Федеральное агентство по образованию

Московский инженерно-физический институт
(государственный университет)

А.Н. Тихомирова

Теория алгоритмов

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

Москва 2008

УДК 519.712(075)
ББК 22.176.я7
Т46

Тихомирова А.Н. Теория алгоритмов: Учебное пособие. М.: МИФИ, 2008. 176 с.

Книга посвящена теории алгоритмов и содержит основные сведения о свойствах алгоритмов и способах их формального представления (машины Тьюринга, алгоритмы Маркова, рекурсивные функции). Изложены основы теории бесконечных множеств, рассмотрены вопросы нахождения эффективных процедур для перечисления объектов различной природы. Затронуты проблемы алгоритмической неразрешимости и базовые понятия сложности алгоритмов.

Книга представляет интерес для студентов технических вузов, аспирантов, научных работников и всех, интересующихся теорией алгоритмов и связанными с ней аспектами истории развития математики.

Работа подготовлена в рамках Инновационной образовательной программы МИФИ.

Рецензент д-р техн. наук, доц. Андросов В.А.

ISBN 978-5-7262-1078-0

Оглавление

Предисловие **5**

Введение **6**

Глава 1. Формальные описания алгоритмов **7**
§ 1.1. Алгоритмы: определение и основные свойства 7
§ 1.2. Классические машины Тьюринга 9
§ 1.3. Специальные машины Тьюринга 20
§ 1.4. Теоремы Шеннона 27
§ 1.5. Алгоритмически неразрешимые задачи 34
§ 1.6. Нормальные алгоритмы 47
§ 1.7. Эффективные перечислимость и распознаваемость 49
Задачи к главе 1 56

Глава 2. Числовые множества **62**
§ 2.1. Множества: определение и основные свойства 62
§ 2.2. Классификация множеств 64
§ 2.3. Счетные множества 67
§ 2.4. Несчетные множества 78
§ 2.5. Множества с мощностью, больше чем мощность континуума 91
§ 2.7. Вычислимые числа 104
Задачи к главе 2 105

Глава 3. Арифметические вычисления **109**
§ 3.1. Арифметические функции 109
§ 3.2. Частичные арифметические функции 115
§ 3.3. Эффективное распознавание и сравнение функций 119
Задачи к главе 3 124

Глава 4. Рекурсивные функции **126**
§ 4.1. Роль рекурсивных функций 126
§ 4.2. Примитивно-рекурсивные функции 127
§ 4.3. Частично рекурсивные функции 129
§ 4.4. Общерекурсивные функции 134
§ 4.5. Задание рекурсивных функций 136
§ 4.6. Мажорируемые неявные функции 142
§ 4.7. Возвратная рекурсия и функция Фибоначчи 143
§ 4.8. Эффективная перечислимость и распознаваемость 145
§ 4.9. Нерекурсивные и непримитивно рекурсивные функции 151

§ 4.10. Границы применимости формальных моделей алгоритмов 153
Задачи к главе 4 155

Глава 5. Сложность алгоритмов..... 159
§5.1 Сравнение функций с точки зрения сложности..... 159
§5.2 Полиномиальные и экспоненциальные алгоритмы..... 161
§5.3 Временная и пространственная сложность машин Тьюринга.. 162
Задачи к главе 5 164

Краткий справочник имен 165

Список литературы..... 173

Предисловие

Основной предмет книги – теория алгоритмов – ныне общепризнанно является дисциплиной, образующей теоретический фундамент для создания вычислительных и управляющих систем. В этой связи интерес к основам этой теории возникает при изучении многих разделов современной и классической математики. Основное внимание уделено способам формального представления алгоритмов (машины Тьюринга, алгоритмы Маркова, рекурсивные функции).

Рассмотренные в книге фрагменты теории бесконечных множеств, а также возникающие в связи с этим парадоксы, призваны расширить математическую широту восприятия абстрактного подготовленным читателем.

Ряд вопросов, связанных с проблемами алгоритмической неразрешимости, включен в данную книгу с той же целью.

Настоящая книга возникла в результате обработки конспектов лекций по теории алгоритмов, читавшихся автором с 2003 года в Московском инженерно-физическом институте (государственном университете). Основной лекционный материал сформировался в результате общения с выдающимся математиком, философом и исследователем, профессором и просто удивительным по своей душевной теплоте человеком, Поваровым Гелием Николаевичем.

Памяти проф. Поварова Г.Н. посвящается данная книга.

Введение

Еще на самых ранних ступенях развития математики в ней стали возникать различные вычислительные процессы чисто механического характера. С их помощью искомые величины ряда задач вычислялись из исходных величин по определенным правилам и инструкциям. Со временем все такие процессы в математике получили название алгоритмов.

Первоначально теория алгоритмов возникла в связи с внутренними потребностями теоретической математики. Математическая логика, алгебра, геометрия и анализ являются основными областями приложения теории алгоритмов.

Другая область теории алгоритмов возникла в 1940-х годах в связи с созданием быстродействующих электронных вычислительных и управляющих машин. Появление ЭВМ способствовало развитию теории алгоритмов, вызвало к жизни разделы этой теории (алгоритмические системы и алгоритмические языки), имеющие ярко выраженную прикладную направленность. Наконец, теория алгоритмов оказалась тесно связанной и с рядом областей лингвистики, экономики, физиологии мозга и психологии, философии, естествознания.

Задача нахождения единообразной формы записи алгоритмов, решающих различные задачи, является одной из основных задач теории алгоритмов. В настоящей книге будут рассмотрены следующие подходы:

- определение алгоритма через понятие вычислительной машины (машины Тьюринга, предложено Тьюрингом в 1937 году);
- определение алгоритма через процедуру переработки слов по заданным правилам (нормальные алгоритмы, предложены Марковым в 1956 году);
- определение алгоритма через рекурсивную функцию (предложено Клини и Геделем в 1936 году).

Как выяснилось, все эти понятия эквивалентны, что в некоторой степени свидетельствует о правильном направлении исследований в этой области.

Глава 1. Формальные описания алгоритмов

§ 1.1. Алгоритмы: определение и основные свойства

Понятие алгоритма является одним из основных понятий современной математики. Само слово “алгоритм” является производным от имени среднеазиатского ученого Аль Хорезми, уроженца Хивы, жившего в IX веке нашей эры.

Чтобы иметь возможность более уверенно решать алгоритмические задачи, возникающие в различных разделах теоретической и прикладной математики, необходимо иметь достаточно развитую теорию алгоритмов.

> ***Алгоритм*** *– это точное предписание о выполнении в некотором порядке системы операций, определяющих процесс перехода от исходных данных к искомому результату для решения задачи данного типа.*

Это понятие относится к исходным математическим понятиям, которые не могут быть определены через другие, более простые понятия. Иногда такое или подобное определение называют ***интуитивным***, т.е. понятным из опыта.

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

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

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

- множеством допустимых исходных данных,

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

Перечисленных свойств вполне достаточно, чтобы можно было определить, является ли данное конкретное предписание алгоритмом.

На протяжении многих веков понятие алгоритма связывалось с числами и относительно простыми действиями над ними, да и сама математика была, по большей части, наукой о вычислениях, наукой прикладной. Чаще всего алгоритмы представлялись в виде математических формул. Порядок элементарных шагов алгоритма задавался расстановкой скобок, а сами шаги заключались в выполнении арифметических операций и операций отношения (проверки равенства, неравенства и т.д.). Часто вычисления были громоздкими, а вычисления вручную – трудоемкими, но суть самого вычислительного процесса оставалась очевидной. У математиков не возникала потребность в осознании и строгом определении понятия алгоритма, в его обобщении. Но с развитием математики появлялись новые объекты, которыми приходилось оперировать: векторы, графы, матрицы, множества и др. Как определить для них однозначность или как установить конечность алгоритма, какие шаги считать элементарными? В 1920-х годах задача точного определения понятия алгоритма стала одной из центральных проблем математики. В то время существовало две точки зрения на математические проблемы:

- Все проблемы алгоритмически разрешимы, но для некоторых алгоритм еще не найден, поскольку еще не развиты соответствующие разделы математики.
- Есть проблемы, для которых алгоритм вообще не может существовать.

Идея о существовании алгоритмически неразрешимых проблем оказалась верной, но для того, чтобы ее обосновать, необходимо было дать точное определение алгоритма. Попытки выработать такое определение привели к возникновению теории алгоритмов.

Кроме того математики стремились создавать более общие алгоритмы, которые могли решать разные классы задач. Возник вопрос: "А нельзя ли создать Всеобщий Алгоритм, который решал бы любую математическую задачу аксиоматической теории"? Известный математик Г.В.Лейбниц считал, что такой алгоритм будет найден. Однако в 1936 году американский ученый Черч формально доказал, что Всеобщего Алгоритма не существует.

В любом случае возникла необходимость рассматривать алгоритм как математический объект, что оказалось невозможным, так как не было формального определения алгоритма.

В 1935 году американский математик Эмиль Пост опубликовал в «Журнале символической логики» статью «Финитные комбинаторные процессы, формулировка 1». В этой статье и в появившейся одновременно в Трудах Лондонского математического общества статье английского математика Тьюринга «О вычислимых числах с приложением к проблеме решения» были даны первые уточнения понятия «алгоритм». Важность идей Поста и Тьюринга состоит в том, что был предложен простейший способ преобразования информации, построена алгоритмическая система.

В теории алгоритмов предполагается, что каждый шаг алгоритма таков, что его может выполнить достаточно простое устройство (машина). Желательно, чтобы это устройство было универсальным, т.е. чтобы на нем можно было выполнять любой алгоритм. Механизм работы машины должен быть максимально простым по логической структуре, но настолько точным, чтобы эта структура могла служить предметом математического исследования.

§ 1.2. Классические машины Тьюринга

Определение ***алгоритма*** через понятие вычислительной машины основано на понимании эффективной процедуры, представляющей собой множество сообщаемых исполнителю время от времени правил, которые точно определяют его поведение. Чтобы решить проблему интерпретации (понимания) правил необходимо установить:

- язык, на котором описывается множество правил поведения,
- конструкцию интерпретирующего устройства, которое может «понимать» утверждения, сделанные на этом языке, и, таким образом, выполнять шаг за шагом каждый точно определенный процесс.

Следовательно, задача описания алгоритма может быть сведена к построению машины некоторого типа, которая способна воспринимать набор правил, выраженных на некотором языке, и делать то, что предписано этими правилами.

Английский математик А.М.Тьюринг в 1937 году предложил модель вычислительной машины, известной теперь под названием *машина Тьюринга*.

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

С помощью машины Тьюринга можно доказать существование или не существование алгоритмов решения различных задач. Так как машина выполняет определенный алгоритм, то к машине предъявляются требования, вытекающие из свойств алгоритмов. Во-первых, машина должна быть полностью детерминированной (вычисления должны быть точные и общепонятные) и действовать в соответствии с заданной системой правил. Во-вторых, она должна допускать ввод различных “начальных данных” (соответствующих различным задачам из данного класса задач). В-третьих, заданная система правил работы машины и класс решаемых задач должны быть согласованы так, чтобы всегда можно было “прочитать” результат работы машины.

Исследования в этом же направлении примерно в то же самое время велись и американским математиком Э. Л. Постом, который дал одно из первых определений понятия алгоритма в терминах "абстрактной вычислительной машины" и сформулировал основной тезис теории алгоритмов о возможности описать любой

конкретный алгоритм посредством этого определения. Впоследствии стало принято считать, что алгоритмы этого класса осуществляются особыми машинами, называемыми в настоящее время машинами Тьюринга-Поста или иногда просто машинами Тьюринга. Машины Тьюринга копируют в существенных чертах работу человека, решающего некоторую логическую задачу, и часто рассматриваются в качестве математической модели для изучения функционирования человеческого мозга. Обе машины являются абстрактными вычислительными моделями, тем не менее в 1970 году машина Поста была разработана в металле в Симферопольском университете, а машина Тьюринга была физически построена чуть позже, в 1973 году в Малой Крымской академии наук.

1.2.1. Одноленточная машина Тьюринга

Классической моделью считается одноленточная машина Тьюринга. Под ***одноленточной машиной Тьюринга*** понимают кибернетическое устройство, состоящее из следующих элементов:

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

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

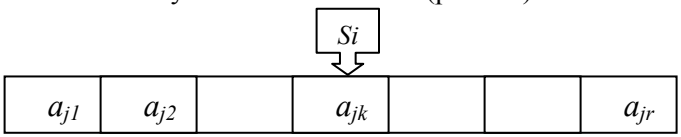


Рис. 1.1. Схематичное изображение машины Тьюринга

Лента

Поскольку бесконечную ленту физически смоделировать затруднительно, обычно предполагается что она конечная, например, магнитная дорожка или бумажная лента типа типографской, и разбита на конечное число ячеек. В процессе работы к существующим ячейкам машина может пристраивать новые ячейки, так что лента может считаться потенциально неограниченной в обе стороны. Каждая ячейка ленты может находиться в одном из конечного множества состояний. Эти состояния мы будем обозначать символами a_0, a_1, ..., a_m или другими символами. По сути это и есть символы, записанные в ячейках ленты. Совокупность таких символов называется ***внешним алфавитом*** машины, а сама лента часто называется ***внешней памятью*** машины. Если ячейка пустая, будем считать, что в ней записан условный символ λ. В процессе работы машины ячейки ленты могут изменять свое содержимое, но могут и не делать этого. Все вновь пристраиваемые ячейки пристраиваются пустыми (содержат условный символ λ). Без ограничения общности ленту можно считать бесконечной лишь с одной стороны (рис.1.2). В этом случае для удобства введем специальный символ ∂, обозначающий начало ленты. Этот символ имеет строго определенное место, его нельзя ни стирать, ни сдвигать. Тогда ленту можно считать однонаправленной (бесконечной вправо) и ее ячейки удобно просматривать слева направо.

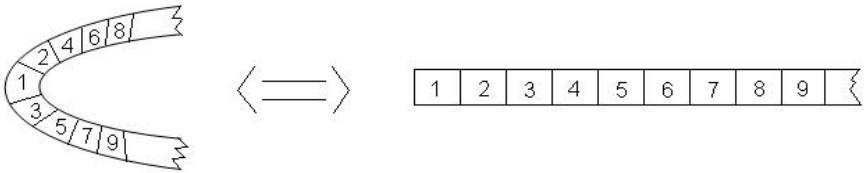


Рис. 1.2. Преобразование двусторонне-бесконечной ленты в односторонне-бесконечную

Управляющая головка

> ***Управляющая головка*** *– это некоторое устройство, которое может перемещаться вдоль ленты так, что в каждый рассматриваемый момент времени оно находится напротив определенной ячейки ленты.*

Иногда, наоборот, считают управляющую головку неподвижной, а движущейся частью становится лента. В таком случае предполагается, что в управляющую головку вводится то одна, то другая ячейка ленты. Если какая-нибудь ячейка находится в управляющей головке, то говорят также, что машина в данный момент «воспринимает» или «обозревает» эту ячейку.

Внутренняя память

> ***Внутренняя память машины*** *– это выделенная ячейка памяти, которая в каждый рассматриваемый момент находится в некотором «состоянии».*

Предполагается, что число возможных состояний внутренней памяти конечное и для каждой машины фиксированное. Состояние внутренней памяти мы будем обозначать символами S_0, S_1, …, S_n или любыми другими символами, не входящими во внешний алфавит машины. Совокупность символов, обозначающих состояния внутренней памяти, называется ***внутренним алфавитом*** машины или ***внутренними состояниями*** машины. Одно из этих состояний называется ***начальным***, с него начинает работу любая машина, пусть это будет состояние S_0. В процессе работы машина может какое-то количество шагов оставаться в состоянии S_0 или возвращаться в него позднее. Еще одно специальное состояние – ***заключительное***. Символ, обозначающий заключительное состояние, будет называться ***стоп-символом***. Роль стоп-символа далее будет играть символ Ω. Если в какой-то момент времени внутренняя память машины приходит в заключительное состояние Ω, то дальнейших изменений в машине не происходит и машина называется ***остановившейся.*** Может случиться, что в

машине никогда не будет происходить никаких изменений и при каком-то другом внутреннем состоянии S_i. Однако в этом случае мы будем говорить, что машина продолжает работать «вечно». В большинстве случае неостанавливающиеся машины не имеют никакой ценности, так как невозможно запротоколировать факт окончания выполнения алгоритма и, соответственно, считать полученный ответ. Обычно говорят, что если машина T не останавливается на входном слове t, то она к этому слову ***неприменима.*** Устройство машины Тьюринга представлено на рис.1.3.

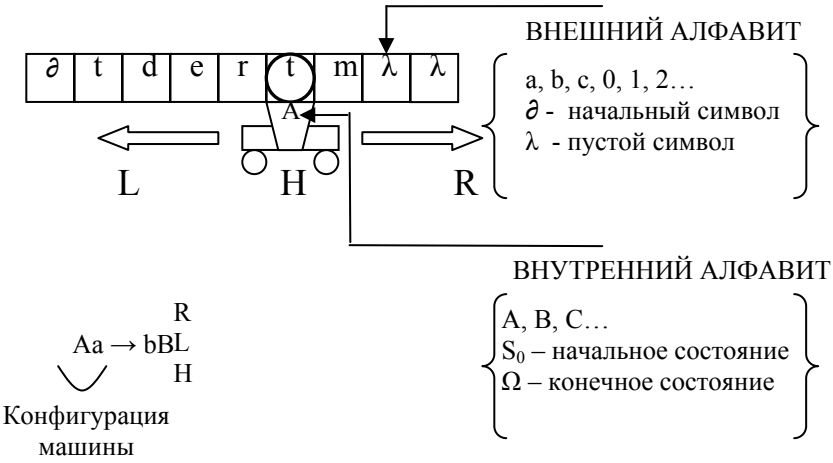


Рис.1.3. Устройство машины Тьюринга

Механическое устройство

Предполагается, что машина снабжена особым механизмом, который в зависимости от символа в воспринимаемой ячейке и состояния внутренней памяти может выполнить следующие действия:

- изменить состояние внутренней памяти,
- одновременно изменить содержимое воспринимаемой ячейки,
- сдвинуть (вправо, влево) управляющую головку в соседнюю ячейку.

В частном случае содержимое воспринимаемой ячейки и/или состояние внутренней памяти могут оставаться неизменными, а управляющая головка – неподвижной (Н). Если управляющая головка находится в самой правой ячейке и по ходу работы машина должна сдвинуть управляющую головку в соседнюю справа (отсутствующую) ячейку, то предполагается, что, сдвигая головку, машина одновременно пристроит недостающую ячейку с пустым символом. Аналогично работает машина и в случае, когда головка воспринимает самую левую ячейку и по ходу дела машине надо сдвинуть головку еще левее. Впрочем, далее в книге предполагается, что лента бесконечна вправо, а ее самая левая ячейка занята специальным символом начала ленты, обозначаемым ∂.

> ***Конфигурация машины Тьюринга*** *– совокупность, образованная содержимым текущей обозреваемой ячейки* a_j *и состоянием внутренней памяти* S_i.

Работа машины состоит в том, что она из данного «состояния» по истечении одного такта работы механического устройства переходит в следующее «состояние», затем из этого состояния по истечении такта работы переходит в новое состояние и так далее. Таким образом, если машина, имея внутреннее состояние S_i и воспринимая ячейку ленты с символом a_j, изменяет свое внутреннее состояние на S_q и одновременно содержимое воспринимаемой ячейки заменяет символом a_r, а управляющая головка сдвинется на одну ячейку вправо (R), то говорят, что машина выполняет команду соответственно $S_i\ a_j \rightarrow a_k\ R\ S_l$. Если управляющая головка сдвинется влево (L) или останется неподвижной (Н), то команды записываются соответственно:

$S_i\ a_j \rightarrow a_k\ L\ S_l$

$S_i\ a_j \rightarrow a_k\ H\ S_l$

Программа машины Тьюринга

> ***Программа машины Тьюринга*** *– совокупность команд установленного формата.*

Так как работа машины по условию целиком определяется состоянием в данный момент ее внутренней памяти S_i и содержимым воспринимаемой ячейки a_j, то для каждых S_i a_j ($i=1, \ldots, n$; $j=1, \ldots, m$), программа машины должна содержать одну и только одну команду, начинающуюся словом S_i a_j. Таким образом, программа машины с символами $\{a_0, a_1, \ldots, a_n\}$ и состояниями $\{S_0, S_1, \ldots, S_m\}$ содержит максимум $n \cdot m$ команд. При этом некоторые команды являются «мертвыми», в том случае, если ни при каких входных словах в данном алфавите невозможно наступление этой конфигурации. В грамотной, с точки зрения реализации, программе таких строк быть не должно, хотя формально их наличие ошибкой не является.

> ***Тезис Тьюринга*** *– любой алгоритм можно преобразовать в машину Тьюринга.*

Эту гипотезу невозможно доказать, потому что она оперирует неформальным понятием алгоритма. Однако обоснование гипотезы есть: все алгоритмы, придуманные в течение столетий, могут быть реализованы на машине Тьюринга. Кроме того, эквивалентность всех формальных определений алгоритма неслучайна и говорит в пользу гипотезы. Чтобы опровергнуть основную гипотезу, необходимо придумать такой алгоритм, который невозможно было бы реализовать на машине Тьюринга, но пока такого алгоритма нет.

1.2.2. Многоленточные машины Тьюринга

Для некоторых задач составление алгоритмов работы для одноленточной машины Тьюринга представляет значительные трудности, связанные с необходимостью где-то хранить результаты промежуточных вычислений или производить посимвольное

сравнение нескольких групп элементов. В некоторых случаях наличие дополнительной (рабочей) ленты или размещение входных слов на нескольких лентах одновременно позволяет получить более лаконичное решение.

Однако, как это ни покажется парадоксальным, вычислительная способность таких машин совершенно не превосходит их одноленточных аналогов. Иными словами, задачи, которые можно решить на многоленточной машине с произвольным количеством лент, всегда решаются и при помощи одноленточной машины.

Многоленточная машина для каждой ленты в общем случае может иметь свой внешний алфавит. Ленты в машине движутся независимо друг от друга. Однако состояние для всех лент машины единое, по сути, это состояние управляющего механизма. Ранее, при рассмотрении одноленточных машин, было принято считать, что лента неподвижна, а управляющая головка перемещается в заданном направлении. Но для рассмотрения многоленточных машин это может оказаться не вполне удобно, потому что ленты являются независимыми, а наглядно представить перемещение единого управляющего механизма по разным направлениям весьма проблематично.

Первое решение проблемы: считаем, что управляющий механизм неподвижен, а ленты могут свободно перемещаться вправо, влево или оставаться на месте независимо друг от друга. Тогда при составлении программы машины Тьюринга при указании направления движения следует придерживаться обратной записи: если обычно управляющий механизм на первом шаге двигался вправо, то теперь ленты машины начинают движение влево относительно неподвижного механизма.

Второе решение проблемы: если подобное «зеркальное» представление не кажется удобным, можно пойти по классическому пути. Допустим, что управляющий механизм подсоединен к набору окошек, в которые обозреваются непосредственно символы на лентах. Тогда считаем, что эти окошки на каждой из лент движутся независимо друг от друга. Виртуально этот процесс становится хоть как-то представимым, если вообразить наличие неких пружинок, которые соединяют

между собой окошки и привязывают их к единому управляющему механизму. В этом случае нотация для записи команд в программе машины Тьюринга остается неизменной.

Далее в книге будет использоваться классический способ (двигается управляющий механизм, ленты неподвижны) и принят следующий формат записи:

$S_i\{a,b,c\} \rightarrow \{a',b',c'\}\{R,L,H\}S_j$.

> *Машина **B** **эквивалентна** машине **A**, если в соответствующие такты их работы лента машины **B** содержит всю информацию о ленте машины **A**.*

Одна из машин может работать гораздо медленнее другой, т.к. каждый такт она моделирует несколькими тактами, поэтому мы говорим о соответствующих тактах. Если в конце концов ***A*** остановится, то ***B*** тоже остановится и к этому моменту будет содержать всю информацию о ленте машины ***A***.

Т.1.2.(1) Теорема

Всякая k -ленточная машина Тьюринга M может быть преобразована в эквивалентную машину M* с одной лентой.

Доказательство

Пусть есть k-ленточная машина M и одноленточная машина M*. Запишем содержимое лент машины M в виде матрицы с 2k строками и бесконечным числом столбцов. В матрице нечетные строки (1, 3, 5 и т.д. до 2k-1 -й) занимают непосредственно ленты машины M, а четные (2, 4, 6, …, 2k-я) являются служебными. На каждой из служебных строк записан только один символ «*», и его расположение указывает на положение смотрового окошка управляющего механизма на соответствующей ленте. Например, на 2-ой строке специальный символ * стоит в той клетке, которая находится непосредственно под клеткой, обозреваемой управляющей головкой на первой ленте (рис.1.4.).

Рис. 1.4. Матрица, содержащая информацию о лентах машины М

Построим ленту машины М*. Для этого содержимое матрицы перенесем на ленту одноленточной машины по столбцам: сначала первый столбец, затем второй и т.д. Одна гиперклетка (набор клеток) машины М* равна целому столбцу в таблице (рис.1.5.).

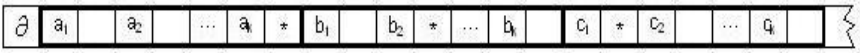


Рис. 1.5. Лента машины М*

В этом случае работа машины М* будет повторять работу машины М с тем отличием, что для воспроизведения команды машины М потребуется набор передвижений с целью определения истинной конфигурации. Для этого на ленте машины М* поочередно отыскиваются все символы *, и считываются находящиеся слева от них символы внешнего алфавита соответствующих лент. Таким образом определяется текущая конфигурация машины М. Далее по программе машины М определяются необходимые действия, и они моделируются, исходя из формата записи машины М*. Например, перемещение управляющего механизма вправо на какой-нибудь из лент имитируется переносом соответствующего символа * на 2k клеток вправо.

Здесь стоит заметить, что в общем случае не вполне очевидно как машина М* будет опознавать, на какой из лент

находится символ, расположенный левее очередной обнаруженной *. В случае, если алфавиты на всех k лентах различны, это трудностей не составляет и будет учтено при составлении программы соответствующим подбором возможных конфигураций. В случае если алфавиты на некоторых или всех лентах совпадают или имеют непустое пересечение, возникает потребность в различных символах, например r_1, r_2, r_3 и т.д. вместо предложенного выше единого символа *.

Таким образом показано, что k-ленточная машина Тьюринга может быть преобразована в эквивалентную ей одноленточную машину Тьюринга, **Q.E.D.**

Здесь и далее при завершении доказательства теорем будет записано общепринятое сокращенное обозначение окончания этого процесса, **Q.E.D.**

> ***Q.E.D.*** *(аббревиатура от лат.* ***quod erat demonstrandum*** *— «что доказывалось», «что и требовалось доказать») латинское выражение, обозначающее завершение доказательства теоремы.*

§ 1.3. Специальные машины Тьюринга

Самоанализирующие машины Тьюринга

Разделим знаки таблицы машины Тьюринга на ленточные и служебные:

- ленточные знаки – это символы внешнего алфавита,
- служебные знаки – это символы внутреннего алфавита (состояния машины), разделитель (он нужен, если таблица записывается в 1 строчку), знаки движения.

> **Самоанализирующие машины** – это машины, в которых все служебные символы как-либо изображаются ленточными символами.

Приведем пример самоанализирующей машины М. Допустим:

A – знак единственного внутреннего состояния

R – знак движения вправо

H – знак остановки

X – табличный разделитель команд

Тогда введем ленточные образы этих знаков:

A – α

R – ρ

H –δ

X – ξ

Можно предложить нарочито упрощенный пример программы машины M.

A α → ξ R A //если машина находится в состоянии A (ленточный образ α), то на это место ставится образ разделителя команд X (ξ).

A ξ → ξ R A //переход через ленточный образ разделителя команд X (ξ).

A ρ → δ H A //если машине предписано двигаться вправо, она меняет ленточный образ символа R (ρ) на ленточный образ символа H (δ) и останавливается.

Посмотрим, как будет проистекать самоанализ машины M.

Как уже упоминалось, образы служебных знаков могут совпадать с самими служебными знаками, но в нашем описании рабочего процесса мы должны их различать.

Код программы машины выглядит так:

A α ξ R A **X** A ξ ξ R A **X** A ρ δ H A **X**

Тогда на ленте запишем этот же код, заменив знаки ленточными образами (пробелы поставлены для повышения читаемости, на ленте символы реально записаны друг за другом). Машина находится в состоянии A, ее положение на ленте показано графически.

1 шаг: α α ξ ρ α ξ α ξ ξ ρ α ξ α ρ δ δ α ξ

2 шаг: ξ α ξ ρ α ξ α ξ ξ ρ α ξ α ρ δ δ α ξ

3 шаг: ξ ξ ξ ρ α ξ α ξ ξ ρ α ξ α ρ δ δ α ξ

4 шаг: ξ ξ ξ ρ α ξ α ξ ξ ρ α ξ α ρ δ δ α ξ

5 шаг: ξ ξ ξ δ α ξ α ξ ξ ρ α ξ α ρ δ δ α ξ

Если в понятие машины Тьюринга включить условие, что при появлении конфигураций, не предусмотренных в таблице машины, она останавливается, то из любой машины Тьюринга легко получить самоанализирующую машину, включив в ее ленточный алфавит все служебные знаки, ранее в нем отсутствующие. Если в первоначальном алфавите их вовсе не было, то новая машина при самоанализе сразу остановится (т.н. тривиальный самоанализ).

Универсальные машины

Тьюринг показал возможность построения определённой вычислительной машины U, универсальной в том смысле, что на U можно выполнить любое вычисление.

Универсальная машина *– машина Тьюринга, обладающая способностью путём подходящего кодирования выполнить любое вычисление.*

Однако к этому, несомненно, надо присоединить то условие, что кодирование должно быть в некотором смысле простым. В самом деле, нет особой выгоды считать универсальной машину Тьюринга, у которой кодирование, по существу, требует работы другой универсальной машины. Отсюда возникает задача явного определения ***универсальной машины Тьюринга.*** Далее универсальная машина Тьюринга будет сокращенно именоваться ***УМТ***.

На ленте УМТ записывается кодовая последовательность, характеризующая данную МТ (программа произвольной МТ и входное слово), и УМТ должна читать этот кодовую последовательность и выполнять всю работу той машины, программа которой записана на её ленте. УМТ должна печатать знаки в таком же коде, как и в таблице. Соответственно для такой машины необходимы входные слова, записанные по какому-то методу (единообразный метод записи программ и входных слов). Поскольку число возможных знаков – бесконечно, а у машины конечный алфавит, поэтому всё (и состояния, и символы) кодируется последовательностью других знаков (чисел).

Пусть машина A имеет m символов a_j и n внутренних состояний S_i . Закодируем знаки, используемые при написании программы работы такой машины следующим образом:

$a_j = 1…1$ ($a_1=1$, $a_2=11$, $a_3=111$ и т.д.)

$S_i = 2…2$ ($S_1=2$, $S_2=22$, $S_3=222$ и т.д.)

$R = 3$

$L = 33$

$H = 333$

В этом случае всю программу работы машины можно записать неким числом, причем возможны два варианта записи:

1. С разделителем команд, допустим X, которые можно закодировать числом 4. В этом случае классическая запись $S_{old}\ a_{old}\rightarrow a_{new}\ R\ S_{new}$,

2. Без разделителя команд. В этом случае команды следует писать в формате $a_{old}\ S_{old}\ a_{new}\ R\ S_{new}$, тогда две команды, записанные непосредственно друг за другом, будут явно различаться элементарным анализатором.

Возьмем в качестве примера машину Тьюринга, которая на пустой ленте бесконечно много раз печатает последовательность *001*. Сразу оговоримся, что из соображений удобства формат записи будет следующим $aS_i \rightarrow a'\{R,L,H\}S_j$. При такой записи обозначение символа указывается ранее обозначения состояния. Цель данной перестановки довольно банальная: избежать наличия лишнего разделителя.

Программа такой машины выглядит следующим образом (λ-пустой знак):

λ A 0 R B
λ B 0 R C
λ C 1 R A

Закодируем символы и состояния:

λ=1 *A=2*
0=11 *B=22*
1=111 *C=222*

Тогда запись программы машины будет выглядеть так (пробелы поставлены для повышения читаемости, в реальности их нет):

1 2 11 3 22 1 22 11 3 222 1 222 111 3 2

В итоге каждая МТ представлена числом – это дескриптивное (описательное) число машины. Вместе с тем оно является кодом для входного слова. Таким образом решается проблема унификации записи программы машины Тьюринга и входного слова на её ленте.

Несмотря на это, построение реально работающей универсальной машины Тьюринга представляет собой довольно сложный процесс.

Пример построения универсальной машины Тьюринга

По сути УМТ является программируемой машиной Тьюринга, с тем отличием, что собственно программа не является внутренним элементом функциональной схемы, которая управляет считывающей головкой, а просто записана на ленте. Следовательно, одна и та же машина получает способность выполнять любую задачу (разумеется, если для этой задачи существует алгоритм решения).

Рассмотрим УМТ. Ее алфавит ограничен символами 1,2,3,4. Тогда, если цифру 5 использовать для разделения описания машины и описания входного слова, лента выглядит следующим образом (рис.1.6):

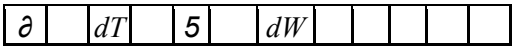


Рис. 1.6. Схематичное изображение ленты УМТ

На рис.1.6. *dT* - описание машины, *dW* – описание входного слова, в котором каждый символ слова a_i записан в виде наборов по несколько *1*, разделенных специальным символом *4* (разделитель).

Описание машины – слово, разбитое на команды. УМТ читает описание данной машины, а затем перерабатывает входное слово так, как бы это сделала конкретная МТ. УМТ имеет такой же алфавит, как и у предъявленной ей машины. Отсюда вытекает необходимость наличия меток для указания положения исходной МТ. Такие метки удобно ставить вместо разделителя, стоящего непосредственно перед группой ячеек, содержащих код рассматриваемого символа. Поскольку первые пять цифр алфавита заняты под кодирование программы МТ, будем заменять разделитель перед символом, на котором остановилась исходная МТ, на цифру *6*.

Составление таблицы для одноленточной УМТ – длительная и малопригодная для понимания в учебных целях процедура. Поэтому для наглядности построим трехленточную машину, которую всегда можно преобразовать в одноленточную по принципу, рассмотренному в Теореме 1.2(1). Ленты трехленточной универсальной машины Тьюринга будут иметь следующее назначение:

- исходная лента, на которой записан код таблицы исходной МТ,
- рабочая лента, на ней записываются внутренние состояния исходной МТ,
- выходная лента – закодированная лента исходной МТ.

