1 Начальные понятия и термины (первая лекция).

Алгоритм. Область возможных исходных данных. Словарное пространство (частный случай). Результат вычисления. Область результативности (применимости).

Обозначение определенности (результативности) выражения $!(\dots)$

Вычислимая функция – вычислимая некоторым алгоритмом.

Разрешимое мн-во в данном словарном пространстве – вычислимость соответстующей характеристической функции.

Полуразрешимое мн-во в данном словарном пространстве – вычислимость соответстующей полухарактеристической функции.

Перечислимое мн-во – пустое или область значений вычислимой последовательности (тотальной вычислимой функции $N \to \Sigma^*$).

1.1 Факты

1. Перечислимость объединения двух перечислимых мн-в.
2. Перечислимость N^2, вычислимые функции проекции $\xi(x)$, $\eta(x)$ – по *номеру* x пары $\langle m, n\rangle$ вычисляют m и n.
3. Перечислимость пересечения двух перечислимых мн-в.
4. Перечислимость прямого произведения перечислимых мн-в и проекций "плоского" перечислимого мн-ва.
5. Перечислимость разрешимого подмножества словарного пр-ва.
6. Теорема Чёрча-Поста. Подмножество словарного пр-ва разрешимо т. и т.т., когда оно и его дополнение перечислимы.
7. Теорема: каждое перечислимое множество есть область определения некоторой вычислимой функции ($f(x) = \mu n.(x = a_n)$).
8. Следствие: каждое перечислимое мн-во полуразрешимо.
9. Теорема: у каждой вычислимой функции область определения и область значений – перечислимы. (док-во не закончено)

2 Перечислимость области определения и области значений выч. ф-ции.

Пошаговость вычислительного процесса. Обогатим наше абстрактное понятие алгоритма (математическую модель) дополнительным допущением:

> Процесс применения алгоритма $\mathcal{A}$ к исходному данному x всегда происходит дискретно – по шагами. Шаги удается занумеровать натуральными числами: 0 (до начала вычисления), 1, 2, ... Каждый шаг обязательно заканчивается. При этом само вычисление можит не закончится только в результате того, что шагов бесконечно много. Предполагается возможность контроля пошагового исполнения в следующем смысле. Пусть X – область возможных исходных данных для алгоритма $\mathcal{A}$. Тогда *сигнализирующее* множество
>
> $$E = \{\langle x, n\rangle \mid \text{вычисление } \mathcal{A} \text{ на входе } x \text{ заканчивается за } \leq n \text{ шагов}\}$$
>
> всегда является разрешимым подмножеством $X \times N$.

Следствие 2.1 *Область определения каждой вычислимой функции – перечислима.*

Доказательство. Пусть $f : X \to Y$ вычисляется алгоритмом A, а E – его сигнализирующее мн-во. Оно разрешимо и, как следствие, перечислимо. Его проекция $pr_1 E$ совпадает с $D(f)$, а проектирование сохраняет перечислимость. Отсюда $D(f)$ –перечислимо.

■

Следствие 2.2 *Область значений каждой вычислимой функции – перечислима.*

Доказательство. Если $D(f) = \emptyset$, то $E(f) = \emptyset$, а потому, перечислима. Иначе $D(f)$ есть область значений вычислимой последовательности $\{a_n\}$. Тогда $E(f) = \{f(a_0), f(a_1), \dots\}$, причем последовательность $b_n = f(a_n)$ также вычислима. (По n вычисляем a_n, а потом к a_n применяем алгоритм вычисления f.)

■

Следствие 2.3 *(Теорема о графике) Пусть X, Y – словарные пр-ва. Функция $f : X \to Y$ вычислима т. и т.т., когда ее график $\Gamma_f = \{\langle x, y\rangle \mid x \in X, y = f(x)\}$ – перечислим.*

Доказательство. ($\Rightarrow$). Если $D(f) = \emptyset$, то $\Gamma_f = \emptyset$, а потому, перечислим. Иначе $D(f)$ есть область значений вычислимой последовательности $\{a_n\}$. Тогда $\Gamma_f = \{\langle a_0, f(a_0)\rangle, \langle a_1, f(a_1)\rangle, \dots\}$, причем последовательность $c_n = \langle a_n, f(a_n)\rangle$ также вычислима.

