Невычислимость, неразрешимость, недоказуемость

Сосинский Алексей Брониславович

Дубна, 2001 г.

Занятие №1. 16 июля 2001 г.

Сегодня речь пойдёт о *теории алгоритмов*. Её можно считать частью *теоретической информатики* (по-английски — theoretical computer science). Этой наукой занимается особая категория математиков, большую часть которых составляют математические логики. Другую часть работающих в ней людей составляют теоретические программисты, т. е. люди, не занимающиеся собственно программированием, а скорее теорией программирования.

Теория алгоритмов занимается алгоритмами. Поэтому, естественно, сначала нужно объяснить, что такое алгоритм. Мы не будем сразу давать формального определения алгоритма, а ограничимся интуитивными пояснениями.

Замечание. На самом деле, не удаётся придумать строгое определение алгоритма, соответствующее нашему интуитивному пониманию, хотя такое определение и пытались дать такие великие математики, как, например, Андрей Николаевич Колмогоров.

Наше лапидарное пояснение таково: *алгоритм* — это детерминированное устройство, которое дискретно перерабатывает информацию.

Как это нужно понимать? Можно представлять алгоритм в виде ящика, которому на вход подаётся *информация*, т.е. слово в некотором алфавите. *Алфавит* $A = \{a_1, a_2, \ldots, a_n\}$ — это непустой набор символов, называемых *буквами*. *Словом* называется любая конечная последовательность таких букв. В том числе рассматривается и *пустое слово*, т. е. слово, не содержащее ни одной буквы. Множество всех слов в алфавите A будем обозначать через W_A. Таким образом, информация всегда конечна. Итак, на вход в начальный момент времени поступает некоторое слово w_0.

Дискретность означает, что алгоритм работает пошагово. Можно, например, считать, что алгоритм совершает действия каждую секунду. В каждый момент времени (после каждого шага) алгоритм находится в некотором *состоянии*, число возможных состояний конечно. В начальный момент алгоритм находится в специальном *начальном состоянии* s_0.

То, какое действие он совершит на i-м шаге, полностью определяется текущей информацией w_{i-1} и текущим состоянием алгоритма s_{i-1}. В этом проявляется *детерминированность*. А действие, совершаемое алгоритмом, состоит в некотором изменении информации: слово w_{i-1} превращается в слово w_i, и, кроме того, алгоритм переходит в новое состояние s_i (быть может совпадающее с s_{i-1}). При этом, как уже было сказано, если фиксированы w_{i-1} и s_{i-1}, то однозначно определены s_i и w_i.

Возможно, после некоторого n-го шага алгоритм останавливается (заканчивает работу). В таком случае *результатом* его работы будет слово w_n. Возможен также случай, когда алгоритм продолжает работать и никогда не останавливается, тогда говорят, что алгоритм *работает вечно* на слове w_0.

Основная неприятность теории алгоритмов состоит в следующем: допустим, у нас есть некоторый алгоритм, мы запустили его и ждём результата. Вот он проработал час, а результата нет, два часа — результата по-прежнему нет. Вопрос: до каких пор мы должны ждать? Можем ли мы в какой-то момент с уверенностью сказать, что если алгоритм работает слишком долго, значит, он будет работать вечно? Оказывается, что нет. Эта неприятность носит фундаментальный характер и отделаться от неё невозможно. Более точно этот факт будет сформулирован несколько позднее.

Машины Тьюринга

Рассмотрим теперь тип алгоритмов, который будет для нас основным, а именно машины Тьюринга. Он был изобретён британским математиком, работавшим в английском шпионском ведомстве, Аланом Тьюрингом.

Машину Тьюринга можно представлять в виде устройства, состоящего из экрана и арифметического модуля. Экран показывает бесконечную в обе стороны *ленту*, поделённую на *ячейки* (можно считать, что она конечна, но, когда мы подходим к её краю, к ленте подклеивается ещё один кусок). В каждой ячейке ленты написана буква некоторого фиксированного алфавита $A = \{a_1, a_2, \ldots, a_n\}$, называемого *ленточным алфавитом*. На экране видно содержимое ровно одной ячейки. Арифметический модуль в каждый момент времени находится в определённом *состоянии*. Поэтому кроме ленточного алфавита имеется ещё *алфавит состояний* $S = \{s_0, s_1, s_2, \ldots, s_m, \times\}$, в котором имеется два выделенных состояния: s_0 — начальное и $\times$ — конечное, *останов*. На корпусе машины Тьюринга есть электрическая лампочка; когда машина приходит в состояние $\times$, она останавливается и зажигается лампочка. Это означает, что машина закончила работу.