В начале моделирования каждого такта работы исходной МТ, УМТ занимает на 3-й ленте первую клетку. Эта клетка

соответствует той клетке на ленте исходной МТ, которую в данный момент воспринимает считывающая головка исходной МТ.

УМТ движется по входной ленте, пока не дойдёт до команды, в которой внутреннее состояние совпадает с записью на 2-й ленте, а воспринимаемый символ – с тем, который записан на входной ленте, в кодовой ячейке которой изображено положение исходной машины. Сравнение происходит по разрядам. В итоге УМТ находит на входной ленте нужную для исполнения команду МТ.

Очень важен механизм опознания того факта, что внутреннее состояние совпадает с записью на 2-й ленте, а воспринимаемый символ – с тем, который записан на входной ленте. Любое запоминание прочитанных символов может производиться только путем изменения внутреннего состояния, однако при программировании УМТ количество состояний должно быть ограничено. В этой связи метод введения все новых и новых состояний типа S^{1} S^{11} S^{111} $S^{111}{}_{2}$ $S^{111}{}_{22}$ $S^{111}{}_{222}S^{111}{}_{2222}$ и т.д. для запоминания количества «2» на второй ленте и «1» на третьей ленте является неосуществимым. Нам просто не хватит описательных возможностей для перебора всех допустимых состояний. Реально требуется механизм ***поразрядного*** сравнения. Способов естественно огромное множество.

По сути, на первой ленте нужно найти некий набор последовательностей «1» и «2», идущих сразу после разделителя команд, который соответствует находящемуся на второй ленте коду состояния исходной машины (последовательность «2») и обозреваемому на третьей ленте коду символа (последовательность «1»). Для оптимизации поэтапного сравнения, возможно, первоначально стоит проверять совпадение символа (т.е. двигать смотровые окошки на 1-й и 3-й ленте) и, в случае удачи, продолжить сравнение, двигая окошки на 1-й и 2-й ленте.

Вероятны и другие способы поразрядного сравнения (например, копирование текущей конфигурации на 4-ую ленту и поиск соответствия её содержимому на 1-ой ленте), но важным остается сам факт: запоминать слово неопределенной длины целиком невозможно, в рамках наших возможностей только поразрядное сравнение содержимого лент.

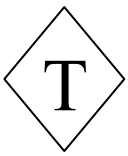
Так или иначе, УМТ считывает инструкцию, соответствующую этой команде. Затем эта инструкция выполняется, при этом совершается три действия.

- Изменяется положение исходной МТ. Например, если инструкция гласит «шаг вправо», то на 3-й ленте делается необходимое количество шагов вправо до ближайшего нового разделителя команд, который заменяется меткой текущего положения (вместо 4 ставится символ 6).
- Изменяется внутреннее состояние исходной МТ. На второй ленте вместо старого записывается ее новое состояние.
- Изменяется символ, содержащийся в рассматриваемой ячейке. Поскольку разные символы кодируются различным количеством единичек, то в случае замены символа необходимо предусмотреть процедуру сдвига (растягивание или сжимание слова на нужное число клеток).

После этого моделируется следующий такт работы исходной МТ. Для этого снова анализируется содержание второй ленты (это теперь текущее состояние МТ) и символ, записанный справа от указателя текущего положения управляющей головки (специальная метка 6).

§ 1.4. Теоремы Шеннона

Т.1.4.(1) Теорема Шеннона №1



Всякая машина Тьюринга *A* может быть преобразована в эквивалентную машину *B* не более чем с двумя внутренними состояниями.

Доказательство

Доказательством теоремы будет схема построения такой машины. Причем строить будем, по сути, универсальную машину Тьюринга, использующую одну ленту и имеющую лишь два внутренних состояния, которая сможет моделировать работу любой машины Тьюринга.

Пусть машина ***A*** содержит:

- **m** символов внешнего алфавита $\mathbf{a_{1,\ldots,j,\ldots,m}}$,

• **n** внутренних состояний $S_{1,...,i,...,n}$,

Тогда машина ***B*** будет содержать:

• два внутренних состояния **α** и **β**,

• **m** обычных символов внешнего алфавита b_i , являющихся аналогами a_i,

• **4mn** особенных символов **b**, за счет которых производится расширение внутренней памяти.

Общая идея построения

В общих чертах метод построения таков. Для произвольной машины Тьюринга ***A*** с алфавитом из ***m*** букв (символов, записываемых на ленте, включая пустой символ) и с ***n*** внутренними состояниями построим машину ***B*** с двумя внутренними состояниями и алфавитом не более чем из ***4mn+m*** символов. Машина ***B*** будет работать, по существу, так же, как и машина ***A***. Во всех клетках ленты, кроме воспринимаемого считывающей головкой и одного смежного с ним, на ленте машины ***B*** записано то же, что и на ленте машины ***A*** в соответствующие такты работы двух машин.

Машина ***B*** моделирует поведение машины ***A***, но хранит информацию о внутреннем состоянии машины ***A*** с помощью символов, записанных в клетке под считывающей головкой, и в клетке, которую считывающая головка машины ***A*** собирается посетить. Основная задача — своевременно освежать эту информацию и держать ее под считывающей головкой. Если последняя передвигается, то информацию о состоянии надо перенести в новый квадрат, используя всего два внутренних состояния машины ***B***. Пусть, например, следующим состоянием машины ***A*** должно быть состояние 17 (согласно какой-нибудь системе счисления). Чтобы перенести его символ, считывающая головка "качается" вперед - назад между старой и новой клеткой 17 раз (точнее, 18 раз в новую клетку и 17 раз назад в старую). В течение этого процесса символ, стоящий в новой клетке, пробегает своего рода последовательность счета, которая оканчивается символом, соответствующим состоянию 17 и в то же время сохраняющим информацию о предыдущем символе в этой клетке. Процесс качания возвращает также старую клетку к одному из

элементарных символов (находящихся во взаимно однозначном соответствии с символами, используемыми машиной ***A***), а именно: к тому элементарному символу, который должен быть записан в этой клетке после окончания данной операции.

Формальная схема построения

Формально машина ***B*** строится так. Пусть символы алфавита машины ***A*** суть $a_1, a_2, ..., a_m$ и пусть ее состояния $S_1, S_2, ..., S_n$. В машине ***B*** мы поставим в соответствие алфавиту машины ***A*** ***m*** элементарных символов $b_1, b_2, ..., b_m$. Затем определим **4mn** новых символов, соответствующих парам из состояния и символа машины **A**, снабженных двумя новыми двузначными индексами. Такие новые символы будем называть ***особыми***. Особый знак машины ***B*** имеет формат $\mathbf{b}_{ijxy}$, где:

i- номер ленточного символа, i=1,2,…,m

j- номер внутреннего состояния машины ***A***, j=1,2,…,n

x- назначение (роль) клетки: если клетка передает информацию во время «качания», то x = ”+”, а если получает – то x = ”-”. Сами клетки назовем соответственно: передатчик / приёмник.

y- положение другой особой клетки (машина ***B*** не может запомнить откуда она ушла): в зависимости от того, вправо или влево от воспринимаемой клетке должна передвинуться считывающая головка при качании, y = R или L.

Два состояния машины ***B*** назовем **α** и **β**. Эти состояния используются двояко. Во-первых, при первом шаге качания они переносят в ближайшую подлежащую посещению клетку информацию о том, вправо (**α**) или влево (**β**) от новой клетки лежит старая клетка. Эта информация нужна в новой клетке, чтобы управляющий элемент передвинул считывающую головку назад в нужном направлении. После первого шага информация об этом сохраняется в новой клетке с помощью записанного там символа (последний индекс **y***)*. Во-вторых, состояния **α** и **β** используются, чтобы сообщить из старой клетки в новый о факте окончания качания. После первого шага качания состояние **β** переносится в новую клетку вплоть до конца качания, когда переносится **α**. Это означает конец операции, и новая клетка начинает затем

действовать как передатчик и управляет следующим шагом вычисления.

Команда машины ***A*** указывает, что она при считывании конфигурации выполняет три действия: запись нового символа, переход в новое состояние и передвижение считывающей головки вправо или влево. Соответствующий фрагмент программы машины ***B*** приведен ниже:

α	b_i	$\rightarrow$	$b_{i,1,-,R}$	R	α	$i=1,2,\ldots,m$	
β	b_i	$\rightarrow$	$b_{i,1,-,L}$	L	α	$i=1,2,\ldots,m$	
β	$b_{i,j,-,y}$	$\rightarrow$	$b_{i,j+1,-,L}$	L	α	$i=1,2,\ldots,m$	$j=1,2,\ldots,n-1;$ $y=R,L$
$\frac{\alpha}{\beta}$	$b_{i,j,+,y}$	$\rightarrow$	$b_{i,j-1,+,y}$	y	β	$i=1,2,\ldots,m$	$j=1,2,\ldots,n-1;$ $y=R,L$
$\frac{\alpha}{\beta}$	$b_{i,1,+,y}$	$\rightarrow$	b_i	y	α	$i=1,2,\ldots,m$	$y=R,L$

Эти операции пока что никак не зависят от таблицы работы машины ***A*** (кроме числа используемых символов). Операции же следующего и последнего типа, формулируются в терминах таблицы работы моделируемой машины. Предположим, что машина ***A*** имеет команду:

$$S_j\, a_i \rightarrow a_k\ \frac{R}{L}\ S_l$$

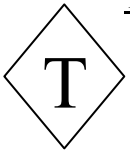
Тогда машина ***B*** будет иметь команду:

$$\alpha\ \ b_{i,j,-,y} \qquad \rightarrow b_{k,1,+,\frac{R}{L}}\ \ \frac{R}{L}\ \frac{\beta}{\alpha}$$

Чтобы заставить машину ***B*** работать аналогично машине ***A***, мы заполняем начальную ленту машины ***B*** соответственно начальной ленте машины ***A*** (с заменой a_i на b_i), за исключением клетки, занимаемой считывающей головкой в начальный момент. Если S_i - начальное состояние машины ***A*** и a_i начальный символ в этом квадрате, то в соответствующем квадрате ленты машины ***B***

записываем $b_{i,j,-,y}$ и приводим машину ***B*** в состояние **α**. Далее инструкция машины ***A*** заменяется приведенной выше инструкцией для машины ***B***. Машина ***B*** работает вплоть до момента, когда вместо особого символа в одной из двух особых клеток окажется записанным элементарный символ, соответствующий символу из внешнего алфавита машины ***A***. Таким образом будет произведен набор действий, эквивалентный первой команде (инструкции) машины ***A.*** Далее аналогично эмулируется вторая команда и т.д. вплоть до остановки машины ***A*** и эквивалентной её машины ***B***. Таким образом показано, как машина ***A*** преобразуется в эквивалентную ей машину ***B*** с двумя внутренними состояниями, **Q.E.D.**

Т.1.4.(2) Теорема Шеннона №2



Всякая машина Тьюринга *A* может быть преобразована в эквивалентную машину *C* не более чем с двумя знаками внешнего алфавита.

Доказательство

Как и в случае теоремы 1.4.(1), доказательством будет схема построения. Покажем, что можно построить машину ***C,*** работающую подобно любой заданной машине Тьюринга ***A*** и использующую только два символа внешний алфавит, например символы 0 и 1.

Пусть машина ***A*** содержит:

- ***n*** внутренних состояний S_j,
- ***m*** символов внешнего алфавита $\mathbf{a}_i$,

Тогда машина **C** будет содержать:

- ***n*** внутренних состояний T_j, являющихся аналогами S_j,
- не более чем ***8nm*** специальных внутренних состояний,
- 2 символа внешнего алфавита: 0 и 1.

Общая идея построения

Пусть l – наименьшее целое число, для которого $m \leq 2^l$. Тогда символам машины ***A*** можно сопоставить двоичные последовательности длины l таким образом, что различным

символам будут соответствовать различные же последовательности. При этом пустому символу машины ***A*** мы ставим в соответствие последовательность из l нулей. Машина ***C*** будет работать с двоичными последовательностями. Элементарная операция машины ***A*** будет соответствовать в машине ***C*** переходу считывающей головки на l - 1 клеток вправо (с сохранением считанной информации во внутреннем состоянии головки), затем обратному переходу на l - 1 клеток влево, записи новых символов по пути и, наконец, движению вправо или влево на l клеток, в соответствии с движением считывающей головки машины ***A***. В течение этого процесса состояние машины ***A***, конечно, сохраняется и в машине ***C***. Замена старого состояния новым происходит в конце операции считывания.

Формальная схема построения

Начальная лента машины ***C*** представляет собой начальную ленту машины ***A***, где каждый символ замещен соответствующей ему двоичной последовательностью. Если работа машины ***A*** начинается с какого-то определенного символа, то работа машины ***C*** начнется с самого левого двоичного символа соответствующей группы. Если машина ***A*** начинает работу в состоянии S_j, то ***C*** начнет работу в состоянии T_j.

Формально машина ***C*** строится так. Состояниям $S_1, S_2, \ldots, S_n$ машины ***A*** мы ставим в соответствие состояния $T_1, T_2, \ldots, T_n$ машины ***C*** (последние будут встречаться, когда машина ***C*** начинает операцию, считывая первый символ в двоичной последовательности длины l). Для каждого из этих T_j определим два состояния T_{j0} и T_{j1}. Если машина ***C*** находится в состоянии T_i и читает символ 0, то она движется вправо и переходит в состояние T_{j0}. Если она читает 1, то движется вправо и переходит в состояние T_{j1}. Таким образом, с помощью этих двух состояний машина запоминает, каким был первый символ двоичной последовательности. Для каждого из этих T_{j0} и T_{j1} определим опять по два состояния: T_{j00}, T_{j01} и T_{j10}, T_{j11}. Если, например, машина находится в состоянии T_{j0} и читает символ 0, то она переходит в состояние T_{j00} и т. д. Таким образом, с помощью этих состояний запоминается начальное состояние и два первых символа,

прочитанных в процессе работы машины. Продолжим такое построение состояний вплоть до l-1 шагов. Получившееся в итоге разнообразие состояний можно обозначить через $T_{j,\ x1,\ x2,\ \ldots,\ xs}$, где j=1,2,…,n; x_i=0,1; s=1,…, l - 1.

Если машина находится в одном из этих состояний ($s < l$-1) и читает 0 или 1, то она движется вправо и 0 или 1 делается дальнейшим индексом состояния. В тот момент, когда s становится равен l – 1, машина читает последний символ последовательности длины l. Теперь правила операций зависят от конкретных правил машины ***A***.

Допустим, текущей инструкцией машины ***A*** была команда

$$S_j\, a_i \rightarrow a_k\ \frac{R}{L}\ S_p$$

Машина ***C*** уже готова к выполнению соответствующей инструкции, а значит в дальнейших состояниях должна быть закодирована информация о трех параметрах:

- о новом символе a_k, который следует записать на место старого символа a_i,
- о направлении дальнейшего движения машины: R или L,
- о номере нового состояния S_p.

Новый символ a_k может быть закодирован двузначным кодом в виде последовательности $y_1,\ y_2,\ \ldots,\ y_{s-1},\ y_s$, где y_i=0,1. Определим два новых множества состояний, которые несколько похожи на введенное выше множество состояний T, но соответствуют не считыванию, а записи: $R_{p,\ y1,\ y2,\ \ldots,\ ys}$ и $L_{p,\ y1,\ y2,\ \ldots,\ ys}$. Название состояния (R или L) будет индикатором движения машины ***A***, первое число в индексе (p) – будет показывать номер нового состояния S_p машины ***A***, а индексы $y_1,\ y_2,\ \ldots,\ y_{s-1},\ y_s$ - значения кода нового символе a_k.

Пусть последовательность $x_1,\ x_2,\ \ldots,\ x_{s-1},\ x_s$ соответствует некоторому символу машины ***A***. Допустим машина ***A*** читает этот символ в состоянии Sj, тогда она записывает символ, соответствующий двоичной последовательности $y_1,\ y_2,\ \ldots,\ y_{s-1},\ y_s$, переходит в состояние S_p и движется, скажем, вправо. В этом случае, по определению, машина ***C***, будучи в состоянии $T_{i,\ x1,\ x2,\ \ldots,\ xl-1}$ и читая символ x_l , переходит в состояние $R_{i,\ x1,\ x2,\ \ldots,\ xl-1}$, записывает символ y_l и движется влево. В любом из состояний $R_{p,\ y1,\ y2,\ \ldots,\ ys}$ (или

$L_{p, y1, y2, ..., ys}$) машина ***C*** записывает y_s, движется влево и переходит в состояние $R_{p, y1, y2, ..., ys-1}$ (или $L_{p, y1, y2, ..., ys-1}$). Посредством этого процесса двоичная последовательность, соответствующая новому символу, записывается вместо старой двоичной последовательности. При s=1 эта запись заканчивается на символе y_1. Остается только передвинуть считывающую голову на l клеток вправо или влево, в зависимости от того, находится ли машина в состоянии R или в состоянии L. Это делается с помощью множеств состояний $U_{i,s}$ и $V_{i,s}$ (i= 1, 2, ..., n; s= 1, 2, ..., l-1). В состоянии R_{ix1} машина записывает x_1, движется вправо и переходит в состояние U_{i1}. В каждом из состояний U машина продолжает движение вправо, не записывая ничего и переходя в состояние U со следующим по величине индексом, пока не будет достигнуто последнее состояние U. Таким образом, $U_{i,s}$ вызывает движение вправо и состояние $U_{i,s+1}$ (s<l-1). Наконец состояние $U_{i,l-1}$ приводит после движения вправо к состоянию T_i, завершая тем самым цикл. Аналогично, $L_{i,x}$ приводит к движению влево и состоянию $V_{i,1}$. $V_{i,s}$ дает движение влево и $V_{i,s+1}$ (s< l -1), наконец, $V_{i,l-1}$ дает движение влево и T_i.

§ 1.5. Алгоритмически неразрешимые задачи

После того, как Тьюрингом была предложена столь удобная формальная модель алгоритма, возник вопрос о границах ее применимости. Если любой алгоритм можно преобразить в машину Тьюринга, то возникает вопрос о том, для любой ли задачи, вообще говоря, существует алгоритм решения.

Далее будут рассмотрены примеры алгоритмически неразрешимых проблем и доказательства их неразрешимости. Вначале отметим, что доказательствам невозможности, приводимым в теории алгоритмов, присуща та же математическая строгость, что и доказательствам невозможности, проводимым в других областях математики.

Существуют два метода доказательства неразрешимости.

Первый, ***прямой метод***, состоит в том, что, исходя из предположения о разрешимости данной проблемы, путем набора логических преобразований (переходов) мы приходим к противоре-

чию с доказанными ранее утверждениями либо с элементарной логикой (со здравым смыслом).

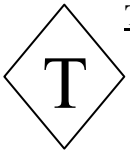
Второй, ***косвенный метод*** (называемый ещё ***методом сведения***), состоит в следующем: показывается, что разрешимость исследуемой проблемы влечёт разрешимость проблемы, о которой уже известно, что она неразрешима. Метод сведения часто бывает более удобным, чем прямой метод.

Первый метод используется вообще достаточно широко в доказательстве теорем различного типа. В общем виде он называется reductio ad absurdum - *доказательство* ***«от противного»***. Это один из самых часто используемых методов доказательства утверждений. Доказательство утверждения *A* проводится следующим образом. Сначала принимают предположение, что утверждение *A* неверно, а затем доказывают, что при таком предположении было бы верно некоторое утверждение *B*, которое заведомо неверно. Полученное противоречие показывает, что исходное предположение было неверным, и поэтому верно утверждение «отрицание отрицания A», которое по *закону двойного отрицания* равносильно утверждению A.

Кроме этого, во многих доказательствах применяется так называемая ***апагогия*** (греч. лат. deductio) – логический прием, которым доказывается несостоятельность какого-нибудь мнения таким образом, что или в нем самом, или же в необходимо из него вытекающих следствиях, мы открываем *противоречие*. Поэтому апагогическое доказательство является доказательством косвенным: здесь доказывающий обращается сначала к противоположному положению, чтобы показать его несостоятельность, и затем по *закону исключения третьего* делает вывод о справедливости того, что требовалось доказать. Этот род доказательства называется также приведением к нелепости: deductio ad absurdum. Существенной его принадлежностью является довод, что третье не существует: tertium non datur, т. е., что кроме мнения, справедливость которого должно доказать, и второго, ему противоположного, которое служит исходным пунктом доказательства, никакой третий факт не допускается. Поэтому косвенное доказательство исходит из факта,

противоречащего положению, справедливость которого требуется доказать.

1.5.1. Задача об остановке произвольной машины Тьюринга при произвольном входном слове



Т.1.5.(1) Теорема

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

Иными словами, нельзя придумать универсальный алгоритм, в результате выполнения которого будет получен однозначный ответ: “да”- если произвольная машина ***Т*** остановится на ленте с произвольным входным словом t и “нет” – если ***Т*** не остановится на ленте t.

Доказательство

Докажем теорему от противного. Предположим, что задача об остановке произвольной машины ***Т*** при обработке произвольного слова t алгоритмически разрешима. Это означает, что существует алгоритм решения, а значит и существует реализующая его машина Тьюринга ***Д***.

Построим машину ***Д***, на ленте которой записано кодовое описание машины ***Т*** и кодовое описание входного слова t (рис.1.7).

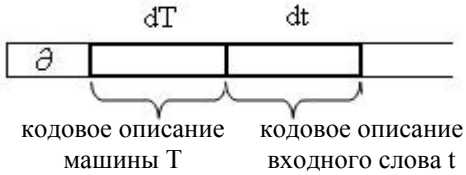


Рис. 1.7. Начальное состояние ленты машины ***Д***

Д обрабатывает эти два слова и пишет “да”, если ***Т*** останавливается при обработке t, и “нет“ если этого не происходит. Если этот алгоритм работает для любого случая (любого входного слова t), то должен работать и для частного случая. В качестве

начального слова, в том числе, можно выбрать описание машины, т.е. возьмем dt = dT.

Построим машину ***E***, которая будет на своей ленте иметь описание произвольной машины ***T***. Работа машины ***E*** состоит в том, что она копирует описание dT , а затем работает как ***Д***. Если существует ***Д***, то существует ***E***.

Построим машину ***F***. ***F*** работает как ***E***, но отличие состоит в том, что, если после работы на входном слове dT машина ***E*** останавливается с ответом "да" (что означает что машина ***T*** останавливается), то машина ***F*** не останавливается, а продолжает бесконечное движение вправо без изменения знаков на ленте. Если же вдруг машина ***E*** останавливается с ответом «нет» (это означает, что машина ***T*** не останавливается), то ***F*** просто останавливается.

Если данный алгоритм работает для произвольной машины ***T***, то и для конкретной тоже будет работать. Положим ***T***=***F***. Таким образом, на ленте машины ***F*** оказывается описание самой машины ***F***. В результате получим, что машина ***F*** анализирует саму себя (***F*** – самоанализирующая машина).

Получаем, что ***F*** не останавливается в том случае, если ***E*** остановится с ответом "да", а это означает, что машина ***T*** остановится. Поскольку ***T***=***F***, имеем ситуацию: ***F*** остановится, если ***F*** не остановится, и ***F*** не остановится, если ***F*** остановится, что явно противоречит здравому смыслу. Такой машины существовать не может.

Поскольку все рассуждения были логически обоснованы, полученное противоречие означает, что ошибка в изначальном предположении о существовании машины ***Д*** (в данном доказательстве используется прямой метод). Отсюда напрямую следует, что задача об остановке произвольной машины ***T*** на произвольном слове t алгоритмически неразрешима, **Q.E.D.**

1.5.2. Задача об остановке произвольной машины Тьюринга на пустой ленте

Если бы имелся алгоритм (а значит и соответствующая машина Тьюринга) для решения проблемы остановки произвольной машины при обработке произвольного слова, то эта машина могла бы решить и проблему остановки на пустой ленте

как частный случай. Но неразрешимость проблемы остановки при анализе произвольного слова непосредственно не означает неразрешимости проблемы остановки на пустой ленте, потому как последняя задача может оказаться более простой, так как ее сфера применения явно уже.

Однако мы можем показать, что эти задачи эквиваленты, в том случае, если для каждой пары машина-лента (***T***-t) докажем наличие соответствующей машины $\boldsymbol{M_{Tt}}$, работающей на чистой ленте.

Т.1.5.(2) Теорема

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

Доказательство

Для каждой пары машина-лента (***T***-t) докажем наличие соответствующей задачи остановки на пустой ленте для некоторой другой машины, предположим $\boldsymbol{M_{Tt}}$. Машина $\boldsymbol{M_{Tt}}$ строится непосредственно по описанию ***T*** и t, если диаграмму состояний ***T*** дополнить последовательностью новых состояний.

Предположим, что для пары ***T***, t в некоторый момент времени лента выглядит таким образом (рис.1.8):

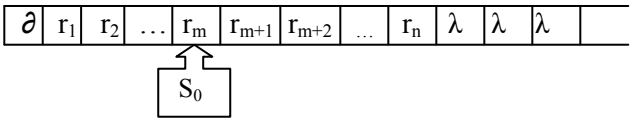


Рис 1.8. Лента машины ***T*** в выбранной момент времени

Новая машина M_{Tt} начинает работу на пустой ленте и работает по следующей программе:

$S_1\ \lambda \rightarrow r_1\ R\ S_2$

$S_2\ \lambda \rightarrow r_2\ R\ S_3$

…

$S_m\ \lambda \rightarrow r_m\ R\ S_{m+1}$

$S_{m+1}\ \lambda \rightarrow x\ R\ S_{m+2}$
$S_{m+2}\ \lambda \rightarrow r_{m+2}\ R\ S_{m+3}$

...
$S_n\ \lambda \rightarrow r_n\ L\ S_x$
$S_x\ r_{n-1} \rightarrow r_{n-1}\ L\ S_x$
$S_x\ r_{n-2} \rightarrow r_{n-2}\ L\ S_x$

...
$S_x\ r_{m+2} \rightarrow r_{m+2}\ L\ S_x$
$S_x\ x \rightarrow r_{m+1}\ L\ S_0$
$S_0\ r_m \rightarrow$ далее работа аналогично машине ***T*** на ленте t,

где:

x – некоторая буква, в других случаях не встречающаяся на входной ленте t;

$S_{1, \ldots,} S_n, S_x$ – новые состояния машины $\boldsymbol{M_{Tt}}$, которых не было в машине ***T.***

Следовательно, машина $\boldsymbol{M_{Tt}}$ при запуске на пустой ленте эквивалента машине ***T***, работающей на ленте t, так как, по сути, машина $\boldsymbol{M_{Tt}}$ просто печатает копию ленты t на своей ленте, затем выбирает нужную позицию и после этого становится полностью идентичной машине ***T***. Значит $\boldsymbol{M_{Tt}}$ и ***T*** - эквивалентные машины.

Предположим, что задача об остановке машины на чистой ленте алгоритмически разрешима, значит ее можно решить и применительно к машине $\boldsymbol{M_T}$, начинающей работу на чистой ленте. Соответственно, такая задача решается и для машины ***T*** на ленте t, что есть противоречие с доказанным в теореме 1.5.(1) утверждением о том, что для произвольной машины ***T*** задача остановки при обработке произвольного слова t алгоритмически неразрешима. Отсюда следует, что задача об остановке машины на чистой ленте алгоритмически неразрешима, **Q.E.D.**

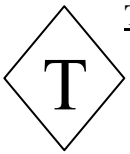
В данном доказательстве используется косвенный метод, называемый методом сведения.

В п.1.5.1. была доказана неразрешимость проблемы останова для произвольных пар (***T***, t). Только что показано, что каждой паре (***T***, t) соответствует легко конструируемая машина $\boldsymbol{M_{Tt}}$, которая останавливается на пустой ленте тогда и только тогда, когда имеет место останов пары (***T***, t).

Способность решать частные проблемы останова (***T***, пустая лента), которая включает все ситуации ***(M_{T_t}*** пустая лента), предоставила бы возможность решить все проблемы останова для произвольных пар (***T***, t). Можно сказать, что значительно более трудная проблема для пар (***T***, t) ***свелась*** к значительно более простой проблеме для пар (***T***, пустая лента).

Сводимость – достаточно тонкое и важное понятие в теории вычислимости. Оказывается, можно определить бесконечную иерархию проблем неразрешимости возрастающей сложности, ни одна из которых не сводится к предшествующей. Было даже показано, что существуют пары неразрешимых проблем, ни одна из которых не сводится к другой.

1.5.3. Задача об остановке конкретной универсальной машины на произвольной ленте специального типа



Т.1.5.(3) Теорема (без доказательства)

Задача об остановке конкретной универсальной машины на произвольной ленте специального типа алгоритмически неразрешима.

Помещение описания машины ***T*** и ленты t на ленте универсальной машины ***U*** и запуск последней дает некоторое упрощение проблемы останова. В случае когда ***T*** на ленте t остановится, логично утверждать что остановится и машина ***U***. Соответственно, если ***T*** на ленте t никогда не остановится, логично утверждать что никогда не становится и машина ***U***. Поэтому общая проблема с произвольной машиной и произвольной лентой легко сводится к проблеме *конкретной* универсальной машины ***U*** с лентой вида, представленной на рис.1.9.:

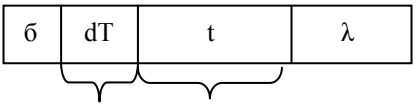


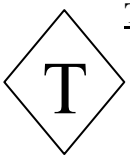
Рис. 1.9. Лента универсальной машины ***U***

На первый взгляд утверждение кажется спорным на основе обычных познаний в комбинаторике. Если предположить, что процесс вычислений в конечном итоге завершится, то с математической точки зрения можно найти верхнюю границу времени, затраченного на вычисления. Такая граница, безусловно, существует, потому что машина ***T*** имеет известное число n состояний, ее алфавит состоит из известного числа m символов, а лента t содержит известное число q непустых ячеек. Таким образом, в природе существует только конечное число вычислений, связанных со значениями (n,m,q), и, следовательно, только конечное число вычислений, кончающихся остановом. Тогда f(n,m,q) – точная длина самого длинного процесса вычисления из этого конечного множества.

Осталось всего лишь заставить машину ***U*** вычислять эту верхнюю границу, т.е. функцию f(n,m,q), значение которой равно количеству тактов, за которые машина должна перейти к заключительному состоянию и остановиться, если ей вообще судьба когда - нибудь остановиться. При этом машина ***U*** (а точнее, ее более продвинутая модификация), может начать имитацию поведения машины ***T*** с лентой t и вести счет числа циклов. Если вычисление не завершилось за время f(n,m,q), то ***U*** может смело останавливаться и сообщать что вычисления на машине ***T*** никогда не закончатся.

При всей логичности такого хода решения проблемы оказывается, что не существует машины, которая может вычислить верхнюю границу функции f(n,m,q), при всем при том, что сама эта граница *существует*. Эту печальную ситуацию не улучшает даже тот очевидный факт, что нам вовсе не обязательно точно знать вид функции f(n,m,q) – достаточно использовать любую функцию с достаточно большими значениями g(n,m,q), такую, что $g(n,m,q) \geq f(n,m,q)$. Это первый звонок в направлении довольно печальной ситуации, с которой мы столкнемся позднее: оказывается, что нет *вычислимых* функций, которые могут расти столь быстро (подробнее о вычислимых функциях рассказано в главе 2). А раз так – то нет и соответствующих машин Тьюринга.

1.5.4. Задачи о печатании символов



Т.1.5.(4) Теорема

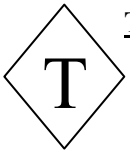
Задача о печатании данного символа на чистой ленте точно один раз алгоритмически неразрешима.

Доказательство

Без ограничения общности, возьмем знак «0». Итак, машина Тьюринга ***T*** работает на чистой ленте. Преобразуем ее в новую машину Тьюринга ***D***. Если ***T*** не содержит в своем алфавите знака «0», то ***D*** просто совпадает с ***T***. Если ***T*** имеет этот знак в своем алфавите, то в алфавите машины ***D*** знак «0» будет заменен любым знаком, ранее не входящим в алфавит машины. Очевидно, что ***D*** остановится тогда, когда остановится ***T***.

Построим машину ***E***, которая работает как ***D*** вплоть до ее остановки. После этого машина ***E*** печатает «0» и тоже останавливается.

Получим, что символ «0» печатается точно один раз в том случае, если произвольная машина ***T*** останавливается на чистой ленте. Значит, задача о печатании ровно одного нуля равносильна задаче об остановке машины на чистой ленте. Поскольку задача останова алгоритмически неразрешима, то и задача о печатании символа точно один раз тоже неразрешима, **Q.E.D.**



Т.1.5.(5)Теорема

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

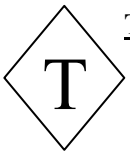
Доказательство

Без ограничения общности, возьмем знак «0». Итак, машина Тьюринга ***T*** работает на чистой ленте. Преобразуем ее в новую машину Тьюринга ***D***. Если ***T*** не содержит в своем алфавите знака «0», то ***D*** просто совпадает с ***T***. Если ***T*** имеет этот знак в своем

алфавите, то в алфавите машины ***D*** знак «0» будет заменен любым знаком, ранее не входящим в алфавит машины. Очевидно, что ***D*** остановится тогда, когда остановится ***T***.

Построим машину ***E***, которая работает как ***D*** вплоть до ее остановки. После этого машина ***E*** переходит в состояние A и печатает «0» бесконечно много раз ($A\lambda \rightarrow 0RA$).

Таким образом, получим, что символ «0» печатается бесконечно много раз в том случае, если произвольная машина ***T*** останавливается на чистой ленте. Значит, задача о печатании бесконечно большого числа нулей равносильна задаче об остановке машины на чистой ленте. Поскольку задача останова алгоритмически неразрешима, то и задача о печатании символа бесконечно много раз тоже неразрешима, **Q.E.D.**



Т.1.5.(6) Теорема

Задача о печатании данного символа на чистой ленте хотя бы один раз алгоритмически неразрешима.

Доказательство

Без ограничения общности возьмем знак «0». Итак, машина Тьюринга ***T*** работает на чистой ленте. Построим машину ***E***, которая работает как ***T*** вплоть до ее остановки. После чего машина ***E*** печатает «0» и останавливается. В итоге будет напечатан хотя бы один символ «0» (возможно и больше, если ранее машиной ***T*** такой символ уже печатался).