($\Leftarrow$). Если $\Gamma_f = \emptyset$, то f нигде не определена, а потому, вычислима алгоритмом "не выдавать никакого результата". Иначе Γ_f есть область значений вычислимой последовательности $\{c_n\}$, где $c_n = \langle a_n, b_n\rangle$, причем $b_n = f(a_n)$, а последовательность $\{a_n\}$ перечисляет $D(f)$. Алгоритм вычисления $f(x)$ следующий:

> На входе x порождать последовательно $c_0 = \langle a_0, b_0\rangle$, $c_1 = \langle a_1, b_1\rangle$, ... и проверять условие "$x = a_n$?". Как только оно станет истинным, выдать в качестве результата b_n и остановится.

■

Итог:

M – перечислимо	$\Leftrightarrow$	M – полуразрешимо
	$\Leftrightarrow$	M – область определения выч. функции
	$\Leftrightarrow$	M – область значений выч. функции.
$M \subset X$ – разрешимо	$\Leftrightarrow$	M, $X \setminus M$ – перечислимы.
f – вычислима	$\Leftrightarrow$	ее график Γ_f перечислим.

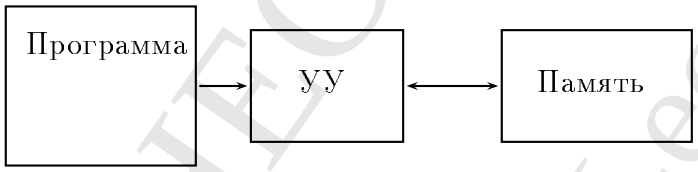
Замечание. Заметим, что до сих пор все доказываемые факты имели следующую форму: если существуют одни алгоритмы, то существуют другие. Доказательство состояло в построении желаемых алгоритмов из имеющихся; по существу, в программировании. Однако центральная задача теории алгоритмов несколько иная: изучить границы возможности программирования, т.е. выяснить, для каких задач не существует (и не может существовать) алгоритмов их решения. Одна из таких границ – тривиальная: нельзя запрограммировать функцию, у которой область определения или область значений не влезает в область возможных исходных данных ни для одного алгоритма, например, несчетна. Но есть и гораздо более глубокие ограничения, актуальные уже для счетных областей.

Мы приступаем к разработке техники доказательства отрицательных фактов про вычислимость (т.е. что такая-то функция не является вычислимой). Для этого оказывается важным иметь возможность анализировать не только (абстрактные) алгоритмы, но и их программы, т.е. записи (реализации) алгоритмов на каком-нибудь языке программирования. Какой язык выбрать?

Программист и аналитик выбирают язык по-разному: программист – чтобы было удобно программировать, а аналитик – чтобы было удобно анализировать. Для анализа удобнее самый примитивный язык, который все же обладает свойством универсальности (т.е. принципиально позволяет реализовать все алгоритмы). Мы пойдем именно этим путем.

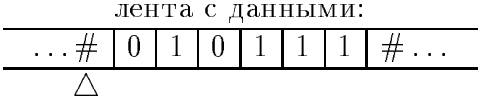
3 Машины Тьюринга.

Машина Тьюринга состоит из управляющего устройства (УУ) и потенциально бесконечной внешней памяти, структура которой не меняется со временем. Она снабжена программой, задающей правила ее функционирования.



В простейшем случае память представляет собой бесконечную в обе стороны ленту, разбитую на одинаковые ячейки, предназначенные для хранения букв фиксированного конечного алфавита Σ (одна буква в ячейке, в "пустых" ячейках – тоже буква #, обо-

значающая пустоту). Имеется одна читающе-пишущая головка.