Объясним теперь, что такое вход и выход машины Тьюринга. *Входом* называется то, что написано на ленте до начала работы, плюс её расположение относительно экрана. Расположение относительно экрана важно, потому что машина Тьюринга видит не всё слово, написанное на ленте, а максимум одну его букву. Входное слово должно быть конечно. Все остальные ячейки пусты, мы будем обозначать это при помощи пустого символа $\square$. Как уже было сказано, в начальный момент времени машина находится в состоянии s_0. Затем машина начинает работать. Если она останавливается (приходит в состояние $\times$), то слово, написанное на ленте в этот момент, и есть *выход* или *результат* работы. Нетрудно понять, что это слово также конечно.

Осталось описать, как машина работает. К машине прилагается программа, состоящая из конечного числа команд следующего вида:

$$s_i a_j \rightsquigarrow a_k d s_m.$$

Означает эта запись следующее. Каждый раз, когда машина находится в состоянии s_i и видит на экране букву a_j, она пишет в этой ячейке букву a_k (это может быть и та же самая буква), переходит в состояние s_m и лента совершает *движение* d, где $d \in D = \{\leftarrow, \rightarrow, -\}$, а символы $\leftarrow, \rightarrow, -$ соответствуют сдвигу в левую ячейку, в правую ячейку и отсутствию сдвига. Мы будем считать, что для каждой комбинации буквы a_j и состояния s_i (кроме, конечно, состояния останова $\times$) в программе присутствует команда, говорящая, как машина должна поступить, оказавшись в такой ситуации. При этом, в отличие от реальных компьютеров, позволяющих писать разные программы, здесь сама машина Тьюринга уже однозначно определяет свою программу.

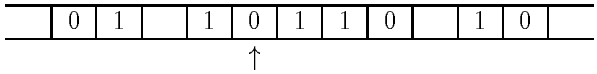
Вообще говоря, машины Тьюринга можно определить и несколько иначе. Рассматриваемые нами машины представляют собой лишь один из возможных типов. Бывают, например, машины Тьюринга с многими лентами, с односторонними лентами, машины, у которых лента движется только в одну сторону и т. д.

Примеры машин Тьюринга

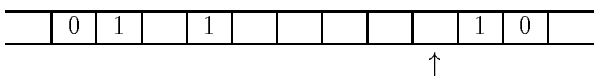
Рассмотрим несколько примеров программ для машин Тьюринга. Во всех этих примерах алфавит будет состоять всего лишь из двух букв: $A = \{0, 1\}$.

1) $s_0\ 0 \rightsquigarrow \square \rightarrow s_0$
   $s_0\ 1 \rightsquigarrow \square \rightarrow s_0$
   $s_0 \square \rightsquigarrow \square - \times$

Попробуем понять, как она работает. Для этого нам придётся как-то изображать её на бумаге. Так как рисовать каждый раз экран, лампочку и т. д. неудобно, мы будем изображать машину Тьюринга схематически, а именно, мы будем рисовать ленту с буквами, записанными в ячейках, а букву, которая в данный момент находится на экране, будем помечать стрелочкой. При этом опять-таки удобнее будет говорить, что это стрелочка двигается по ленте, а не лента протягивается через экран. Поэтому экран мы будем теперь называть *головкой* (по аналогии с воспроизводящей головкой магнитофона). Пусть начальное состояние машины таково (мы не пишем $\square$ в пустых ячейках):



Тогда машина остановится через пять шагов и лента будет выглядеть следующим образом:



Естественно назвать эту машину «правым ластиком». Он стирает все символы вправо от начального положения головки до первой пустой клетки. (Она обязательно найдётся, поскольку входное

слово конечно.) Нетрудно придумать другую машину Тьюринга — «абсолютный правый ластик», которая бы стирала все символы правее начального положения, но, к сожалению, она никогда не остановится. Алфавит состояний «правого ластика» состоит всего лишь из двух символов: s_0 и $\times$. В дальнейшем мы не будем указывать алфавит состояний явно, так как он однозначно определяется из программы (в программе присутствуют все возможные состояния, как и все возможные буквы). Можно также придумать машину Тьюринга, которая будет стирать в обе стороны от начального положения головки до первой пустой ячейки с каждой стороны.

$s_0\square \rightsquigarrow 1 - \times$

2) $s_0\ 1 \rightsquigarrow 0{\leftarrow}s_0$

$s_0\ 0 \rightsquigarrow 1 - \times$

Если в начале работы этой машины Тьюринга головка смотрит на самую правую цифру некоторого двоичного числа, то результатом работы будет прибавление к этому числу 1 (предполагается, что на ленте написано лишь это число). Можно назвать эту машину «счётчиком».

