^{x-2}q_11\beta0 \vdash \alpha1^{x-3}q_21\beta\beta0 \vdash \ldots \vdash q_2\alpha1^{x-2}\beta\beta0 \vdash q_20\alpha1^{x-2}\beta\beta0 \vdash \alpha q_3\alpha1^{x-2}\beta\beta0 \vdash \ldots \vdash q_10\alpha^x\beta^x0 \vdash 0q_4\alpha^x\beta^x0 \vdash 01q_4\alpha^{x-1}\beta^x0 \vdash \ldots \vdash 01^xq_4\beta^x0 \vdash 01^x1q_4\beta^{x-1}0 \vdash \ldots \vdash 01^x1^xq_40 \vdash 01^{2x}q_00.$

§4. Алгоритмы Маркова

4.1. Алфавит и понятие слова

В середине прошлого века выдающийся русский математик А.А. Марков ввел понятие *нормального алгоритма (алгорифма)* с целью уточнения понятия «алгоритм». Позже это понятие получило название нормального алгоритма Маркова (НАМ). НАМ представляет собой некоторые правила по переработке слов в каком-либо алфавите, так что исходные данные и искомые результаты для НАМ являются словами в некотором алфавите.

Алфавитом называется любое непустое множество символов, его элементы называются буквами.

В алфавит также включается пустой символ, который мы будем обозначать греческой буквой Δ. Под словом понимается любая последовательность непустых символов алфавита либо пустой символ, который обозначает пустое слово.

Если X и $\overline{X}$ – два алфавита, причем $X \subset \overline{X}$, то алфавит $\overline{X}$ называется расширением алфавита X.

Слова будем обозначать латинскими буквами: $B, C, P, \ldots$ (или этими же буквами с индексами). Одно слово может быть составной частью другого слова. Тогда первое называется подсловом второго или вхождением во второе.

4.2. Марковская подстановка

Марковской подстановкой называется операция над словами, задаваемая с помощью упорядоченной пары слов (B, C), состоящая в следующем. В заданном слове P находят первое вхождение слова B (если таковое имеется) и, не изменяя остальных частей слова P, заменяют в нем это вхождение словом C. Полученное слово называется

результатом применения марковской подстановки (B, C) к слову P. Если же первого вхождения B в слово P нет (и, следовательно, вообще нет ни одного вхождения B в P), то считается, что марковская подстановка (B, C) неприменима к слову P.

Частными случаями марковских подстановок являются подстановки с пустыми словами: (Δ, C), (B, Δ), (Δ, Δ).

Для обозначения марковской подстановки (B, C) используется запись $B \to C$. Она называется формулой подстановки (B, C). Некоторые подстановки (B, C) будем называть заключительными и обозначать их формулой заключительной подстановки $B \to C^*$. Слово B называется левой частью, а C – правой частью в формулах подстановки.

4.3. Схема нормального алгоритма Маркова

Упорядоченный конечный список формул подстановок

$$\begin{cases} B_1 \to C_1\alpha_1, \\ B_2 \to C_2\alpha_2, \\ \ldots\ldots\ldots\ldots, \\ B_s \to C_s\alpha_s, \end{cases} \qquad (1)$$

где $\alpha_i \in \{\Delta, *\}$, в алфавите X называется *схемой нормального алгоритма*. Данная схема определяет алгоритм преобразования слов, называемый нормальным алгоритмом Маркова.

Пусть нам дан некоторый алфавит X и схема нормального алгоритма (1), тогда нормальным алгоритмом Маркова в алфавите X, определенным схемой (1), будем называть описываемый ниже процесс построения последовательности слов A_i $(i = 0,1,\ldots)$, исходя из данного слова A. Опишем этот процесс:

1. Если $i = 0$, то $A_0 = A$. В данном случае считаем процесс построения последовательности слов незавершенным.

2. Если $i \geq 0$, то слова $A_0, A_1, ...A_i$ построены и

а) если каждая из подстановок схемы (1) не применима к слову A_i, то полагаем:

- $A_{i+1} = A_i$,
- процесс построения слов считаем завершенным
- A_{i+1} – результат применения нормального алгоритма Маркова к слову A;

б) если среди подстановок схемы (1) есть применимые к слову A_i, то:

- A_{i+1} – слово, полученное из A_i применением *первой* применимой к нему подстановки из схемы (1),
- если подстановка была заключительной, то процесс построения слов считается завершенным и A_{i+1} – результат применения нормального алгоритма Маркова к слову A,
- если же подстановка не является заключительной, то процесс построения слов считается незавершенным.