В каждый момент времени головка обозревает одну ячейку памяти. За один такт работы головка может изменить содержимое этой ячейки и переместиться к одной из соседних (или остаться на месте). Функционированием головок управляет УУ. В каждый момент времени УУ находится в одном из фиксированного конечного множества внутрнних состояний Q. Один такт работы состоит в следующем: машина читает содержимое обозреваемой ячейки и внутреннее состояние, выбирает из программы инструкцию, однозначно определяемую этими данными, и исполняет ее. В инструкции сказано, каким должно стать новое внутреннее состояние, какая буква должна быть записаны в обозреваемую ячейку и куда должны переместиться головка.

Таким образом, работа машины описывается тремя конечными функциями:

$$\begin{array}{ll} \alpha : Q \times \Sigma \to Q & \text{– новое состояние} \\ \beta : Q \times \Sigma \to \Sigma & \text{– новое содержимое ячейки} \\ \gamma : Q \times \Sigma \to \{N, L, R\} & \text{– движение головок} \end{array}$$

(N – на месте, L – влево, R – вправо). Их удобно записывать в виде программы – набора непротиворечивых команд вида:

$$qa \mapsto \alpha(q,a)\beta(q,a)\gamma(q,a)$$

Непротиворечивость набора означает, что в нем нет двух команд с одинаковой левой частью qa. Для сокращения письма предполагают также, что если команда с левой частью qa не выписана явно, то она все-таки есть в программе, но имеет специфический вид $qa \mapsto qa\,N$. Такие команды не меняют содержимого ленты, состояния и положения головки, поэтому они вызывают зацикливание, если исполняются.

На ленту записывается входное слово и головка располагается в "пустой" ячейке перед ним. При включении машины УУ оказывается в заранее фиксированном начальном состоянии $q_{нач.} \in Q$, а переход в также фиксированное заключительное состояние $q_{закл.} \in Q$ вызывает ее отключение (остановку).

Определение 3.1 (Итог:) Полная спецификация машины Тьюринга есть набор

$$M =< Q, \Sigma, \alpha, \beta, \gamma, q_{нач.}, q_{закл.} > .$$

Пример. Договоримся натуральные числа представлять в унарной записи:

$$n \quad \longleftrightarrow \quad \bar{n} = \underbrace{11\ldots1}_{n \text{ раз}}.$$

Описать машину Тьюринга, реализующую операцию сложения: $111 + 11 \longmapsto 11111$.

$$\begin{array}{lcl} q_1\# & \mapsto & q_1\#R \\ q_1 1 & \mapsto & q_1 1 R \\ q_1 + & \mapsto & q_2 1 R \\ q_2 1 & \mapsto & q_2 1 R \\ q_2\# & \mapsto & q_0\#L \end{array}$$

Вопрос: Что именно задает спецификация машины Тьюринга?

Ответ: Процесс вычисления. Но что именно этот процесс вычисляет – еще не сказано!

Определение 3.2 Пусть $\Sigma_0 \subseteq (\Sigma \backslash \{\#\})$. Машина Тьюринга M вычисляет (частичную) словарную функцию $f : \Sigma_0^* \to \Sigma_0^*$, если

1) на входах $v \in \Sigma_0^*$ машина M останавливается т. и т.т., когда значение $w = f(v)$ определено, и,

2) если такое произошло, то слово w оказалось записанным на ленте, непосредственно справа и слева от него стоят буквы $\notin \Sigma_0$, а головка остановилась внутри или непосредственно перед w.

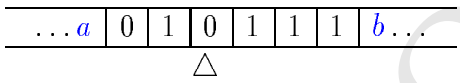
Замечание. В этом определении машине разрешается оставлять на ленте "мусор" справа и слева от результата.

Вопрос: Машина, вычисляющая словарную функцию $f : \Sigma_0^* \to \Sigma_0^*$, на входе $v \in \Sigma_0^*$ остановилась, но ни в обозреваемой ячейке, ни непосредственно справа от нее нет букв алфавита Σ_0. Чему равно $f(v)$?