Задачи

Придумайте машину Тьюринга, которая

(1) вычитает единицу из двоичного числа;

(2) складывает два двоичных числа: $\overline{c_1 \ldots c_m}$ и $\overline{e_1 \ldots e_k}$, — записанных на ленте следующим образом:

	e_1	$\ldots$	e_k	$\star$	c_m	$\ldots$	c_1	
				$\uparrow$				

в алфавит в этой задаче добавлена специальная буква $\star$, выполняющая роль метки, разделяющей два числа; правое число написано наоборот для удобства, тем не менее, результат нужно выдать в виде обычной двоичной записи;

(3) переворачивает двоичное слово, т. е. на входном слове $\overline{c_1 \ldots c_m}$ выдаёт $\overline{c_m \ldots c_1}$;

(4) вставляет пустую ячейку справа от начального положения головки;

(5) вырезает ячейку, на которой головка стоит в начале.

Указание: задачи (4) и (5) могут быть использованы при решении остальных задач, поэтому имеет смысл сначала решить именно эти две задачи.

Замечание. В некоторых задачах условия работы алгоритма заданы не полностью. Например, не говорится, как должна работать машина из задачи (1) на входе 0 и т. п. Это означает, что все эти детали предоставляются на усмотрение решающего.

Тезис Чёрча

Тезис Чёрча был придуман американским математическим логиком Алонзо Чёрчем, и состоит он в следующем:

Любой алгоритм может быть реализован при помощи машины Тьюринга.

Это не теорема и не аксиома. Тезис Чёрча не может быть доказан просто потому, что у нас нет строгого определения алгоритма. Если рассмотреть алгоритмы какого-нибудь другого типа, описанного столь же полно и подробно, как машины Тьюринга, то для них тезис Чёрча превращается в теорему, которую можно строго доказать. Но так как понятие алгоритма не имеет определения, тезис Чёрча остаётся как бы над математикой, его можно воспринимать как закон природы.

Основные теоремы

Будем называть машины Тьюринга, которые мы рассматривали до сих пор, машинами Тьюринга от одной переменной, их множество обозначим через $\mathcal{T}^1$. Определим теперь множество машин Тьюринга от двух переменных $\mathcal{T}^2$.

Машиной Тьюринга T' от двух переменных называется обычная машина Тьюринга, вход которой (слово в алфавите A) имеет вид px, где p и x — слова в алфавите A, головка смотрит на первую букву слова x, *результатом* (выходом) машины называется слово, стоящее непосредственно справа от головки после остановки машины. Слово p называется *инструкцией*, а x — *исходным данным*.

Фактически машина от двух переменных и машина от одной переменной отличаются друг от друга только способом интерпретации входа и выхода.

Теорема 1 (основная положительная теорема). *Существует такая машина Тьюринга U от двух переменных, называемая универсальной машиной Тьюринга, что для любой машины Тьюринга T от одной переменной существует такая инструкция p_T, что для любого слова x результат,*

получаемый машиной T на входе x, совпадает с результатом работы машины U на входе с инструкцией p_T и исходным данным x, т. е. машина U на входе $p_T x$ останавливается, если останавливается машина T на входе x, и даёт тот же выход, а если T не останавливается, то и U работает вечно.

Эту формулировку можно кратко записать так:

$$\exists U \in \mathcal{T}^2 \; \forall T \in \mathcal{T}^1 \; \exists p_T \in W_A \; \forall x \in W_A \; T(x) \doteq U(p_T, x).$$

Здесь значок $\doteq$ означает равенство выходов в случае остановки и вечную работу обеих машин в противном случае.

Эту теорему мы приводим без доказательства, а лишь вкратце набросаем идею того, как должна работать универсальная машина. Она должна состоять из подпрограммы, которая умеет имитировать один шаг работы машины Тьюринга от одной переменной. Для этого она будет читать очередную команду из инструкции (а инструкция должна некоторым удобным образом кодировать программу машины T), запоминать, какие действия должна была бы выполнить машина T, затем идти в область исходного данного x (уже как-то изменённого) и выполнять эти действия. Сложность здесь состоит в том, что машина Тьюринга в каждый момент времени видит лишь одну букву, поэтому реализовать, например, запоминание команды на ней уже не так просто. Эта подпрограмма запускается в цикле, признаком выхода из которого будет переход машины T в состояние останова.