Таким образом, если нормальный алгоритм Маркова, применимый к слову A, завершается на слове B, то говорят, что данный алгоритм перерабатывает слово A в слово B. Если же нормальный алгоритм Маркова никогда не завершается, то говорят, что данный алгоритм не применим к слову A. Запись $A_i \Rightarrow A_{i+1}$ будет означать, что слово A_i переработано в слово A_{i+1}.

Как и машина Тьюринга, нормальные алгоритмы не производят собственно вычислений: они лишь производят преобразования слов, заменяя в них одни буквы другими по предписанным им правилам. В свою очередь, мы предписываем им такие правила, результаты применения которых мы можем интерпретировать как вычисления.

Пример:

Дана функция

$$\varphi_3(11...1)=\begin{cases}1, \textit{ если } n \textit{ делится на } 3,\\ \Delta, \textit{ если } n \textit{ не делится на } 3,\end{cases}$$

где n – число единиц в слове 11…1. Рассмотрим нормальный алгоритм в алфавите $A=\{1\}$ со следующей схемой:

$$\begin{cases}111\to\Delta\\ 11\to\Delta*\\ 1\to\Delta*\\ \Delta\to 1*\end{cases}$$

Этот алгоритм записывается по следующему принципу: пока число букв 1 в слове не меньше 3, алгоритм последовательно стирает по три буквы. Если число букв меньше 3, но больше нуля, то оставшиеся буквы 1 или 11 стираются заключительно; если слово пустое, оно заключительно переводится в слово 1. Например:

$$1111111\Rightarrow 1111\Rightarrow 1\Rightarrow\Delta;$$
$$111111111\Rightarrow 111111\Rightarrow 111\Rightarrow\Delta\Rightarrow 1.$$

Таким образом, рассмотренный алгоритм реализует (или вычисляет) данную функцию.

4.4. Работа нормального алгоритма Маркова

Пусть дано преобразуемое слово – цепочка символов фиксированного алфавита и нормальная схема подстановок, содержащая фиксированную последовательность простых и заключительных подстановок. Механизм работы нормального алгоритма состоит в следующем:

1) Слово всегда просматривается слева направо. Схема подстановок просматривается, начиная с первой подстановки, и, если подстановку можно применить, то она применяется к самому левому вхождению указанного в подстановке левого слова в заданном слове.

2) Работа алгоритма заканчивается если:

- ни одна из подстановок не применима,

- использована заключительная подстановка.

Может возникнуть ситуация, когда процесс не закончится никогда. В этом случае считают, что алгоритм не применим к слову.

Пример:

Пусть имеем алфавит $X = \{x, y, z\}$ и нормальную схему подстановок:

$$\begin{cases} xx \to y \\ xy \to x \\ yzy \to x \\ zz \to z* \\ yy \to x \end{cases}$$

Процесс построения последовательности слов для данного слова $\underline{xx}xyyyzzz$ имеет вид:

$$\underline{xx}xyyyzzz \to y\underline{xy}yyzzz \to y\underline{xy}yzzz \to y\underline{xy}zzz \to yx\underline{zz}z \to yxzz.$$

4.5. Нормальная вычислимость функций

Частичная функция f, заданная на множестве слов в алфавите X, называется ***нормально вычислимой***, если найдется такое расширение $\overline{X}$ ($\overline{X} \supseteq X$) алфавита X и такой НАМ, который всякое слово A из области определения функции f перерабатывает в слово $f(A)$, то есть $A \Rightarrow f(A)$ и который неприменим к словам не входящим в область определения функции f.

Пример: 1. В алфавите $\{1\}$ схема НАМ $\Delta \to 1*$ нормально вычисляет функцию $f(x) = x + 1$.

Пример: 2. Покажем, что функция $f(x, y) = x + y$ нормально вычислима. Для этого предъявим схему НАМ вычисляющую ее.

В алфавите $X = \{0,1\}$, где 0 – разделитель. Схема НАМ

имеет вид:

$$\begin{cases} 0 \to \Delta, \\ \Delta \to \Delta *. \end{cases}$$

Действительно, пусть $x = 2, y = 4$. Тогда начальное слово $\Delta 1101111\Delta$ перерабатывается в конечное слово $\Delta 111111\Delta$, а это и есть значение функции $f(x, y) = x + y$ при $x = 2, y = 4$.

4.6. Принцип нормализации Маркова

Создатель теории нормальных алгоритмов советский математик А. А. Марков выдвинул гипотезу, подобную тезисам Черча и Тьюринга. Она получила название «Принцип нормализации Маркова». Согласно этому принципу, *частичная числовая функция является вычислимой тогда и только тогда, когда она является нормально вычислимой*.