Ответ: Пустому слову Λ.

Обозначение. Пусть машина Тьюринга M на входе v остановилась. Обозначим через $M(v)\big|_{\Sigma_0}$ то максимальное слово в алфавите Σ_0 на ленте, на которое указывает головка.

после остановки:



$$w = 010111 = M(v)\big|_{\{0,1\}}$$

Определение 3.3 Пусть $\Sigma_0 \subseteq (\Sigma \backslash \{\#\})$. Машина Тьюринга M вычисляет (частичную) словарную функцию $f : \Sigma_0^* \to \Sigma_0^*$ В СИЛЬНОМ СМЫСЛЕ (или ЧИСТО), если 1) – то же самое условие и

2') = 2) + дополнительно требуется, чтобы на ленте ничего, кроме результата $w \in \Sigma_0^*$ и букв # не оставалось, а головка остановилась в клетке непосредственно перед словом w. (Это условие наз. условием чистоты.)

Замечание. Понятие ЧИСТО работающей машины Тьюринга относительно – надо указывать алфавит Σ_0. Иначе неясно, что считается результатом, а что – мусором.

Замечание. Можно аналогично опрелить вычислимость и "чистую" вычислимость по Тьюрингу словарных функций от нескольких (фиксированного числа) аргументов, т.е. функций типа $\Sigma_0^* \times \ldots \times \Sigma_0^* \to \Sigma_0^*$. Для этого надо объявить какой-нибудь символ из $\Sigma \setminus \Sigma_0$ (например, "$\square$" или "+") разделителем и подавать все аргументы на вход манине Тьюринга в виде одного слова, отделяя один от другого разделителем.

Вопрос: Какую функцию $\{1\}^* \times \{1\}^* \to \{1\}^*$ вычисляет машина из примера выше? Является ли это "чистым" вычислением ?

Теорема 3.4 *Пусть словарная функция $f : \Sigma_0^* \to \Sigma_0^*$ вычислима машиной Тьюринга M. Можно построить машину Тьюринга M', которая чисто вычисляет f.*

Доказательство. Сначала преобразуем машину $M =< Q, \Sigma, \ldots >$ в другую – M_1, которая также будет вычислять f, но при этом никогда не будет писать # в ячейки, которые читала. Добавим к ленточному алфавиту Σ новую букву $\square$ (двойник #). В программе M изменим команды с участием # так:

вместо	напишем	
$qa \mapsto q'\#X$	$qa \mapsto q'\square X$	X – символ движения
$q\# \mapsto q'a'X$	$q\# \mapsto q'a'X$ $q\square \mapsto q'a'X$	
$q\# \mapsto q'\#X$	$q\# \mapsto q'\square X$ $q\square \mapsto q'\square X$	

В результате получим машину M_1, которая записывает $\square$ в те ячейки, в которые M писала #, а в остальном – повторяет вычисление M.

Теперь добавим группу команд, которые "чистят мусор" на ленте после того, как M_1 выполнит основную работу. Разжалуем заключительное состояние машины M_1 в обычное – q_n. Допишем команды:

$$
\begin{array}{rll}
q_n a \mapsto q_{n+1} a N, & a \in \Sigma \cup \{\square\}; & \\
q_{n+1} a \mapsto q_{n+1} a L, & a \in \Sigma_0; & \text{к началу ответа} \\
q_{n+1} a \mapsto q_{n+2} a N, & a \in (\Sigma \cup \{\square\}) \setminus \Sigma_0; & \\
q_{n+2} a \mapsto q_{n+2} \# L, & a \in (\Sigma \cup \{\square\}) \setminus \{\#\}; & \text{к началу зоны} \\
q_{n+2} \# \mapsto q_{n+3} \# R; & & \\
q_{n+3} \# \mapsto q_{n+3} \# R; & & \text{чистим слева} \\
q_{n+3} a \mapsto q_{n+4} a R, & a \in \Sigma_0; & \\
q_{n+4} a \mapsto q_{n+4} a R, & a \in \Sigma_0; & \text{к концу ответа} \\
q_{n+4} a \mapsto q_{n+5} \# R, & a \in (\Sigma \cup \{\square\}) \setminus \Sigma_0; & \\
q_{n+5} a \mapsto q_{n+5} \# R, & a \in (\Sigma \cup \{\square\}) \setminus \{\#\}; & \text{чистим справа} \\
q_{n+5} \# \mapsto q_{n+6} \# L; & & \\
q_{n+6} \# \mapsto q_{n+6} \# L; & & \text{возврат головки} \\
q_{n+6} a \mapsto q_{n+7} a L, & a \in \Sigma_0; & \\
q_{n+7} a \mapsto q_{n+7} a L, & a \in \Sigma_0; & \\
q_{n+7} \# \mapsto q_{n+8} \# N. & &
\end{array}
$$