Таким образом, получим, что символ «0» печатается хотя бы один раз в том случае, если произвольная машина ***T*** останавливается на чистой ленте. Значит, эта задача равносильна задаче об остановке машины на чистой ленте. Поскольку эта задача согласно теореме 1.5.(2), алгоритмически неразрешима, то и задача о печатании символа хотя бы один раз тоже неразрешима, **Q.E.D.**

1.5.5. Обобщение проблемы алгоритмической неразрешимости

Доказанные ранее теоремы об алгоритмической неразрешимости имеют общие черты. Для обобщения введем новые определения.

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

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

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

Т.1.5.(7) Теорема Райса (без доказательства)

Ни для какого нетривиального инвариантного свойства машин Тьюринга не существует алгоритма, позволяющего для любой машины Тьюринга узнать, обладает ли она этим свойством.

К таким ситуациям относится и факт, доказанный в теореме 5.2.(1). Свойство машины останавливаться инвариантное и нетривиальное, и поэтому узнать обладает ли произвольная машина этим свойством невозможно. Теорема 5.2.(1) является центральной в череде доказательств неразрешимости, поэтому для более наглядного понимания этой фундаментальной проблемы приведем пример, взятый у самого Алана Тьюринга.

Требуется составить программу U, которая бы по любому подаваемому ей на вход файлу X, содержащему текст программы

(например, на каком-нибудь языке программирования), определяла бы, остановится ли когда-нибудь программа из файла X в процессе своей работы, получив на вход известные данные, или "зациклится". Если программа X зациклится, то программа U должна показать на экране фотографию юноши, иначе — девушки, после чего закончить свою работу. Кстати, такого рода "проверяющая на зацикливаемость" программа U была бы, очевидно, довольно полезна для проверки создаваемых компьютерных программ. Оказывается, написать эту "проверяющую программу" U невозможно в принципе, даже если допустить, что компьютер, на котором выполняется U, имеет сколь угодно большую память и может работать неограниченно долгое время. Если бы такая программа U была написана, то ее можно было бы легко переделать так, чтобы вместо команды вывода изображения девушки на экран она бы зацикливалась (вставить для этого, скажем, "вечный цикл"). Пусть эта переделанная программа называется U2. Что будет делать программа U2, если ей на вход подать текст программы U2 (текст себя самой)? Если она зацикливается, то она должна показать фотографию юноши и остановиться, т.е. не зациклиться. Но если она не зацикливается, это значит, что она должна зациклиться (поскольку вывод девушки в программе U был заменен на "вечный цикл"). Тем самым программа U2 оказывается в безвыходном положении, несовместимом с допущением возможности ее существования.

1.5.6. Задача об остановке конкретной машины на конкретной ленте

Продолжая уменьшать неопределенность при решении проблемы останова, сформулируем проблему еще более конкретно. Является ли задача останова конкретной машины T_0 при конкретном входном слове t_0 алгоритмически разрешимой?

Если дана пара (T_0 t_0), то может оказаться, что никому и никогда не удастся выяснить, остановится ли машина T_0 на входном слове t_0. Самое удивительное, что хотя может быть неизвестен способ решения этой задачи, но точно *исключена*

ситуация, чтобы кому-то удалось *доказать,* что не существует способа ее решения.

Т.1.5.(8) Теорема

Невозможно доказать алгоритмическую неразрешимость задачи об остановке конкретной машины T_0 на конкретном входящем слове t_0.

Доказательство

Предположим противное. Пусть доказано, что не существует способа определить, остановится машина T_0 на ленте t_0 или нет. Пусть тем или иным образом построили реальный прототип или обычный программный эмулятор данной машины. Если это прототип, то у нас есть достаточный запас ленты для медленной, но по существу, неограниченной ее подачи (например, небольшая бумажная фабрика). Тогда возможны лишь два случая: либо машина T_0 остановится, либо нет. В первом случае проводимый эксперимент закончится однозначным выводом об остановке машины, что явно противоречило бы предположению о том, что доказана неразрешимость данной проблемы. Значит, остается единственная возможность сохранить непротиворечивость утверждения – а именно, признать, что машина T_0 никогда не остановится. Поэтому, доказательство того, что не существует способа определить, остановится машина T_0 на ленте t_0 или нет, автоматом означает доказательство того, что этой остановки никогда не произойдет. Следовательно, ответ на вопрос все-таки существует, что есть явное противоречие с принятым предположением, **Q.E.D.**

Итак, появился принципиально другой тип проблемы. С одной стороны, доказать неразрешимость этой проблемы нельзя, при всем при том, что дать эффективный алгоритм решения тоже не представляется возможным.

§ 1.6. Нормальные алгоритмы

Если попытаться уйти от наличия самого управляющего механизма машины Тьюринга со своим собственным регистром для записи внутреннего состояния и перенести информацию о состоянии некоторого агрегата непосредственно на ленту, можно получить новую нотацию для записи алгоритмов. Проблемы движения управляющего механизма в этом случае не существует: для выполнения предписания необходимо будет проанализировать заданное слово и найти первое попавшееся соответствие между левой частью инструкции и каким-либо фрагментом содержимого ленты. Для удобства допустим, что анализ производится укрупненно, не по одному символу, а сразу по нескольким. Кроме того, лента является «растяжимой», т.е. вместо одного символа можно вписывать произвольное их количество и наоборот, без процедуры сдвигания части слова.

Таким образом, формируется идея нормального алгоритма переработки входного слова, называемого в литературе ***алгоритмом Маркова***.

> ***Нормальный алгоритм Маркова*** *– математическое построение, предназначенное для уточнения понятия алгоритм, которое задается алфавитом и нормальной схемой подстановок, выполняемых по заранее определенной схеме.*

Нормальные алгоритмы можно рассматривать как обобщение машины Тьюринга. В свою очередь работу машин Тьюринга можно рассматривать как переработку начального слова некоторого нормального алгоритма. Этот алгоритм получается сразу же из таблицы машины.

Пусть существует следующая машина Тьюринга, которая печатает на чистой ленте последовательность 001001001001….

Aλ → 0RB
Bλ → 0RC
Cλ → 1RA

Работа алгоритма происходит следующим образом (через запятую указана пара: символ-состояние)

A, 0B, 00C, 001A

Построим алгоритм Маркова, для чего к внешнему алфавиту {0,1} добавляем внутренний алфавит {A,B,C...}

A → 0B

B → 0C

C → 1A

Λ → A

Изначально лента пустая (содержит только символ Λ). Последняя строчка выполнится первой, что позволит записать на ленте символ A. Имея на ленте символ A, в процессе работы алгоритма, мы последовательно получим:

Λ

A

0B

00C

001A

0010B

…и т.д., что, по сути, означает бесконечно написание последовательности 001001001…

Таким образом, всегда на основе машины Тьюринга довольно легко можно получить работающий алгоритм Маркова.

Любой нормальный алгоритм можно в свою очередь преобразовать в машину Тьюринга, но это более сложно. Сложности связаны с тем, что у Маркова укрупненный алгоритм, т.к. сразу читается и может быть записано несколько символов.

По своей сути основная операция при работе алгоритмов Маркова – это переработка слов в некотором алфавите. Эта переработка заключается в производстве некоторого количества замен определенных последовательностей символов. Эти замены совершаются в СТРОГО определенном порядке, а именно: ***после каждой замены алгоритм читается с самого начала, а слово анализируется с самого первого (левого) символа***. В отличие от машин Тьюринга, алгоритмы Маркова выполняются без какого-либо устройства, осуществляющего движения и имеющего внутреннюю память. В данном случае мы можем оперировать

только ленточными знаками. Сама лента в этом случае не разделяется на строгие ячейки, а имеет гибкую основу, что позволяет ей растягиваться и сжиматься исходя из того, увеличивается ли в слове число символов или уменьшается.

Формат команды (строки) следующий:

$\{a_i\} \rightarrow \{b_j\}$ [•],

где

$\{a_i\}$ - последовательность символов, которая ищется в слове,

$\rightarrow$ - знак перехода к операции записи,

$\{b_j\}$ - последовательность символов, которая записывается вместо найденной последовательности,

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

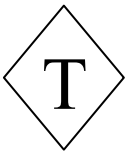
Λ – служебный знак, обозначающий пустой символ, присутствует везде: изначально на ленте (если она пустая), справа и слева от каждого символа (если на ленте записано слово).

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

> ***Тезис Маркова**: любой вычислительный процесс можно преобразовать в нормальный алгоритм.*

§ 1.7. Эффективные перечислимость и распознаваемость

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



Т.1.7.(1) Теорема

Множество машин Тьюринга эффективно перечислимо.

Доказательство

Идея: описать произвольную машину Тьюринга некоторым числом, которое эффективно распознаётся среди натуральных чисел. Тогда, применив алгоритм распознавания к натуральному ряду, удастся перенумеровать все машины Тьюринга. А это будет означать, что они эффективно перечислимы.

Произведём кодировку. Для этого перечислим и пронумеруем состояния (символы внутреннего алфавита) и закодируем их единичками:

S_0	Ω	A	B	C
1-й	2-й	3-й	4-й	5-й
1	11	111	1111	11111

Пронумеруем ленточные знаки (символы внешнего алфавита) и закодируем их двойками:

∂	λ	a	b	c
1-й	2-й	3-й	4-й	5-й
2	22	222	2222	22222

Символы, определяющие движение управляющей головки получат коды, состоящие из троек. Стрелку в строке таблицы обозначим четверкой, а разделитель между строк (переход на новую строку) обозначим пятеркой.

R	L	H	➔	новая строка
3	33	333	4	5

Например: A a ➔ b L B ←→ 111222422223311115.

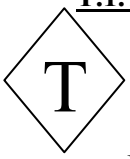
Поменяем местами символы элемента алфавита и состояния (в левой части), получим: a A ➔ b L B ←→ 22211142222331115.

Теперь можно убрать “4” и “5”, тогда получим 222111222233111, тем самым уменьшив длину кода (данный шаг не

является обязательным). Таким образом, каждой машине Тьюринга сопоставлено натуральное число.

Теперь возьмем ряд натуральных чисел и применим к нему алгоритм распознавания, определяющий, является ли данное число кодом какой-либо машины Тьюринга. Если в результате проверки выясняется, что данный формат допустим, то соответствующему числу присваивается очередной свободный номер. Следовательно, каждая машина Тьюринга получит свой собственный номер, следовательно, такие машины эффективно перечислимы, **Q.E.D.**

Т.1.7.(2) Теорема



Множество алгоритмов Маркова эффективно перечислимо.

Доказательство

Идея: описать произвольный алгоритм некоторым числом, которое эффективно распознаётся среди натуральных чисел. Тогда применив алгоритм распознавания к натуральному ряду, удастся перенумеровать все нормальные алгоритмы. А это означает, что они эффективно перечислимы.

Произведём кодировку следующим образом:

Нормальный алгоритм представляет собой набор строк, например:

ab → c

Λ → d •

Пронумеруем символы алфавита. Первые три символа служебные, для них зарезервируем числа 1, 2, 3.

→ "1"

Λ "2"

• "3"

Остальные символы получают следующие вакантные номера:

a (x_1) "4"

b (x_2) "5"

c (x_3) "6"

. . . .

(x_k) "k+3"

Отступление *(из области свойств простых чисел). Если мы возьмем ряд простых чисел и каждое из них возведём в какую-нибудь степень, а затем их перемножим, то по полученному числу мы сможем сказать, в какой именно степени были соответствующие простые числа. Это делается путем деления получившегося числа поочередно на простые числа, начиная с двойки, до тех пор, пока возможно деление без остатка. Потом производится деление на 3, потом на 5 и т.д. до тех пор, пока не получится единица.*

$2^x \cdot 3^y \cdot 5^z \cdot \ldots = N$

Например, мы можем рассчитать степени, в которые были возведены простые числа, произведение которых образует 540:

$540 = 2^x \cdot 3^y \cdot 5^z$

$540/2=270;\ 270/2=135;\ 135/3=45;\ 45/3=15;\ 15/3=5;\ 5/5=1$

Двойка участвовала в делении 2 раза, тройка - 3 раза, пятерка – 1 раз. Проверим: $2^2 \cdot 3^3 \cdot 5^1 = 4 \cdot 27 \cdot 5 = 540$

Возьмём первую строчку алгоритма Маркова и представим её следующим образом:

a b → c

$2^4 \cdot 3^5 \cdot 5^1 \cdot 7^6 = A$

Полученное натуральное число ***A*** является кодом данной строчки (такая нумерация называется Гёделевой).

Аналогично поступаем со второй строчкой алгоритма:

Λ → d •

$2^2 \cdot 3^1 \cdot 5^7 \cdot 7^3 = B$

B является кодом второй строчки.

Далее формируем кодовое число всего алгоритма. Для этого выпишем ряд простых чисел и возведем каждое число в соответствующую этой строчке степень.

$2^A, 3^B, \ldots$

Перемножив эти числа, получим кодовый номер алгоритма.

$2^A \cdot 3^B \ldots = K$

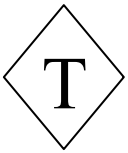
Таким образом, алгоритм представлен в виде произведения простых чисел, взятых в степенях, соответствующих коду каждой строки. Полученное число **К** является кодом данного алгоритма.

При этом по данному коду можно восстановить полностью весь алгоритм.

Теперь возьмем ряд натуральных чисел и применим к нему алгоритм распознавания, определяющий является ли данное число кодом какого либо алгоритма Маркова. Такой алгоритм распознавания будет состоять из двух процедур: сначала по данному числу **К** вычисляются числа **A, B, C**… (коды строк), а затем из этих чисел извлекаются соответствующие x, y, z,… (коды символов в рамках каждой строки). Полученные значения проверяются на предмет соответствия формату строк алгоритмов Маркова. Если ответ положительный, то соответствующему числу **К** присваивается очередной свободный номер. Таким образом, каждый алгоритм Маркова получит свой собственный номер, следовательно, нормальные алгоритмы эффективно перечислимы, **Q.E.D.**

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

Т.1.7.(3) Теорема Поста



Если множество A эффективно перечислимо, то подмножество B эффективно распознается в A тогда и только тогда, когда B и A\B оба эффективно перечислимы.

Доказательство

необходимость

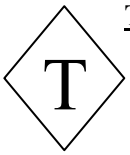
Пусть B распознается в A. Множество A представляет собой набор элементов $A=\{a_1,a_2,\ldots\}$. Вытаскиваем $\forall$ элемент a_i из множества A.

- Если $a_i \in B$, то нумеруем его в B,
- Если $a_i \notin B$, то нумеруем его в A\B.

Так как A эффективно перечислимо, то таким образом будут вытащены все его элементы. Таким образом эффективно перечислены множества B и A\B.

достаточность

Пусть B и A\B эффективно перечислимы. Множества B и A\B представляет собой набор элементов $B=\{x_1,x_2,\ldots\}$ и $A\backslash B=\{y_1,y_2,\ldots\}$. В силу того, что множества B и A\B эффективно перечислимы, существуют алгоритмы их перечисления. Т.к. A эффективно перечислимо, то начнем перечислять его элементы a_i. Для каждого элемента запустим параллельно алгоритмы перечисления B и A\B (поочерёдно по одному элементу из каждого множества) и будем сравнивать элементы множеств с a_i. Так как a_i принадлежит либо B, либо A\B, то он встретится на каком-то конечном шаге перечисления. Таким образом, можно определить, к какому множеству он относится, а значит, B эффективно распознается в A, **Q.E.D.**



Т.1.7.(4) Теорема

Множество останавливающихся машин Тьюринга эффективно перечислимо.

Доказательство

Возьмем множество машин, останавливающихся на первом такте, и пронумеруем их: $T_{11},T_{12},T_{13},T_{14}\ldots$

Затем пронумеруем множество машин, останавливающихся на втором такте: $T_{21},T_{22},T_{23},T_{24}\ldots$

Аналогично, на третьем такте: $T_{31},T_{32},T_{33},T_{34}\ldots$, и т.д.

Затем расположим их в виде бесконечной матрицы. Используя диагональный метод (рис.1.10.), перечислим их (пронумеруем натуральными числами):

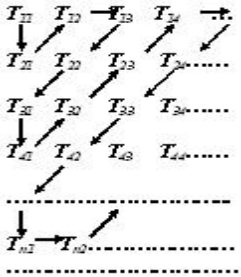


Рис. 1.10. Диагональная нумерация

Таким образом, каждая останавливающаяся машина Тьюринга получит свой собственный номер. Отсюда следует, что останавливающиеся машины можно эффективно перечислить, **Q.E.D.**

Т.1.7.(5) Теорема

Множество не останавливающихся машин Тьюринга невозможно эффективно перечислить.

Доказательство

Предположим противное, а именно, что множество не останавливающихся машин эффективно перечислимо. Тогда используем теорему Поста 1.7.(3), которая гласит, что подмножество B эффективно перечислимого множества A эффективно распознается в нем тогда и только тогда, когда это подмножество и его дополнение до всего множества эффективно перечислимы.

Пусть множество A – это множество всех машин Тьюринга (оно эффективно перечислимо по теореме 1.7.(1)), а подмножество B – это множество останавливающихся машин Тьюринга (оно эффективно перечислимо по Теореме 1.7.(4)). Тогда, по теореме 1.7.(3) и из нашего предположения об эффективной перечислимости не останавливающихся машин Тьюринга следует, что множество останавливающихся машин Тьюринга (B) эффективно распознается среди всех машин Тьюринга (A). Однако этого быть не может, так как общая задача об остановке машины

Тьюринга на произвольной ленте неразрешима. Получили противоречие. Значит, исходное предположение неверно, т.е. множество не останавливающихся машин Тьюринга эффективно не перечислимо, **Q.E.D.**

Задачи к главе 1

Одноленточные машины Тьюринга

При написании алгоритмов рекомендуется придерживаться следующей схемы (порядка символов в команде)

S_0∂->∂RA
Aa->bRA
Aλ->cH**Ω**

Жирным выделены зарезервированные символы (начальные состояние и символ, конечное состояние и пустой символ).

Необходимо придерживаться определенных правил при написании программ для машины Тьюринга:

1. Начальный символ трогать нельзя, он должен быть строго один на ленте в фиксированном месте.
2. Пустых символов бесконечно много, и если такой символ появился – далее слово считается законченным.
3. Начальное состояние S0 можно использовать и как рабочее, т.е. смена состояния в первой строке должна производиться только по необходимости.

Задача 1.1

Дано слово в алфавите {0,1}. Написать "1" в конце слова.

Решение:

S_o∂→∂RS_o
S_o1→1RS_o
S_o0→0RS_o
S_oλ→1HΩ

Задача 1.2

Дано слово в алфавите {0,1}. Написать в конце слова последний символ.

Решение:

Существуют два различных варианта решения:

1. Доходим до конца слова и возвращаемся назад с целью запомнить последний символ

$S_0∂ \rightarrow ∂RS_0$
$S_01 \rightarrow 1RS_0$
$S_00 \rightarrow 0RS_0$
$S_0λ \rightarrow λLA$ //A – дошли до конца слова
$A0 \rightarrow 0RB$
$A1 \rightarrow 1RC$
$A∂ \rightarrow ∂ΩH$ //Слова нет
$Bλ \rightarrow 0ΩH$
$Cλ \rightarrow 1ΩH$

2. Идем по слову, каждый раз запоминаем результат, по окончании слова последний результат записываем в пустую ячейку.

$S_0∂ \rightarrow ∂RS_0$
$S_01 \rightarrow 1RA_1$ // A_1 – нашли 1
$S_00 \rightarrow 0RA_0$ // A_0 – нашли 0
$S_0λ \rightarrow λHΩ$ //слова нет
$A_11 \rightarrow 1RA_1$
$A_10 \rightarrow 0RA_0$
$A_1λ \rightarrow 1HΩ$
$A_01 \rightarrow 1RA_1$
$A_00 \rightarrow 0RA_0$
$A_0λ \rightarrow 0HΩ$

Задача 1.3

Дано слово в алфавите {0,1}. Скопировать 1-й символ, записать его в конце слова.

Задача 1.4

Дано слово в алфавите {0,1,2}. Все последовательности «012» заменить на последовательность из трех звездочек «***».

Задача 1.5

Дано число n в унарном коде. Найти значение функции псевдоразности f(n)= n÷2. Результат записать вместо слова.

$$n \div 2 = \begin{cases} n-2, \text{ если } n \geq 2 \\ 0, \text{ если } n < 2 \end{cases}$$

Унарный код оперирует символами только одного типа, поэтому количество предметов кодируется поштучно, например | (одна палочка) это число “0”, || (две палочки) –число “1”, ||| - “2”, и т.д.

Решение:

$S_0 \partial \rightarrow \partial$ R S_0
S_0 | $\rightarrow$ | R S_0
$S_0 \lambda \rightarrow \lambda$LA //дошли до конца слова
A | $\rightarrow \lambda$L A_1 //стерли последнюю палочку
$A \partial \rightarrow \partial H\Omega$ //слова нет
A_1 | $\rightarrow \lambda A_2$L //стерли предпоследнюю палочку
$A_1 \partial \rightarrow \partial$BR //задан аргумент n=0 (одна палочка), и мы ее стерли
$B\lambda \rightarrow$ | $H\Omega$ //восстанавливаем палочку, т.к. f(0)=0 и f(1)=0
A_2 | $\rightarrow$ | $H\Omega$ //задан аргумент n≥2, задача выполнена
$A_2 \partial \rightarrow \partial$RB //задан аргумент n=1 (было две палочки, мы их стерли)

Задача 1.6

Дано слово в алфавите {a,b}. Скопировать его в обратном порядке, записав копию сразу вслед за исходным словом.

Задача 1.7

Дано слово в алфавите {a,b}. Скопировать его в прямом порядке, записав копию сразу за исходным словом.

Задача 1.8

Дано слово в алфавите {a,b}. Скопировать его в обратном порядке, записав копию вместо исходного слова.

Задача 1.9

Дано слово в алфавите {a,b}. Определить, является ли слово палиндромом. Палиндром – слово, которое читается одинаково справа налево и слева направо. Если да – написать после слова «+», если нет – «-».

Задача 1.10

На ленте в унарном коде записаны числа x и y, между ними знак - . Вместо исходной последовательности символов написать результат вычисления функции f(x,y)=x-y.

Многоленточные машины Тьюринга

Основные принципы работы многоленточных машин Тьюринга:

1. Управляющие головки на каждой из лент двигаются независимо.
2. Внутреннее состояние у машины единое. Это свойство самой машины, а не управляющих головок и лент.

При написании алгоритмов следует придерживаться следующей схемы: $S_i\{a,b,c\}\rightarrow\{a',b',c'\}\ \{R,L,H\}\ S_j$

Задача 1.11

Алфавит {0,1}. Машина двухленточная. Дано слово, оно записано на первой ленте. Скопировать это слово на вторую ленту в обратном порядке.

Задача 1.12

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

Важно: в ходе сложения, возможно, выяснится, что разрядов не хватает – тогда надо будет сдвинуть результат на одну клетку вправо для записи первой единицы.

Задача 1.13

Алфавит унарный. Машина трехленточная. Даны два числа в унарном коде, они записаны на первой и второй ленте соотв. Определить, чему равно произведение этих двух чисел и результат записать на третью ленту.

Алгоритмы Маркова

При написании алгоритмов следует придерживаться следующей схемы:

$\{a_i\} \rightarrow \{b_j\}$ [•], где

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

$\rightarrow$ – знак перехода к операции записи,

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

[•] – знак окончания алгоритма (необязательный параметр),

Λ – знак пустого символа, присутствует везде: изначально на ленте если она пустая, справа и слева от каждого символа, если на ленте записано слово).

Программа (алгоритм) представляет собой совокупность строк указанного вида, причем порядок строк имеет важнейшее значение. Основная операция при работе алгоритмов Маркова – это переработка слов в некотором алфавите. Эта переработка заключается в производстве некоторого количества замен определенных последовательностей символов. Эти замены совершаются в СТРОГО определенном порядке, а именно: после каждой замены алгоритм читается с самого начала, а слово анализируется с самого первого символа. Окончание работы алгоритма происходит в тот момент, когда выполняется строка, содержащая знак принудительной остановки, либо тогда, когда более ни одна строка не может быть выполнена (в слове нет ни одной из искомых последовательностей символов).

Задача 1.14

Алфавит {0,1}. Переработать произвольное слово в 0.

Задача 1.15

Алфавит {0,1}. Удвоить все символы в слове, результат записать вместо слова.

Задача 1.16

Алфавит {0,1}. Удалить каждый третий символ.

Задача 1.17
Алфавит {0,1}. Поставить в конце слова +, если в нем присутствует хотя бы один ноль и – , если это не так.

Задача 1.18
Алфавит {0,1}. Поставить в конце слова +, если в нем есть строго два нуля (не важно, соседние они или нет) и поставить в конце – , если это не так.

Задача 1.19
Алфавит {0,1}. Поставить в конце слова +, если в нем есть точно два нуля и они стоят рядом (т.е. они соседние) и в – иначе.

Задача 1.20
Алфавит {0,1}. Скопировать слово в прямом порядке после слова.

Задача 1.21
Даны два числа в унарном коде, разделенные знаком “+”. Вычислить сумму. Результат записать вместо слова.

Задача 1.22
Даны два числа в унарном коде, разделенные знаком “-”. Вычислить разность. Результат записать вместо слова.

Задача 1.23
Алфавит {a,b}. Подсчитать количество символов a. Результат записать после слова в унарном коде.

Задача 1.24
Даны два числа в унарном коде, разделенные знаком “/”. Вычислить целую часть от деления первого числа на второе. Результат записать вместо слова.

Задача 1.25
Алфавит {0,1}. Определить – каких символов 0 или 1 в слове больше. Если больше нулей – поставить в конце +, если больше 1 – поставить -. Если поровну – поставить в конце =.

Глава 2. Числовые множества

§ 2.1. Множества: определение и основные свойства

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

Можно привести другое определение множества:

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

Дополнительные определения и операции над множествами

1. Множество, которое не имеет ни одного элемента, называется пустым и обозначается $\emptyset$.
2. Единичное множество – множество, все элементы которого тождественны.
3. Множество M^1 называется подмножеством множества M тогда и только тогда, когда любой элемент множества M^1 принадлежит множеству M.
4. Множества называются равными, если они имеют одни и те же элементы.
5. Подмножество M^1 множества M называется *собственным* подмножеством множества M, если M^1 является его подмножеством, но при этом существует хотя бы один элемент, принадлежащий M, но не принадлежащий M^1.
6. Пусть A и B – два множества. Множество $M=A \cup B$ такое, что его каждый элемент принадлежит A или B (а возможно и A и B), называется суммой или объединением множеств A и B.
7. Пусть A и B – два множества. Множество $M=A \cap B$ такое, что его каждый элемент принадлежит и A и B одновременно, называется пересечением множеств A и B.
8. Пусть A и B – два множества. Множество $M=A \setminus B$ такое, что оно состоит из тех элементов множества A, которых

нет во множестве B, называется разностью множеств A и B, или дополнением B до A.

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

Пусть A = {a,b}; B = {m,n}

Тогда A×B = {(a,m),(a,n),(b,m),(b,n)}

10. Пусть A – множество. Множество M, элементами которого являются подмножества множества A, включая само A и пустое множество, называется множеством всех подмножеств множества A или булеаном A и обозначается P(A).

Пусть A = {a,b,c}

Тогда M= P(A)={Ø, (a), (b), (c), (a,b), (a,c), (b,c), (a,b,c)}

11. Отображением f множества A в множество B называется некое правило, по которому каждому элементу множества A ставят в соответствие элемент множества B.

12. Множество всех отображений множества A в B обозначается как B^A (B в степени A).

Пусть A = {a,b,c}; B = {m,n}

Тогда B^A это набор функций f_i приведенных в табл. 2.1.

Таблица 2.1. Отображения множества A в множество B.

A	$f_1(A)$	$f_2(A)$	$f_3(A)$	$f_4(A)$	$f_5(A)$	$f_6(A)$	$f_7(A)$	$f_8(A)$
a	*m*	*m*	*m*	*m*	*n*	*n*	*n*	*n*
b	*m*	*m*	*n*	*n*	*m*	*m*	*n*	*n*
c	*m*	*n*	*m*	*n*	*m*	*n*	*m*	*n*

Каждая такая функция задана своими значениями в каждой из трех точек области определенности.

§ 2.2. Классификация множеств

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

Можно привести другое определение мощности:

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

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

Кардинальные числа некоторых множеств

1. Мощность пустого множества равна 0: $|\varnothing|=0$.
2. Мощность множества из одного элемента равна 1: $|\{a\}|=1$.
3. Если множества равномощны ($A \sim B$), то их кардинальные числа равны: $|A|=|B|$.
4. Если A – подмножество B и $\exists$ C: $(A \sim C)\&(C \subseteq B)$, то кардинальное число A не превосходит кардинального числа B, т.е. $|A| \leq |B|$.
5. Мощность булеана множества A равна $2^{|A|}$: $|P(A)|=2^{|A|}$
6. Мощность множества всех отображений A в B равна $|B^{|A|}|$: $|B^A|=|B^{|A|}|$

Конечные, счетно-бесконечные и несчетные множества чисел

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

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

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

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

Несчетные *– бесконечные множества, не равномощные множеству натуральных чисел.*

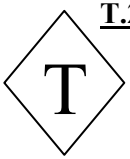
Можно классифицировать множества и по другому признаку: счетности. Тогда все множества делятся на два класса: **счетные** и **несчетные. Счетные** множества в свою очередь делятся на два подкласса: **конечные** и **счетно-бесконечные**.

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

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

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

Иллюстрацией данного факта могут служить два известных парадокса.



Т.2.2.(1) Парадокс Галилея

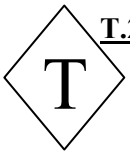
Хотя большинство натуральных чисел не является квадратами, всех натуральных чисел не больше, чем квадратов (если сравнивать эти множества по мощности).

Доказательство

Рассмотрим множество квадратов натуральных чисел:

1, 4, 9, 16, 25, 36,… Назовем его N_1. Пусть его мощность равна $|N_1|$. По построению $N_1 \subset N$ (N_1 собственное подмножество N). Пронумеруем множество N_1 натуральным рядом: $1 \to 1$, $4 \to 2$, $9 \to 3$, $16 \to 4$, $25 \to 5$, $36 \to 6$, ...

Следовательно, можно построить взаимнооднозначное соответствие, доказав, что $|N_1|=|N|$, значит, квадратов натуральных чисел столько же, сколько и самих натуральных чисел, **Q.E.D.**



Т.2.2.(2) Парадокс Гильберта

Если гостиница с бесконечным количеством номеров полностью заполнена, в неё можно поселить ещё посетителей, даже бесконечное число.

Доказательство

Первого постояльца следует поселить во второй номер, второго – в третий, далее аналогично, n-го в (n+1)-й. Поскольку номеров бесконечное количество, места всем хватит, т.к. для каждого натурального n найдется число n+1. Подобную процедуру можно повторять столько, сколько потребуется, **Q.E.D.**

Это оригинальная версия парадокса, в ней под бесконечным числом постояльцев следует понимать счетно-бесконечное число. Для обозначения мощности конечных множеств используются натуральные числа. Для обозначения мощности бесконечных множеств нужны числа иного рода, их называют трансфинитными.

Трансфинитное число *(finis – "конец", лат.) – кардинальное число бесконечного множества.*

Алеф-нуль ***($\aleph_0$)*** – первое трансфинитное число. По определению – это мощность множества всех натуральных чисел. Это наименьшая бесконечная мощность.

§ 2.3. Счетные множества

Счетно-бесконечными также будут все множества, для которых удасться доказать равномощность с множеством натуральных чисел. Далее в этой главе для краткости и соответствия общепринятым формулировкам теорем вместо термина «счетно-бесконечные» при доказательстве равномощности рассматриваемого множества и множества натуральных чисел будет использоваться термин «счетные». Это не является ошибочным утверждением, любое счетно-бесконечное множество является счетным, но не наоборот. Строго говоря, доказать факт того, что множество счетное проще, чем доказать том факт, что оно счетно-бесконечное (в последнем случае требуется показать, что множество не является конечным).

Для доказательства того, что множества равномощны, обычно используется какой-либо способ, позволяющий поставить в соответствие каждому элементу рассматриваемого множества какое-то натуральное число. Подобный прием использовался при доказательстве теорем 2.2.(1) и 2.2.(2). В общем случае оказывается вовсе не обязательным конкретное указание эффективного способа установления такого соответствия. Достаточно доказательства самого факта. Более того, если в процессе доказательства равномощности такой (обязательно эффективный, т.е. основанный на алгоритме) способ будет найден, то помимо собственно требуемого доказательства счетности, попутно будет доказан факт эффективной перечислимости исследуемого множества. При этом уже становится обязательным наличие процедуры, которая устанавливает взаимно - однозначное соответствие между элементами исследуемого множества и элементами множества натуральных чисел.

Помимо указанного способа, зачастую используется методика оценки кардинального числа множества сверху и снизу, что зачастую позволяет точно вычислить реальное значение мощности исследуемого множества.

В дальнейшем в ряде задач рассматривается «расширенное» множество натуральных чисел, включающее в себя стандартный ряд натуральных чисел (1,2,3,…) и число 0. Доказательство равномощности этих множеств не составляет труда. Будем обозначать множество натуральных чисел буквой N, а расширенное множество натуральных чисел N*.

Обычное и расширенное множество натуральных чисел являются эффективно перечислимыми (первое по определению, второе по причине простейшего установления нумерации 0->1, 1->2, 2->3,….и т.д., позволяющей установить взаимно-однозначное соответствие между элементами исследуемого множества и элементами множества натуральных чисел).

Стоит также отметить, что любое конечное множество также эффективно перечислимо. Важно обратить особое внимание на тот факт, что исходя из сформулированных определений, счетность конкретного множества вовсе не означает, что это множество гарантированно будет эффективно перечислимым. Более того, как будет показано в последующем, найдутся множества, являющиеся счетными и эффективно не перечислимыми одновременно.

2.3.1. Множество целых чисел

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

Т.2.3. (1) Теорема

Множество целых чисел счетно и эффективно перечислимо.

Доказательство

Ряд целых чисел: -n, …, -3,-2,-1,0,1,2,3,…, n,…Будем обозначать множество целых чисел буквой Z. Расположим целые числа следующим образом:

0, 1, -1, 2, -2, 3, -3, …., n, -n, …

Тогда каждому числу можно поставить в соответствие натуральное число

0,	1,	-1,	2,	-2,	3,	….,n, -n, …
1,	2,	3,	4,	5,	6,	….,2n, 2n+1, …

Таким образом доказано, что множество Z равномощно множеству N, а значит оно ***счетно.***

Для доказательства эффективной перечислимости множества Z необходимо установить тот факт, что все элементы множества Z могут быть перебраны по алгоритму и должны получить в результате такого перебора порядковые номера, без пропусков и повторений.