Теорема. Класс всех нормально вычислимых функций совпадает с классом всех функций, вычислимых по Тьюрингу.

Действительно, пусть машина Тьюринга с внешним алфавитом $A = \{a_0, a_1, ..., a_m\}$ и алфавитом внутренних состояний $Q = \{q_0, q_1, ..., q_n\}$ вычисляет некоторую функцию f, заданную и принимающую значения во множестве слов алфавита A. Попытаемся теперь представить программу этой машины Тьюринга в виде схемы некоторого нормального алгоритма. Для этого нужно каждую команду машины Тьюринга $q_i a_j \to q_k a_l X$ представить в виде совокупности марковских подстановок. Конфигурации, возникающие в машине Тьюринга в процессе ее работы, представляют собой слова в алфавите $A \cup Q$. Эти слова имеют вид: $a_{j_1} ... a_{j_k} q_i a_{j_{k+1}} ... a_j$. Нам понадобиться различать начало слова и его конец (или его левый и правый концы). Для этого к алфавиту $A \cup Q$ добавим еще два символа (не входящие ни в

A, ни в Q): $A \cup Q \cup \{u, v\}$. Эти символы будут ставить соответственно в начало и конец каждого машинного слова w: uwv.

Пусть на данном шаге работы машины Тьюринга к машинному слову w предстоит применить команду $q_i a_j \to q_k a_l X$. Это означает, что машинное слово w (а вместе с ним и слово $q_i a_j \to q_k a_l X$) содержит подслово $q_i a_j$. Посмотрим, какой совокупностью марковских подстановок можно заменить данную команду в каждом из следующих трех случаев:

1) Пусть машина Тьюринга остается на месте, то есть команда имеет вид: $q_i a_j \to q_k a_l$. Ясно, что в этом случае следующее слово получается из слова uwv с помощью подстановки $q_i a_j \to q_k a_l$, которую мы и будем считать соответствующей команде $q_i a_j \to q_k a_l$;

2) Пусть теперь $X = L$, то есть команда имеет вид: $q_i a_j \to q_k a_l L$. Нетрудно понять, что в этом случае для получения из слова uwv следующего слова надо к слову uwv применить ту подстановку из совокупности

$$a_0 q_i a_j \to q_k a_0 a_l;$$
$$a_1 q_i a_j \leftarrow q_k a_1 a_l;$$
$$\ldots\ldots\ldots\ldots$$
$$a_m q_i a_j \to q_k a_m a_l;$$
$$u q_i a_j \to u q_k a_0 a_l,$$

которая применима к слову uwv. В частности, последняя подстановка применима только тогда, когда q_i — самая левая буква в слове w, то есть когда надо пристраивать ячейку слева;

3) И, наконец, пусть $X = \mathrm{R}$, то есть команда имеет вид: $q_i a_j \to q_k a_l \mathrm{R}$. В этом случае аналогично, чтобы получить из

слова uwv следующее слово, нужно к слову uwv применить ту из подстановок совокупности

$$q_i a_j a_0 \to a_l q_k a_0;$$
$$q_i a_j a_1 \leftarrow a_l q_k a_1;$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$q_i a_j a_m \to a_l q_k a_m;$$
$$q_i a_j v \to a_l q_k a_0 v,$$

которая применима к слову uwv.

Поскольку слово $q_i a_j$ входит в слово w только один раз, то к слову uwv применима только одна из подстановок, перечисленных в пунктах 2) и 3). Поэтому порядки следования подстановок в этих пунктах безразличны, важны лишь их совокупности.

Заменим каждую команду из программы машины Тьюринга указанным способом совокупностью марковских подстановок. Мы получим схему некоторого нормального алгоритма. Теперь ясно, что применить к слому w данную машину Тьюринга – это все равно, что применить к слову uwv построенный нормальный алгоритм. Другими словами, действие машины Тьюринга равнозначно действию подходящего нормального алгоритма. Это и означает, что всякая функция, вычислимая по Тьюрингу, нормально вычислима.

А. А. Марковым так же доказано, что класс нормально вычислимых функций совпадает с классом частично рекурсивных функций (и, следовательно, с классом вычислимых по Тьюрингу функций). Из этого результата вытекает эквивалентность принципа нормализации Маркова тезисам Черча и Тьюринга. Таким образом, имеет место следующее утверждение:

Теорема. Следующие классы функций (заданных на

множестве N_0^n и принимающих натуральные значения) совпадают:

а) класс всех функций, вычислимых по Тьюрингу;

б) класс всех частично рекурсивных функций;

в) класс всех нормально вычислимых функций.