Осталось объявить q_{n+8} заключительным состоянием и машина M' готова.

■

Вопрос: Зачем сначала машину M преобразовали в M_1 ?
Ответ: Чтобы было ясно, какую зону надо очищать от мусора.

Структурные факты.

Теорема 3.5 *Если словарные функции $f, g : \Sigma_0^* \to \Sigma_0^*$ вычислимы на машинах Тьюринга, то их композиция $\lambda v.g(f(v))$ также вычислима на подходящей машине Тьюринга. (Ее можно построить явно по машинам, вычисляющим f и g.)*

Доказательство. Пусть машины M_f, M_g вычисляют f и g. Преобразуем M_f в M'_f, вычисляющую f чисто. Переименуем состояния M_g так, чтобы ее начальное состояние совпало с заключительным состоянием M'_f, а остальные состояния не совпадали с состояниями M'_f. Объединим две получившиеся программы в одну. Объявим общим начальным состоянием начальное состояние M'_f, а общим заключительным – заключительное состояние переименованной M_g. Получившаяся машина вычисляет композицию $\lambda v.g(f(v))$.

■

Вопрос: Зачем сначала машину M_f преобразовали в M'_f ?

Ответ: Чтобы обеспечить стандартное начало работы для переименованной M_g.

Все традиционные для языков программирования управляющие конструкции сохраняют свойство "быть вычислимым на машинах Тьюринга". Например, следующие (без док-ва):

$$\texttt{IF}\ g(v) = \Lambda\ \texttt{THEN}\ f_1(v)\ \texttt{ELSE}\ f_2(v)$$

$$\texttt{WHILE}\ g(v) = \Lambda\ \texttt{DO}\ v := f(v)\ \texttt{DONE};\ \texttt{RETURN}\ v$$

Замечание. Конечное множество состояний на самом деле может представлять любую ограниченную (для данной машины) структуру данных. Это имитирует статически распределенную память с возможностью за один шаг переприсваивать ее всю целиком (например, всю оперативную память компьютера). Единственное ограничение – новое состояние определяется однозначно по предыдущему состоянию и содержимому обозреваемой ячейки ленты. Т.к. допускается сколь угодно большой (но конечный) ленточный алфавит (например, из 2^{64} букв), то машин Тьюринга оказывается достаточно для моделирования принципиальных вычислительных возможностей компьютера, снабженного внешней памятью (лента – потенциально бесконечный HDD). Эта модель адекватна в качественных вопросах ("Можно запрограммировать данную функцию или нет?").

Тезис Тьюринга.

Результат сравнительного изучения запасов вычислимых словарных функций для различных (не только "игрушечных") моделей вычислений может быть сформулирован в виде следующего неформального утверждения:

> **Тезис Тьюринга (неформальный):** Каждую вычислимую (программой какого угодно языка программирования) частичную функцию типа $\Sigma^* \to \Sigma^*$ можно вычислить на подходящей машине Тьюринга.