Факт ***эффективной перечислимости*** множества Z напрямую следует из приведенного способа нумерации элементов натуральными числами. Итак, множество Z счетно и эффективно перечислимо, **Q.E.D.**

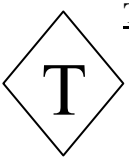
Если оперировать трансфинитными числами, получим:

$\aleph_0 + 1 + \aleph_0 = \aleph_0$

2.3.2. Множество упорядоченных пар натуральных чисел

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

Т.2.3. (2) Теорема



Множество упорядоченных пар натуральных чисел счетно и эффективно перечислимо.

Доказательство

Обычно, употребляя термин «упорядоченная» пара, считают, что допустим пара (1,5) и пара (5,1) имеют разный смысл и рассматриваются как различные. Чтобы установить взаимно-однозначное соответствие между упорядоченными парами натуральных чисел и натуральными числами, достаточно расположить пары (p,q) в таблицу так, что (p,q) находится в p-й строке и в q-м столбце.

(1,1)	(1,2)	(1,3)	...
(2,1)	(2,2)	(2,3)	…
…	…	…	…

Затем указанные пары перечисляются диагональным методом, начиная с левого верхнего угла. Последовательность обхода матрицы по сути может быть любой. Например, можно расположить пары в последовательность по возрастающей сумме p + q, а при равной сумме – по возрастанию p. Получим ряд:

n	1	2	3	4	5	6	…
(p,q)	(1,1)	(1,2)	(2,1)	(1,3)	(2,2)	(3,1)	…

Таким образом доказано, что множество упорядоченных пар натуральных чисел равномощно множеству N, а значит, оно ***счетно***.

Иногда под термином «***упорядоченные***» понимают ситуацию, при которой в паре (p,q), например, гарантированно $p \leq q$, т.е. первый член пары меньше или равен второму. В этом случае пары (p,q) и (q,p) считаются тождественными, т.е. пара воспринимается как неупорядоченное множество из двух элементов, в котором на первом месте пишется меньшее число, а на втором – большее. Множество таких пар является собственным подмножеством множества рассмотренных выше пар и по логике вещей тем более будет ***счетно***.

Факт ***эффективной перечислимости*** множества упорядоченных пар натуральных чисел независимо от конкретной трактовки термина «упорядоченный» представляется вполне очевидным. В первом случае он напрямую следует из приведенного способа нумерации элементов натуральными числами. Во втором случае к предложенному алгоритму перечисления необходимо добавить процедуру проверки соотношения между элементами p и q, и если, например, $p \leq q$, то присваивать очередной номер этой паре, а в противном случае пропускать её. Итак, множество упорядоченных пар натуральных чисел счетно и эффективно перечислимо, **Q.E.D.**

Если оперировать трансфинитными числами, то при основной трактовке термина «упорядоченные» получим что $\aleph_0 \bullet \aleph_0 = \aleph_0$. Важно, что при второй трактовке этого термина получим тот же результат. Действительно, $(\aleph_0 \bullet \aleph_0 - \aleph_0)/2 + \aleph_0 = \aleph_0$ (на основе алгоритма построения пар формируем матрицу, из матрицы формата $\aleph_0$ на $\aleph_0$ выбрасываем элементы главной диагонали, остальное множество сокращаем вдвое и добавляем обратно элементы главной диагонали).

2.3.3. Множество упорядоченных n-ок натуральных чисел

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

T

Т.2.3. (3) Теорема

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

Доказательство

Чтобы установить взаимно-однозначное соответствие между упорядоченными n-ками натуральных чисел и натуральными числами, достаточно разложить n-ку вида $(m_1, m_2, m_3, \ldots, m_n)$

следующим образом: $(m_1, m_2, m_3,\ldots,m_n) = (m_1, (m_2, m_3,\ldots,m_n)) = (m_1, (m_2, (m_3,\ldots,m_n))) = (m_1, (m_2, (m_3, (\ldots(m_{n-1},m_n)))))$

Расположив по горизонтали таблицы пары натуральных чисел, а по вертикали – натуральные числа, диагональным методом получим нумерацию троек натуральных чисел. Далее по горизонтали таблицы располагаются тройки натуральных чисел, а по вертикали – натуральные числа, диагональным методом получаем нумерацию четверок натуральных чисел и т.д.

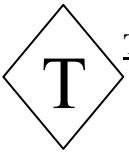
Таким образом доказано, что множество n-ок натуральных чисел равномощно множеству N, а значит, оно ***счетно***.

Факт ***эффективной перечислимости*** множества напрямую следует из приведенного способа нумерации элементов натуральными числами. Итак, множество упорядоченных n-ок натуральных чисел счетно и эффективно перечислимо, **Q.E.D.**

Если оперировать понятием кардинального числа (мощности), то получим, что произведенное n раз (n – натуральное число) умножение первого трансфинитного числа само на себя не изменяет его значения $\aleph_0 \bullet \aleph_0 \bullet \ldots \bullet \aleph_0 = \aleph_0$ или $\aleph_0^n = \aleph_0$

2.3.4. Множество конечных комплексов натуральных чисел

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



Т.2.3. (4) Теорема

Множество конечных комплексов натуральных чисел счетно и эффективно перечислимо.

Доказательство

Чтобы установить взаимно-однозначное соответствие между конечными комплексами натуральных чисел и натуральными числами, можно использовать двоичное разложение вида:

$n=2^{(p_1-1)} + 2^{(p_1+p_2-1)}+ \ldots+2^{(p_1+p_2+ \ldots+p_k -1)}$, где ^ - значок степени.

Например, в двоичном коде $27 = 11011= 1\cdot2^0 + 1\cdot2^1 +0\cdot2^2 +1\cdot2^3 +1\cdot2^4 = 2^0 + 2^1 +2^3 + 2^4$, откуда получим:

$p_1-1 = 0$	$p_1=1$	$p_1=1$
$p_1+ p_2 -1 = 1$	$p_1+ p_2=2$	$p_2=1$
$p_1+ p_2 + p_3 -1 = 3$	$p_1+ p_2 + p_3 = 4$	$p_3 =2$
$p_1+ p_2 + p_3 + p_4 -1 = 4$	$p_1+ p_2 + p_3+ p_4 = 5$	$p_4 =1$

Итак, натуральное число 27 является кодом комплекса (1,1,2,1).

В свою очередь комплексу (2,1,1,1) соответствует следующий код:

$p_1-1 = 2 -1 = 1$

$p_1+ p_2 -1 = 2 + 1 – 1 = 2$

$p_1+ p_2 + p_3 -1 = 2 + 1 - 1 -1 = 3$

$p_1+ p_2 + p_3 + p_4 -1 = 2 + 1 – 1 + 1 – 1 = 4$

В итоге число $n = 2^1 + 2^2 + 2^3 + 2^4 = 11110$ (в двоичном коде) или 2 + 4 + 8 + 16 = 30 (в десятичном коде). Таким образом, комплексу (2,1,1,1) соответствует натуральное число 30.

В результате доказано, что множество конечных комплексов натуральных чисел равномощно множеству N, а значит, оно ***счетно***.

Факт ***эффективной перечислимости*** множества напрямую следует из приведенного способа нумерации элементов натуральными числами. Итак, множество конечных комплексов натуральных чисел счетно и эффективно перечислимо, **Q.E.D.**

Если оперировать трансфинитными числами, то получим:

$$\aleph_0+\aleph_0{}^2+\aleph_0{}^3+\ldots+\aleph_0{}^k=\sum_{i=1}^{k}\aleph_0{}^k=\sum_{i=1}^{k}\aleph_0=k\cdot\aleph_0=\aleph_0$$

2.3.5. Множество рациональных чисел

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

Т.2.3. (5) Теорема

Т

Множество рациональных чисел счетно и эффективно перечислимо.

Доказательство

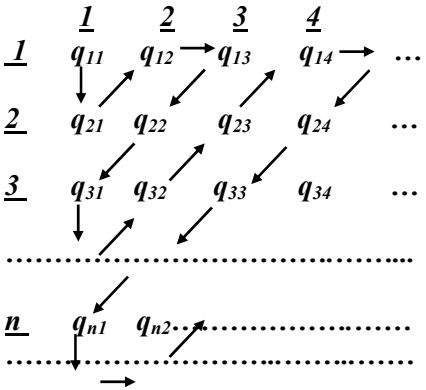
Обозначим множество рациональных чисел Q.

Рассмотрим сначала положительные рациональные числа – множество Q^+. Определим положительное рациональное число как q=n/m, где n и m – натуральные числа.

Запишем их в виде бесконечной матрицы, строки и столбцы которой пронумерованы натуральными числами, начиная с 1. Элемент, стоящий на пересечении i-й строки и j-го столбца, получит наименование q_{ij}

Используя диагональный метод, перечислим их (пронумеруем натуральными числами):

q_{11} q_{21} q_{12} q_{13} q_{22} q_{31} q_{41} q_{32} q_{23} q_{14} q_{15} q_{24} q_{33}
1 2 3 4 5 6 7 8 9



Все (и положительные, и отрицательные) рациональные числа в совокупности перечисляются по аналогии с целыми числами путем чередования положительной дроби и её отрицательного аналога. При этом некоторые рациональные числа мы нумеруем по нескольку раз: например, 1 будет пронумерована как 1/1, 2/2, и т.д., а, например, 4/5 как 8/10, 12/15 и т.д.

Таким образом, показано, что множество рациональных чисел не превосходит по мощности множество натуральных чисел, $|Q| \leq |N|$, т.к. каждое рациональное число получит соответствующий номер, а если быть точным – то даже несколько номеров. С другой стороны, то, что множество натуральных чисел не превосходит по мощности множество рациональных чисел, очевидно, $|N| \leq |Q|$ (хотя бы потому, что оно является его подмножеством). Следовательно, доказано, что множество рациональных чисел равномощно множеству натуральных чисел $|Q| = |N| = \aleph_0$, а значит оно ***счетно***.

Факт ***эффективной перечислимости*** множества Q напрямую следует из приведенного способа нумерации элементов натуральными числами. В ходе этой нумерации каждое рациональное число получает соответствующий номер, и если к алгоритму добавить процедуру, проверяющую дробь на предмет сокращаемости (если числитель и знаменатель имеют общие делители) и исключающую из нумерации сокращаемые дроби, то мы в чистом виде получим перечисление рациональных чисел по алгоритму без пропусков и повторений, что совпадает с определением эффективной перечислимости. Итак, множество рациональных чисел счетно и эффективно перечислимо, **Q.E.D.**

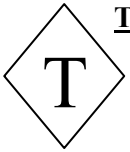
Это, вероятно, следующий, после парадоксов Галилея и Гильберта, не вполне очевидный и поначалу воспринимаемый как парадоксальный, факт. Ведь множество рациональных чисел расположено на прямой повсюду плотно, и между любыми двумя рациональными числами существует другие рациональные числа, однако мощность множества таких чисел оказывается не больше, чем у множества целых или натуральных чисел.

2.3.6. Множество действительных алгебраических чисел

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

Множество алгебраических действительных чисел обозначим латинской буквой A.

Общий вид алгебраического уравнения: $a_0+a_1 \cdot x^1+a_2 \cdot x^2+\ldots+a_n \cdot x^n=0$, где a_0, a_1,… – рациональные коэффициенты. Исходя из определения, можно утверждать, что рассмотренные ранее классы натуральных, целых и рациональных чисел являются подмножествами множества алгебраических чисел.



Т.2.3. (6) Теорема

Множество алгебраических чисел счетно и эффективно перечислимо.

Доказательство

Доказательство построим привычным образом, а именно предложим процедуру нумерации всех алгебраических чисел числами натурального ряда. При этом каждое число будем задавать через образующее его алгебраическое уравнение. Так, для линейных уравнений будем иметь упорядоченные пары рациональных чисел, для квадратных уравнений – тройки, в общем случае получаем упорядоченную n-ку рациональных чисел: $(a_{i1},a_{i2}\ldots a_{in})$ для каждого i-ого алгебраического уравнения (n-1)-й степени. Располагать элементы будем в двусторонне бесконечной матрице.

Выпишем на первой строке будущей матрицы все упорядоченные пары рациональных чисел. Это возможно, т.к. пары рациональных чисел эффективно перечислимы (рациональные числа эффективно перечисляются, их можно записать в матрицу и перечислить пары чисел диагональным способом). Такие пары рациональных чисел соответствуют линейным уравнениям и имеют по одному корню: каждая пара однозначно определяет корень линейного уравнения.

На второй строке выпишем все упорядоченные тройки рациональных чисел. Это возможно, т.к. тройки рациональных чисел эффективно перечислимы (рациональные числа эффективно перечисляются, их пары тоже эффективно перечисляются, значит можно записать в матрицу по строкам пары, по столбцам числа и перечислить тройки чисел диагональным способом). Такие тройки соответствуют квадратным уравнениям и имеют максимум по два

корня: таким образом, в процессе формирования матрицы каждую тройку рациональных чисел нужно будет повторить два раза для обеспечения процесса получения соответствующего номера для двух чисел, являющихся решением соответствующего уравнения.

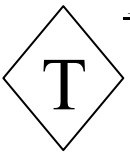
На третьей строке – по три числа на каждое кубическое уравнение соотв. упорядоченным четверкам и т.д.

Следовательно, получим матрицу, которую можно обойти при помощи диагонального процесса Кантора. Если часть корней алгебраического уравнения комплексная, при нумерации их просто пропускаем. Каждое алгебраическое число получит соответствующий номер, и это подтверждает тот факт, что множество алгебраических действительных чисел ***счетно***.

Факт ***эффективной перечислимости*** множества А напрямую следует из приведенного способа нумерации элементов натуральными числами, т.к. попутно указана эффективная процедура нумерации наборов рациональных чисел, однозначно задающих алгебраические уравнения соответствующей степени. При этом важно то, что алгебраическое уравнение n-й степени имеет эффективный алгоритм решения, т.е. процедура полностью эффективна. Итак, множество алгебраических действительных чисел счетно и эффективно перечислимо, **Q.E.D.**

Счетными также будут множества, составленные из всех пар, троек и т.д. алгебраических чисел.

2.3.7. Счетные числовые множества: обобщение



Т.2.3. (7) Теорема (без доказательства)

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

В реальной жизни мы используем различные конечные системы знаков, например цифры, буквы, ноты.

Рассмотрим систему знаков, например, числа в любой конечной системе счисления, допустим десятичной. Имея 10 знаков в нашем распоряжении: 0,1,2,3,4,5,6,7,8,9 мы можем

составлять два типа множеств: фиксированной длины и произвольной длины.

В первом случае речь идет о чисто комбинаторной задаче, например можно составить 10^5 различных последовательностей из пяти символов. Это немаленькое число, но оно натуральное и мощность рассматриваемого множества всех возможных последовательностей такого рода выражается натуральным числом. Во втором случае множество таких последовательностей будет счетно-бесконечно, по аналогии с множествами комплексов натуральных чисел, и его мощность есть число алеф-ноль.

Можно обобщить, что полученное в результате применения теоремы 2.3.(7) множество будет счетно-бесконечно, если в случае конечной системы знаков допустить сколь угодно длинные комплексы знаков (сколько угодно длинные, но при этом все равно конечные!).

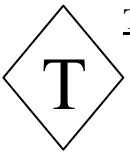
Счетно-бесконечными являются, например:

- множество «слов», которое можно составить при помощи конечного алфавита («слово» здесь - комплекс букв, не важно имеющих смысл или нет),
- множество всех книг, написанных на любом или даже на всех языках,
- множество всех симфоний и т.д.

§ 2.4. Несчетные множества

2.4.1. Несчетность множества действительных чисел (континуума)

Множество действительных чисел обозначим латинской буквой R.



Т.2.4. (1) Теорема

Множество действительных чисел несчётно.

Доказательство

Предположим противное: пусть множество действительных чисел счетное. Тогда любое подмножество счетного множества тоже счетное. Возьмём на множестве действительных чисел подмножество R_1 – интервал (0,1) и выкинем из этого отрезка числа, содержащие хотя бы в одном своём разряде нули или девятки (примеры таких чисел: 0.9, 0.0001 и т.д.). Множество R_2, составленное из оставшихся чисел, является подмножеством множества R_1 . Это означает, что R_2 – счетное.

Из того факта, что R_2 – счетное, напрямую следует, что возможен какой-либо способ перечисления его элементов для установления взаимно-однозначного соответствия между элементами R_2 и элементами множества натуральных чисел. Это следует из самого определения мощности множества, согласно которому предполагается, что в равномощных множествах каждый элемент одного множества имеет парный элемент из другого множества и наоборот. Обратите внимание, фундаментальное отличие данного определения от определения эффективной перечислимости состоит в том, что в данном случае мы даже не говорим о наличии какого-либо алгоритма перечисления, мы просто утверждаем, что можно привести список действительных чисел из множества R_2 и список соответствующих им натуральных чисел из множества N. Алгоритм построения связи $N \leftrightarrow R_2$ нас в данном случае не интересует, достаточно того, что такое соответствие возможно.

Построим такой список чисел из множества R_2 и пронумеруем числа в разрядах:

$0.a_{11}a_{12}a_{13}\ldots$

$0.a_{21}a_{22}a_{23}\ldots$

……

$0.a_{n1}a_{n2}a_{n3}\ldots$

Теперь построим число $b=0.b_1b_2\ldots$, причём

$b_i=a_{ii}+1$, где + обозначает операцию сложения, результатом которого не могут быть числа 0 и 9, т.е. если $a_{ii}=1$, то $b_i=2$; если $a_{ii}=2$, то $b_i=3$, …., если $a_{ii}=8$, то $b_i=1$).

Таким образом, построенное число b будет отличаться от каждого из чисел множества R_2 хотя бы в одном разряде, и, следовательно, не попадёт в составленный список. Однако по своей структуре число b должно содержаться в множестве R_2. Получили противоречие, значит исходное предположение неверно и множество R_2 – несчётно.

Так как множество R_2 является по условию подмножеством множества R_1, то R_1 – несчетно, а т.к. R_1 несчетно – то значит и множество R несчётно, **Q.E.D.**

Примечание: можно и не выбрасывать числа, содержащие 0 и 9. Таким образом, в наш ряд некоторые числа войдут дважды. Это связано с тем, что конечные дроби могут быть превращены в бесконечные. Например ½=0,5=0,5(0)=0,4(9).

В общем случае это могло стать причиной того, что не удалось сосчитать множество действительных чисел. Но множество чисел, представимых двояким образом (конечные дроби), – это множество рациональных чисел. Как было доказано ранее, их счетное количество. Можно даже показать, что это множество эффективно перечислимо. Таким образом, даже двойное представление множества таких чисел образует счетное множество, следовательно, доказательство верно даже без такого упрощения.

Получен принципиально новый результат – найдено несчетное множество чисел. Его мощность, согласно доказанной теореме, не равна алеф-нуль ($\aleph_0$), а значит, необходимо новое число в трансфинитной шкале.

Алеф ($\aleph$) – второе трансфинитное число. По определению – это мощность континуума (всех действительных чисел). Это вторая по величине бесконечная мощность. Доказанная только что теорема 2.4.(1) о несчетности множества действительных чисел является убедительным доказательством того, что мощность этого множества больше, чем алеф-ноль (больше множества натуральных

чисел). И это весьма важный результат после череды доказательства счетности разнообразных множеств чисел.

Если оперировать понятием кардинального числа (мощности), то получим, что, так как каждое число сегмента (0,1) может быть представлено десятичной дробью вида $0.a_1a_2a_3\ldots$ не менее одного раза и не более двух, то:

$\aleph \le 10^{\aleph_0} \le 2\aleph$,

а так как $2\aleph=\aleph$, то получим что $10^{\aleph_0}=\aleph$. Те же рассуждения справедливы в случае, если мы будем разлагать числа не в десятичные, а, например, в двоичные дроби, дроби с основанием 3, 15, 10005 или даже $\aleph_0$ (если вы можете такое себе представить).

Таким образом, $\aleph=2^{\aleph_0}=3^{\aleph_0}=\ldots=10^{\aleph_0}=\ldots n^{\aleph_0}=\ldots\aleph_0{}^{\aleph_0}$

Если задуматься, можно обнаружить очередной не вполне очевидный факт из теории множеств. $\aleph^2=\aleph\cdot\aleph$ есть мощность множества пар действительных чисел. Пара действительных чисел, вообще говоря, соответствует точке на плоскости. В свою очередь, $\aleph^3=\aleph\cdot\aleph\cdot\aleph$ есть мощность множества троек действительных чисел, а это точки в пространстве. Рассуждения можно продолжить далее вплоть до $\aleph_0$-мерного пространства или множества всех последовательностей действительных чисел счетной длины. Следовательно, все конечно-мерные или счетно-мерные пространства имеют одинаковую мощность $\aleph$ (здесь $\aleph$ – количество точек в пространстве).

Для $\aleph_0$- мерного действительного пространства или множества всех последовательностей действительных чисел счетной длины с точки зрения операций над кардинальными числами получим $\aleph^{\aleph_0}=(2^{\aleph_0})^{\aleph_0}=2^{\aleph_0\cdot\aleph_0}=2^{\aleph_0}=\aleph$.

В этом месте интересно будет обратиться к историческим событиям, связанным с чередой доказательств в этой сфере. С тем, что на бесконечной прямой столько же точек, сколько и на отрезке, математики, хотя и не сразу, но в итоге примирились. Но следующий результат Кантора оказался еще более неожиданным. В поисках множества, имеющего больше элементов, чем отрезок на действительной оси, он обратил внимание на множество точек квадрата. Изначально сомнений в результате не было: ведь отрезок целиком размещается на одной стороне квадрата, а множество всех

отрезков, на которые можно разложить квадрат, само по себе имеет ту же мощность, что и множество точек отрезка. На протяжении почти трех лет (с 1871 по 1874) Кантор искал доказательство того, что взаимно однозначное соответствие между точками отрезка и точками квадрата невозможно. И в какой-то момент совершенно неожиданно получился прямо противоположный результат: ему удалось построить соответствие, которое он искренне считал невозможным. Кантор не верил сам себе и даже написал немецкому математику Рихарду Дедекинду: «Я вижу это, но не верю этому». Когда шок от этого факта прошел, стало интуитивно понятно и вскоре доказано, что и куб имеет столько же точек, сколько отрезок. Вообще говоря, любая геометрическая фигура на плоскости (геометрическое тело в пространстве), содержащая хотя бы одну линию, имеет столько же точек, сколько отрезок. Такие множества назвали множествами мощности континуума (от латинского continuum – непрерывный). Следующий шаг почти очевиден: размерность пространства в определенных пределах несущественна. Например, 2-мерная плоскость, 3-мерное привычное пространство, 4-х, 5-ти и далее n-мерные пространства с точки зрения количества точек, содержащихся в соответствующем n-мерном теле, равномощны. Такая ситуация будет наблюдаться даже в случае пространства с бесконечным количеством измерений, важно только чтобы это количество было счетным.

На данном этапе обнаружены два типа бесконечностей и соответственно два трансфинитных числа, обозначающих их мощности. Множества первого типа имеют мощность, эквивалентную мощности натуральных чисел (алеф-ноль). Множества второго типа имеют мощность, эквивалентную количеству точек на действительной оси (мощность континуума, алеф). Показано, что во множествах второго типа элементов больше, чем во множествах первого типа. Естественно, возникает вопрос – а нет ли в природе «промежуточного» множества, которое имело бы мощность больше, чем количество натуральных чисел, но при этом меньше, чем множество точек на прямой? Этот непростой вопрос получил название ***«проблема континуума»***. Она же

известна как ***«континуум-гипотеза»*** или «*первая проблема Гильберта*». Точная формулировка звучит следующим образом:

> ***Континуум-гипотеза:*** *с точностью до эквивалентности, существуют только два типа бесконечных числовых множеств: счетное множество и континуум.*

Гипотеза предполагает, что не существует множества *промежуточной мощности*, т. е. такого множества X, $\mathbb{N} \subset X \subset \mathbb{R}$, которое не эквивалентно ни $\mathbb{N}$, ни $\mathbb{R}$. Этой проблемой занимались очень многие математики. Сам Георг Кантор неоднократно заявлял, что доказал эту гипотезу, но всякий раз находил у себя ошибку.

Название «*первая проблема Гильберта*» относится к публикации Давидом Гильбертом на II Международном Конгрессе математиков в Париже в 1901 г. списка из 23 кардинальных проблем математики. Тогда эти проблемы не были решены, позднее окончательное решение получили 16 проблем из 23.

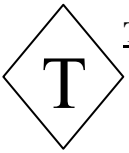
В результате после долгих исследований по вопросу континуум-гипотезы в 1938 году немецкий математик Курт Гедёль доказал, что существование промежуточной мощности не противоречит остальным аксиомам теории множеств. И позднее, в 1963 – 1964 годах почти одновременно, но независимо друг от друга, американский математик Коэн и чешский математик Вопенка показали, что наличие такой промежуточной мощности не выводимо из остальных аксиом теории множеств. Кстати, интересно заметить, что этот результат очень похож на историю с постулатом о параллельных прямых. Как известно, две тысячи лет его пытались вывести из остальных аксиом геометрии, но только после работ Лобачевского, Гильберта и других удалось получить тот же результат: этот постулат не противоречит остальным аксиомам, но и не может быть выведен из них.

2.4.2. Множества комплексных, трансцендентных и иррациональных чисел

Приведем в дополнение к множеству действительных чисел еще несколько несчетных множеств.

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

Множество комплексных чисел обозначим латинской буквой C.



Т.2.4.(2) Теорема

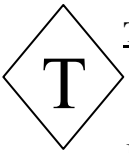
Множество комплексных чисел несчетно.

Доказательство

Так как множество действительных чисел R, несчётное по доказанной ранее Теореме 2.4.(1), является подмножеством множества комплексных чисел C, то множество комплексных чисел также несчётно, **Q.E.D.**

Иррациональным *называется действительное число, не являющееся рациональным.*

Множество иррациональных чисел обозначим латинской буквой I.



Т.2.4.(3) Теорема

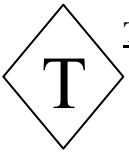
Множество иррациональных чисел несчетно.

Доказательство

Поскольку действительных чисел – несчетное множество, а рациональных – счетное, то иррациональных чисел – несчетное множество, **Q.E.D.**

Трансцендентное число *– действительное число, не являющееся алгебраическим.*

Множество трансцендентных чисел обозначим латинской буквой Т. Каждое трансцендентное действительное число является иррациональным, но обратное неверно. Например, число $\sqrt{2}$ иррациональное, но не трансцендентное: оно является корнем уравнения $x^2 - 2=0$.



Т.2.4. (3) Теорема

Множество трансцендентных чисел несчетно.

Доказательство

Поскольку действительных чисел – несчетное множество, а алгебраических – счетное, и при этом множество А является подмножеством R, то R \ A (множество трансцендентных чисел) представляет собой несчетное множество, **Q.E.D.**

Это несложное доказательство существования трансцендентных чисел опубликовано Кантором в 1873 году и произвело большое впечатление на научную общественность, так как доказывало существование множества чисел, не строя ни одного конкретного примера, а лишь исходя из общих соображений. Из этого доказательства нельзя извлечь ни одного конкретного примера трансцендентного числа, про доказательство такого типа говорят, что оно ***неконструктивно***.

Важно отметить, что долгое время математики имели дело лишь с алгебраическими числами. Потребовались усилия, чтобы найти хотя бы несколько трансцендентных чисел. Впервые это удалось французскому математику Лиувиллю в 1844 году, который доказал набор теорем, позволяющий строить конкретные примеры таких чисел. Например, трансцендентным числом является число 0,101001000001…, в котором после первой единицы стоит один нуль, после второй – два, после третьей – 6, после n-й соответственно n! нулей.

Было доказано, что трансцендентным является десятичный логарифм любого целого числа, кроме чисел 10^n . Также к множеству трансцендентных чисел относятся sin**α,** cos**α** и tg**α** для

любого ненулевого алгебраического числа **α**. Наиболее яркими представителями трансцендентных чисел обычно считают числа **π** и **e.** Кстати, доказательство трансцендентности числа **π**, проведенное немецким математиком Карлом Линдерманом в 1882 году, было большим научным событием, ведь из него следовала невозможность квадратуры круга. История нахождения квадратуры круга длилась четыре тысячелетия, а сам термин стал синонимом неразрешимых задач.

> ***Квадратура круга*** *— задача, заключающаяся в нахождении построения с помощью циркуля и линейки квадрата, равновеликого по площади данному кругу.*

Наряду с трисекцией угла (разбиение произвольного угла на три равные части) и удвоением куба (построение отрезка, являющегося ребром куба в два раза большего объёма, чем куб с данным ребром), эта задача является одной из самых известных неразрешимых задач на построения с помощью циркуля и линейки. В такого рода задачах на построение циркуль и линейка считаются идеальными инструментами, при этом, в частности линейка не имеет делений и имеет только одну сторону бесконечной длины, а циркуль может иметь сколь угодно большой раствор. Напомним, что отношение длины окружности к ее диаметру есть величина постоянная, не зависящая от радиуса круга, именно она обозначается буквой **π**. Таким образом, длина окружности круга радиуса r равна $2\pi r$, а площадь круга равна $S = 1/2\pi r^2$. Если принять за единицу измерения радиус круга и обозначить x длину стороны искомого квадрата, то задача сводится к решению уравнения: $x^2 = \pi$, откуда: $x = \sqrt{\pi}$. Как известно, с помощью циркуля и линейки можно выполнить все 4 арифметических действия и извлечение квадратного корня. Это означает, что квадратура круга возможна в том и только в том случае, если с помощью конечного числа таких действий можно построить отрезок длины π. Таким образом, неразрешимость этой задачи следует из неалгебраичности (трансцендентности) числа **π.** Собственно задача о квадратуре круга сводится к задаче построения треугольника с основанием πr и

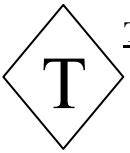
высотой r. Для него потом уже без труда может быть построен равновеликий квадрат.

В упоминаемом ранее списке из 23 кардинальных проблем математики под номером 7 шла проблема, касающаяся трансцендентности чисел, образованных определенным образом.

Седьмая проблема Гильберта. *Пусть a --- положительное алгебраическое число, не равное 1, b --- иррациональное алгебраическое число. Доказать, что* a^b *есть число трансцендентное.*

В 1934 году советский математик Гельфонд и чуть позже немецкий математик Шнайдер доказали справедливость этого утверждения, и таким образом, эта проблема была решена.

С принципом деления чисел на рациональные и иррациональные связаны еще два занимательных факта, не сразу воспринимаемые как истинные.



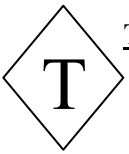
Т.2.4.(5) Теорема

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

Доказательство

Пусть есть два рациональных числа – a и b. Построим линейную, а стало быть, взаимно-однозначную, функцию $f(x) = (x - a) / (b - a)$. Так как $f(a) = 0$ и $f(b) = 1$, то $f(x)$ взаимно-однозначно отображает отрезок $[a; b]$ в отрезок [0; 1], при этом сохраняется рациональность чисел. Поэтому мощности множеств $[a; b]$ и [0; 1] действительных чисел равны, а, как доказано, мощность отрезка [0; 1] равна мощности континуума. Выбрав из полученного множества только иррациональные числа, мы получим, что между любыми двумя рациональными числами всегда найдется континуум иррациональных чисел, **Q.E.D.**

В целом данная теорема интуитивно кажется вполне логичной. Следующая, на первый взгляд, воспринимается скептически.



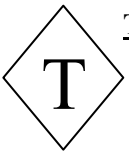
Т. 2.4.(6) Теорема

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

Доказательство

Пусть есть два иррациональных числа a и b, запишем их соответствующие разряды как $a_1a_2a_3...$ и $b_1b_2b_2...$, где a_i, b_i - десятичные цифры. Пускай $a < b$, тогда найдется такое N, что $a_N < b_N$. Построим новое число c, для чего положим $c_i = a_i = b_i$ для $i = 1,\ldots, N-1$. Пускай $c_N = b_N - 1$. Очевидно, что $c < b$. Поскольку все разряды числа a после N-го не могут быть девятками (тогда это будет периодическая дробь, т.е. рациональное число), то обозначим через M >= N такой разряд числа a, что $a_M < 9$. Положим $c_j = a_j$, при $N < j < M$, и $c_M = 9$. В таком случае $c > a$. Итак, мы получили одно рациональное число c, такое что $a < c < b$. Дописывая к десятичной записи числа c любое конечное число цифр сзади мы можем получить сколь угодно много рациональных чисел между a и b. Поставив в соответствие каждому такому числу его порядковый номер, получим взаимно – однозначное соответствие между множеством этих чисел и множеством натуральных чисел, поэтому полученное множество будет счетным, **Q.E.D.**

На этом этапе становится интересным и важным доказательство следующей теоремы, смысл которой до введения шкалы трансфинитных чисел был вообще-то очевидным, а при появлении такой специфической арифметики требует строгого доказательства.



Т.2.4. (7) Теорема Кантора

Для любого кардинального числа α справедливо $\alpha<2^{\alpha}$.

Доказательство

1. Докажем, что по крайней мере $\alpha \le 2^{\alpha}$

Как известно, мощность булеана множества M равна $2^{|M|}$. Пусть множество M = {m_1, m_2, m_3, …}. В булеан множества M (множество всех его подмножеств) в том числе входят множества, состоящие каждое из единственного элемента, например {m_1},{m_2},{m_3}, …. Только такого вида подмножеств будет |M|, а кроме них в булеан входят и другие подмножества, значит, в любом случае $|M| \le 2^{|M|}$

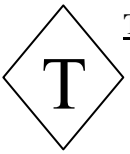
2. Докажем строгость неравенства $\alpha < 2^{\alpha}$

С учетом доказанного в п.1. достаточно показать, что недопустима ситуация, при которой $\alpha=2^{\alpha}$. Предположим противное, пусть $\alpha=2^{\alpha}$, т.е. $|M| = 2^{|M|}$. Это означает, что M равномощно P(M), значит существует отображение множества M на его булеан P(M). Таким образом, каждому элементу m множества M взаимно однозначно соответствует некоторое подмножество Mm, принадлежащее P(M). Значит любой элемент m или принадлежит соответствующему ему подмножеству Mm, или не принадлежит. Построим множество M*, образованное из всех элементов второго рода (т.е. тех m, которые не принадлежат соответствующим им подмножествам Mm).