Введём два новых класса машин Тьюринга. В первый из них включим все машины Тьюринга от одной переменной, которые останавливаются на всех возможных входах, его мы обозначим $\mathcal{T}_0^1$. Второй класс, обозначаемый $\mathcal{T}_0^2$, также будет состоять из всегда останавливающихся машин, но уже от двух переменных.

Оказывается, что факт, аналогичный основной положительной теореме, для всегда останавливающихся машин неверен.

Теорема 2 (основная отрицательная теорема). *Не существует всегда останавливающейся машины Тьюринга от двух переменных U_0, такой что для каждой всегда останавливающейся машины Тьюринга T_0 от одной переменной можно найти некоторую инструкцию p_{T_0}, так что для каждого слова x результат работы машины T_0 на входе x будет совпадать с результатом работы машины U_0 на входе с инструкцией p_{T_0} и исходным данным x.*

$$\nexists U_0 \in \mathcal{T}_0^2 \; \forall T_0 \in \mathcal{T}_0^1 \; \exists p_{T_0} \in W_A \; \forall x \in W_A \; T_0(x) = U_0(p_{T_0}, x).$$

Доказательство основной отрицательной теоремы. Сначала объясним, почему попытка предложить в качестве U_0 универсальную машину U из основной положительной теоремы не даёт желаемого результата: U_0 должна *всегда* останавливаться, а не только на входах, содержащих инструкции всегда останавливающихся машин Тьюринга от одной переменной.

Для доказательства мы используем тот же метод, что и при доказательстве существования несчётного множества. Доказываем от противного. Предположим, требуемая машина U_0 существует. Тогда мы построим всегда останавливающуюся машину Тьюринга от одной переменной T_0, такую что не существует слова p_{T_0}, такого что для любого слова x совпадают результаты работы машин U_0 и T_0 на соответствующих входах: $U_0(p_{T_0}, x) = T_0(x)$. Машина Тьюринга T_0, получив на вход слово x, будет дублировать его ещё раз справа от его исходного положения, устанавливать головку на первую слева букву этого дубликата, после чего запускать машину Тьюринга U_0. После того как U_0 закончит работу (а она всегда останавливается), T_0 будет прибавлять к её выходу единицу (для простоты мы считаем, что машины работают в алфавите $\{0, 1\}$):

$$T_0(x) := U_0(x, x) + 1.$$

Значок $:=$ здесь обозначает «по определению». Машина T_0, очевидно, останавливается на всех входах. Возьмём произвольное слово p. Если бы оно подходило в качестве требуемой инструкции p_{T_0}, то $U_0(p, p) = T_0(p)$ по определению универсальной машины, но, с другой стороны, $T_0(p) = U_0(p, p)+1$ по определению машины T_0. Полученное противоречие доказывает теорему. □

Занятие №2. 19 июля 2001 г.

Кодирование

Часто оказывается, что, для того чтобы применить к исходным данным некоторый алгоритм, в частности машину Тьюринга, приходится привести их к виду, приемлемому для алгоритма, т. е. «закодировать». Например, любая текстовая информация в компьютерах на самом деле закодирована нулями и единицами. Мы будем кодировать слова одного алфавита в другом, т. е. каждому слову одного алфавита мы будем сопоставлять некоторое слово второго, требуя при этом, чтобы разным словам первого алфавита соответствовали разные слова второго, именно это и даёт нам возможность произвести обратную кодировку, расшифровку.

Сначала попробуем закодировать слова алфавита $\{a_1, \ldots, a_k\}$ в алфавите $\{0, 1\}$. Вероятно, самый естественный способ таков: сопоставим слову $a_{i_1}a_{i_2}\ldots a_{i_m}$ следующую последовательность:

$$0\underbrace{11\ldots1}_{i_1}0\underbrace{11\ldots1}_{i_2}0\ldots0\underbrace{11\ldots1}_{i_m}0.$$

Пусть теперь слова того же алфавита нужно закодировать в однобуквенном алфавите $\{1\}$. Очень красивое решение этой задачи придумал великий немецкий, а впоследствии американский математический логик Курт Гёдель, тот самый, которому принадлежат теоремы Гёделя о неполноте. Каждое слово должно кодироваться определённым количеством единиц. Гёдель предложил слово $a_{i_1}a_{i_2}\ldots a_{i_m}$ кодировать последовательностью из $2^{i_1}\cdot 3^{i_2}\cdot\ldots\cdot p_m^{i_m}$ единиц, где основания степеней: $2, 3, 5, \ldots, p_m$ — представляют собой расположенные по возрастанию первые m простых чисел. Пустое слово при этом кодируется одной единицей. Из теоремы о единственности разложения натурального числа на простые сомножители вытекает возможность декодирования, таким образом, это действительно кодировка.