Пример: Составьте нормальный алгоритм в трёхэлементном расширении $B = A \cup \{a, b, c\}$ основного алфавита $A = \{0,1,2,3,4,5,6,7,8,9\}$, вычисляющий функцию $f(x) = 2x$.

Вначале осуществим переход $w \to aw \to wa \to wb$, а затем приступим к поразрядному умножению.

Таким образом, нормальный алгоритм будет задаваться следующей схемой:

$\Delta \to a$	$0a \to 0b$	$0b \to b0$	$0c \to b1$	$b \to \Delta^*$
$a0 \to 0a$	$1a \to 1b$	$1b \to b2$	$1c \to b3$	$c \to 1^*$
$a1 \to 1a$	$2a \to 2b$	$2b \to b4$	$2c \to b5$	
$a2 \to 2a$	$3a \to 3b$	$3b \to b6$	$3c \to b7$	
$a3 \to 3a$	$4a \to 4b$	$4b \to b8$	$4c \to b9$	
$a4 \to 4a$	$5a \to 5b$	$5b \to c0$	$5c \to c1$	
$a5 \to 5a$	$6a \to 6b$	$6b \to c2$	$6c \to c3$	
$a6 \to 6a$	$7a \to 7b$	$7b \to c4$	$7c \to b5$	
$a7 \to 7a$	$8a \to 8b$	$8b \to c6$	$8c \to b7$	
$a8 \to 8a$	$9a \to 9b$	$9b \to c8$	$9c \to b9$	
$a9 \to 9a$				

Проверим:

1) $\underline{\Delta}24 \to \underline{a}\underline{2}4 \to 2\underline{a}\underline{4} \to 24\underline{a} \to 24\underline{b} \to 2\underline{b}8 \to \underline{b}\underline{4}8 \to \Delta 48$;

2) $\underline{\Delta}39 \to \underline{a}\underline{3}9 \to 3\underline{a}\underline{9} \to 39\underline{a} \to 39\underline{b} \to 3\underline{c}8 \to \underline{b}78 \to \Delta 78$;

3) $\underline{\Delta}185 \to \underline{a}\underline{1}85 \to 1\underline{a}\underline{8}5 \to 18\underline{a}\underline{5} \to 185\underline{a} \to 185\underline{b} \to 18\underline{c}0 \to$

$\to 1\underline{c}70 \to \underline{b}\underline{3}70 \to \Delta 370$.

§5. Нумерации

5.1. Геделевская нумерация

Теория нумераций – раздел теории алгоритмов, в котором изучаются свойства классов объектов, занумерованных с помощью натуральных чисел. Присвоив номер объектам (машинам Тьюринга, НАМ, частично рекурсивным функциям и т.д.), мы можем во многих случаях получить новые свойства этих объектах. Идея использования нумерации объектов для получения различных утверждений об этих объектах принадлежит К. Геделю.

Пусть M – произвольное непустое конечное или счетное множество.

Определение. Однозначной нумерацией множества M будем называть взаимно-однозначное отображение φ множества натуральных чисел N во множество M.

Если $\varphi(n) = a$, то натуральное число n называют номером элемента $a \in M$ при нумерации φ. Обозначим элемент a с номером n через a_n, тогда элементы занумерованного множества M можно представить в виде последовательности $a_1, a_2, a_3, \ldots$.

Если на множестве введена однозначная нумерация, то оно является счетным множеством.

Определение. Множество M называется эффективно счетным множеством, если существует однозначная нумерация $\varphi : N \to M$ такая, что выполнены следующие два условия:

1) Существует алгоритм $A1$, вычисляющий по элементу $a \in M$ его номер $\varphi(a) \in N$.

2) Существует алгоритм $A2$, устанавливающий по номеру $n \in N$ элемент $a \in M$ с данным номером.

Нумерация φ называется в этом случае геделевской нумерацией. Условия 1) и 2) можно заменить на условия:

1) Функция $\varphi : N \to M$ вычислима.

2) Функция $\varphi^{-1} : M \to N$ вычислима.

Используя вместо самих элементов их номера при некоторой нумерации, мы сводим рассуждений об этих элементах к рассуждениям о натуральных числах.

Аналогичным образом можно рассмотреть нумерацию алгоритмов.

Пример: Проверить, что $f(m,n) = 2^m \cdot 5^n, m,n \in N$ – геделевская нумерация.