По построению видно, что если какой-либо элемент m принадлежит M*, значит он автоматически не принадлежит Mm. Это, в свою очередь означает, что ни для какого m невозможна ситуация M*=Mm. Значит, множество M* отлично от всех множеств Mm и для него нет взаимно-однозначного элемента m из множества M. Это в свою очередь означает, что равенство $|M|= 2^{|M|}$ неверно. Следовательно, доказано, что $|M|< 2^{|M|}$ или $\alpha<2^{\alpha}$, **Q.E.D.**

В приложении к рассмотрению бесконечных множеств, это убедительно доказывает, что множество всех подмножеств

натуральных чисел (а это по сути множество комплексов бесконечной длины) НЕ равномощно множеству самих натуральных чисел. Т.е. $\aleph_0 \neq 2^{\aleph 0}$. И значит, по аналогии, можно построить еще более обширное множество, например на основе действительных чисел. Иными словами, вопрос относительно других типов бесконечных множеств заключается в следующем: а существует ли множество мощности большей, чем мощность множества действительных чисел? Если такой вопрос будет решен положительно, сразу же встанет следующий: а существует ли множество еще большей мощности? Потом еще больше. И, наконец, логичный глобальный вопрос: а существует ли множество самой большой мощности?



Т.2.4. (8) Теорема

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

Доказательство

Рассмотрим множество **B** всех функций, заданных на множестве **A** и принимающих значения 0 и 1. Каждой точке **a** множества **A** поставим в соответствие функцию $f^a(x)$, принимающую в этой точке значение 1, а в остальных точках значение 0. Ясно, что разным точкам соответствуют разные функции. Отсюда следует, что мощность множества **B** не меньше мощности множества **A** ($|\mathbf{B}| \geq |\mathbf{A}|$).

Предположим, что мощности множество **A** и **B** равны друг другу. В этом случае существует взаимно-однозначное соответствие между элементами множеств **A** и **B**. Обозначим функцию, соответствующую элементу **a** из множества **A**, через $f_a(x)$. Все функции семейства $f_a(x)$ принимают значение или 0 или 1.Построим новую функцию $\varphi(x)=1- f_x(x)$. Таким образом, чтобы найти значение функции $\varphi(x)$ в некоторой точке **a**, принадлежащей множеству **A**, надо сначала найти соответствующую функцию $f_a(\mathbf{a})$ и затем вычесть из единицы значение этой функции в точке **a**. Из построения видно, что функция $\varphi(x)$ также задана на множестве **A** и принимает значения 0 и 1. Следовательно, $\varphi(x)$ является

элементом множества **B**. Тогда существует такое число b в множестве A, такое что $\varphi(x) = f_b(x)$. С учетом ранее введенного определения функции $\varphi(x)=1- f_x(x)$, получим что для всех x, принадлежащих множеству **A**, верно $1 - f_x(x)= f_b(x)$. Пусть x = b. Тогда $1 - f_b(b) = f_b(b)$ и значит $f_b(b)=1/2$. Данный результат явным образом противоречит тому, что значения функции $f_b(x)$ равны нулю или единице. Следовательно, принятое предположение неверно, а значит не существует взаимно-однозначного соответствия между элементами множеств **A** и **B** ($|\mathbf{A}|\neq|\mathbf{B}|$). Поскольку $|\mathbf{A}|\neq|\mathbf{B}|$ и при этом $|\mathbf{B}|\geq|\mathbf{A}|$, значит $|\mathbf{B}|>|\mathbf{A}|$. Это означает, что для любого множества **A** можно построить множество **B** большей мощности. Отсюда можно сделать вывод, что множества самой большой мощности не существует, **Q.E.D.**

Существует довольно тесная связь между построенным множеством функций и булеаном множества **A** (множеством всех подмножеств **A**). Рассмотрим множество **B** всех подмножеств множества **A**. Пусть **C** – некоторое подмножество в **A**. Возьмем функцию *f(x)*, принимающую значении 1, если *x* принадлежит **C**, и значение 0 в противном случае. Таким образом, разным подмножествам **C** соответствуют различные функции. Наоборот, каждой функции *f(x)*, принимающей два значения 0 и 1, соответствует подмножество в **A**, состоящее из тех элементов *x*, в которых функция принимает значение 1. Таким образом, установлено взаимно-однозначное соответствие между множеством функций, заданных на множестве **A** и принимающих значения 0 и 1, и множеством всех подмножеств в **A**.

§ 2.5. Множества с мощностью, больше чем мощность континуума

Итак, множества самой большой мощности не существует. Первые два трансфинитных числа имели в природе образующие их множества (множество натуральных чисел и множество действительных чисел). Если отталкиваться от множества континуума, то можно построить множество всех подмножеств континуума, получим его булеан, назовем это множество B_R. По

определению мощность множества B_R равна $2^{\aleph}$. Согласно теореме Кантора $2^{\aleph} \neq \aleph$. Очевидно, что множество B_R бесконечно, следовательно, его кардинальное число является числом трансфинитным и оно никак не может совпадать ни с одним из двух рассмотренных ранее трансфинитных чисел. А значит, в нашу шкалу пора вводить третье трансфинитное число.

Алеф-один ($\aleph_1$) – третье трансфинитное число. По определению, это мощность множества всех подмножеств континуума. Это же число соответствует мощности многих других множеств, например:

- Множества всех линейных функций, принимающих любые действительные значения. По сути это множества всех возможных кривых в счетно-мерном пространстве, где количество измерений n – любое конечное число или даже $\aleph_0$.
- Множества фигур на плоскости, т.е. множества всех подмножеств точек на плоскости или множества всех подмножеств пар действительных чисел.
- Множества тел в обычном трехмерном пространстве, а также, вообще говоря, в любом счетно-мерном пространстве, где количество измерений n – любое конечное число или даже $\aleph_0$.

Поскольку число $\aleph_1$ вводится как мощность булеана множества с мощностью $\aleph$, получаем утверждение, что $\aleph_1 = 2^{\aleph}$.

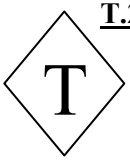
§ 2.6. Парадоксы теории множеств

Возникает резонный вопрос: а что дальше? Что будет, если построить множество всех подмножеств множества B_R. Чему будет равно его кардинальное число (конечно по аналогии можно предположить, что это $2^{\aleph 1}$) и, главное, какому реально существующему множеству это будет соответствовать? Есть ли вообще большие, чем B_R бесконечные множества и сколько их?

Хотя нами показано, что наибольшего трансфинитного числа не существуют, как показывают исследования, восходить всё далее и далее к новым большим кардинальным числам небезопасно – это приводит к антиномии (парадоксам). Действительно, каково бы ни было множество кардинальных чисел, всегда можно найти

кардинальное число, большее, чем все числа данного множества и, следовательно, не входящее в него. Следовательно, ни одно такое множество не содержит все кардинальные числа и множество всех кардинальных чисел немыслимо.

Вполне естественно, что каждому математику хочется иметь дело с непротиворечивой теорией, т.е. такой, что в ней нельзя одновременно доказать две теоремы, явно отрицающие друг друга. Является ли теория Кантора непротиворечивой? До каких пределов можно расширять круг рассматриваемых множеств? К сожалению, не все так безоблачно. Если ввести такое внешне безобидное понятие как «множество всех множеств U», то возникает ряд любопытных моментов.

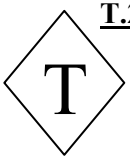


Т.2.6.(1) Парадокс Кантора

Кардинальное число множества всех подмножеств P(U) множества всех множеств U не больше чем |U|.

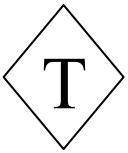
Доказательство

Так как U содержит все мыслимые и возможные множества, то оно по логике вещей, содержит в частности и множество всех своих подмножеств. Более того, все элементы множества P(U) принадлежат U, следовательно, $|P(U)| \leq |U|$. Однако существует доказанная ранее Теорема Кантора 2.4.(7), согласно которой для любого кардинального числа α справедливо $\alpha<2^{\alpha}$. Т.о., ввиду того, что P(U) - множество всех подмножеств U (булеан U), получим что $|P(U)| > |U|$. Два полученных вывода $|P(U)| \leq |U|$ и $|P(U)| > |U|$ прямо противоречат друг другу, что в принципе не должно быть возможно и является иллюстрацией парадокса, **Q.E.D.**



Т.2.6.(2) Парадокс Рассела

Пусть В – множество всех множеств, которые не содержат самих себя в качестве своих собственных элементов. Тогда можно доказать две теоремы.

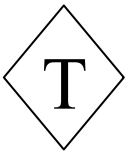


Теорема 2.6.(2).1.

В принадлежит В.

Доказательство

Предположим противное, т.е. **В** не принадлежит **В**. По определению, это означает, что **В** принадлежит **В**. Получили противоречие – следовательно, исходное предположение неверно и **В** принадлежит **В**, **Q.E.D.**



Теорема 2.6.(2).2.

В не принадлежит В.

Доказательство

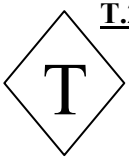
Предположим противное, т.е. **В** принадлежит **В**. По определению множества **В** любой его элемент не может иметь себя в качестве собственного элемента, следовательно, **В** не принадлежит **В**. Противоречие – следовательно, исходное предположение неверно и **В** не принадлежит **В**, **Q.E.D.**

Нетрудно видеть, что Теоремы 2.6.(2).1. и 2.6.(2).2. исключают друг друга.

К сожалению, даже исключение из рассмотрения всех суперобширных множеств не спасает теорию Кантора. По сути, парадокс Рассела затрагивает логику, т.е. способы рассуждения, с помощью которых при переходе от одного истинного утверждения к другому образуются новые понятия.

Уже при выводе парадокса используется логический закон исключенного третьего, являющийся одним из неотъемлемых приемов рассуждений в классической математике (т.е. если истинно утверждение не-А, то ложно А). Если задуматься о сути

вещей, то можно в целом уйти и от теории множеств, и от математики в целом.



Т.2.6.(3) Парадокс

Пусть P – некоторое свойство. Обладает ли само P этим свойством P?

Доказательство

Интересная постановка, не правда ли? Например, свойство быть сладким не применимо само к себе, потому что свойство быть сладким само по себе не сладкое. Зато свойство быть абстрактным, будучи абстрактным, разумеется, абстрактно, т.е. применимо само к себе. Дадим следующее определение:

Импредикабельным *называется свойство, которое не применимо само к себе.*

Теперь возникает риторический вопрос: свойство быть импредикабельным будет ли само по себе импредикабельно? Нетрудно показать две ветки рассуждений:

1. если это свойство импредикабельно, то значит не применимо само к себе, и, следовательно, оно не является импредикабельным.
2. если это свойство не импредикабельно само по себе, то значит оно применимо само к себе, и, следовательно, импредикабельно.

Итак, напрашивается неутешительный вывод: свойство быть импредикабельным импредикабельно тогда и только тогда, когда оно не импредикабельно. С виду нелепость, на самом деле серьезный и даже в некотором смысле плачевный вывод. Возникает невольное ощущение, что сами законы мышления по сути противоречивы.

Далее приведем итоги многолетних исследований в этой сфере. Причинами появления парадоксов считаются:

- Допущения теорией сверхобширных множеств типа «множества всех множеств».

• Сверхобширные множества допускаются в качестве элементов некоторых объектов.

• Наблюдается непредикативность определений, т.е. в парадоксе та сущность, о которой идет речь, определяется или характеризуется посредством некоторой совокупности, к которой она сама принадлежит.

В этой связи были сформулированы условия, которым должна удовлетворять теория множеств, свободная от парадоксов:

• Включение в теорию всех основных достижений канторовской теории множеств.

• Непротиворечивость.

Интересно, что был создан набор теорий множеств, для которых доказано первое условие, но ни для одной из них не доказано второе (впрочем показано, что известных парадоксов они не содержат, но это вовсе не означает, что нас не подстерегают парадоксы пока не известные). Если углубиться в первопричину противоречивости теории Кантора, надо с грустью признать, что как теория оперирования с бесконечностями она не полна в своем анализе понятия бесконечного. Пресловутое понятие «множество всех множеств» или «бесконечность бесконечностей» не поддается анализу Кантора. Следовательно, теория Кантора имеет определенную границу применимости и попытки применить ее за границами этой области автоматом ведут к парадоксам. Иными словами, с философской точки зрения, попытки выразить в понятиях конкретной теории явление, находящееся вне пределов области, где эти понятия применимы, вызывает появление парадоксов.

Итак, ни для одной из созданных теорий непротиворечивость не была доказана, хотя показано, что известных парадоксов они не содержат. В частности, в 1908 году немецкий математик Эрнст Цермело предложил формальную систему аксиом для теории множеств, которая охватывает все современные математические рассуждения и при этом считается свободной от парадоксов. Ранее, в 1889 году, такая система аксиом для обычной арифметики была предложена итальянским математиком Джузеппе Пеано. Пеано впервые сформулировал аксиомы арифметики, казавшиеся до

смешного очевидными (существует нуль; за каждым числом следует еще число и т.д.), но на самом деле абсолютно исчерпывающие. Они играли ту же роль, что и постулаты великого Евклида в геометрии. Исходя из подобных утверждений, с помощью логических рассуждений, можно было получить основные арифметические теоремы.

В тот же период немецкий математик Готлиб Фреге выдвинул еще более амбициозную задачу. Он предложил не просто аксиоматически утвердить основные свойства исследуемых объектов, но и формализовать, кодифицировать сами методы рассуждений, что позволяло записать любое математическое рассуждение по определенным правилам в виде цепочки символов. С именем и научными изысканиями Фреге связана, пожалуй, одна из самых драматических историй в развитии науки о числах. Когда второй том его трудов был уже в печати, ученый получил письмо от молодого английского математика Бертрана Рассела. Поздравив коллегу с выдающимися результатами, Рассел, тем не менее, указал на одно обстоятельство, прошедшее мимо внимания автора. Коварным «обстоятельством» был получивший впоследствии широкую известность «парадокс Рассела», рассмотренный выше под названием Теорема 2.6.(2) и по сути представлявший собой вопрос: будет ли множество всех множеств, не являющихся своими элементами, своим элементом? Фреге не смог немедленно разрешить загадку, очень расстроился, взял академический отпуск в своем университете, потратил массу сил, пытаясь подправить свою теорию, но все было тщетно. Он прожил еще более двадцати лет, но не написал больше ни одной работы по арифметике.

Однако Расселу удалось вывести вариант формальной системы, позволяющий охватить всю математику и свободный от всех известных к тому времени парадоксов, с опорой именно на идеи и работы Фреге. Полученный им результат, опубликованный в 1902 году в книге Principia Mathematica, фактически стал аксиоматизацией логики, а Гильберт считал, что его «можно рассматривать как венец всех усилий по аксиоматизации науки».

Была и еще одна причина столь пристального интереса математиков к основаниям своей дисциплины. Дело в том, что на рубеже XIX и XX столетий в теории множеств были обнаружены

противоречия, для обозначения которых был придуман эвфемизм «парадоксы теории множеств». Наиболее известный из них — знаменитый парадокс Рассела — был, увы, не единственным. Более того, для большинства ученых было очевидно, что за открытием новых странностей дело не станет. Их появление оказало на математический мир, по выражению Гильберта, «катастрофическое воздействие», поскольку теория множеств играла роль фундамента, на котором возводилось все здание науки о числах. «Перед лицом этих парадоксов надо признать, что положение, в котором мы пребываем сейчас, на длительное время невыносимо. Подумайте, в математике, этом образце надежности и истинности, понятия и умозаключения, как их всякий изучает, преподает и применяет, приводят к нелепостям. Где же тогда искать надежность и истинность, если даже само математическое мышление дает осечку?» — сокрушался Гильберт в своем докладе на съезде математиков в июне 1925 года.

Таким образом, впервые за три тысячелетия, математики вплотную подошли к изучению самых глубинных оснований своей дисциплины. Сложилась любопытная картина: любители цифр научились четко объяснять, по каким правилам они ведут свои вычисления, им оставалось лишь доказать «законность» принятых ими оснований с тем, чтобы исключить любые сомнения, порождаемые злополучными парадоксами.

В первой половине 20-х годах Гильберт, вокруг которого сложилась к тому времени школа блестящих последователей, в целой серии работ наметил план исследований в области оснований математики, получивший впоследствии название «Геттингенской программы». В максимально упрощенном виде ее можно изложить следующим образом: математику можно представить в виде набора следствий, выводимых из некоторой системы аксиом, и доказать, что:

- Математика является полной, т.е. любое математическое утверждение можно доказать или опровергнуть, основываясь на правилах самой дисциплины.
- Математика является непротиворечивой, т.е. нельзя доказать и одновременно опровергнуть какое-либо утверждение, не нарушая принятых правил рассуждения.

• Математика является разрешимой, т.е., пользуясь правилами, можно выяснить относительно любого математического утверждения, доказуемо оно или опровержимо.

Фактически программа Гильберта стремилась выработать некую общую процедуру для ответа на все математические вопросы или хотя бы доказать существование таковой. Сам ученый был уверен в утвердительном ответе на все три сформулированные им вопроса: по его мнению, математика действительно была полной, непротиворечивой и разрешимой. Оставалось только это доказать. Ранее им же была сформулирована так называемая «вторая проблема Гильберта». Она сводилась к необходимости строго доказать, что система аксиом — базовых утверждений, принимаемых в математике за основу без доказательств, — совершенна и полна, то есть позволяет математически описать всё сущее. Надо было доказать, что можно задать такую систему аксиом, что они будут, во-первых, взаимно непротиворечивы, а во-вторых, из них можно вывести заключение относительно истинности или ложности любого утверждения.

Однако в 1931 году венский математик Курт Гёдель разрушил все надежды. Он опубликовал короткую статью, попросту опрокинувшую весь мир так называемой «математической логики». После долгих и сложных математико-теоретических преамбул он установил буквально следующее удивительное свойство любой системы аксиом: *«Если можно доказать утверждение А, то можно доказать и утверждение не-А».* То есть, возвращаясь к формулировке второй задачи Гильберта, если система аксиом полна (то есть любое утверждение в ней может быть доказано), то она противоречива.

Единственным выходом из такой ситуации остается принятие неполной системы аксиом. То есть, приходиться мириться с тем, что в контексте любой логической системы у нас останутся утверждения «типа А», которые являются заведомо истинными или ложными, — и мы можем судить об их истинности лишь вне рамок принятой аксиоматики. Если же таких утверждений не имеется, значит, наша аксиоматика противоречива, и в ее рамках неизбежно будут присутствовать формулировки, которые можно

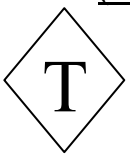
одновременно и доказать, и опровергнуть. Итак, краткая формулировка ***первой***, или слабой теоремы Гёделя о неполноте: *«Любая формальная система аксиом содержит неразрешенные предположения»*. Её можно сформулировать более строго.

Т.2.6.(4) Слабая теорема Геделя о неполноте (без доказательства)

Пусть T — корректная формальная система. Тогда множество утверждений, которые T может доказать, и множество истинных утверждений не совпадают. Но т.к. все доказуемые с помощью T утверждения истинны, отсюда следует, что есть истинные утверждения, недоказуемые в T.

Но на этом Гёдель не остановился, сформулировав и доказав вторую, или сильную теорему Гёделя о неполноте: *«Логическая полнота (или неполнота) любой системы аксиом не может быть доказана в рамках этой системы. Для ее доказательства или опровержения требуются дополнительные аксиомы (усиление системы)»*. Её можно сформулировать иначе.

Т.2.6.(5) Сильная теорема Геделя о неполноте (без доказательства)



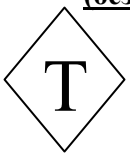
Пусть T — корректная формальная система. Тогда можно построить конкретное утверждение G, обладающее следующим свойством: G истинно, но недоказуемо в T.

*Формальная система аксиом называется **консистентной**, если она не может доказать одновременно какое-то утверждение и его отрицание, т.е. доказать противоречие.*

Неконсистентная формальная система — это плохо и практически бесполезно, т.к. можно легко показать, что из доказательства противоречия можно получить доказательство **чего**

угодно. Корректность отличается от консистентности. Если система корректна, то она автоматически консистентна: ведь она доказывает только истинные утверждения, а какое-то утверждение и его отрицание не могут одновременно быть истинными: одно из них будет истинным, а другое ложным. Заметим, однако (это важно), что "консистентность", как и "доказуемость" есть свойство синтаксическое, не зависящее от ***смысла*** формул и их интерпретации. При этом ***корректность*** системы есть свойство семантическое, требующее понятия "истинности". На основании определения консистентности построена 3-я теорема Геделя.

Т.2.6.(6) Третья теорема Геделя о неполноте (без доказательства)



Пусть T — консистентная формальная система. Тогда T не является полной системой, т.е. существует утверждение G такое, что T не может его ни доказать, ни опровергнуть; более того, мы можем построить такое конкретное G (называемое "гёделевым утверждением").

Неполнота системы **T** утверждается в качестве результата только в третьей версии, но легко видеть, что она сразу следует из заключения и в первых двух теоремах. Уже в них видно, что существует какое-то истинное, но недоказуемое утверждение. Такое утверждение **T** не доказывает, но и опровергнуть его — доказать его отрицание — система не может, т.к. его отрицание ложно, а **T** (в первых двух вариантах теоремы) **корректна** и доказывает только истинные утверждения. Поэтому **T** не может ни доказать, ни опровергнуть такое утверждение **G** и, следовательно, **T** неполна.

Спокойнее было бы думать, что теоремы Гёделя носят отвлеченный характер и касаются лишь областей возвышенной математической логики, однако фактически оказалось, что они напрямую связаны с устройством человеческого мозга. Английский математик и физик Роджер Пенроуз показал, что теоремы Гёделя можно использовать для доказательства наличия принципиальных различий между человеческим мозгом и

компьютером. Смысл его рассуждения прост. Компьютер действует строго логически и не способен определить, истинно или ложно утверждение А, если оно выходит за рамки аксиоматики, а такие утверждения, согласно теореме Гёделя, неизбежно имеются. Человек же, столкнувшись с таким логически недоказуемым и неопровержимым утверждением А, всегда способен определить его истинность или ложность — исходя из повседневного опыта. По крайней мере, в этом человеческий мозг превосходит компьютер, скованный чистыми логическими схемами. Человеческий мозг способен понять всю глубину истины, заключенной в теоремах Гёделя, а компьютерный - никогда.

Можно рассмотреть суть доказательства более подробно. Пенроуз утверждает, что предположение о существовании компьютерной программы, воспроизводящей функции человеческого интеллекта, в частности, воспроизводящей функции, составляющие математические способности человека, ведет к противоречию.

Предположим, что математические способности некоторого математика, назовем его Человек_Разумный, полностью описываются некоторой формальной системой **F**. Это означает, что любое математическое утверждение, которое Человек_Разумный признает "неоспоримо верным", является теоремой, доказываемой в **F**, и наоборот. Предположим, также, что Человек_Разумный знает, что **F** описывает его математические способности. Человек_Разумный также полагает, что тот факт, что **F** описывает его математические способности, эквивалентен вере в непротиворечивость и непогрешимость **F**. В противном случае мы должны были бы поставить под сомнение истины, которые представляются нам "неоспоримо истинными". Согласно теореме Геделя о неполное формальных систем, поскольку **F** непротиворечива, существует геделевское предложение G(**F**), которое должно быть истинным, но которое не является теоремой в системе **F**. Однако, поскольку Человек_Разумный верит, что **F** - непротиворечивая система и знает, что **F** представляет его способность к математическим рассуждениям, он должен прийти к выводу, что G(**F**) является "неоспоримой истиной". Таким образом, мы получаем математическое утверждение G(**F**), которое

Человек_Разумный признает истинным, но которое не является теоремой в **F** , что противоречит первоначальному предположению, что **F** представляет целиком и полностью математические способности Человека_Разумного.

Отсюда вывод, что никакая формальная система не может быть адекватным выражением математических способностей человека и, следовательно, невозможна полная компьютерная имитация человеческого сознания. Попыткой создать твердую эмпирическую почву для решения этого вопроса и стал тест, разработанный Аланом Тьюрингом.

Первый вариант теста, опубликованный в 1950 году, был несколько запутанным. Современная версия теста Тьюринга представляет собой следующее задание. Группа экспертов общается с неизвестным существом. Они не видят своего собеседника и могут общаться с ним только через какую-то изолирующую систему - например, клавиатуру. Им разрешается задавать собеседнику любые вопросы, вести разговор на любые темы. Если в конце эксперимента они не смогут сказать, общались ли они с человеком или с машиной, и если на самом деле они разговаривали с машиной, можно считать, что эта машина прошла тест Тьюринга.

Нет нужды говорить, что сегодня ни одна машина не может даже близко подойти к тому, чтобы пройти тест Тьюринга, хотя некоторые из них весьма неплохо работают в очень ограниченной области. Предположим, тем не менее, что в один прекрасный день машина все-таки сможет пройти этот тест. Будет ли это означать, что она разумна и обладает интеллектом?

Джон Р. Сирл, преподаватель философии Калифорнийского университета в Беркли, разработал воображаемую систему, которая показывает, что ответ на этот вопрос отрицательный. Эта система под названием «Китайская комната» работает следующим образом: вы сидите в комнате, в стене этой комнаты есть две щели. Через первую щель вам передают вопросы, написанные по-китайски. (Предполагается, что Вы, как и Джон Сирл, не знаете китайского. Если это не так, выберите какой-нибудь другой язык, неизвестный Вам). Затем вы просматриваете книги с инструкциями типа: «Если вы получили такой-то набор символов, напишите на листке бумаги

такой-то (отличный от исходного) набор символов и передайте его обратно через другую щель».

Ясно, что если книги с инструкциями достаточно полны, «машина», состоящая из Вас и комнаты, сможет пройти тест Тьюринга. При этом очевидно, что Вам совсем не обязательно понимать, что Вы делаете. По мнению Сирла, это показывает, что даже если машина прошла тест Тьюринга, это еще не значит, что она разумна и обладает интеллектом.

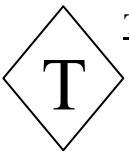
§ 2.7. Вычислимые числа

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

Например, все рациональные числа вычислимы, так как существует алгоритм их вычисления до любого знака, то есть с любой степенью точности.

Алгебраические числа являются вычислимыми, так как существуют численные методы их вычисления (алгоритм Ньютона), позволяющие их вычислить с любой степенью точности.

Важно отметить, что все рациональные числа суть алгебраические, т.к. они могут быть представлены как корни уравнения $a_0+a_1x=0$, где a_0 , a_1 – целые числа.



Т.2.7. (1) Теорема

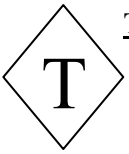
Множество вычислимых действительных чисел счетно.

Доказательство

Вычислимые числа включают в себя все алгебраические и некоторые трансцендентные числа.

Так как каждому вычислимому действительному числу соответствует хотя бы одна машина Тьюринга, а машин Тьюринга - $\aleph_0$, то значит вычислимых чисел никак не больше, чем $\aleph_0$. С другой стороны, поскольку все алгебраические числа вычислимы, а их тоже $\aleph_0$, то вычислимых действительных чисел никак не

меньше, чем $\aleph_0$. Значит, мощность множества вычислимых действительных чисел равна $\aleph_0$ и, следовательно, это множество счетно, **Q.E.D.**



Т.2.7. (2). Теорема

Существуют невычислимые действительные числа и их несчетное множество.

Доказательство

Существование невычислимых чисел следует хотя бы из того факта, что всего действительных чисел несчетное множество, а вычислимых – счетное. Значит, должно существовать несчетное множество невычислимых действительных чисел, **Q.E.D.**

Пример невычислимого действительного числа:

Пусть имеются $T_i,\ldots\ldots,T_n,\ldots$, где i – i-ая машина Тьюринга. Действительное число X можно представить так:

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

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

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

Задачи к главе 2

Установление взаимно-однозначных соответствий

Задача 2.1.

Установить взаимно-однозначное соответствие между промежутком $0<x<1$ и всей числовой прямой.

Задача 2.2.
Установить взаимно-однозначное соответствие между промежутком $a \le x \le b$ и $c \le x \le d$.

Задача 2.3.
Установить взаимно-однозначное соответствие между промежутком $0 \le x < 1$ и $0 \le x < \infty$.

Задача 2.4.
Установить взаимно-однозначное соответствие между промежутком $-1 < x < 1$ и $0 \le x < \infty$.

Задача 2.5.
Установить взаимно-однозначное соответствие между промежутком $-1 < x \le 1$ и $0 < x < \infty$.

Задача 2.6.
Установить взаимно-однозначное соответствие между промежутком $0 \le x \le 1$ и множеством иррациональных чисел того же отрезка.

Задача 2.7.
Установить взаимно-однозначное соответствие между всеми точками плоскости и точками сферы радиуса 1.

Задача 2.8.
Установить взаимно-однозначное соответствие между точками открытого квадрата $0 < x < 1$, $0 < y < 1$ и точками плоскости.

Задача 2.9.
Установить взаимно-однозначное соответствие между множеством всех рациональных чисел отрезка $0 \le x \le 1$ и множеством всех точек плоскости, обе координаты которых рациональны.

Задача 2.10.
Установить взаимно-однозначное соответствие между точками окружности радиуса a и b, $a \ge b$.

Счетность множеств

Задача 2.11.
Является ли счетным множество всех конечных подмножеств счетного множества?

Задача 2.12.
Является ли счетным любое множество попарно непересекающихся букв Т на плоскости?

Задача 2.13.
Является ли счетным любое множество попарно непересекающихся букв Г на плоскости?

Задача 2.14.
Является ли счетным любое множество попарно непересекающихся открытых интервалов на действительной прямой?

Задача 2.15.
Является ли счетным множество точек разрыва монотонной функции на действительной прямой?

Задача 2.16.
Можно ли построить на плоскости несчетное множество попарно непересекающихся окружностей?

Задача 2.17.
Можно ли построить на плоскости несчетное множество попарно непересекающихся букв N?

Задача 2.18.
Можно ли построить на плоскости несчетное множество попарно непересекающихся букв Б?

Мощность множеств

Задача 2.19.
Найти мощность множества всех четырехугольников на плоскости, координаты всех вершин которых рациональны.

Задача 2.20.
Найти мощность множества всех многоугольников на плоскости, координаты всех вершин которых рациональны.

Задача 2.21.
Найти мощность множества всех многочленов с действительными коэффициентами.

Задача 2.22.
Найти мощность множества всех действительных чисел, в десятичном разложении которых встречается цифра 3.

Задача 2.23.
Найти мощность множества всех действительных чисел, в десятичном разложении которых не встречается цифра 6.

Задача 2.24.
Найти мощность множества всех действительных чисел $0<x<1$, в десятичном разложении которых после запятой стоит цифра 4 и больше эта цифра не встречается.

Задача 2.25.
Найти мощность множества всех непрерывных функций на действительной прямой.

Задача 2.26.
Найти мощность множества всех функций, определенных на сегменте $a\le x\le b$ при $a<b$ и разрывных хотя бы в одной точке.

Глава 3. Арифметические вычисления

§ 3.1. Арифметические функции

Расширенное множество натуральных чисел, помимо обычного множества натуральных чисел, включает также число ноль (ранее это множество обозначалось N*). Далее в этой главе и в последующих главах слово «расширенное» для краткости будет опускаться, и под терминами «N», «натуральный ряд», «натуральное число», «множество натуральных чисел» следует понимать «N*», «расширенный натуральный ряд», «натуральное число или 0», «расширенное множество натуральных чисел» соответственно.

Арифметическая функция *– функция, определенная на множестве натуральных чисел и принимающая значения из множества натуральных чисел.*

Т.3.1.(1) Теорема

Множество арифметических функций n-переменных несчетно.

Доказательство

Для доказательства несчетности множества достаточно доказать несчетность какого-нибудь его подмножества. Рассмотрим арифметические функции одной переменной вида $f_i(x)$. Эти арифметические функции одной переменной образуют подмножество множества арифметических функций *n* переменных.

Предположим противное. Пусть арифметических функций одной переменной счетное множество, т.е. их можно перечислить. Тогда их можно расположить в виде бесконечной последовательности $f_0(x), f_1(x), f_2(x), \ldots, f_n(x), \ldots$

Построим новую функцию $g(x)=f_x(x)+1$.

Это так называемая диагональная функция, например:

$g(0)= f_0(0)+1,\ g(1)= f_1(1)+1,\ g(2)= f_2(2)+1, \ldots$

$g(x)$ отлична от всех перечисленных функций, т.к. от каждой из функций она отличается хотя бы в одной точке. Например, от функции $f_0(x)$ функция $g(x)$ отличается в точке x=0, от функции $f_1(x)$ функция $g(x)$ отличается в точке x=1 и т.д. Однако по построению $g(x)$ принадлежит множеству арифметических функций одной переменной, значит должна быть в списке перечисления $f_i(x)$, т.е. совпадать с одной из перечисленных функций. Получили противоречие, следовательно, исходное предположение неверно, и функций одной переменной несчетное множество.

Поскольку множество арифметических функций одной переменной является подмножеством множества арифметических функций n переменных, то значит множество арифметических функций n переменных ***несчетно***, **Q.E.D.**

> *Арифметическая функция называется **вычислимой**, если существует алгоритм для ее вычисления в каждой точке.*

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

Т. 3.1.(2) Теорема

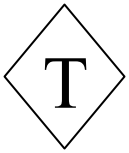
Множество вычислимых арифметических функций счетно.

Доказательство

Так как каждой вычислимой арифметической функции соответствует хотя бы одна машина Тьюринга, а машин Тьюринга $\aleph_0$, то значит вычислимых арифметических функций никак не больше, чем $\aleph_0$. Получим: $|\text{ВАФ}| \leq \aleph_0$.

С другой стороны, подмножеством множества вычислимых арифметических функций являются, например, функции вида $f(x)=n$, где n – натуральное число. Поскольку натуральных чисел $\aleph_0$, то вычислимых арифметических функций никак не меньше чем $\aleph_0$. Получим: $|\text{ВАФ}| \geq \aleph_0$. Значит, мощность множества вычислимых арифметических функций равна $\aleph_0$, а значит оно ***счетно***, **Q.E.D.**

Т. 3.1.(3) Теорема Тьюринга



Множество вычислимых арифметических функций *n* переменных не поддается эффективному перечислению.