Поставим, наконец, эту задачу в общем виде: закодировать алфавит $A = \{a_1, \ldots, a_k\}$ в алфавите $B = \{b_1, \ldots, b_m\}$. Решив эту задачу, мы, естественно, получим новые решения обеих предыдущих задач. Все слова в алфавите можно упорядочить. Например, в словарях используется так называемый *лексикографический порядок*, при котором сначала идут слова на букву «а», затем на букву «б» и т. д. Среди слов, начинающихся на «а» сначала идут слова со второй буквой «а», затем со второй «б» и т. д. Но в словаре конечное число слов, а нам нужно занумеровать все слова, поэтому мы немного изменим способ нумерации: первым назовём пустое слово, затем лексикографически перенумеруем все однобуквенные слова (т. е. перечислим алфавит), затем также лексикографически занумеруем все двухбуквенные слова и т. д. Занумеруем таким способом слова в первом алфавите и во втором. Теперь каждому натуральному числу поставлено в соответствие два слова: одно в алфавите A, другое — в алфавите B. Нам осталось поставить такие пары слов в соответствие друг другу напрямую, минуя натуральный ряд. Заметим, что в решении алфавиты совершенно равноправны, так что такой кодировкой мы сразу решили задачу кодировки в обе стороны.

Теперь нам совершенно всё равно, с каким алфавитом работает машина Тьюринга. Мы умеем любой алфавит кодировать в нужные нам символы. В частности, нас не будет смущать необходимость подавать машине Тьюринга, работающей в алфавите $\{0, 1\}$, на вход натуральные числа. Ведь числа можно записать в десятичной системе, т. е. в десятибуквенном алфавите, а его закодировать в двухбуквенном.

Вычислимые функции

Определение 1. Функция f из $\mathbb{N}$ в $\mathbb{N}$ (не обязательно определённая на всём множестве натуральных чисел) называется *вычислимой*, если существует машина Тьюринга $T \in \mathcal{T}^1$, такая что для любого натурального числа n

$$T(n) = \begin{cases} f(n), & \text{если } f(n) \text{ определена;} \\ \text{не останавливается} & \text{в противном случае.} \end{cases}$$

В этом случае говорят, что машина T вычисляет функцию f. Множество вычислимых функций мы будем обозначать через $\mathcal{CF}$. Если же вычислимая функция f к тому же всюду определена, то мы будем записывать это так: $f \in \mathcal{CF}_0$ (при этом, очевидно, вычисляющая её машина Тьюринга всегда останавливается).

Теорема 3. *Существует невычислимая функция из* $\mathbb{N}$ *в* $\mathbb{N}$.

Доказательство. Функций из $\mathbb{N}$ в $\mathbb{N}$ несчётное число, а вычислимых функций — счётное, так как для каждой вычислимой функции существует некоторая машина Тьюринга (на самом деле, бесконечно много машин), вычисляющая её. Машины же Тьюринга можно кодировать в конечном алфавите. Собственно, мы это и делаем, когда пишем программу машины Тьюринга. Поэтому различных машин не больше, чем слов в конечном алфавите, т. е. не более чем счётное число. □

Это — правильное доказательство, но оно обладает существенным недостатком: оно не даёт способа найти конкретную невычислимую функцию. Несколько позже мы докажем эту теорему ещё раз, на этот раз просто построив невычислимую функцию.

Перечислимые и разрешимые множества

Определение 2. *Перечислимым множеством* называется множество $A \subset \mathbb{N}$, для которого найдётся вычислимая функция $f \in \mathcal{CF}$, множество значений которой совпадает с A, $\operatorname{Im} f = A$. В таком случае говорят, что функция f перечисляет множество A.

Замечание. При этом не подразумевается, что функция перечисляет элементы множества в порядке возрастания, убывания и т. п.

Примеры

1) $\varnothing$ — перечислимое множество, так как оно является множеством значений нигде не определённой функции, а она вычислима, её вычисляет, например, абсолютный правый ластик.
2) Множество чётных чисел перечислимо.
3) Любое конечное множество перечислимо.

Задача. Докажите, что существуют неперечислимые множества.

Определение 3. Множество $A \subset \mathbb{N}$ называется *разрешимым*, если функция

$$\chi_A(n) = \begin{cases} 1, & n \in A; \\ 0, & n \notin A \end{cases}$$

вычислима. Эта функция называется *характеристической функцией* множества A.

Теорема 4. *Всякое разрешимое множество перечислимо.*

Задача. Докажите эту теорему.

Теорема 5 (третья основная теорема). *Существует перечислимое, но неразрешимое множество.*

Занятие №3. 23 июля 2001 г.