$f(2,3) = 2^2 \cdot 5^3 = 4 \cdot 125 = 500 \Rightarrow$ по номеру (2,3) мы вычислили элемент.

Разложим 500 на простые множители $500 = 2^2 \cdot 5^3 \Rightarrow m = 2, n = 3$, т.е. по введённому элементу определили его номер. Т.о. $f(m,n) = 2^m \cdot 5^n$, $m,n \in N$ – геделевская нумерация.

5.2. Нумерация алгоритмов

Поскольку любой алгоритм можно задать конечным описанием (словом), а множество всех конечных слов в фиксированном конечном алфавите счетно, то множество всех алгоритмов счётно. Это означает наличие взаимно-однозначного соответствия между множеством N натуральных чисел и множеством всех алгоритмов, рассматриваемым как подмножество множества Al всех слов в алфавите X, выбранном для описания алгоритмов. Функция $\varphi : N \to Al$ называется нумерацией алгоритма. Если $\varphi(n) = A$, то натуральное число n называют номером алгоритма A. Будем обозначать алгоритм A с номером n через A_n.

Из взаимной однозначности отображения φ следует существование обратной функции φ^{-1}, восстанавливающей по описанию алгоритма A_n его номер в этой нумерации $\varphi^{-1}(A_n) = n$.

Нумерация всех алгоритмов является одновременно и нумерацией всех вычислимых функций в следующем смысле: номер функции f – это номер некоторого алгоритма, вычисляющего f. Ясно, что в разных нумерациях всякая функция будет иметь бесконечное множество различных номеров. Однако, в любом случае множество вычислимых функций счетно. Существование нумераций позволяет работать с алгоритмами как с числами.

5.3. Нумерация машин Тьюринга

Рассмотрим машины Тьюринга (МТ) с внешним алфавитом $A = \{a_0, ..., a_i, ...\}$ и алфавитом внутренних состояний $Q = \{q_0, q_1, ..., q_j, ...\}$. Пусть A и Q – счетные множества. Пусть для всех МТ a_0 – пустой символ, q_0 – заключительное состояние, q_1 – начальное состояние, L, R, W – символы сдвига: влево, вправо, на месте.

Каждому символу из множества $\{L, R, W, a_0, a_1, ..., a_i, ..., q_0, q_1, ...q_j, ...,\}$ поставим в соответствие двоичное число:

	Символ	Код	Число нулей в коде
d	L	10	1
	R	100	2
	W	1000	3
A	a_0	10000	4
	a_1	1000000	6

	…		
	a_i	10……0	$2i+4$
	…		
Q	q_0	100000	5
	q_1	10000000	7
	…		
	q_j	10……0	$2j+5$
	…		

Команде МТ $I: qa \to q'a'd$ поставим в соответствие двоичное число: $Код(I) = Код(q)Код(a)Код(q')Код(a')Код(d)$.

Упорядочим команды МТ в соответствии с лексикографическим порядком левых частей команд $q_1a_0, q_1a_1, ..., q_2a_0, ...$ и т. д. Получим последовательность команд $I_1, ..., I_{n(m+1)}$, где n – число символов в алфавите A, m – число состояний в множестве Q.

Тогда МТ можно поставить в соответствие двоичное число вида: $Код(T) = Код(I_1)Код(I_1)...Код(I_{n(m+1)})$.

Пример. $A = \{a_0, a_1\}$, $Q = \{q_0, q_1\}$, $I_1: q_1a_0 \to q_0a_0W$, $I_2: q_1a_1 \to q_0a_1W$.

Тогда $\underbrace{Код(T) = 10^7 10^4 10^5 10^4 10^3}_{I_1} \underbrace{10^7 10^6 10^5 10^6 10^3}_{I_2}$.

Такое кодирование является алгоритмической процедурой. Зная код машины Тьюринга можно однозначно восстановить множество ее команд. Код МТ можно рассматривать как двоичную запись натурального числа, которое можно записать в десятичной системе счисления. Это число будем называть номером машины Тьюринга. Машину Тьюринга с номером k обозначим M_k. Таким образом, множество всевозможных машин Тьюринга счетно.

Теорема. Существует функция, не вычислимая по

Тьюрингу, т. е. не вычислимая ни на одной машине Тьюринга.

Доказательство теоремы следует из того факта, что множество всевозможных числовых функций имеет мощность континуума, а множество функций вычислимых на МТ счетно, как и множество самих машин Тьюринга. Континуальная мощность строго больше счетной. Следовательно, существуют функции, не вычислимые по Тьюрингу.