Доказательство

Предположим противное. Пусть множество вычислимых арифметических функций *n* переменных эффективно перечислимо. Тогда существует алгоритм, по которому его можно перечислить. Применим этот алгоритм. Получим последовательность:

$f_0(x_1,\ldots,x_n), f_1(x_1,\ldots,x_n),\ldots, f_n(x_1,\ldots,x_n),\ldots$

Построим диагональную функцию:

$$g(x_1,\ldots,x_n)=\begin{cases} f_{x1}(x_1,\ldots,x_n)+1, \text{ при } x_1=\ldots=x_n \\ 0, \text{ в противном случае} \end{cases}$$

Пример (пусть n=3)

$g(0,0,0)=f_0(0,0,0)+1$

$g(0,0,1)=0$

$g(0,1,0)=0$

$g(1,1,1)=f_1(1,1,1)+1$

$g(1,1,2)=0$

По построению видно, что функция $g(x_1,\ldots,x_n)$ – арифметическая. Докажем, что, кроме этого, она является вычислимой. Для этого должен существовать алгоритм её вычисления. Укажем алгоритм вычисления $g(x_1,\ldots,x_n)$. Для любых значений $x_1,\ldots,x_n$ мы можем сначала провести операцию сравнения.

1) Если $x_1=\ldots=x_n$, запускаем алгоритм перечисления вычислимых арифметических функций $f_i(x_1,\ldots,x_n)$. Этот алгоритм существует в силу нашего предположения. Находим функцию с номером x_1, т.е. $f_{x1}(x_1,\ldots,x_n)$. Далее применим к ней алгоритм вычисления в точке $(x_1,\ldots,x_1)$, т.е. вычислим $f_{x1}(x_1,\ldots,x_1)$. Такой алгоритм существует в силу вычислимости функций вида $f_i(x_1,\ldots,x_n)$. Прибавление к результату вычисления единички есть тривиальная арифметическая операция, т.о. при одинаковых значениях аргументов $g(x_1,\ldots,x_n) = f_{x1}(x_1,\ldots,x_1) +1$ вычислима.

2) Если условие $x_1=...=x_n$ не выполняется, т.е. не все значения аргументов равны, то значение $g(x_1,...,x_n)$ приравнивается нулю, т.о. при различных значениях аргументов $g(x_1,...,x_n)=0$ тоже вычислима.

Отсюда видно, что диагональная функция $g(x_1,...,x_n)$ принадлежит множеству вычислимых арифметических функций n переменных.

Раз построенная функция принадлежит к множеству вычислимых арифметических функций, то она должна быть среди ранее эффективно перечисленных функций, но по построению она не может быть среди них, так как от каждой функции она отличается хотя бы в одной точке. Получили противоречие, следовательно, исходное предположение неверно и вычислимые арифметические функции n переменных нельзя эффективно перечислить, **Q.E.D.**

Т. 3.1.(4) Теорема

Множество невычислимых арифметических функций несчетно.

Доказательство

Ранее доказаны два утверждения:

- АФ (арифметических функций) несчетное множество.
- ВАФ (вычислимых арифметических функций) счетное множество.

Но при этом ВАФ есть подмножество АФ, а значит дополнение ВАФ до АФ (т.е. множество невычислимых функций) является несчетным. Значит, множество невычислимых арифметических функций несчетно, **Q.E.D.**

Следовательно, доказано, что невычислимые арифметические функции существуют. Более того, их очень много – несчетное множество. Приведем пример *невычислимой функции* одной переменной.

Пусть T_0 ,T_1 ,T_2 … ,T_x - машины Тьюринга. Определим функцию $f_1(x)$ следующим образом:

$$f_1(x) = \begin{cases} 1, \text{ если } T_x \text{ остановится на чистой ленте;} \\ 0, \text{ в противном случае.} \end{cases}$$

Эта функция везде определена, но алгоритма для решения проблемы остановки произвольной машины Тьюринга на чистой ленте не существует, значит, не существует алгоритма вычисления $f_1(x)$.

По большому счету, при построении такого рода функций (невычислимых) необходимо соблюдать два основных правила:

- Следить за тем, чтобы значение функции в каждой точке было определено и являлось числом натуральным (значит, функция принадлежит множеству арифметических функций);
- В описание функции добавить условие выбора одного из нескольких значений при заданном аргументе, причем само условие выбора должно гарантированно являться алгоритмически неразрешимой проблемой (как минимум для одной точки области определенности, а в общем случае для всех).

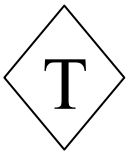
Тогда полученная функция будет являться невычислимой арифметической функцией.

Важно понимать, что могут существовать различные функции, которые не являются вычислимыми ни в одной точке. Например, приведенная ниже функция также *невычислима* ни в одной точке:

$$f_2(x) = \begin{cases} 8, \text{ если } T_x \text{ напечатает бесконечное число нулей на ленте;} \\ 3, \text{ в противном случае.} \end{cases}$$

С фактической точки зрения ничего не изменится: по-прежнему ни в одной точке функция $f_2(x)$ не может быть вычислена, точно так же, как ни в одной точке не может быть вычислена функция $f_1(x)$. Но, тем не менее, $f_1(x)$ и $f_2(x)$ - это совсем разные функции, так как они имеют различные *описания*.

Т.о. с точки зрения формально-описательной, даже если рассматривать нигде не вычислимые функции, все приведенные в соответствии с изложенными выше рекомендациями примеры будут определять (задавать) разные функции. Этих функций бесконечно много, однако вопрос о том, счетно ли множество функций, описываемых таким образом, не вполне очевиден.



Т. 3.1.(5) Теорема

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

Доказательство

Среди функций, описываемых конечным числом слов, содержатся ВСЕ вычислимые функции (алгоритм их вычисления и есть описание) и еще какие-то иные, типа приведенных выше примеров $f_1(x)$ и $f_2(x)$. Однако на примере нумерации Гёделя, рассмотренной в ходе доказательства Теоремы 1.7.(2), видим, что множество функций, описываемых конечным числом слов (понятий / терминов / идей и т.д.), тоже может быть сопоставлено с рядом натуральных чисел, т.е. является счетным и даже эффективно перечислимым множеством, **Q.E.D.**

Исходя из вышесказанного, получим, что существует несчетное множество арифметических функций, которые не могут быть даже описаны конечным числом слов (разумеется, об их вычислимости не может идти и речи). Примеры таких функций нельзя привести по определению, в силу того что любой завершенный по своему описанию пример является уже законченным описанием функции, а значит, сама функция попадает в множество функций, описываемых конечным числом слов.

§ 3.2. Частичные арифметические функции

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

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

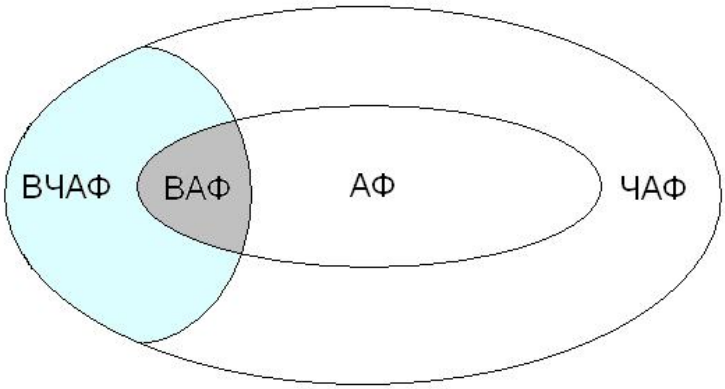


Рис. 3.1. Диаграмма вхождения множеств друг в друга

Примеры:

f(n) = n – 1.
Для этой функции область определенности: M= [1, ∞), область значений: N. Т.о. функция строит соответствие *{1, 2, ...}* $\rightarrow$ *N.*

f(n) = 1 – n.
Для этой функции область определенности: M= [0,1], область значений: [0,1]. Т.о. функция строит соответствие *{0, 1}* $\rightarrow$ *{0, 1}.*

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

1) *всюду определенные функции.* M = N. Множество всюду определенных частичных арифметических функций совпадает с множеством арифметических функций. Все остальные частичные арифметические функции имеют точки неопределенности;

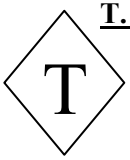
2) *нигде неопределенные функции.* $M = \varnothing$. Например, $f(n) = 0 - (n + 1)$. Нигде не определенные функции также являются подмножеством множества частичных арифметических функций.

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

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

Например:

- характеристическая функция $\chi_{\varnothing}=0$ (для пустого множества характеристическая функция всюду равна 0);
- характеристическая функция $\chi_N=1$ (для множества натуральных чисел характеристическая функция всюду равна 1);
- характеристическая функция $\chi_A=\mathrm{unsg}(|x-a_1|\cdot|x-a_2|\cdot\ldots\cdot|x-a_n|)$ для множества $A=\{a_1,a_2,..,a_n\} : a_1<a_2<,..,<a_n$.



Т. 3.2.(1) Теорема

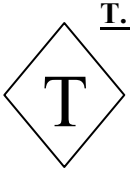
Множество частичных арифметических функций несчетно.

Доказательство

Поскольку множество АФ есть подмножество ЧАФ, и множество АФ несчетно, то и множество ЧАФ также несчетно, **Q.E.D.**

Вычислимая частичная арифметическая функция *(ВЧАФ) – это функция, для которой существует алгоритм вычисления ее значения в любой точке области определенности.*

Т. 3.2.(2) Теорема



Множество вычислимых частичных арифметических функций счетно и эффективно перечислимо.

Доказательство

Рассмотрим произвольную функцию *f(x)*, принадлежащую множеству ВЧАФ. Раз функция вычислима – значит, ее можно вычислить на машине Тьюринга Т. Пусть на входной ленте будет записан параметр функции *x* в унарном коде: | | | … | (x+1 палочка). Результат вычисления – значение *f(x)* также выдается в унарном коде: | | | **…** |. Если для данного *x* такой результат получен, значит, в точке *x* функция определена. Однако если произошли следующие события:

- машина испортила исходный параметр,
- машина не остановилась,
- машина напечатала нечитаемый результат,

то будем считать, что в данной точке *x* функция *f(x)* не определена.

Если именно так интерпретировать «неудобное» для нас поведение конкретной машины Тьюринга, то тогда можно утверждать, что любая машина Тьюринга вычисляет какую-нибудь ВЧАФ. Даже если машина при любом входном значении аргумента работает по совершенно непохожему на вычисление алгоритму, например стирает любое слово на ленте, то мы просто считаем, что функция, которую вычисляет машина, вообще нигде не определена.

С другой стороны, для вычисления одной и той же функции может существовать несколько алгоритмов. Т.о. любой ВЧАФ может соответствовать несколько машин Тьюринга.

Рассмотрим множество машин Тьюринга. Они эффективно перечислимы. Перечислим их, попутно т.о. перечисляя все

ВЧАФы, правда, возможно, некоторые ВЧАФ мы укажем несколько раз. Это довольно тонкий момент. Формально повторное указание одной и той же ВЧАФ в списке перечисления всех существующих вычислимых частичных арифметических функций несколько раз есть явной нарушение идеологии эффективной перечислимости. Строго говоря, по определению, термин «эффективно перечислимо» означает «возможность перечисления по алгоритму без пропусков и повторений». В данном случае в первом приближении наблюдается нарушение того определения. Однако если посмотреть на проблему более пристально, то можно исправить указанную «погрешность» доказательства следующим образом.

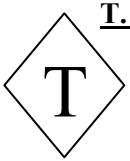
По сути, вычислимая частичная арифметическая функция задается не более и не менее, чем ***алгоритмом*** её вычисления в любой точке области определенности. Этот алгоритм может быть задан любым удобным образом: в виде аналитической формы (формулы), в виде набора инструкций машины Тьюринга, в виде алгоритма Маркова, в виде эффективного описания с помощью заранее согласованного языка и т.д. В этой связи, например, функция $f(x)=x+1$ может быть задана различными способами, и даже если сосредоточиться только на описании алгоритма в терминах набора инструкций (программы) машины Тьюринга, таких программ тоже можно составить несколько (точнее счетно-бесконечное множество). Подтверждением этого является хотя бы тот факт, что добавление набора «холостых» инструкций (типа «напечатать число n в унарном коде и затем его стереть) к работающей программе не изменит её смысла, если таковым смыслом считать полученный после остановки результат.

При такой интерпретации термина «вычислимая частичная арифметическая функция» все становится совсем «чисто» и при перечислении машин Тьюринга мы действительно ***строго один раз*** перечисляем каждую вычислимую частичную арифметическую функцию, т.о. доказано что множество ВЧАФ является счетным и эффективно перечислимым, **Q.E.D.**

§ 3.3. Эффективное распознавание и сравнение функций

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

Т. 3.3.(1) Теорема



Невозможно эффективно распознать функции-константы среди вычислимых арифметических функций.

Доказательство

Общая идея.

Пусть машина Тьюринга **T** вычисляет некоторую функцию *f(x)*. Если существует такая процедура эффективного распознавания – значит, существует и машина Тьюринга **F**, ее реализующая. В ходе своей работы машина **F** должна поочередно подставлять в качестве входных значений для машины **T** натуральные числа и сравнивать результат работы машины **T** с заданным числом (константой). Поскольку натуральных чисел – счетно-бесконечное множество, процесс сравнения может продолжаться бесконечно, а значит, однозначного ответа дать нельзя.

Строгое доказательство.

Без ограничения общности возьмем в качестве константы число 0. Построим функцию:

$$f_T(x)=\begin{cases} 0, \text{ если машина } \mathbf{T} \text{ не остановится за первые } (x+1) \text{ шагов;} \\ 1, \text{ в противном случае.} \end{cases}$$

Например, если **T** остановится на n–ом шаге, значения функции $f_T(x)$ будут выглядеть так:

x	$f_T(x)$
0	0
1	0
2	0
…	…
n-2	0
n-1	1
n	1

Далее значение функции будет всегда равно 1.

Т.о. функция *f(x)* окажется константой - ноль только в том случае, если машина **T** никогда не остановится на чистой ленте.

Предположим противное: пусть мы можем распознать константу - ноль среди ВАФ. Тогда для произвольной машины Тьюринга мы сможем с уверенностью сказать, остановится ли она когда-нибудь в ходе своей работы. Подобный вывод прямо противоречит теореме о неразрешимости проблемы остановки, следовательно, предположение неверно и константа-ноль (а также любые другие константы) не распознается эффективно среди вычислимых арифметических функций, **Q.E.D.**

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

Т. 3.3.(2) Теорема

T

Вычислимые арифметические функции не поддаются эффективному сравнению.

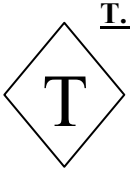
Доказательство

Пусть имеются две вычислимые арифметические функции $f_1(x)$ и $f_2(x)$. Построим функцию $g(x) = |f_1(x) - f_2(x)|$. Данная функция является арифметической (т.к. определена на всем множестве натуральных чисел) и вычислимой (в силу вычислимости $f_1(x)$ и $f_2(x)$, а также вычислимости процедуры нахождения модуля разности их значений).

Функция *g(x)* является функцией константой - ноль, если сравниваемые функции равны. Далее продолжим доказательство теоремы методом «от противного».

Предположим противное. Пусть существует алгоритм эффективного сравнения двух функций. Это значит, что существует алгоритм распознавания функции-константы ноль среди всех вычислимых арифметических функций, что противоречит доказанной ранее теореме. Значит, исходное предположение неверно, и вычислимые арифметические функции не поддаются эффективному сравнению, **Q.E.D.**

Т. 3.3.(3)Теорема



Невозможно эффективно распознать функции-тождества среди вычислимых арифметических функций.

Доказательство

Построим функцию

$$f_T(x) = \begin{cases} x, \text{ если машина } \mathbf{T} \text{ не остановится за первые } (x+1) \text{ шагов;} \\ x+1, \text{ в противном случае.} \end{cases}$$

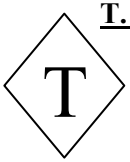
Например, если **T** остановится на *n* – ом шаге, значения функции $f_T(x)$ будут выглядеть так:

x	$f_T(x)$
0	0
1	1
…	…
n-2	n-2
n-1	n
…	

Т.о. функция $f_T(x)$ окажется функцией - тождеством только в том случае, если машина **T** никогда не остановится на чистой ленте.

Предположим противное. Пусть мы можем распознать функции-тождества среди ВАФ. Тогда для произвольной машины Тьюринга мы сможем с уверенностью сказать, остановится ли она когда-нибудь в ходе своей работы. Подобный вывод прямо противоречит теореме о неразрешимости проблемы остановки произвольной машины Тьюринга на чистой ленте – следовательно, сделанное предположение неверно, и функции-тождества не распознаются эффективно среди вычислимых арифметических функций, **Q.E.D.**

Т. 3.3.(4)Теорема



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

Доказательство

Общая идея.

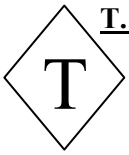
По сути это означает, что не существует алгоритма, по которому можно распознать является ли вычислимая частичная арифметическая функция вычислимой арифметической функцией. Действительно, для того чтобы понять, является ли ВЧАФ всюду определенной, нам пришлось бы вычислить ее для всех натуральных чисел, что, по словам Г.Н. Поварова, «под силу лишь бесконечному разуму».

Строгое доказательство.

1.По теореме Поста (применительно к множествам ВАФ и ВЧАФ) множество ВАФ эффективно распознается в ВЧАФ тогда и только тогда, когда ВАФ эффективно перечислимо и ВЧАФ\ВАФ эффективно перечислимо.

2. Предположим противное. Пусть ВАФ можно эффективно распознать в ВЧАФ. Тогда ВАФ эффективно перечислимо, а это противоречит Теореме 3.1.(3), согласно которой ВАФ эффективно не перечислимо.

Следовательно, исходное предположение неверно и ВАФ эффективно не распознаваемо в множестве ВЧАФ, **Q.E.D.**



Т. 3.3.(5)Теорема Черча

Невозможно эффективно распознать точки неопределенности вычислимой частичной арифметической функции.

Доказательство

Общая идея.

Это означает, что не существует алгоритма, по которому для произвольной ВЧАФ можно выявить все точки неопределенности. Действительно, для того чтобы распознать даже одну точку неопределенности, нам бы пришлось ждать результата работы алгоритма вычисления функции сколь угодно долго (до тех пор, пока не будет посчитано значение функции, чего может и вовсе не случится, если это точка неопределенности).

Строгое доказательство.

1. Поскольку множество ВЧАФ n переменных эффективно перечислимо по Теореме 3.2.(2) , то перечислим их:

$f_1(x_1, ..., x_n), f_2(x_1, ..., x_n), ...$

2. Построим диагональную функцию

$$g(x_1,...,x_n)= \begin{cases} 0, \text{ если } f_{x1}(x_1,...,x_n) \text{ не определена;} \\ \text{не определена, если } f_{x1}(x_1,...,x_n) \text{ определена.} \end{cases}$$

Видно, что $g(x_1,...,x_n)$ является частичной арифметической функцией.

3. Предположим противное, а именно что теорема Черча не верна и существует алгоритм эффективного распознавания точек неопределенности ВЧАФ. Тогда $g(x_1,...,x_n)$ становится вычислимой частичной арифметической функцией, то есть ВЧАФ.

4. Значит, $g(x_1,...,x_n)$ должна была попасть в перечисление множества ВЧАФ n переменных. Однако по построению $g(x_1,...,x_n)$ отличается от всех перечисленных функций, значит в перечислении ее нет.

5. Получили противоречие. Значит, предположение неверно, т.е. не существует алгоритма эффективного распознавания точек неопределенности ВЧАФ, **Q.E.D**.

Задачи к главе 3

Условие для задач 3.1.-3.6.
Пусть задана частичная арифметическая функция *f(x)*. Множество A- её область определенности, а множество B – её область значений. Найти характеристические функции множеств A и B.

Задача 3.1.
$f(x) = x–15$.

Задача 3.2.
$f(x) = 2x+5$.

Задача 3.3.
$f(x) = 3x–5$.

Задача 3.4.
$f(x) = x – 5$.

Задача 3.5.
$f(x) =(x-6) / (9-x)$.

Задача 3.6
$f(x) = 3x+4$.

Условие для задач 3.7.-3.12.
Пусть задана функция f от одного или двух аргументов. Укажите все множества (АФ, ВАФ, ЧАФ, ВЧАФ), к которым принадлежит функция f.

Задача 3.7.
$f(x, y) = -x-1-y$.

Задача 3.8.
$f(x, y) = 2x-1+3y$.

Задача 3.9

$f(x) = 2x+3$.

Задача 3.10

$$f(x) = \begin{cases} x+4, \text{ если машина Тх не остановится на чистой ленте;} \\ 1, \text{ если остановится за первые 15 тактов;} \\ 3x-1, \text{ если остановится позднее 15-ого такта.} \end{cases}$$

Задача 3.11

$$f(x) = \begin{cases} 1, \text{ если машина Тх остановится за первые 18 тактов;} \\ \\ x+2, \text{ если машина Тх остановится позднее 18-ого такта.} \end{cases}$$

Задача 3.12

$$f(x, y) = \begin{cases} 3x-2y, \text{ если } x > y; \\ y-x, \text{ если } x < y; \\ (x+y)/2 \text{ если } x=y. \end{cases}$$

Условие для задач 3.13.-3.15.

Для некоторой частичной арифметической функции $f(x)$ заданы множество A (область определенности) и множество B (область значений). Найти аналитический вид функции $f(x)$ и задать множества A и B с помощью характеристических функций.

Задача 3.13

A: $(x \geq 11) \cap (\text{mod}(x,2)=0)$;
B: N.

Задача 3.14

A: $\text{mod}(x,2)=1$;
B: $y \geq 8$.

Глава 4. Рекурсивные функции

§ 4.1. Роль рекурсивных функций

Задача точного определения понятия алгоритма была полностью решена в 30-х годах XX века в двух формах: на основе описания алгоритмического процесса и на основе понятия рекурсивной функции.

Первый подход подробно описан в главе 1 и заключается в том, что был сконструирован формальный автомат, способный осуществлять ограниченный набор строго определённых элементарных операций (машина Тьюринга). Алгоритмом стали называть конечную последовательность таких операций и постулировали предложение, что любой интуитивный алгоритм является алгоритмом и в сформулированном выше смысле. То есть для каждого алгоритма можно подобрать реализующую его машину Тьюринга.

Развитие этой теории позволило строго доказать алгоритмическую неразрешимость ряда важных математических проблем. Однако её существенным недостатком является невозможность реализации такой полезной конструкции, как рекурсия.

Этого недостатка лишён другой подход к формализации понятия алгоритма, развитый Гёделем и Чёрчем. Здесь теория построена на основе широкого класса так называемых частично рекурсивных функций.

Замечателен тот факт, что оба данных подхода, а также другие подходы (в том числе алгоритмы Маркова, машины Поста) приводят к одному и тому же классу алгоритмически вычислимых функций. Поэтому далее представляется целесообразным рассмотреть теорию частично рекурсивных функций Чёрча, так как она позволяет доказать не только алгоритмическую разрешимость конкретной задачи, но и существование для неё рекурсивного алгоритма.

Важно отметить, что рекурсия сама по себе не является алгоритмом, она является методом построения алгоритмов. Ее

очень удобно применять (но не обязательно эффективно) в тех случаях, когда можно выделить самоподобие задачи, т.е. свести вычисление задачи некоторой размерности N к вычислению аналогичных задач меньшей размерности.

При этом, если получается сделать алгоритм без применения рекурсии, то, скорее всего, им и надо воспользоваться. Рекурсивные вызовы подпрограмм имеют свойство решать одну и ту же задачу бесчисленное количество раз (во время повторов), что значительно сказывается на времени.

Кроме того, рекурсивные функции очень опасны. Несмотря на то, что существует множество задач, на решение которых прямо напрашивается рекурсия, не стоит сразу же бросаться реализовывать рекурсивные вызовы. Вполне вероятно, что все это обернется либо большими и неоправданными расходами оперативной памяти, либо будет очень медленно работать.

Широкий класс частично-рекурсивных функций, как будет показано далее, можно детализировать, строго выделив довольно интересные по своим свойствам подклассы примитивно-рекурсивных и общерекурсивных функций. Гипотеза Черча (о тождественности класса общерекурсивных функций и класса всюду определенных вычислимых функций) и гипотеза Клини (о тождественности класса частичных функций, вычисляемых посредством алгоритмов, и класса частично-рекурсивных функций) хотя и являются недоказуемыми, позволили придать необходимую точность формулировкам алгоритмических проблем и в ряде случаев сделать возможным доказательство их неразрешимости.

§ 4.2. Примитивно-рекурсивные функции

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

В качестве базисных функций обычно берутся следующие простейшие функции:

- функция **следования**;

- функция **тождества** (функция проекции, или функция выбора аргументов);
- функция **константы**.

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

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

Таким образом, *базис Клини* состоит из:

- трех простых функций;
- двух разрешенных операций.

Рассмотрим сначала функции:

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

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

3) функция **константы**: $C_q^n(x_1,...,x_n) = q$, где n – число переменных, q – значение, которое принимает функция. Функция константа принимает всюду одно значение. Например, $C_2^3(16,9,10)=2$.

Далее рассмотрим операции.

1. Операция **примитивной рекурсии R.**

Если мы имеем функцию одной переменной *f(x)*, то схема рекурсии называется «рекурсия без параметров» и задается системой уравнений**:**

$$\begin{cases} f(0) = q \\ f(x') = \chi(f(x), x) \end{cases}$$

Функция, заданная такими уравнениями, кратко задается схемой вида $\mathbf{R_q}(\chi)$**.** Поскольку вид системы уравнений (и способ

задания трех разрешенных функций) строго определен, то схема является однозначной.

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

$$\begin{cases} f(0, x_2,\ldots, x_n) = \Psi(x_2,\ldots, x_n) \\ f(x'_1, x_2,\ldots, x_n) = \chi(f(x_1,\ldots, x_n), x_1,\ldots, x_n) \end{cases}$$

Функция, заданная такими уравнениями, кратко задается схемой вида $\mathbf{R^n(\Psi, \chi)}$

2. Операция **подстановки (суперпозиции) S.**

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

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

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

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

При этом функция Φ при подсчете внутренних функций не учитывается.

§ 4.3. Частично рекурсивные функции

Класс частично-рекурсивных функций определяется путем указания конкретных исходных функций и фиксированного множества операций получения новых функций из ранее заданных.

В качестве базисных функций обычно берутся следующие:

- функция **следования**;

- функция **тождества** (функция проекции, или функция выбора аргументов);
- функция **константы**.

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

> *Функция называется **частично-рекурсивной**, если она может быть получена из простейших функций C_q^n, S, U_m^n конечным числом операций подстановки, примитивной рекурсии и минимизации (т.е. задана в расширенном базисе Клини).*

Таким образом, *расширенный базис Клини* состоит из:

- трех простых функций (аналогичны стандартному базису Клини);
- трех разрешенных операций (две из них аналогичны стандартному базису Клини, третья называется операцией минимизации).

Оператор минимизации

Пусть имеется функция $f(x_1,....x_n)$, принадлежащая множеству частичных арифметических функций (ЧАФ). Пусть существует какой то механизм ее вычисления, причем значение функции не определено тогда и только тогда, когда этот механизм работает бесконечно, не выдавая никакого определенного результата.

Фиксируем какие-нибудь значения $x_1,.....,x_{n-1}$ для первых $n-1$ аргументов функции f и рассмотрим уравнение:

$f(x_1,.....,x_{n-1},y)=x_n$

Чтобы найти решение y (натуральное) этого уравнения, будем вычислять при помощи указанного выше "механизма" последовательно значения $f(x_1,...,x_{n-1},y)$ для y=0,1,2,... Наименьшее значение a, для которого получится $f(x_1,...x_{n-1},a)=x_n$, обозначим через

$\mu_y(f(x_1,...,x_{n-1},y)=x_n)$

Описанный процесс нахождения выражения $\mu_y(f(x_1,...,x_{n-1},y)=x_n)$ будет продолжаться бесконечно в следующих случаях:

- Значение $f(x_1,.....,x_{n-1},0)$ не определено;
- Значения $f(x_1,.....,x_{n-1},y)$ для $y=0,1,...,a-1$ определены, но отличны от x_n, а значение $f(x_1,.....,x_{n-1},a)$ – не определено;
- Значения $f(x_1,.....,x_{n-1},y)$ определены для всех $y=0,1,2,...$ и отличны от x_n.

Во всех этих случаях значение выражения $\mu_y(f(x_1,...,x_{n-1},y)=x_n)$ считается неопределенным. В остальных случаях описанный процесс обрывается и дает наименьшее решение $y=a$ уравнения $f(x_1,.....,x_{n-1},y)=x_n$. Это решение, как сказано, и будет значением выражения $\mu_y(f(x_1,...,x_{n-1},y)=x_n)$.

Например, для функции разности $f(x,y)=x-y$ в соответствии с указанным смыслом символа μ, для всех x,y имеем:

$f(x,y)=x-y=\mu_z(y+z=x)$

Подсчет значений функции $f(x,y)=x-y$ в этом случае будет проходить по двум сценариям.

Пример 1: Пусть x=5, y=3. Последовательно, начиная с z=0, перебираем возможные значения для нахождения $\mu_z(y+z=x)$:

- при z=0 получим выражение 3+0=5, его значение «ложь»;
- при z=1 получим выражение 3+1=5, его значение «ложь»;
- при z=2 получим выражение 3+2=5, его значение «истина», следовательно, процесс обрывается, и получаем результат: $\mu_z(3+z=5)=2$.

Таким образом, не умея производить операцию вычитая (которая отсутствует в базисе Клини), мы смогли посчитать значение частичной функции $f(x,y)=x-y$ для двух заданных значений аргументов.

Пример 2: Пусть x=3, y=5. Последовательно, начиная с z=0, перебираем возможные значения для нахождения $\mu_z(y+z=x)$:

- при z=0 получим выражение 5+0=3, его значение «ложь»;
- при z=1 получим выражение 5+1=3, его значение «ложь»;
- при z=2 получим выражение 5+2=3, его значение «ложь»;

и т.д. – т.е. при увеличении z ситуация, при которой выражение $5+z=3$ будет иметь значение «истина», так и не наступит, а значит значение функции $\mu_z(5+z=3)$ так и не будет найдено.

Пример 2 – это иллюстрация случая, при котором оператор минимизации не дает результата именно тогда, когда такой результат не может быть получен в принципе, по той причине, что в заданной точке функция действительно не определена.

Это третий случай, в котором процесс поиска выражения $\mu_y(f(x_1,...,x_{n-1},y)=x_n)$ продолжается бесконечно.

Напротив, значение выражения $\mu_y((y+1)(y-(x+1))=0)$ не определено, так как уже при $y=0$ значение терма $1\cdot(0-(x+1))$ для любого x не определено. В то же время уравнение $(y+1)(y-(x+1))=0$ имеет решение $y=x+1$, но оно не совпадает со значением выражения $\mu_y((y+1)(y-(x+1))=0)$.

Этот пример показывает, что для частичных функций $f(x_1,.....,x_{n-1},y)$ выражение $\mu_y(f(x_1,...,x_{n-1},y)=x_n)$, строго говоря, не есть наименьшее решение уравнения $f(x_1,.....,x_{n-1},y)=x_n$. Если же функция $f(x_1,.....,x_{n-1},y)$ всюду определенная и уравнение $f(x_1,.....,x_{n-1},y)=x_n$ имеет решения, то $\mu_y(f(x_1,...,x_{n-1},y)=x_n)$ есть наименьшее решение для этого уравнения. Отсюда возникает закономерное желание использовать под оператором минимизации только всюду определенные функции. Наилучший способ убедиться в том, что функция всюду определена, – использовать заведомо примитивно рекурсивные функции (или функции, примитивная рекурсивность которых уже доказана ранее или вытекает из способа их задания). В этом случае значения $f(x_1,.....,x_n)$ будет неопределенно только в одном из трех случаев, а именно если значения $f(x_1,.....,x_{n-1},y)$ определены для всех $y=0,1,2,...$ и отличны от x_n. Для этого имеющуюся функцию нужно преобразовать таким образом (путем переноса слагаемых, умножения обеих частей, и т.д.), чтобы по обе стороны равенства стояли примитивно рекурсивные (а значит всюду определенные) выражения. В общем виде, под оператором минимизации получим некое рекурсивное уравнение, части которого зависят от переменных $(x_1,.....,x_{n-1},x_n,y)$. Это уравнение принимает значение «истина» или «ложь» при последовательно выбираемых значениях параметра y.

Пример 1. Нигде неопределенная функция $f(x) = -x - 1$.

Построим рекурсивное уравнение:

$$f(x) = \mu_Z(z + x + 1 = 0)$$

Результат операции минимизации не определен даже для точки $x=0$.

Пример 2. Функция, определенная в одной точке, $f(x) = -x$.

Построим рекурсивное уравнение:

$$f(x) = \mu_Z(z + x = 0)$$

Результат операции минимизации определен только для точки $x=0$, при остальных значениях x вычисление (подбор нужного значения z) никогда не будет закончено.

Значение операции минимизации, примененное к функции $f(x,y)$, можно записать как $Mf(x,y)$.

Оператор минимизации для функций одной переменной (обращение функций)

Представляет интерес частный случай применения операции минимизации – функции одной переменной. Тогда вместо $Mf(x)$ используется запись $f^{-1}(x)$. В этом случае $f^{-1}(x)=\mu_y(f(y)=x)$ называется обращением функции $f(x)$.

Пример 1. Для знаковой функции $sg(x)$ обращение будет выглядеть следующим образом:

$$sg^{-1}(x) = \mu_y(f(y)=x) = \begin{cases} x \text{ при } x=0,1; \\ \text{не определена при } x>1. \end{cases}$$

Пример 2. Для функции следования $s(x)$ обращение будет выглядеть следующим образом:

$$s^{-1}(x) = \mu_y(f(y)=x) = \begin{cases} x-1 \text{ при } x>0; \\ \text{не определена при } x=0. \end{cases}$$

Тезис Черча. *Класс алгоритмически (или машинно) вычислимых частичных арифметических функций совпадает с классом всех частично рекурсивных функций.*

Иными словами, частичная арифметическая функция, вычислимая в общем смысле, есть суть частично рекурсивная

функция (рис. 4.1.). Это не строгое определение, это вывод на уровне тезиса (не доказанное, но и не опровергнутое утверждение).