Неформальность Тезиса Тьюринга состоит в том, что он не может быть полностью обоснован математическими средствами (доказан как математическая теорема). В то же время все многочисленные попытки его опровергнуть, т.е. предложить язык программирования с большими вычислительными возможностями, оказались безуспешными (в результате чего в настоящее время подобные проекты считаются абсолютно бесперспективными). Причина невозможности полного математического обоснования кроется в словах "какого угодно языка программирования" – они не определяют ни самого семейства языков, о которых идет речь, ни какого-либо свойства этого семейства.

Если их заменить на достаточно информативное описание семейства языков программирования (которое необходимо окажется менее общим), то соответствующий частный случай Тезиса станет обычным, "поддающимся доказательству" математическим утверждением.

Пример. Пусть семейство состоит из языков C и PASCAL, заданных полным описанием их синтаксиса и операционной семантики. Оба языка обладают компиляторами в ASSEMBLER, поэтому для обоснования соответствующего частного случая Тезиса Тьюринга достаточно построить компилятор, преобразующий ассемблерный код в программу для машины Тьюринга. Последнее является весьма трудоемкой, но абсолютно реалистичной задачей по программированию. Для аккуратного математического доказательства потребуется еще верифицировать все три компилятора, т.е. доказать, что они работают правильно.

Достаточно двух служебных букв.

Ограничимся классом машин Тьюринга с данным ленточным алфавитом Σ. Пусть $\Sigma_0 \subset \Sigma$ и $\#, \square \notin \Sigma_0$. Оказывается, что ограничение на запас "служебных" букв, т.е. на мощность множества $\Sigma \setminus \Sigma_0$, почти не влияет на возможность вычисления словарных функций типа $\Sigma_0^* \to \Sigma_0^*$:

Теорема 3.6 (без док-ва) *Если словарная функция $\Sigma_0^* \to \Sigma_0^*$ вычислима на машине Тьюринга, то существует машина Тьюринга с ленточным алфавитом $\Sigma = \Sigma_0 \cup \{\square, \#\}$, которая чисто вычисляет f.*

Соглашения.

Раз и навсегда фиксируем какой-нибудь большой алфавит Σ в качестве ленточного (напр., всю ASCII-таблицу), несколько символов из него будем использовать как служебные ($\Sigma_{\text{сл}}$), несколько других – как разделители ($\Sigma_{\text{разд}}$), а остальное – как буквы для формирования алфавитов изучаемых словарных пространств: $\Sigma_0, \Sigma_1, \ldots \subseteq \Sigma \setminus (\Sigma_{\text{сл}} \cup \Sigma_{\text{разд}})$.

Разделители нужны для словарного представления исходных данных $\langle v_1, \ldots, v_n \rangle \in \Sigma_1^* \times \ldots \times \Sigma_n^*$ – в виде одного слова $v = v_1 \square \ldots \square v_n \in \Sigma^*$ для определения вычислимости по Тьюрингу словарных функций от многих переменных, $f : \Sigma_1^* \times \ldots \times \Sigma_n^* \to \Sigma_0^*$. Такая функция считается вычислимой по Тьюрингу, если соответствующее отображение $v_1 \square \ldots \square v_n \mapsto f(v_1, \ldots, v_n)$ вычислимо по Тьюрингу (как функция одного аргумента).

Условимся представлять натуральные числа их унарными записями: $n \longleftrightarrow \bar{n} \in \{1\}^*$. Это даст определение вычислимости по Тьюрингу функции, некоторые аргументы и/или значения которой – натуральные числа.

Кодирование программ машин Тьюринга словами в алфавите программ.

Вопрос. Как определить вычислимость по Тьюрингу для функций, аргументами (и/или значениями) которых являются машины Тьюринга?

Ответ. Надо научится кодировать машины Тьюринга словами в "неслужебной части" алфавита Σ.