§6. Разрешимые и перечислимые множества

Пусть $X \subseteq N^n$. Множество X называется *разрешимым (рекурсивным)*, если существует алгоритм A, который по любому объекту x определяет, принадлежит ли он множеству X. При этом алгоритм A *называют разрешающим.*

Разрешающим алгоритмом для функции f будем называть такой (алгоритм), с помощью которого вычисляется функция f. Таким образом, *функция разрешима, если она вычислима.*

Проблема разрешимости для функций следующим образом связана с проблемой разрешимости для множеств. Пусть X произвольное подмножество N^n. Рассмотрим характеристическую функцию $\chi(x_1, x_2, ..., x_n)$ множества X, определяемую следующим образом:

$$\chi(x_1, x_2, ..., x_n) = \begin{cases} 1, \textit{если } (x_1, x_2, ..., x_n) \in X, \\ 0, \textit{если } (x_1, x_2, ..., x_n) \notin X. \end{cases}$$

Теорема. X разрешимо тогда и только когда характеристическая функция

$$\chi(x_1, x_2, ..., x_n) = \begin{cases} 1, \textit{если } (x_1, x_2, ..., x_n) \in X, \\ 0, \textit{если } (x_1, x_2, ..., x_n) \notin X. \end{cases}$$

вычислима (рекурсивна).

Доказательство. Множество X разрешимо тогда и только тогда, когда имеется алгоритм A, умеющий определять для $x \in N^n$ что верно из следующих двух условий:

1) $x \in X$ или 2) $x \notin X$.

Это означает, что алгоритм A умеет вычислять значение характеристической функции $\chi(x_1, x_2, ..., x_n)$ Поэтому X разрешимо $\Leftrightarrow$ функция $\chi(x_1, x_2, ..., x_n)$ вычислима. С учетом тезиса Черча получаем, что X разрешимо $\Leftrightarrow$ функция $\chi(x_1, x_2, ..., x_n)$ рекурсивна.

Что и требовалось доказать.

Очевидно, пересечение, объединение и разность разрешимых множеств – разрешимы. Любое конечное множество разрешимо.

Множество $M \subseteq N$ называется (рекурсивно, или эффективно, или алгоритмически) *перечислимым*, если M либо пусто, либо существует алгоритм, позволяющий перечислить все элементы этого множества (возможно с повторениями). Существует много эквивалентных определений перечислимого множества, например:

1. Множество перечислимо, если оно есть область определения некоторой вычислимой функции.

2. Множество перечислимо, если оно есть область значений некоторой вычислимой функции.

3. Множество M перечислимо, если существует такая вычислимая функция $\psi_M(x)$, (которая называется перечисляющей множество M), что $m \in M \Leftrightarrow (\exists x)(m = \psi_M(x))$.

Примерами эффективно перечислимых множеств являются:

1) множество квадратов натуральных чисел.

2) множество упорядоченных пар натуральных чисел с нулем.

Теорема 1. Если множества M и L перечислимы, то

перечислимы множества $M \cup L$ и $M \cap L$.

Теорема 2. (Поста). Множество M разрешимо тогда и только тогда, когда оно само и его дополнение эффективно перечислимы.

Теорема 3. Существует перечислимое, но неразрешимое множество натуральных чисел.

Теорема 4. Функция f с натуральными аргументами и значениями вычислима тогда и только тогда, когда ее график $F = \{(x, y) \mid f(x) \textit{ определено и равно } y\}$ является перечислимым множеством пар натуральных чисел.

Разрешимые и перечислимые множества составляют простейшие и в тоже время важнейшие примеры множеств, строение которых задается с помощью тех или иных алгоритмических процедур. Изучение множеств конструктивных объектов с точки зрения таких свойств этих множеств, которые связаны с наличием тех или иных алгоритмов, образует алгоритмическую теорию множеств.

§6.Алгоритмически неразрешимые задачи

6.1. Алгоритмические проблемы математики

Проблема построения алгоритма, обладающего теми или иными свойствами, называется *алгоритмической проблемой.* Алгоритмическая проблема решена, если: 1) найден искомый алгоритм (т.е. представлено его описание) или 2) доказано, что такого алгоритма не существует. Причем в первом случае мы имеем дело с алгоритмически разрешимой проблемой, а во втором алгоритмически неразрешимой.

Строгая формализация понятия алгоритма позволяет уточнить вопрос об алгоритмической разрешимости данной

массовой проблемы. Теперь этот вопрос можно сформулировать так: существует ли машина Тьюринга (или алгоритм Маркова, или рекурсивная функция), решающая данную массовую проблему, или же такой машины (нормального алгоритма, или рекурсивной функции) не существует?