Доказательство. Введём следующее обозначение: пусть U — универсальная машина Тьюринга из основной положительной теоремы. *Диагональной функцией* будем называть функцию

$$d(x) = \begin{cases} U(x,x) + 1, & \text{если } U(x,x) \text{ заканчивает работу,} \\ \text{не определена} & \text{в противном случае.} \end{cases}$$

Нетрудно понять, что $d \in \mathcal{CF}$ (d вычислима).

Основная лемма. *Невозможно доопределить диагональную функцию d до всюду определённой вычислимой функции.*

Доказательство. Фактически, нам нужно доказать, что не существует такой функции $\tilde{d} \in \mathcal{CF}_0$ (вычислимой всюду определённой), что $\tilde{d}(x) = d(x)$ всюду, где d определена. Доказываем (опять!) диагональным методом. От противного. Пусть такая функция $\tilde{d}$ существует. Она вычислима, следовательно, найдётся машина Тьюринга T, вычисляющая её, причём эта машина будет всегда останавливаться, так как $\tilde{d}$ всюду определена. По основной положительной теореме мы можем найти такую инструкцию p_T, что $U(p_T, x) = T(x)$ (здесь написано именно равенство, а не $\doteq$, ведь обе машины всегда останавливаются). Осталось выписать стандартное противоречие: $U(p_T, p_T) = T(p_T) = \tilde{d}(p_T) = d(p_T) = U(p_T, p_T) + 1$. То, что $d(p_T)$ определена, следует из определённости $U(p_T, p_T)$. □

Заметим, что мы попутно доказали и что сама диагональная функция не всюду определена. Также эта лемма позволяет наконец-то предъявить конкретную невычислимую функцию. В качестве примера можно взять такую:

$$\hat{d}(x) = \begin{cases} d(x), & \text{если } d(x) \text{ определена,} \\ 17 & \text{в противном случае.} \end{cases}$$

Предложение 1. *Множество* $\operatorname{Im} d$ *перечислимо и непусто.*

Доказательство. Это множество перечислимо по определению, поэтому нужно только предъявить хоть один аргумент, на котором функция d определена. Для этого достаточно взять инструкцию любой всегда останавливающейся машины, на ней и совпадающем с ней исходном данном остановится универсальная машина, а значит, и машина, вычисляющая d. □

Предложение 2. *Область определения функции* d*, обозначаемая через* $\operatorname{Def} d$*, — неразрешимое множество.*

Доказательство. Доказываем от противного. Если бы оно было разрешимо, то его характеристическая функция была бы вычислима. Но это сразу позволило бы нам вычислить функцию $\hat{d}$. Действительно, можно сначала запустить на входе x машину Тьюринга, вычисляющую характеристическую функцию $\operatorname{Def} d$. Она всегда останавливается. Если она выдала 0, то мы выдаём на выход 17, если же 1, то мы запускаем машину, вычисляющую d, в полной уверенности, что она остановится. Конечно, здесь надо было бы описать, как построить такую машину Тьюринга, но мы, как это принято, сошлёмся на тезис Чёрча. В таком случае предъявленного алгоритма достаточно. □

Предложение 3. *Множество* $\operatorname{Def} d$ *перечислимо.*

Доказательство. Нам нужно построить машину Тьюринга, перечисляющую $\operatorname{Def} d$. Это нетрудно сделать, несколько видоизменив машину, вычисляющую d. Мы добавим в конец этой программы указание вместо получившегося выхода выдать её вход x. Конечно, это требует хранения x во время работы программы где-то на ленте и, следовательно, некоторого числа дополнительных команд, но мы опять не будем обсуждать детали, сославшись на тезис Чёрча. □

Замечание. На самом деле, это просто другое, эквивалентное определение перечислимости: множество перечислимо тогда и только тогда, когда оно является областью определения некоторой вычислимой функции. Мы только доказали это в одну сторону.

Задача. Докажите это в другую сторону.

Из предложений 2–3 следует, что $\operatorname{Def} d$ являет собой пример перечислимого, но неразрешимого множества. Тем самым, доказана третья основная теорема. □

Предложение 4. *Множество* $\operatorname{Im} d$ *неразрешимо.*

Задача. Докажите предложение 4.

Замечание. Вместо терминов *перечислимый*, *разрешимый* довольно часто используют также термины *рекурсивно-перечислимый* и *рекурсивный* соответственно.

В заключение приведём несколько примеров множеств, связанных с формальными исчислениями, и укажем, какие из них являются разрешимыми или перечислимыми. Под номером формулы мы естественно будем понимать гёделев номер.