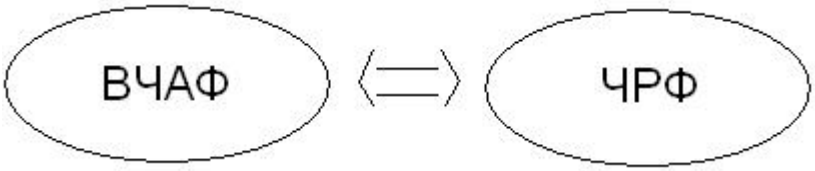


Рис. 4.1. Эквивалентность классов функций ВЧАФ и ЧРФ

Итак, для того чтобы задать частично-рекурсивную функцию в виде элементарной схемы ее вычисления, необходимо расширить введенный ранее базис Клини за счет появления новой операции – операции минимизации. Здесь важно понимать, что в расширенном базисе Клини в итоге могут оказаться заданы также и примитивно-рекурсивные функции – но для примитивно-рекурсивных функций использование этого оператора, по сути, является избыточным.

Пример 1. Рассмотрим всюду определенную функцию $f(x) = x + 8$. Построим рекурсивное уравнение: $f(x) = \mu_Z(x + 8 = z)$. Несмотря на то, что в данном определении функции *f(x)* присутствует оператор минимизации, сама функция примитивно-рекурсивна, её можно выразить и без данного оператора в виде обычной схемы

$f(x) = x + 8.$

$$\begin{cases} f(0) = 0 + 8 = 8 \\ f(x') = x + 1 + 8 = (x + 8) + 1 = (x + 8)' = [f(x)]' = S(f(x)) = \\ = S(U_1^2(f(x), x)) = S(U_1^2) = S_1^2(S, U_1^2) \end{cases}$$

$$f(x) = R_8\ (S_1^2(S, U_1^2))$$

§ 4.4. Общерекурсивные функции

Операция минимизации, введенная ранее, ставит в соответствие любой заданной функции *f* определенную частичную функцию *Mf*. Введем еще одну операцию, которую будем обозначать символом $M^l f$ и называть **слабой минимизацией**.

По определению положим $M^l f = Mf$, если функция Mf всюду определена.

> *Частичная арифметическая функция f называется* ***общерекурсивной****, если она может быть получена из из простейших функций C_q^n, S, U_m^n конечным числом операций подстановки, примитивной рекурсии и* ***слабой*** *минимизации.*

Так как операции подстановки, примитивной рекурсии и слабой минимизации, выполненные над всюду определенными функциями, либо ничего не дают, либо дают снова функции, всюду определенные, то все общерекурсивные функции всюду определенные.

С другой стороны, если результат операции слабой минимизации получен, то он совпадает с результатом обычной минимизации. Поэтому все общерекурсивные функции являются всюду определенными частично-рекурсивными функциями. Обратное тоже верно: каждая всюду определенная частично-рекурсивная функция является общерекурсивной (это утверждение, вообще говоря, требует доказательства, которое в данной книге не рассматривается).

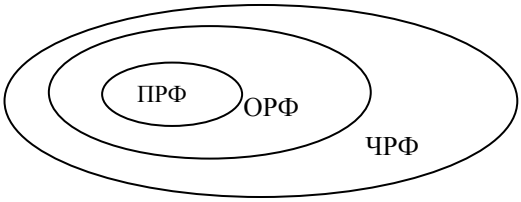


Рис. 4.2. Отношения между классами рекурсивных функций

Согласно определению, каждая примитивно-рекурсивная функция является общерекурсивной. Классы рекурсивных функций отображены на рис. 4.2.

Согласно тезису Черча, класс всюду определенно вычислимых числовых функций также будет совпадать с классом общерекурсивных функций (рис. 4.3.).

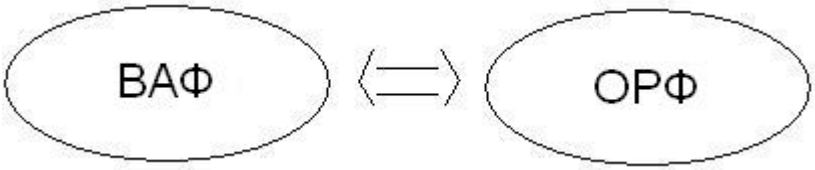


Рис.4.3. Эквивалентность классов функций ВАФ и ОРФ

§ 4.5. Задание рекурсивных функций

4.5.1. Задание рекурсивных функций при помощи системы уравнений

В общем случае **рекурсией** называется такой способ задания функции, при котором значения определяемой функции для произвольных значений аргументов выражаются известным образом через значения определяемой функции для меньших значений аргументов.

Рассмотрим равенство $f(x)=g(x)$, где $f(x)$ и $g(x)$ некоторые функции. Когда говорят, что это равенство есть уравнение, то это означает, что равенство рассматривается как неопределенное высказывание, при одних значениях x истинное, при других ложное.

Произвольная система рекурсивных уравнений задает частично – рекурсивную функцию. Если данная функция определена на всем множестве натуральных чисел, то она будет ***общерекурсивной*.** Также верно другое утверждение. Функция является общерекурсивной, если она *определена* посредством ряда уравнений некоторого типа.

Затруднение вызвано тем, что просмотр набора уравнений обычно не убеждает нас в том, что эти уравнения “определяют” какую-либо функцию вообще. Когда говорится “функция”, обычно понимается под этим “правило, которое каждому значению (или набору из n значений, если мы имеем функцию n переменных) ставит в соответствие результирующее значение”. Но в общем случае система уравнений не всегда дает определенное значение.

Например, система уравнений:

$$\begin{cases} \Phi(x) + 3x = f(x); \\ f(x+1) = f(x) + 2; \\ f(0) = 3 \end{cases}$$

задает некоторую функцию $\Phi(x)$. Однако ответить на вопрос, *определяет* ли данная система эту функцию Φ в общем случае невозможно. В данном конкретном случае такие выводы очевидно можно сделать на основе обычного подсчета значений и наших наблюдений касательно поведения функции при росте ее аргумента. Результаты расчетов представлены в табл. 4.1.

Таблица 4. 1. Поведение функции при росте ее аргумента.

x =	Уравнение	$f(x)$	Уравнение	$\Phi(x)$
0		$f(0)=3$	$\Phi(0)+0=3$	$\Phi(0)=3$
1	$f(0+1)=f(0)+2=3+2=5$	$f(1)=5$	$\Phi(1)+3*1=5$	$\Phi(1)=2$
2	$f(1+1)=f(1)+2=5+2=7$	$f(2)=7$	$\Phi(2)+3*2=7$	$\Phi(2)=1$
3	$f(2+1)=f(2)+2=7+2=9$	$f(3)=9$	$\Phi(3)+3*3=9$	$\Phi(3)=0$
4	$f(3+1)=f(3)+2=9+2=11$	$f(4)=11$	$\Phi(4)+3*4=11$	$\Phi(4)$ не опред

Из таблицы видно, что для значений $x=4$ и выше значение функции Φ не определено. Следовательно, функция $\Phi(x)$ – частично-рекурсивная функция.

Таким образом, вполне допустимой является ситуация, когда для некоторых (или даже для всех) точек значение функции не определено – считается, что в этих точках сама функция не определена. Это явление весьма характерно. Поэтому, делая предположение, что система уравнений действительно определяет общерекурсивную функцию, нужно всегда проявлять осторожность. Обычно требуется дополнительное доказательство этого положения, например, в виде индуктивного доказательства того, что для каждого значения аргумента вычисление завершается выдачей единственного значения.

Например, система уравнений

$$\begin{cases} \Phi(x) + x = f(x); \\ f(x+1) = f(x) + 2; \\ f(0) = 3 \end{cases}$$

задает функцию *Ф(x)*, которая, как видно из таблицы наших наблюдений касательно поведения функции при росте ее аргумента (табл. 4.2.), оказывается *всюду* определенной. Необходимо обратить особое внимание на то, что такой вывод мы может сделать только при рассмотрении конкретной заданной функции, в общем же случае, как будет показано позднее, невозможно придумать алгоритм, который на основе анализа системы рекурсивных уравнений некоторого типа сможет ответить на вопрос о принадлежности функции к классу общерекурсивных.

В данном случае вычисление значения функции *Ф(x)* дает следующие результаты (табл. 4.2):

Таблица 4.2. Поведение функции при росте ее аргумента.

x=	Уравнение	*f(x)*	Уравнение	*Ф(x)*
0		*f(0)=3*	*Ф(0)+0=3*	*Ф(0)=3*
1	*f(0+1)=f(0)+2=3+2=5*	*f(1)=5*	*Ф(1)+ 1=5*	*Ф(1)=4*
2	*f(1+1)=f(1)+2=5+2=7*	*f(2)=7*	*Ф(2)+ 2=7*	*Ф(2)=5*
3	*f(2+1)=f(2)+2=7+2=9*	*f(3)=9*	*Ф(3)+ 3=9*	*Ф(3)=6*
4	*f(3+1)=f(3)+2=9+2=11*	*f(4)=11*	*Ф(4)+ 4=11*	*Ф(4)=7*

Очевидно, что разница между значением функции *f(x)* и собственно значением аргумента *x* с ростом аргумента только увеличивается, а значит уравнение *Ф(x) +x=f(x)* всегда будет иметь единственное решение. Таким образом, функция *Ф(x)* – общерекурсивная функция.

Система уравнений может быть задана через базисные операции в виде схемы рекурсии. Например, рассмотрим систему уравнений, задающую функцию *F(x)*:

$$\begin{cases} E(x)=R_0(U^2_1); \\ F(x)=E^{-1}(x). \end{cases}$$

Если раскрыть операции рекурсии и обращения функций, получим:

$$\begin{cases} E(0)=0; \\ E(x+1)=E(x); \\ F(x)=\mu_y[E(y)= x]. \end{cases}$$

Для $x=0$ значение $F(x)$ определено и равно 0. Но для $x=1$ наше определение "не работает", так как *выражение* $F(1)=\mu_y[E(y)=1]$, означает:

-сначала посмотри, истинно ли $E(0) = 1$;

-если нет, посмотри, истинно ли $E(1) = 1$;

-если нет, посмотри, истинно ли $E(2) = 1$;

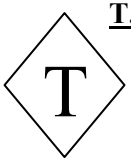
-если нет, то ... и т.д.

Вычисление никогда не заканчивается, потому что $E(x)$ всегда равна нулю, и для $F(1)$ не будет найдено никакого значения.

Вследствие этого при рассмотрении функций, заданных системами рекурсивных уравнений, будем обычно говорить о *частично-рекурсивной функции*. Когда говорится о частично-рекурсивной функции $F(x)$, то надо понимать, что может не существовать значения, определенного для некоторого (или даже любого!) значения x. Если $F(x)$ *о*казывается определенной для всех значений x, то будем называть ее *общерекурсивной функцией*. Конечно, любая общерекурсивная функция также является частично-рекурсивной.

4.5.2. Дополнительные способы задания примитивно-рекурсивных функций

Расширим класс операций, которые, так же как и суперпозиция с примитивной рекурсией, дают в качестве результата примитивно-рекурсивные функции.



Т.4.5.(1)Теорема

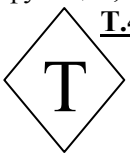
Пусть n-местная функция $g(x_1,.....,x_{n-1},x_n)$ примитивно-рекурсивная. Тогда n-местная функция $f(x_1,.....,x_{n-1}, x_n)$, определенная равенством $f(x_1,.....,x_{n-1}, x_n)=\sum g(x_1,.....,x_{n-1}, i)$, где i изменяется от 0 до x_n, также примитивно-рекурсивная.

Доказательство

Запишем схему примитивной рекурсии по последней переменной:

$$\begin{cases} f(x_1,.....,x_{n-1}, 0) = g(x_1,.....,x_{n-1}, 0); \\ f(x_1,.....,x_{n-1}, x_n +1) = f(x_1,.....,x_{n-1}, x_n) + g(x_1,.....,x_{n-1}, x_n +1). \end{cases}$$

Из этого описания очевидно, что f – примитивно рекурсивная функция, **Q.E.D.**



Т.4.5.(2)Теорема

Пусть n-местная функция $g(x_1,.....,x_{n-1},x_n)$ примитивно-рекурсивна. Тогда n-местная функция $f(x_1,.....,x_{n-1}, x_n)$ определенная равенством $f(x_1,.....,x_{n-1}, x_n)=\prod g(x_1,.....,x_{n-1}, i)$, где i изменяется от 0 до x_n, также примитивно-рекурсивна.

Доказательство

Запишем схему примитивной рекурсии по последней переменной:

$$\begin{cases} f(x_1,.....,x_{n-1}, 0) = g(x_1,.....,x_{n-1}, 0); \\ f(x_1,.....,x_{n-1}, x_n +1) = f(x_1,.....,x_{n-1}, x_n) \cdot g(x_1,.....,x_{n-1}, x_n +1). \end{cases}$$

Из этого описания очевидно, что f – примитивно рекурсивная функция, **Q.E.D.**

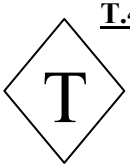
Пусть заданы некоторые функции $f_i(x_1,.....,x_n)$, $i=1,...,s+1$ и указаны какие то условия $Pj(x_1,.....,x_n)$, $j=1,...,s$, которые для любого набора чисел $x_1,.....,x_n$ могут быть истинными или ложными. Допустим, что ни для одного набора чисел $x_1,.....,x_n$ никакие два из упомянутых условий не могут быть одновременно истинными. Функция $f(x_1,.....,x_n)$, заданная схемой:

$$f(x_1,.....,x_n) = \begin{cases} f_1(x_1,...,x_n) & \text{если } P_1(x_1,...,x_n) \text{ истинно;} \\ f_2(x_1,...,x_n) & \text{если } P_2(x_1,...,x_n) \text{ истинно;} \\ \ldots\ldots\ldots & \\ f_s(x_1,...,x_n) & \text{если } P_s(x_1,...,x_n) \text{ истинно;} \\ f_{s+1}(x_1,...,x_n) & \text{для остальных } x_1,...,x_n \end{cases}$$

*называется **кусочно-заданной** функцией.*

Вопрос о том будет ли функция f примитивно-рекурсивной, зависит от природы функций f_i и условий P_i. Зато для простейшего случая можно доказать очень полезную теорему.

Т.4.5.(3)Теорема



Пусть заданы n-местные примитивно-рекурсивные функции $f_1(x_1,\ldots,x_n)$, $f_2(x_1,\ldots,x_n)$, …, $f_{s+1}(x_1,\ldots,x_n)$, $a_1(x_1,\ldots,x_n)$, $a_2(x_1,\ldots,x_n)$, …, $a_s(x_1,\ldots,x_n)$, причем ни при каких значениях переменных никакие две из функций a_i одновременно в 0 не обращаются. Тогда функция, определенная кусочной схемой

$$f(x_1,\ldots,x_n)=\begin{cases} f_1(x_1,\ldots,x_n), & \text{если } a_1(x_1,\ldots,x_n)=0;\\ f_2(x_1,\ldots,x_n), & \text{если } a_2(x_1,\ldots,x_n)=0;\\ \ldots\ldots; & \\ f_s(x_1,\ldots,x_n), & \text{если } a_s(x_1,\ldots,x_n)=0;\\ f_{s+1}(x_1,\ldots,x_n), & \text{для остальных } x_1,\ldots,x_n, \end{cases}$$

будет примитивно-рекурсивной.

Доказательство

Функцию f можно представить в виде

$f=f_1unsga_1+\ldots+f_sunsga_s+f_{s+1}sg(a_1\cdot a_2\cdot\ldots\cdot a_s)$.

Все примененные функции – примитивно рекурсивные, следовательно, f – примитивно рекурсивная функция, **Q.E.D.**

В данной теореме рассмотрены простейшие случаи, когда все условия p_i имеют вид $a_i=0$. На самом деле условия могут быть сформулированы иначе, однако с помощью функции псевдоразности их можно преобразовать к виду, указанному в теореме.

Например,

$a_i=b_i$ эквивалентно $|a_i\div b_i|=0$,

$a_i\leq b_i$ эквивалентно $a_i\div b_i=0$,

$a_i<b_i$ эквивалентно $unsg(b_i\div a_i)=0$.

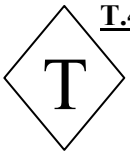
Таким образом, теорема оказывается справедливой и в этих случаях.

§ 4.6. Мажорируемые неявные функции

Как уже говорилось в п.4.3., большой ошибкой было бы думать, что функция, выраженная при помощи оператора минимизации, однозначно является непримитивно-рекурсивной. В ряде случаев использование оператора минимизации есть не более чем удобный способ задания процедуры вычисления, позволяющий обойти определенные формальные трудности. Как мы покажем ниже, иногда пусть и более сложным образом, но все же удается заменить процедуру поиска минимально возможного решения для уравнения, задающего всюду определенную функцию.

Рассмотрим уравнение $g(x_1,\ldots,x_n, y)=0$, левая часть которого есть всюду определенная функция. Допустим, для каждого $x_1,\ldots x_n$ уравнение имеет единственное решение y. Тогда это решение будет однозначной всюду определенной функцией от $x_1,\ldots x_n$. Актуален вопрос: будет ли функция $y=f(x_1,\ldots,x_n)$ – примитивно-рекурсивной, если левая часть уравнения, а именно $g(x_1,\ldots,x_n,y)$ есть примитивно рекурсивная функция.

В общем случае это не так. Зато в некоторых отдельных случаях ответ будет положительным.



Т.4.6.(1)Теорема о мажорируемых неявных функциях

Пусть $g(x_1,\ldots,x_n, y)$, $a(x_1,\ldots,x_n)$ – такие примитивно-рекурсивные функции, что уравнение $g(x_1,\ldots,x_n, y)=0$ для каждых $x_1,\ldots,x_n$ имеет по меньшей мере одно решение и $\mu_y(g(x_1,\ldots,x_n, y)=0) \le a(x_1,\ldots,x_n)$ для любых $x_1,\ldots,x_n$. Тогда функция $f(x_1,\ldots,x_n)= \mu_y(g(x_1,\ldots,x_n, y)=0)$ также примитивно рекурсивна.

Доказательство

Фиксируем какие-нибудь значения $x_1,\ldots,x_n$ и пусть $b= \mu_y(g(x_1,\ldots,x_n, y)=0)$. Рассмотрим последовательность произведений:

$g(x_1,\ldots,x_n, 0)$
$g(x_1,\ldots,x_n, 0) \cdot g(x_1,\ldots,x_n, 1);$
$g(x_1,\ldots,x_n, 0) \cdot g(x_1,\ldots,x_n, 1) \cdot g(x_1,\ldots,x_n, 2);$
$\ldots;$
$g(x_1,\ldots,x_n, 0) \cdot g(x_1,\ldots,x_n, 1) \cdot g(x_1,\ldots,x_n, 2) \cdot\ldots\cdot g(x_1,\ldots,x_n, i);$
$\ldots$

Так как $y=b$ есть наименьшее решение уравнения $g(x_1,\ldots,x_n, y)=0$, то первые b членов этой последовательности нулю точно не равны, зато все остальные содержат равный нулю сомножитель $g(x_1,\ldots,x_n, b)$ и потому равны нулю.

Из условия $\mu_y(g(x_1,\ldots,x_n, y)=0) \le a(x_1,\ldots,x_n)$ вытекает, что

$$b = \sum_{i=0}^{a(x_1,\ldots,x_n)} sg\,(g(x_1,\ldots,x_n, 0) \cdot g(x_1,\ldots,x_n, 1) \cdot\ldots\cdot g(x_1,\ldots,x_n, i)).$$

Введем примитивно-рекурсивную функцию:

$$h(x_1,\ldots,x_n, z)=\prod_{i=0}^{z} g(x_1,\ldots,x_n, i).$$

Теперь получим:

$$f(x_1,\ldots,x_n)= \sum_{i=0}^{a(x_1,\ldots,x_n)} sg\, h(x_1,\ldots,x_n, i).$$

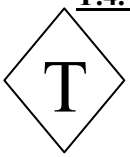
В силу теорем о сложении и мультиплицировании $f(x_1,\ldots,x_n)$ – примитивно-рекурсивная функция, **Q.E.D.**

§ 4.7. Возвратная рекурсия и функция Фибоначчи

Пусть $a_1(x),\ldots, a_s(x)$ – всюду определенные функции, удовлетворяющие для любого значения x условиям $a_i(x+1) \le x$ $(i = 1,\ldots,s)$. Говорят, что функция $f(y)$ получается ***возвратной рекурсией*** *из функции $h(y, z_1,\ldots,z_s)$ и вспомогательных функций $a_1(x),\ldots,a_s(x)$, если для любых значений y выполнено:*

$f(0) = q$ *(константа);*
$f(y+1) = h(f(a_1(y+1)),\ldots, f(a_s(y+1)),y).$

Т.4.7.(1)Теорема (без доказательства)



Пусть функция *f* – возвратная рекурсия относительно функций *h, a_1,…,a_s*. Тогда *f* примитивно рекурсивна, если соответствующие функции примитивно рекурсивны

В качестве примера возвратной рекурсии для функции одной переменной чаще всего приводят функцию Фибоначчи.

Для более доступного изложения проблемы, приведем определение и формулировку для случая одной-единственной переменной.

*Числа Ф(n) , образующие последовательность **1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ...** называются **"числами Фибоначчи"**, а сама последовательность - **последовательностью Фибоначчи,** если каждый новый член последовательности равен сумме двух предыдущих.*

Чтобы иметь дело со всюду определенными функциями, доопределим *Ф(n)* в точке *n=0*. Пусть *Ф(0)=0*. Следовательно, функция Фибоначчи может быть представлена следующим образом:

$$\begin{cases} \Phi(0) = 0; \\ \Phi(1) = 1; \\ \Phi(n+2) = \Phi(n) + \Phi(n+1), \end{cases}$$

или

$$\begin{cases} \Phi(0) = 0; \\ \Phi(n+1) = \Phi(n) + \Phi(n \div 1) + (1 \div sg(n)) \end{cases}, \quad \text{где } sg(x) = \begin{cases} 1, \text{ если } x > 0; \\ 0, \text{ если } x = 0. \end{cases}$$

Здесь и далее знак «÷» – знак псевдоразности. Исходя из определения возвратной рекурсии, получим:

Ф(n) – возвратная рекурсия из функций;

$h(n, a_1, a_2) = (1 \div sg(n)) + a_1 + a_2$

и вспомогательных функций

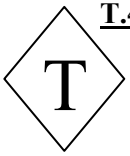
$a_1(n) = n \div 1;$
$a_2(n) = n \div 2$.

Пояснение:

$\Phi(n+1) = (1 \div sg(n)) + \Phi(a_1(n+1)) + \Phi(a_2(n+1)) =$
$= (1 \div sg(n)) + \Phi(n+1 \div 1) + \Phi(n+1 \div 2)) =$
$= (1 \div sg(n)) + \Phi(n) + \Phi(n-1).$

Так как все эти функции примитивно рекурсивны, то функция $\Phi(n)$ также примитивно рекурсивна.

§ 4.8. Эффективная перечислимость и распознаваемость



Т.4.8.(1)Теорема

Примитивно-рекурсивные функции эффективно перечислимы.

Доказательство

Из теоремы Клини следует, что примитивно-рекурсивные функции могут быть представлены в виде операторного терма, т.е. в виде схемы примитивной рекурсии, в которой используется всего несколько функциональных символов: для трех функций C_q^n, S (без индексов), U_m^n и двух операций: S_m^n с двумя индексами для подстановки и R^n для примитивной рекурсии.

Построим доказательство эффективной перечислимости примитивно-рекурсивных функций на основе геделевой нумерации.

Для этого надо ввести кодовые номера для всех символов:

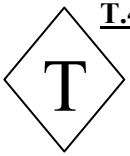
S (следование) $\rightarrow$ 1;
C $\rightarrow$ 2;
U $\rightarrow$ 3;
S (подстановка) $\rightarrow$ 4;
R (примитивная рекурсия) $\rightarrow$ 5;
($\rightarrow$ 6;
) $\rightarrow$ 7;
, $\rightarrow$ 8;
$x_i \rightarrow i+9$ (здесь i – номер переменной).

Возьмем ряд простых чисел: 2,3,5,7,…Дальнейшая процедура аналогична геделевой нумерации, например, алгоритмов Маркова.

Например,

$x+y=R^2[U_1^1,S_1^3(S,U_1^3)]= 2^5*3^{12}*5^6*7^3*11^{10}*13^{10}*17^8*\ldots$

Таким образом, получили геделев номер примитивно-рекурсивного описания функции суммы двух аргументов. Геделевы номера эффективно распознаются среди натуральных чисел, следовательно их описания можно эффективно перечислить, а значит множество ПРФ – эффективно перечислимо, **Q.E.D.**



Т.4.8.(2)Теорема

Множество частично-рекурсивных функций эффективно перечислимо.

Доказательство

Для наших целей важно, чтобы перечисление частично-рекурсивных функций было эффективным, т. е. чтобы у нас был эффективный способ нахождения *n*-й функции списка или хотя бы способ нахождения ее определения. Хотя это может показаться странным, но будет достаточно знать, что перечисление существует - нам даже не потребуется выяснять какие-либо подробности о его свойствах. Следовательно, вместо подробного доказательства, достаточно привести убедительный довод в пользу его возможности, т. е. что мы могли бы построить его, если бы это потребовалось.

Почему мы все же могли предположить, что существует эффективное перечисление частично-рекурсивных функций? В конце концов, они образуют фантастически богатое, причудливое и неупорядоченное множество объектов, *поскольку в это множество входят объекты, соответствующие всем возможным рекурсивным определениям.*

На самом деле при ближайшем рассмотрении ясно, что определение каждой частично-рекурсивной функции состоит из конечного набора уравнений, каждое из которых включает конечное число символов, обозначающих *нуль, тождество, следование, примитивную рекурсию, суперпозицию и наименьшее*

число в сочетании с конечным числом *запятых и скобок.* Точные правила их расположения не важны. Важно то, что в любой такой ситуации, всегда можно ввести некоторого рода бесконечное лексикографическое упорядочение составных объектов. Хотя эта ситуация кажется более сложной, есть способ сделать ее даже проще. Действительно, поскольку большинство частично-рекурсивных функций (или, скорее, определений) вовсе не определяет истинного окончания процесса вычислений, нам не надо слишком заботиться, чтобы при перечислении определений все последовательности символов были правильными. Действительно, мы можем рассмотреть какую-либо процедуру, которая, в конце концов, образует любую (и все) последовательности символов, требуемых для определений. Имеется только одна техническая проблема - остаться в рамках фиксированного алфавита. Она решается использованием, скажем, двоичного кодирования неограниченного числа названий переменных и функций, которые могут потребоваться. Тогда простая схема числового упорядочения будет эффективным перечислением при условии, что существует способ правильной интерпретации *n*-го описания.

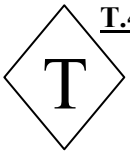
Бесконечное лексикографическое упорядочение составных объектов легко нумеруется при помощи алгоритма Геделя. Из теоремы Клини следует, что частично-рекурсивные функции могут быть представлены в виде операторного терма, т.е. в виде схемы частичной рекурсии, в которой используется всего несколько функциональных символов: для трех функций C_q^n, S (без индексов), U_m^n и трех операций: S_m^n с двумя индексами для подстановки, R^n для примитивной рекурсии и μ для минимизации. Учитывая этот факт, снимается вопрос о правильной интерпретации *n*-го описания: если кодировать будем не системы уравнений, а схемы частично-примитивных рекурсий в виде операторных термов, то такая интерпретация будет автоматической.

Построим доказательство эффективной перечислимости на основе геделевой нумерации. Для этого надо ввести кодовые номера для всех элементов при помощи геделевой нумерации. Введем такие номера:

S (следование) $\rightarrow$ 1;
C (константа) $\rightarrow$ 2;
U (тождества) $\rightarrow$ 3;
S (подстановка) $\rightarrow$ 4;
R (примитивная рекурсия) $\rightarrow$ 5;
M (минимизация) $\rightarrow$ 6;
($\rightarrow$ 7;
) $\rightarrow$ 8;
, $\rightarrow$ 9;
$x_i \rightarrow i+10$.

Возьмем ряд простых чисел: 2,3,5,7,… Дальнейшая процедура аналогична геделевой нумерации примитивно-рекурсивных функций. Таким образом, получили геделев номер частично-рекурсивного описания. Геделевы номера эффективно распознаются среди натуральных чисел, следовательно, частично-рекурсивные функции можно эффективно перечислить, **Q.E.D.**

Предостережение: не делайте никаких далеко идущих предположений! Некоторые функции $f_n(x)$ не могут быть определены даже хотя бы для одного значения x. Некоторые функции с разными индексами, например $f_i(x)$ и $f_j(x)$, могут быть одинаковыми: $f_i(x) = f_j(x)$ для всех x. Действительно, некоторые функции могут появляться в списке бесконечно часто, что соответствует совершенно разным определениям. Единственное, в чем мы уверены, это то, что если функция частично-рекурсивна, то она имеет *по крайней мере* один индекс в перечислении.



Т.4.8.(3)Теорема

Общерекурсивные функции не поддаются эффективному перечислению.

На уровне тезисов:

Из тезиса Чёрча: всякая арифметическая функция, вычислимая в обычном смысле (ВАФ), есть обще рекурсивная функция (ОРФ), а всякая частичная арифметическая функция, вычислимая в обычном смысле (ЧВАФ), есть частично рекурсивная функция (ЧРФ). Известно (на основании теоремы

Тьюринга), что ВАФ эффективно не перечислимы. Отсюда следует, что и ОРФ эффективно не перечислимы.

Доказательство

Допустим, что множество общерекурсивных функций n-переменных эффективно перечислимо.

Тогда существует алгоритм, по которому их можно перечислить. Применим этот алгоритм. Получим последовательность:

$f_0(x_1,\ldots,x_n), f_1(x_1,\ldots,x_n),.. f_n(x_1,\ldots,x_n),..$

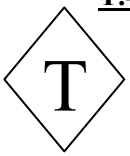
Построим диагональную функцию:

$$g(x1,\ldots,xn)=\begin{cases} f_{x1}(x_1,\ldots,x_n)+1 \text{ при } x_1=\ldots=x_n; \\ 0, \text{ в противном случае.} \end{cases}$$

Укажем процедуру вычисления g. Для любых значений x,y,z мы можем сначала провести операцию сравнения, и затем, если $x=y=z$, отыскать функцию f_x. Это вполне рекурсивная операция (сравнение аргументов). Если результат сравнения отрицательный (ложь), значение функции $g(x_1,\ldots,x_n)=0$. Если результат сравнения положительный (истина), далее можем применить к ней алгоритм вычисления функции $f_x(x,x,x)$ в заданной точке. Такой алгоритм существует в силу рекурсивности функций вида $f_i(x_1,\ldots,x_n)$. Прибавление к результату вычисления единички есть тривиальная рекурсивная функция. Т.о. видно, что эта диагональная функция будет тоже общерекурсивной.

Раз построенная функция принадлежит к множеству общерекурсивных функций, то она должна быть среди ранее эффективно перечисленных функций, но по построению она не может быть среди них, так как от каждой функции она отличается хотя бы в одной точке. Получили противоречие, следовательно, общерекурсивные функции нельзя эффективно перечислить,**Q.E.D.**

Т.4.8.(5)Теорема



Примитивно-рекурсивные функции не поддаются эффективному распознаванию среди общерекурсивных функций.

Доказательство

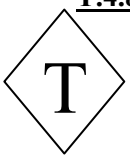
Возьмем общерекурсивную, но не примитивно-рекурсивную функцию *f(x)*. Построим *g(x)* следующим образом:

$$g(x) = \begin{cases} f(x), \text{ если машина } \boldsymbol{T} \text{ не остановится за первые } x+1 \text{ тактов;} \\ 0 \text{ во всех иных случаях.} \end{cases}$$

Функция *g(x)* тоже общерекурсивна (в силу того, что она везде определена и вычислима). Но на самом деле, если ***T*** все-таки остановится, то *g(x)* будет являться также примитивно-рекурсивной (потому что функция, отличная от нуля в конечном числе точек всегда примитивно-рекурсивна). Пусть мы умеем (знаем какой то алгоритм) распознавать примитивно-рекурсивные функции среди общерекурсивных функций. Применим этот алгоритм к общерекурсивной функции *g(x)*. Если удастся сказать, является ли она примитивно-рекурсивной, значит, окажется решенной и задача об остановке машины Тьюринга, что противоречит ранее доказанной теореме.

Значит, наше предположение не верно, следовательно, примитивно-рекурсивные функции не поддаются эффективному распознаванию среди всех общерекурсивных функций, **Q.E.D.**

Т.4.8.(6)Теорема



Общерекурсивные функции не поддаются эффективному распознаванию среди частично рекурсивных функций.

На уровне тезисов:

Понятие общерекурсивных функций совпадает с ВАФ, а понятие частично рекурсивной функции совпадает с ВЧАФ. А так

как ВАФ не распознается среди ВЧАФ, то общерекурсивные функции не поддаются эффективному распознаванию среди частично рекурсивных функций.

Доказательство

Множество частично рекурсивных функций – эффективно перечислимо, а значит, к нему и к его подмножеству, соответствующему классу общерекурсивных функций, можно применить теорему Поста**.** Предположим противное, а именно, что рекурсивные функции поддаются эффективному распознаванию среди частично рекурсивных функций. Это означает, что возможно также эффективно перечислить как общерекурсивные функции, так и частично рекурсивные, но не общерекурсивные функции. Насчет второго ничего неизвестно, а вот первое явно противоречит доказанной ранее теореме 4.8.(3). Следовательно, общерекурсивные функции не распознаваемы эффективно среди частично рекурсивных функций, **Q.E.D.**

§ 4.9. Нерекурсивные и непримитивно рекурсивные функции

> ***Нерекурсивной*** *называется функция, значение которой нельзя вычислить, несмотря на то, что в данной рассматриваемой точке сама функция определена*.

Например, зададим функцию *f(x)* следующим образом:

$$f(x)= \begin{cases} 1, \text{ если м.Тьюринга } T_x \text{ останавливается на чистой ленте;} \\ 0, \text{ если м.Тьюринга } T_x \text{ не останавливается на чистой ленте.} \end{cases}$$

Значения функции везде определены (0 или 1), но вычислены быть не могут, иначе это означало бы, что решена задача об остановке произвольной машины Тьюринга на чистой ленте, что противоречит теореме 1.5.(2).

> ***Непримитивно рекурсивной*** *называется функция, являющаяся общерекурсивной, но не принадлежащая множеству примитивно-рекурсивных функций*.

Одним из методов решения задачи о нахождении непримитивно рекурсивных функций может служить метод построения функции, растущей быстрее любой функции заданного класса. Этот метод очень удобен при исследовании сравнительной силы различного рода рекурсий. Изложим его применительно к уже рассмотренному классу примитивно рекурсивных функций.