На этот вопрос теория алгоритмов в ряде случаев дает отрицательный ответ. Один из первых результатов такого типа получен американским математиком Черчем в 1936 году. Он касается проблемы распознавания выводимости математической логики и имеет следующую формулировку:

Теорема Черча. Проблема распознавания выводимости алгоритмически неразрешима.

В 1946 и 1947 годах российский математик А.А, Марков и американский математик Э. Пост независимо один от другого построили конкретные примеры ассоциативных исчислений, для каждого из которых проблема эквивалентности слов алгоритмически неразрешима, и, следовательно, не существует алгоритма для распознавания эквивалентности слов в любом исчислении.

В 1955 году российский математик П.С. Новиков доказал алгоритмическую неразрешимость проблемы тождества групп, формально эта проблема представляет собой частный случай проблемы эквивалентности слов в ассоциативном исчислении.

Одной из наиболее знаменитых алгоритмических проблем математики являлась 10-я проблема Гильберта, поставленная им в числе других в 1900 г. на Международном математическом конгрессе в Париже. Требовалось найти алгоритм, определяющий для любого диофантова уравнения $F(x, y,..., z) = 0$, имеет ли оно целочисленное решение. Здесь $F(x, y,..., z)$ – многочлен с целыми показателями степеней и с целочисленными коэффициентами. В 1970 году советский

математик Ю.В. Матиясевич доказал алгоритмическую неразрешимость этой проблемы.

6.2. Алгоритмические проблемы в общей теории алгоритмов

Рассмотрим алгоритмические проблемы, связанные с машиной Тьюринга. Предположим, что на ленте машины Тьюринга записана ее собственная функциональная схема в алфавите машины. Если машина применима к такой конфигурации, то будем называть ее самоприменимой, в противном случае – несамоприменимой. Возникает массовая проблема распознавания самоприменимости машин Тьюринга, состоящая в следующем. По заданной программе машины Тьюринга установить, к какому классу относится машина: к классу самоприменимых, или к классу несамоприменимых машин.

Теорема. Проблема распознавания самоприменимости машин Тьюринга алгоритмически не разрешима.

Кроме того, неразрешимой является «проблема остановки» (т.е. проблема распознавания применимости алгоритма): не существует алгоритма, который по номеру x любого алгоритма (в произвольной, но фиксированной нумерации) и исходным данным y определял бы, остановится алгоритм при этих данных или нет. Как следствие этой неразрешимости – невозможность создания общего для любой программы алгоритма отладки программ.

Неразрешимой является также проблема распознавания эквивалентности алгоритмов:

Еще один достаточно обширный круг алгоритмически не разрешимых проблем описывает теорема Райса. Эта теорема устанавливает алгоритмическую неразрешимость вообще всякого нетривиального свойства вычислимых функций.

Теорема Райса. Пусть C - любой непустой собственный класс вычислимых функций от одного аргумента. Тогда не существует алгоритма, который бы по номеру x функции f_x определял бы, принадлежит f_x классу C или нет. Т.е. множество $\{x \mid f_x \in C\}$ неразрешимо.

Смысл этой теоремы: какое бы свойство вычислимых функций ни рассматривать (периодичность, ограниченность, равенство другой, наперёд заданной функции и т.д.), нельзя построить общий алгоритм, например, машину Тьюринга, который по произвольному алгоритму A определял бы, обладает ли этим свойством вычисляемая им функция.

В частности, оказывается неразрешимой проблема эквивалентности алгоритмов: по двум заданным алгоритмам нельзя узнать, вычисляют они одну и ту же функцию или нет (или нельзя построить алгоритм, который по любым двум алгоритмам (программам) выяснял бы, вычисляют они одну функцию или нет). Действительно, по тексту сколько-нибудь сложной программы, не запуская ее в работу, трудно понять, что она делает (какую функцию вычисляет). Если это понимание и приходит, то каждый раз по-своему; единого метода здесь не существует. Это своего рода практическое проявление теоремы Райса.