- Множество номеров правильно построенных формул формального исчисления, в частности формальной арифметики $\mathcal{Z}$, разрешимо.
- Множество номеров теорем $\mathcal{Z}$ перечислимо, но неразрешимо. Это означает, что компьютер способен выводить теоремы, но бессмысленно требовать, чтобы он сказал про конкретное утверждение, является ли оно теоремой.
- Множество номеров арифметических истин, т. е. формул, утверждающих истинные факты про натуральные числа, неперечислимо (это следствие теоремы Тарского). В качестве нетривиального примера такой арифметической истины, не являющейся теоремой $\mathcal{Z}$, можно привести формулу $G(n_0)$ из доказательства теоремы Гёделя о неполноте.

Алгоритмически неразрешимые проблемы

(1) Не существует алгоритма, который по программе машины Тьюринга определяет, всегда ли она останавливается, т. е. множество программ машин из $\mathcal{T}_0^1$ неразрешимо.
(2) Не существует алгоритма, который по программе P и слову x отвечает на вопрос, заканчивает ли работу программа P на входе x. (Это и есть формальный вариант основной неприятности теории алгоритмов.)
(3) Не существует алгоритма, который по программе определяет, заканчивает ли она работу хотя бы на одном входе.
(4) Не существует алгоритма, который по программе машины Тьюринга находит минимум функции, вычисляемой этой машиной.

Доказательство этих фактов предоставляется читателю в виде задачи.

Занятие №4. 25 июля 2001 г.

Теорема Гёделя

Теорема 6 (теорема Гёделя о неполноте). *Если формальная арифметика $\mathcal{Z}$ непротиворечива, то она неполна, т. е. существует предложение формальной арифметики, которое нельзя доказать, но и нельзя опровергнуть — доказать его отрицание.*

Можно было бы подумать, что это недостаток исчисления $\mathcal{Z}$, но, к сожалению, теорема Гёделя имеет следующее обобщение:

Теорема 7. *Если формальное исчисление $\mathcal{F}$ достаточно богато и непротиворечиво, то оно неполно.*

В формулировке этой теоремы остался неопределённым термин *достаточно богато*, и мы не будем давать строгое определение (по причине его громоздкости). Скажем только, что любое исчисление, содержащее формальную арифметику, будет достаточно богато.

Определение 4. *Формальное исчисление* $\mathcal{F} = \langle A, \varphi, \delta \rangle$ содержит три компоненты: алфавит A, состоящий из *неопределённых символов*; алгоритм φ, определяющий по слову в алфавите A, является ли оно правильно построенным словом, т. е. *формулой*; алгоритм δ, который по конечной последовательности формул $(f_1, f_2, \ldots, f_k)$ произвольной длины выдаёт ответ «да» либо «нет»; формулу f будем называть *теоремой*, если существует некоторая последовательность формул, оканчивающаяся на f, на которой алгоритм δ говорит «да».

Рассмотрим множество слов в алфавите A, занумеруем их в нашем лексикографическом порядке, учитывающем длину слов. Обозначим через $\mathbb{F} \subset \mathbb{N}$ множество номеров правильно построенных формул исчисления $\mathcal{F}$, через $\mathbb{T} \subset \mathbb{N}$ множество номеров теорем $\mathcal{F}$, наконец, через $\tilde{\mathbb{T}} \subset \mathbb{N}$ множество *антитеорем*, т. е. утверждений, опровержимых в $\mathcal{F}$.

На этом языке *непротиворечивость* означает, что $\mathbb{T} \cap \tilde{\mathbb{T}} = \varnothing$, а *неполнота*, что $\mathbb{F} \setminus (\mathbb{T} \cup \tilde{\mathbb{T}}) \neq \varnothing$.

Теперь нам придётся ввести новый термин для наших утверждений о формальных исчислениях: теоремами теперь называются определённые слова алфавита A. Поэтому мы будем называть наши утверждения *метатеоремами*.

Метатеорема 1. *Множество $\mathbb{F}$ разрешимо.*

Доказательство. Это тривиальное следствие определения. □

Метатеорема 2. *Множество $\mathbb{T}$ перечислимо.*

Доказательство. Действительно, упорядочим лексикографически (с учётом количества членов) все конечные цепочки формул. Вычислимая функция, перечисляющая множество теорем, будет выглядеть так: получив на входе номер последовательности, она запускает алгоритм δ; в тех случаях когда он отвечает «да», функция выдаёт на выход последнюю формулу последовательности, в противном случае функция не определена. □

Приведём пример исчисления, в котором множество теорем не только перечислимо, но и разрешимо. Простейшим из таких исчислений является исчисление высказываний $\mathcal{S}$. Алфавит его состоит из неопределённых символов $\{(,), \vee, \neg, p, q, r, \ldots\}$, букв $p, q, r, \ldots$ при этом счётное количество (на самом деле, надо конечно писать $p, p', p'', p''', \ldots$, но мы, удобства ради, будем использовать разные буквы).