Итак, нужно найти по возможности простые, но очень быстрорастущие функции. Опыт показывает, что произведение растет быстрее суммы, степень быстрее произведения. Называя сложение, умножение и возведение в степень действиями 0-й, 1-й и 2-й ступени и вводя для них в целях единообразия обозначения:

$P_0(a, x)=a+x, \quad P_1(a, x)=a\cdot x, \quad P_2(a, x)=a^x,$

приходим к знакомой всем идее о продолжении этой последовательности путем введения действий высших ступеней. При этом действие высшей ступени должно возникать из действия предыдущей ступени так же, как умножение возникает из сложения, а возведение в степень из умножения. Функции P_0, P_1, P_2 связаны следующими соотношениями:

$P_1(a, x+1)=P_0(a, P_1(a, x)), \quad P_1(a, 1)=a,$

$P_2(a, x+1)=P_1(a, P_2(a, x)), \quad P_2(a, 1)=a.$

Продолжим эту цепочку, полагая по определению для $n=2, 3,\ldots$

$P_{n+1}(a, 1)=a,$

$P_{n+1}(a, x+1)=P_n(a, P_{n+1}(a, x)).$

Чтобы функции $P_n(a, x)$ были определены всюду, положим $P_{n+1}(a, 0)=1$ при $n=1, 2,\ldots$ и указанные соотношения будем считать определениями функций $P_n(a, x)$ для $n=2, 3,\ldots$ Ясно, что необходимо добавить соотношение $P_1(a,0)=0.$ Тогда, например,

$P_3(a, 0)=1, \quad P_3(a, 1)=a, \quad P_3(a, 2)=a^a, \quad P_3(a, 3)=(a^a)^a.$

Введем новые функции $B(n, x)=P_n(2,x), \quad A(x)=B(x, x).$

Функцию $B(n, x)$ часто называют ***функцией Аккермана,*** а функцию $A(x)$ – *диагональной* функцией Аккермана.

Для функции $B(n, x)$ из указанных соотношений вытекают следующие тождества:

$$\begin{cases} B(n+1, x+1)=B(n, B(n+1, x)); \\ B(n+1, 0) = sg\, n; \\ B(0, x) = 2+x. \end{cases}$$

Подобный способ возникновения рекурсивной функции *B(n,x)* из примитивно-рекурсивных функций называется ***рекурсией второй ступени.***

§ 4.10. Границы применимости формальных моделей алгоритмов

Как показано в этой книге, машина Тьюринга, машина Поста, нормальные алгоритмы Маркова, рекурсивные функции являются разновидностями формализации понятий "вычисление" и "алгоритм". Все эти формализации эквивалентны друг другу, т.е. существуют стандартные алгоритмы, позволяющие программу для машины Тьюринга перевести в нормальный алгоритм или программу для машины Поста и т.д., и также возможен и обратный перевод. Любая функция, вычислимая по Тьюрингу, вычислима также посредством машины Поста, нормальных алгоритмов или рекурсивных функций.

Отсюда можно сделать вывод, что существует (потенциально бесконечный) класс "универсальных вычислительных машин", способных (в силу того, что каждая из них является адекватной формализацией понятия алгоритма) вычислить любую функцию, вычислимую в интуитивном смысле. Т.е. любая формализация алгоритма, принадлежащая к данному классу, позволяет адекватно представить любой вычислительный процесс (при условии, что этот процесс может быть представлен в виде ясной, четкой, однозначной инструкции, написанной, например, на естественном языке - т.е. если этот процесс можно представить как "алгоритмический" в интуитивном смысле этого слова). Утверждение о существовании класса универсальных вычислительных машин, способных вычислить все, что вычислимо в интуитивном смысле, известно как "тезис Черча".

Тезис Черча нередко рассматривают как важный аргумент в пользу возможности искусственного интеллекта. Действительно, из тезиса Черча вытекает, что все универсальные вычислительные устройства качественно эквивалентны друг другу. Иными словами, одна универсальная вычислительная машина не может быть качественно "умнее" другой - в том смысле, что задачи, принципиально неразрешимые для машины одного типа, будут

также неразрешимыми и для машин любых других типов. Различия между универсальными вычислительными машинами могут касаться лишь количественных параметров, а именно, они могут отличаться лишь по скорости вычислений и по объему памяти.

Однако эти тезисы, бесспорно, применимы при сравнении искусственных вычислительных машин. Когда речь заходит о нашем мозге, сознании или в глобальном смысле человеческих способностях, ситуация в корне меняется. Помимо философских аспектов в пользу доводов о возможности или невозможности создания интеллектуальных систем, сопоставимых с человеческим потенциалом, появляются и строго формалистические доводы на основании принятой аксиоматики.

Предположим, что математические способности некоторого математика, назовем его Человек_Разумный, полностью описываются некоторой формальной системой F. Это означает, что любое математическое утверждение, которое Человек_Разумный признает "неоспоримо верным", является теоремой, доказываемой в F, и наоборот. Предположим также, что Человек_Разумный знает, что F описывает его математические способности. Человек_Разумный, также полагает, что тот факт, что F описывает его математические способности, эквивалентен вере в непротиворечивость и непогрешимость F (в противном случае мы должны были бы поставить под сомнение истины, которые представляются нам "неоспоримо истинными").

Согласно теореме Геделя о неполноте формальных систем, поскольку F непротиворечива, существует геделевское предложение G(F), которое должно быть истинным, но которое не является теоремой в системе F. Однако, поскольку Человек_Разумный верит, что F - непротиворечивая система и знает, что F представляет его способность к математическим рассуждениям, он должен прийти к выводу, что G(F) является "неоспоримой истиной". Таким образом, мы получаем математическое утверждение G(F), которое Человек_Разумный признает истинным, но которое не является теоремой в F, что противоречит первоначальному предположению, что F представляет целиком и полностью математические способности Человека_Разумного.

Отсюда вывод, что никакая формальная система не может быть адекватным выражением математических способностей человека и, следовательно, невозможна полная компьютерная имитация человеческого сознания.

Однако эта точка зрения не является бесспорной и у неё довольно много авторитетных противников. Заинтересованный читатель может продолжить изучение этой темы в рамках физиологии мозга, философии бытия, молекулярной физики, квантовой механики и других направлений современных исследований, которые ставят своей целью ответ на простейший по своей формулировке вопрос А.Тьюринга: «А может ли машина МЫСЛИТЬ?».

Задачи к главе 4

Задача 4.1.

Доказать примитивную рекурсивность функции суммы (приведен пример решения для функции двух аргументов – рекурсия с параметрами):

$f(x, y) = x + y.$

$$\begin{cases} f(0, y) = 0 + y = y = U_1^1(y) = U_1^1; \\ f(x', y) = x + 1 + y = (x + y) + 1 = (x + y)' = [f(x, y)]' = S(f(x, y)) = \\ = S(U_1^3(f(x, y), x, y)) = S(U_1^3) = S_1^3(S, U_1^3); \end{cases}$$

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

Задача 4.2.

Доказать примитивную рекурсивность функции произведения двух аргументов:

$f(x,y)=П(x,y)=x \cdot y.$

Задача 4.3.

Доказать примитивную рекурсивность функции факториал (приведен пример решения для функции одного аргумента – рекурсия без параметров):

$$f(x) = x!;$$

$$\begin{cases} f(0) = 1; \\ f(x) = (x+1)! = x! \cdot (x+1) = f(x) \cdot (x+1) = f(x) \cdot S(x) = \\ = \Pi(f(x), S(x)) = \Pi(U_1^2, S(U_2^2)) = \Pi(U_1^2, S_1^2(S, U_2^2)) = \\ = S_2^2(\Pi, U_1^2, S_1^2(S, U_2^2)); \end{cases}$$

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

Задача 4.4.

Доказать примитивную рекурсивность функции псевдоразности:

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

Задача 4.5.

Доказать примитивную рекурсивность знаковой функции:

$$Sg(x) = \begin{cases} 0, \text{ если } x=0; \\ \\ 1, \text{ если } x>0. \end{cases}$$

Задача 4.6.

Доказать примитивную рекурсивность функции равенства:

$$Eql(x,y) = \begin{cases} 1, \text{ если } x = y; \\ \\ 0, \text{ иначе.} \end{cases}$$

Задача 4.7.

Доказать примитивную рекурсивность функции степени:
$f(x,y)=x^y$.

Задача 4.8.

Доказать примитивную рекурсивность функции модуля разности:

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

Задача 4.9.
Доказать примитивную рекурсивность функции больше:

$$More\ (x,y) = \begin{cases} 1, \text{ если } x>y; \\ 0, \text{ иначе.} \end{cases}$$

Задача 4.10.
Доказать примитивную рекурсивность функции больше или равно:

$$Moreql\ (x,y) = \begin{cases} 1, \text{ если } x\geq y; \\ 0, \text{ иначе.} \end{cases}$$

Задача 4.11.
Доказать примитивную рекурсивность функции остаток от деления аргумента x на 2:
$rest_2(x)$.

Задача 4.12.
Доказать примитивную рекурсивность функции минимум:

$$min(x,y) = \begin{cases} x, \text{если } x \leq y; \\ y, \text{в противном случае.} \end{cases}$$

Задача 4.13.
Доказать примитивную рекурсивность функции максимум:

$$max(x,y) = \begin{cases} x, \text{если } x \geq y; \\ y, \text{в противном случае.} \end{cases}$$

Задача 4.14.
Доказать примитивную рекурсивность функции целая часть от деления аргумента x на 2:
$div_2(x)$.

Задача 4.15.
Доказать примитивную рекурсивность функции целая часть от деления аргумента x на y.
$div\ (x,y)$

Задача 4.16.
Доказать примитивную рекурсивность функции остаток от деления аргумента *x* на *y:*
rest (x,y).

Задача 4.17.
Доказать примитивную рекурсивность функции число различных делителей *x* (включая число 1):
nd (x).

Задача 4.18.
Доказать примитивную рекурсивность функции *(n+1)*-е простое число в натуральном ряде чисел:
p(n).

Глава 5. Сложность алгоритмов

§ 5.1. Сравнение функций с точки зрения сложности

Введем понятие ***верхнего и нижнего порядка*** одной функции относительно другой функции.

> *Назовем арифметическую функцию f(x)* ***функцией одного верхнего порядка*** *с функцией g(x) и пишем f(x) = O(g(x)), если существует такое c > 0, что, в конце концов (т. е. начиная с некоторого x) получим* $f(x) \le c \cdot g(x)$.

Пример:
Рассмотрим две функции: $f_1(x) = 2x + 3$ и $f_2(x) = x^2$. Результаты расчетов значений функций при разных значениях аргумента представлены в табл. 5.1.

Таблица 5.1. Значения функций при разных значениях аргумента

x	2x+3	x^2
0	3	0
1	5	1
2	7	4
3	9	9
4	11	16

Таким образом, x^2 рано или поздно обгонит любую линейную функцию и можно записать $2x+3=O(x^2)$.

> *Назовем арифметическую функцию f(x) функцией* ***одного нижнего порядка*** *с функцией g(x) и пишем f(x)=o(g(x)), если существует такое c > 0, что в конце концов (начиная с некоторого x) получим* $f(x) \ge c \cdot g(x)$.

Пример: Рассмотрим две функции $f_1(x) = 2^x$ и $f_2(x) = x + 2$. Результаты расчетов значений функций при разных значениях аргумента представлены в табл. 5.2.

Таблица 5.2. Значения функций при разных значениях аргумента

x	2^x	x+2
0	1	2
1	2	3
2	4	4
3	8	5
4	16	6

Таким образом, 2^x рано или поздно станет больше любого полинома и можно записать 2^x=o(x+2). При малых размерах экспоненциальный алгоритм лучше полиномиального алгоритма

> *Назовем арифметическую функцию f(x) функцией* ***одного порядка с функцией*** *g(x), если она одного верхнего и одного нижнего порядка с функцией g(x), f(x) = o(g(x)) & f(x) = O(g(x)).*

> *Функции одного верхнего порядка с полиномиальными функциями называются* ***полиномиальными функциями.***

Это не только все полиномы, но и некоторые трансцендентные функции. Все остальные функции экспоненциальные в широком смысле этого слова.

> *В строгом смысле слова* ***экспоненциальными*** *называются функции одного нижнего порядка с экспонентой.*

Тогда функции между экспоненциальными и полиномиальными называются ***субэкспоненциальными функциями.*** Обычно их не рассматривают. Экспоненциальные функции разделяются по скорости еще на несколько классов.

> *Арифметические функции f(x) и g(x) называются* ***полиномиально-связанными*** *или* ***полиномиально-эквивалентнтными****, если существуют такие многочлены* $P_1(x)$ *и* $P_2(x)$*, что, в конце концов (начиная с некоторого числа)* $f(x) \le P_1 \cdot g(x)$ *и* $g(x) \le P_2 \cdot f(x)$.

Пример:
$f(x)=2x+3$, $p_1(x)=3x+7$, $p_2(x)=1+x^5$, $g(x)=x^3$.
$2x+3 \le (3x+7)\cdot x^3$ и $x^3 \le (1+x^5)\cdot(2x+3)$
=> f(x) и g(x) полиномиально связаны или эквивалентны.

§ 5.2. Полиномиальные и экспоненциальные алгоритмы

Временная сложность алгоритма *— это, в худшем случае, функция размера входных данных, которая показывает максимальное количество элементарных операций, затрачиваемые алгоритмом для решения экземпляра задачи указанного размера.*

В наилучшем случае - функция размера входных данных, которая показывает минимальное количество элементарных операций, затрачиваемые алгоритмом для решения экземпляра задачи указанного размера.

Пространственная сложность алгоритма *— это, в худшем случае, функция размера входных данных, которая показывает максимальное количество памяти, затрачиваемое алгоритмом для решения экземпляра задачи указанного размера.*

В наилучшем случае - функция размера входных данных, которая показывает минимальное количество памяти, затрачиваемое алгоритмом для решения экземпляра задачи указанного размера.

Теория сложности алгоритмов определяет общие типы алгоритмов, дифференцируя их по трудоемкости на два класса:

- полиномиальные алгоритмы;
- экспоненциальные алгоритмы.

Алгоритм, обе функции сложности которого полиномиальные, называется ***полиномиальным алгоритмом.***

Алгоритм, у которого хотя бы одна из двух функций сложности экспоненциальная, называется ***экспоненциальным алгоритмом.***

Задача, решаемая полиномиальным алгоритмом, называется ***легкоразрешимой*** *задачей.*

Задача, которую нельзя решить полиномиальным алгоритмом, называется ***трудноразрешимой.***

В число трудноразрешимых задач входят алгоритмически неразрешимые задачи. Неразрешимость есть крайний случай экспоненциальности.

§ 5.3. Временная и пространственная сложность машин Тьюринга

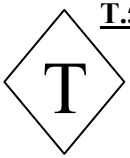
Временная сложность машины Тьюринга *T(x) – это число шагов, необходимых для решения данной задачи размера x в худшем случае.*

Пространственная сложность машины Тьюринга *S(x) – это число единиц памяти, необходимых для решения данной задачи размера x в худшем случае.*

Обычно не нужно точное определение функции сложности, а надо найти ее верхний или нижний порядок.

Для одноленточных машин Тьюринга временная сложность оценивается в тактах работы, а пространственная в количестве клеток, пройденных машиной в ходе работы. $S(n) \le T(n)$ (пространственная сложность всегда меньше или равна временной сложности алгоритма).

Для многоленточной машины Тьюринга временная сложность также определяется в тактах работы машины, но за один такт машина обрабатывает все свои ленты, поэтому $S(n)=O(T(n))$. Пространственная сложность имеет один верхний порядок с временной сложностью.



Т.5.3.(1)Теорема

Многоленточная машина полиномиально преобразуется в одноленточную машину.

Доказательство

k-ленточную машину T_k преобразуем в одноленточную как показано в Теореме 1.2. (1). Составим матрицу из 2k строк. Нечетные строки изображают ленты многоленточной машины. В четных строках изображается особой меткой положение машины T_k на соответствующей ленте. Укрупненный знак машины T состоит из 2k типичных простейших знаков. Все, что записано в одной вертикальной колонке, есть только один укрупненный знак машины T. Теперь смоделируем такт машины T_k на T. Предположим, что вначале моделируемая машина T находится не более чем на две клетки влево от самой левой метки положения, но не правее ее. Машина T движется вправо, считая все метки положения и запоминая соответствующие знаки лент. Меток - конечное число, следовательно, через конечное время машина T дойдет до последней метки и восстановит конфигурацию машины T_k. Таким образом, машина T через конечное время узнает конфигурацию машины T_k. Это произойдет за время T(n), равное сложности машины T_k. На ленте машины T будет записано не более, чем kT(n) символов. Конфигурация при этом будет определятся за O(T(n)) тактов.

Машина T моделирует ответ, она идет влево и обрабатывает каждую встреченную метку и соответствующим образом ее передвигает. В конце концов она дойдет до самой левой метки. В самой левой воспринимаемой клетке может быть сразу несколько меток. Одни пойдут вправо, другие влево. Тем самым мы вернемся в предполагаемое положение. Тактов столько же: O(T(n))+O(T(1))=O(T(n)), всего тактов: T(n)O(T(n))=O(T2(n)). Новая функция сложности полиномиально эквивалентна старой, **Q.E.D.**

Задачи к главе 5

Задача 5.1.
Найти пространственную и временную сложность машины Тьюринга, вычисляющей в унарном коде функцию $f(x)= 2x+3$.

Задача 5.2.
Найти пространственную и временную сложность машины Тьюринга, вычисляющей в унарном коде функцию $f(x)= x^2$.

Задача 5.3.
Найти пространственную и временную сложность машины Тьюринга, вычисляющей в унарном коде функцию $f(x)= 2^x$.

Задача 5.4.
Найти пространственную и временную сложность машины Тьюринга, вычисляющей в унарном коде функцию $f(x)= x \cdot y$.

Задача 5.5.
Найти пространственную и временную сложность машины Тьюринга, вычисляющей в унарном коде функцию $f(x)= x^y$.

Краткий справочник имен

Галилео ГАЛИЛЕЙ
Galileo Galilei, 1564 - 1642
Итальянский философ, математик, физик, механик и астроном, оказавший значительное влияние на науку своего времени. Галилей первым использовал телескоп для наблюдения планет и других небесных тел, а также сделал ряд выдающихся астрономических открытий. При жизни был известен как активный сторонник гелиоцентрической системы мира, что привело Галилея к серьёзному конфликту с католической церковью. Он сформулировал так называемый «парадокс Галилея»: натуральных чисел столько же, сколько их квадратов, хотя бóльшая часть чисел не являются квадратами. Это подтолкнуло научное сообщество в дальнейшем к исследованию природы бесконечных множеств и их классификации. Процесс завершился созданием теории множеств. Галилео Галилей умер 8 января 1642 года, в возрасте 78 лет, в своей постели. Похоронили его в Арчетри без каких-либо почестей.

Курт ГЁДЕЛЬ
Kurt Gödel, 1906 - 1978
Австрийский, затем американский математик. В 1931 году опубликовал теорему, получившую впоследствии его имя. В 1930-е годы эмигрировал в США, но затем вернулся в родную Австрию и женился. В 1940 году, в разгар войны, вынужденно бежал в Америку транзитом через СССР и Японию. Некоторое время проработал в Принстонском институте перспективных исследований. К сожалению, психика ученого не выдержала, и он умер в психиатрической клинике от голода, отказываясь принимать пищу, поскольку был убежден, что его намереваются отравить.

Давид ГИЛЬБЕРТ
David Hilbert, 1862 - 1943
Немецкий математик. Внёс значительный вклад в развитие многих математических разделов. В 1910-х годах Гильберт создаёт в современном виде функциональный анализ, введя понятие, получившее название гильбертова пространства. В 1910—1920-е

годы (после смерти Анри Пуанкаре) был признанным мировым лидером математиков. Умер Гильберт 14 февраля в военном 1943 году в Гёттингене. За его гробом шло всего около десятка человек.

Рихард Юлиус Вильгельм ДЕДЕКИНД
Julius Wilhelm Richard Dedekind, 1831 - 1916
Немецкий математик, член Берлинской АН. Учился у Гаусса и Дирихле в Гёттингенском университете. Основные работы Дедекинда относятся к теории алгебраических чисел. Обобщив теорию многочленов и алгебраических чисел, он ввел в математику абстрактные алгебраические структуры: кольца, идеалы и модули. Совместно с Кронекером он создал общую теорию делимости. Дедекинд известен также как автор одной из первых систем строгого обоснования теории действительных чисел.

ЕВКЛИД
Ενκλειδηζ
Древнегреческий математик. Его научная деятельность протекала в Александрии в III веке до н. э. Евклид – первый математик александрийской школы. Его главная работа «Начала» (в латинизированной форме – «Элементы») содержит изложение планиметрии, стереометрии и ряда вопросов теории чисел. В ней он подвел итог предшествующему развитию греческой математики и создал фундамент дальнейшего развития математики. Из других сочинений по математике надо отметить работу «О делении фигур». Евклид – автор работ по астрономии, оптике, музыке и др.

Георг КАНТОР
Georg Cantor, 1845 - 1918
Немецкий математик. Разработал теорию бесконечных множеств и теорию трансфинитных чисел. Доказал несчетность множества всех действительных чисел. Сформулировал общее понятие мощности множества. Развил принципы сравнения мощности множества и доказал эквивалентность множества точек линейного отрезка и точек n-мерного многообразия. Ввел понятие предельной точки производного множества, развил одну из теорий иррациональных чисел, сформулировал аксиому непрерывности,

названную его именем, получил результаты по проблеме единственности тригонометрических рядов. Созданная Кантором теория множеств не только лежит ныне в основе математического анализа, но и послужила причиной общего пересмотра логических основ математики и оказала влияние на всю современную структуру математики.

Стивен Коул КЛИНИ
Stephen Cole Kleene, 1909 - 1994
Американский логик и математик. Его работы совместно с работами Алонзо Чёрча, Курта Гёделя и Алана Тьюринга дали начало разделу математической логики — теории вычислимости. Кроме того, известен изобретением регулярных выражений. Его именем названы Алгебра Клини, Звёздочка Клини, теорема Клини о рекурсии, теорема Клини о неподвижной точке. Работал также в области интуиционистсткой математики Брауэра. Внёс важный вклад в теорию конечных автоматов.

Пол Джозеф КОЭН
Paul Joseph Cohen, 1934 - 2007
Американский математик. Вершиной профессиональной деятельности Коэна в области теории множеств его коллеги называют опубликованное в 1963 году доказательство невозможности подтверждения так называемой континуум-гипотезы (доказательство невозможности опровергнуть которую было сделано в 1940 году Куртом Гёделем). В 1964 году ученый был удостоен премии Бохера за достижения в области анализа, а за изучение логики в 1966 году получил медаль Филдса (эквивалент Нобелевской премии). В следующем году исследования в области логики принесли ему также Национальную медаль за научные достижения. Коэн не оставлял преподавания до последних месяцев жизни.

Готфрид Вильгельм фон ЛЕЙБНИЦ
Gottfried Wilhelm von Leibniz, 1646 - 1716
Немецкий (саксонский) философ и математик. Опередив время на два века, 20-летний Лейбниц задумал проект математизации

логики. В это время Лейбниц изобретает собственную конструкцию арифмометра, гораздо лучше паскалевского — он умел выполнять умножение, деление и извлечение корней. Необходимо отметить два важнейших математических достижения Лейбница: создание (наряду с Ньютоном) математического анализа (дифференциального и интегрального исчисления) и создание комбинаторики как науки. Он единственный во всей истории математики, кто одинаково свободно работал как с непрерывным, так и с дискретным величинами. Лейбниц также описал двоичную систему счисления с цифрами 0 и 1, на которой основана современная компьютерная техника.

Карл Луи Фердинанд фон ЛИНДЕМАН
Ferdinand von Lindemann, 1852-1939
Немецкий математик. Научные работы Линдемана охватывают многие области математики. Главной сферой его научных интересов была геометрия, особенно дифференциальная геометрия. Определенную значимость получили также работы Линдемана, связанные с теоретической физикой, теорией линий спектра. В 1882 году он доказал, что число π трансцендентно.

Жозеф ЛИУВИЛЛЬ
Joseph Liouville, 1809 – 1882
Французский математик, член Парижской АН, иностранный почетный член Петербургской АН. В 1836 году основал "Журнал чистой и прикладной математики". Научные интересы Лиувилля были очень широкими. Он построил теорию эллиптических функций, которые рассматривал как двояко периодические функции комплексной переменной. На основе общей теории аналитических функций, Лиувилль исследовал краевую задачу для линейных дифференциальных уравнений 2-го порядка. Ему принадлежит фундаментальная теорема статистической механики, а также теорема об интегрировании канонических уравнений динамики. В теории чисел особенно важны результаты исследований Лиувилля, касающиеся рациональных приближений алгебраических чисел. В 1851 году Лиувилль доказал

существование трансцендентных чисел, именно он впервые построил конкретные классы трансцендентных чисел.

Николай Иванович ЛОБАЧЕВСКИЙ
1792 – 1856
Русский математик. Создатель геометрии Лобачевского, деятель университетского образования и народного просвещения. Известный английский математик Уильям Клиффорд назвал Лобачевского «Коперником геометрии».

Андрей Андреевич МАРКОВ
1903-1979
Ученая степень доктора физико-математических наук присвоена без защиты диссертации в 1935 году. Основные труды по теории динамических систем, топологической алгебре, теории алгоритмов и конструктивной математике. Доказал неразрешимость проблемы равенства в ассоциативных системах, создал школу конструктивной математики и логики в СССР, автор понятия нормального алгорифма.

Джузеппе ПЕАНО
Giuseppe Peano, 1858 – 1932
Итальянский математик. Внёс существенный вклад в математическую логику, аксиоматику, философию математики. Создатель вспомогательного искусственного языка латино-сине-флексионе. Более всего известен как автор стандартной аксиоматизации натуральной арифметики — арифметики Пеано.

Роджер ПЕНРОУЗ
Roger Penrose, 1931
Английский математик. Направления исследований Роджера Пенроуза включают в себя многие аспекты геометрии, теорию непериодических мозаик, общую теорию относительности и основы квантовой теории. Пенроуз получил множество наград за свой вклад в науку. Получил приз Science Book Prize в 1990 году за книгу "Новый ум короля".

Гелий Николаевич ПОВАРОВ

1928 – 2004

Русский математик, кибернетик, логик, философ, автор теории кумулятивных сетей. Родился 2 февраля 1928 года в Москве. В 1950 году с отличием закончил механико-математический факультет Московского государственного университета им. М. В. Ломоносова. После окончания университета был призван на действительную военную службу, одновременно учился в заочной аспирантуре Института автоматики и телемеханики АН СССР. После демобилизации, с 1953 по 1960 годы работал в системе АН СССР. В 1965 году перешел на работу в Московский инженерно-физический институт (государственный университет), где с 1967 года работал на кафедре кибернетики, профессор с 1994 года. Поваров Г.Н. внес значительный вклад в развитие и пропаганду советской и российской кибернетики, философию и методологию этой науки. Под его редакцией вышли важные труды по кибернетике и теории систем. Значительный период его научной деятельности был связан с разработкой методов синтеза управляющих контактных схем, результатом чего стало создание математической теории синтеза контактных схем с одним входом и несколькими выходами. Продолжением исследований в области сетей связи стала разработка теории кумулятивных сетей. На протяжении всей жизни Г.Н. Поваров интересовался вопросами математической логики и исследованием булевых функций, методами их сравнения и минимизации. Предложил новую концепцию событийной логики. Гелий Николаевич был полиглот, владел большим количеством европейских языков. В 1993 году избран действительным членом Международной Академии Информатизации при ООН. До самого последнего он дня не прекращал работы, читал лекции, вел исследования по истории кибернетики и вычислительной техники. Скончался 16 ноября 2004 года в возрасте 76 лет.

Эмиль Леон ПОСТ

Post Emil Leon, 1897 - 1954

Американский математик и логик. Им получен ряд фундаментальных результатов в математической логике, в том

числе одно из наиболее употребительных определений понятий непротиворечивости и полноты формальных систем (исчислений), а также доказательство функциональной полноты и дедуктивной полноты (в широком и узком смысле) исчисления высказываний. Пост дал одно из первых (независимое от А.М. Тьюринга) определений понятия алгоритма в терминах «абстрактной вычислительной машины» и сформулировал основной тезис теории алгоритмов о возможности описать любой конкретный алгоритм посредством этого определения. Посту принадлежат доказательства выразимости общерекурсивных функций и предикатов через примитивно рекурсивные функции, в частности так называемая теорема о нормальной форме. Он же автор первых (одновременно с А.А. Марковым) доказательств алгоритмической неразрешимости ряда проблем математической логики и алгебры.

Бертран Артур Уильям РАССЕЛ
Bertrand Arthur William Russell, 1872 – 1970
Английский математик, философ и общественный деятель. Создал концепцию «логического атомизма» и разработал теорию дескрипций. Существенные результаты были получены Расселом в области символической логики и ее применения к философским и математическим проблемам. Профессор Рассел является автором множества работ в области математической логики. Важнейшей является написанная в 1913 году в соавторстве с А. Уайтхедом работа «Принципы математики», которая доказывает соответствие принципов математики принципам логики и возможность определения основных понятий математики в терминах логики. Почти до самой смерти 2 февраля 1970 года 98-летний ученый оставался энергичным и деятельным.

Фридрих Людвиг Готлоб ФРЕГЕ
Friedrich Ludwig Gottlob Frege, 1848 - 1925
Немецкий логик, математик и философ. Представитель школы аналитической философии. Сформулировал идею логицизма, то есть направление в основаниях математики и философии математики, основным тезисом которого является утверждение о «сводимости математики к логике». Фреге является основателем

современной логической семантики. Умер Фреге в Бад-Кляйнене 26 июля 1925 года.

Эрнст Фридрих Фердинанд ЦЕРМЕЛО
Ernst Friedrich Ferdinand Zermelo, 1871 – 1953
Немецкий математик. Основные исследования относятся к теории множеств, где он дал общую аксиоматику и доказал, что всякое множество может быть вполне упорядочено. Работы Цермело оказали большое влияние на развитие этого раздела математики и вызвали оживлённую дискуссию. Занимался также вариационным исчислением и вопросами приложения теории вероятностей к статистической физике.

Алонзо ЧЁРЧ
Alonzo Church, 1903 - 1995
Американский математик и логик, внесший вклад в основы информатики. Чёрч внёс существенный вклад в развитие комбинаторной логики: ему принадлежат исследования в области логической семантики и модальной логики. Кроме того, Чёрч прославился разработкой теории лямбда-исчислений.

Клод Элвуд ШЕННОН
Claude Elwood Shannon, 1916 - 2001
Американский математик и электротехник. Является одним из создателей математической теории информации, в значительной мере предопределил своими результатами развитие общей теории дискретных автоматов, которые являются важными составляющими кибернетики. Клод Шеннон умер 24 февраля 2001 года в возрасте 84 лет после многолетней борьбы с болезнью Альцгеймера.

Список литературы

1. Мальцев А.И. Алгоритмы и рекурсивные функции. М.: Наука, 1965, 1986.
2. М.Минский. Вычисления и автоматы. М.: Мир, 1971
3. Ахо А. Построение и анализ вычислительных алгоритмов. Пер. с англ. М.: Мир, 1979.
4. Ахо А. Хопкрофт Д. Структуры данных и алгоритмы. Пер. с англ. М.: Издательский дом «Вильямс», 2000.
5. Яблонский С.В. Введение в дискретную математику. М.: Наука, 1979; 1996.
6. ПападимитриуХ., Стайглиц К. Комбинаторная оптимизация. Алгоритмы и сложность. Пер. с англ. М.: Мир, 1985.
7. Лавров И.А. ,Максимова Л.Л. Задачи по теории множеств, математической логике и теории алгоритмов. М.: Наука, 1975, 1984.
8. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982.
9. Гашков С.Б., Чубариков В.Н. Аримфетика. Алгоритмы. Сложность вычислений. М.: Высшая школа, 2000.
10. Гуц А.К. Кардинальные и трансфинитные числа: Учебное пособие. Омск: Омск.университет, 1995.
11. Хаусдорф Ф. Теория множеств. Пер. с нем. М.: КомКнига, 2006.
12. Бурова И.Н. Парадоксы теории множеств и диалектика. М.: Наука, 1976.
13. Виленкин Н.Я. Рассказы о множествах. М.: Наука, 1969.
14. Коэн П.Дж. Теория множеств и континуум-гипотеза. Пер. с англ. М.: Мир, 1969.
15. Успенский В.А., А.Л. Семенов. Теория алгоритмов: основные открытия и приложения. М.: Наука, 1987.
16. Бурбаки Н. Теория множеств. Пер. с франц. М.: Мир, 1965.
17. Горбатов В.А. Фундаментальные основы дискретной математики. М.: Наука, 2000.

18. Горбатов В.А., Горбатов А.В., Горбатова М.В. Дискретная математика: учебник для студентов ВТУЗов. М.: АСТ: Астрель, 2006.
19. Кантор Г. Труды по теории множеств. М.: Наука, 1985.
20. Эщби У.Р. Введение в кибернетику. Пер. с англ. М.: КомКнига, 2005.
21. Фалевич Б.Я. Теория алгоритмов: Учебное пособие. М.: Машиностроение, 2004.
22. Пенроуз Р. Новый ум короля: о компьютерах, мышлении и законах физики. Пер. с англ. М.: Едиториал УРСС, 2005.
23. Пойя Д. Математика и правдоподобные рассуждения. Пер. с англ. М.: Издательство иностранной литературы, 1957.
24. Френкель А.А., Бар-Хиллел И.Основания теории множеств. Пер. с англ. М.: КомКнига, 2006.
25. Гельфонд А.О., Трансцендентные и алгебраические числа. М.: КомКнига, 2006.
26. Марков А.А., Нагорный Н.М. Теория алгорифмов. М.: ФАЗИС, 1996.

Анна Николаевна Тихомирова

ТЕОРИЯ АЛГОРИТМОВ

Редактор С.В. Бирюкова

Подписано в печать 10.12.08. Формат 60x84 1/16
Объем 11,0 п.л. Уч.-изд.л. 11,0. Тираж 150 экз.
Изд. № 1/37 Заказ №

Московский инженерно-физический институт
(государственный университет)
115409 Москва, Каширское ш., 31

Типография издательства «Тровант»
г. Троицк Московской области