Список вопросов к экзамену

1. Интуитивное понятие алгоритма, признаки алгоритма, примеры. Алгоритм как задача вычисления функции.
2. Необходимость уточнения понятия алгоритма
3. Аксиоматические теории.
4. Вычислимые функции, простейшие вычислимых функций.
5. Основные операторы формальной теории вычислимых функций и их свойства.
6. Примитивно рекурсивные функции и их свойства, примеры.
7. Частично рекурсивные и общерекурсивные функции свойства, примеры.
8. Теоремы о несчетности числовых функций и счетности вычислимых функций. Тезис Чёрча.
9. Универсальные функции.
10. Определение машины Тьюринга. Машинное слово.
11. Модель машины Тьюринга.
12. Работа машины Тьюринга.
13. Вычислимые по Тьюрингу функции. Тезис Тьюринга. Вычислимость по Тьюрингу ЧРФ.
14. Композиция машин Тьюринга.
15. Универсальная машина Тьюринга.
16. Машина Поста.
17. Ассоциативные исчисления основные определения.
18. Дедуктивные цепочки.
19. Алгоритмы Маркова, определение.
20. Марковские подстановки. Нормально вычислимые функции.
21. Тезисы Чёрча, Тьюринга и Маркова.
22. Неразрешимые алгоритмические проблемы.
23. Алгоритмическая сводимость.
24. Разрешимые и перечислимые множества, свойства, примеры. Характеристические функции.

25. Теорема Поста.
26. Геделевская нумерация.
27. Нумерация алгоритмов.
28. Нумерация машин Тьюринга.
29. Теоремы о параметризации и неподвижной точки. Теорема Райса.
30. Проблема распознавания самоприменимости машин Тьюринга.
31. Проблема останова.
32. Грамматики. Языки, иерархия языков по Хомскому.
33. Сложность алгоритмов. Классы сложности. NP-полнота.

Список литературы

Основная:

1. Алферова З.В. Теория алгоритмов. – М.: Статистика, 1973.
2. Верещагин Н.К., Шень А. Вычислимые функции. – М.: МЦНМО, 1999.
3. Гуц А.К. Математическая логика и теория алгоритмов. – Омск.: Омский ун-т, 2003.
4. Игошин В.И. Математическая логика и теория алгоритмов – М.: Академия, 2004.
5. Игошин В.И. Задачи и упражнения по математической логике и теории алгоритмов. – М.: Академия, 2005.
6. Катленд Н. Вычислимость. Введение в теорию рекурсивных функций. – М.: Мир, 1983.
7. Лавров И.А., Максимова Л.Л. Задачи по теории множеств, математической логике и теории алгоритмов, 3-е изд. – М.: Физматлит, 1995.
8. Лихтарников Л.М., Сукачева Т.Г. Математическая логика /курс лекций/. – СПб.: Лань, 1998.
9. Мальцев А. И. Алгоритмы и рекурсивные функции. – М.: Наука, 1986.
10. Успенский В.А. Лекции о вычислимых функциях. – М.: Физматгиз, 1960.
11. Успенский В.А., Семенов А.Л. Теория алгоритмов: основные открытия и приложения. – М.: Наука, 1987.

Дополнительная:

1. Барвайз Дж. (ред.). Справочная книга по математической логике. Часть III. Теория рекурсии. – М.: Наука, 1982.
2. Боро В., Цагир Д., Рольфс Ю., Крафт Х., Янцен Е. Живые числа. Пять экскурсий. – М.: Мир, 1985.

3. Булос Дж., Джеффри Р. Вычислимость и логика. – М.: Мир, 1994.
4. Ершов Ю.Л., Палютин Е.А. Математическая логика. – М.: Наука, 1987.
5. Ершов Ю.Л. Теория нумераций. – М.: Наука, 1977.
6. Кушнер Б.А. Лекции по конструктивному математическому анализу. – М.: Наука, 1973.
7. Кузнецов О.П., Адельсон-Вельский Г.М. Дискретная математика для инженера. – М.: Энергоатомиздат, 1988.
8. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. – М.: МЦМНО, 2002.
9. Марков А.А., Нагорный Н.М. Теория алгорифмов. – М.: Наука, 1984.
10. Матиясевич Ю. В. Десятая проблема Гильберта. – М.: Физматлит, 1993.
11. Петер Р. Рекурсивные функции. – М.: ИЛ, 1954.
12. Роджерс Х. Теория рекурсивных функций и эффективная вычислимость. – М.: Мир, 1972.
13. Роберт Р. Столл. Множества. Логика. Аксиоматические тоерии. – М.: Просвещение, 1968.
14. Черч А. Введение в математическую логику. – М.: ИЛ, 1960.
15. Шенфилд Дж. Математическая логика. – М.: Наука, 1975.
16. Эббинхауз Г.Д., Якобс К., Ман Ф.К., Хермес Г. Машины Тьюринга и рекурсивные функции. – М.: Мир, 1972.