Не будем останавливаться на подробном определении правильно построенной формулы, это и так очевидно. Введём следующее определение. С точки зрения формальной теории это определение имеет статус обозначения или сокращённой записи.

Определение 5.

$$«p \to q» := «(\neg p) \vee q»$$

Перечислим все аксиомы исчисления $\mathcal{S}$:

1. $(p \vee p) \to p$
2. $p \to (p \vee q)$
3. $(p \vee q) \to (q \vee p)$
4. $(p \to q) \to ((r \vee p) \to (r \vee q))$

В исчислении высказываний всего два правила вывода:

1. $\dfrac{\phi \text{ и } \phi \to \psi}{\psi}$ (*modus ponens*)
2. $\dfrac{\phi(p)}{\Pi_p^{\psi}\phi(p)}$ (правило замены)

Правило замены позволяет заменять в формуле все символы p на формулу ψ.

Осталось описать алгоритм δ. Он отвечает «да» в том и только в том случае, когда каждая формула последовательности либо является аксиомой, либо получается по *modus ponens* из каких-то двух предыдущих формул, либо заменой из одной предыдущей формулы. Последовательности, удовлетворяющие этому правилу, называются *выводами*.

Метатеорема 3. *Исчисление высказываний $\mathcal{S}$ непротиворечиво, полно, множество его теорем $\mathbb{T}_{\mathcal{S}}$ разрешимо.*

Заметим, что один из возможных разрешающих алгоритмов для $\mathbb{T}_{\mathcal{S}}$ основан на так называемых *таблицах истинности*.

Теперь мы сформулируем теорему Гёделя в обобщённом виде, а именно мы ослабим требования полноты, непротиворечивости и достаточного богатства.

Определение 6. Рассмотрим формальное исчисление $\mathcal{F} = \langle A, \varphi, \delta \rangle$. Пусть существует вычислимая инъекция $\sigma : \mathbb{N} \to W_A$. Эта инъекция задаёт некоторую нумерацию слов алфавита A (не обязательно всех): $w_1, w_2, \ldots, w_i = \sigma(i), \ldots$

Формальное исчисление $\mathcal{F}$ называется *слабо богатым*, если существуют два слова $\xi, \tilde{\xi}$ в алфавите $A \cup \{b\}$, $b \notin A$, и перечислимое, но неразрешимое множество X, такие что в случае $n \in X$ теоремой является $\xi(w_n)$, а в случае $n \notin X$ — $\tilde{\xi}(w_n)$. Здесь $\xi(w_n)$ и $\tilde{\xi}(w_n)$ представляют собой результат подстановки слова w_n вместо символа b соответственно в слова ξ и $\tilde{\xi}$.

Формальное исчисление $\mathcal{F}$ называется *слабо полным*, если для каждого натурального числа n в $\mathcal{F}$ доказывается $\xi(w_n)$ или $\tilde{\xi}(w_n)$.

Наконец, $\mathcal{F}$ называется *слабо непротиворечивым*, если ни для какого натурального n в $\mathcal{F}$ не доказывается одновременно и $\xi(w_n)$, и $\tilde{\xi}(w_n)$.

Метатеорема 4 (обобщённая теорема Гёделя). *Формальное исчисление не может быть одновременно слабо богатым, слабо полным и слабо непротиворечивым.*

Доказательство. Доказываем от противного. Пусть все три условия одновременно выполнены. Тогда запустим процесс перечисления множества теорем $\mathcal{F}$. Для каждого натурального числа n по крайней мере одно из слов $\xi(w_n)$ и $\tilde{\xi}(w_n)$ является теоремой в силу слабой полноты. Значит, одно из них рано или поздно встретится в нашем перечислении. Допустим, нам встретилось $\xi(w_n)$. Это могло произойти, если $n \in X$. А в противном случае такого не могло бы случиться, так как теоремой было бы $\tilde{\xi}(w_n)$, а одновременно они не могут быть теоремами в силу слабой непротиворечивости. Аналогично, если в перечислении нам встретится $\tilde{\xi}(w_n)$, мы можем с уверенностью сказать, что $n \notin X$. Таким образом, мы построили разрешающий алгоритм для множества X. Противоречие. □