Заметим, что если договориться, что у машины Тьюринга всегда $q_{\text{нач}} = q_1$ и $q_{\text{закл}} = q_0$, то ее программа (подробная) будет однозначно определять ее полную спецификацию $M =< Q, \Sigma, \alpha, \beta, \gamma, q_{нач.}, q_{закл.} >$. Осталось закодировать программу:

- Буквы $a_i \in \Sigma$ будем записывать словами $a\bar{i} \in \{a, 1\}^*$.
- Состояния $q_j \in Q$ будем записывать словами $q\bar{j} \in \{q, 1\}^*$.
- Переведенные так команды будем записывать в одно слово, разделяя буквой "z"(например).

В результате получим слово в алфавите Ы$= \{1, a, q, \to, N, L, R, z\}$ (алфавит программ), которое называется кодом машины M (обозначение $Code(M)$). Именно оно и будет на ленте представлять машину M как исходное данное или результат вычисления для других машин Тьюринга. (Следует позаботиться, чтобы все символы алфавита программ попали в "неслужебную часть" алфавита Σ.)

Св-ва кодирования:

1. (Очевидное.) Соответствие между машинами и их кодами взаимно однозначно (ленточный алфавит, начальное и заключительное состояния фиксированы).
2. Множество кодов программ является разрешимым подмножеством в Ы*. (Задачка по программированию.)

Универсальная машина Тьюринга.

Условное равенство ($\simeq$). $\mathcal{A}(x)$ означает результат применения алгоритма $\mathcal{A}$ к входному данному x. Но в выкладках приходится использовать это выражение и тогда, когда соответствующего результата не существует (обозначение не имеет денотата). Использовать в таких выкладках обычное равенство не вполне корректно, т.к. истинностное значение утверждения "a=b" при неопределенных a или b также НЕ ОПРЕДЕЛЕНО (а не false, как многие думают). Его правильно заменять другим отношением эквивалентности – *условным равенством*:

$$a \simeq b \Leftrightarrow \begin{cases} true, & \text{если } (!a \wedge !b \wedge a = b) \vee (\neg!a \wedge \neg!b), \\ false, & \text{иначе.} \end{cases}$$

Примеры. 1. Равенство $x \cdot 1/x = 1/x \cdot x$ для произвольного $x \in \mathbf{R}$ бессмысленно, а $x \cdot 1/x \simeq 1/x \cdot x$ – верно.

2. Если определить одно обозначение (u) через другое ($u := v$), то становится верным утверждение "$u \simeq v$". При этом утверждение "u=v" может оказаться бессмыслицей.

Определение 3.7 *Универсальной* машиной для класса всех машин Тьюринга с ленточным алфавитом Σ называется такая машина Тьюринга U, что для всех машин M из этого класса и всех слов $v \in \Sigma^*$ выполняется

$$U(\,Code(M)\square v\,)\,\Big|_{\Sigma\setminus\{\#\}} \simeq \;\; M(v)\;\Big|_{\Sigma\setminus\{\#\}}.$$

Замечание. Здесь левая и правая части равенства означают те (максимальные) слова в алфавите $\Sigma\setminus\{\#\}$, на которые указывают головки после остановки машин. Ленточный алфавит машины U может быть расширением алфавита Σ. Стандартным способом можно дополнительно обеспечить чистоту вычислений относительно алфавита $\Sigma\setminus\{\#\}$.

Теорема 3.8 *Универсальная машина для класса всех машин Тьюринга с ленточным алфавитом Σ существует.*

Доказательство. Рассмотрим словарную функцию $f : \Sigma^* \to \Sigma^*$, которая определена только на таких словах, которые 1) имеют вид $Code(M)\square v$, и 2) значение $M(v)$ существует, причем ее значение $f(Code(M)\square v)$ есть $M(v)$.

Установим, что f – вычислимая функция. Алгоритм $\mathcal{A}$ ее вычисления:

1. Во входном слове $x \in \Sigma^*$ найти самое левое вхождение буквы $\square$ и разделить x в этом месте на части: $x = w\square v$. Если не удалось, то остановиться без результата.
2. Проверить, что $w \in$ Ы* и что $w = Code(M)$ для некоторой машины Тьюринга M (мн-во кодов программ – разрешимо!). Восстановить M. Если не удалось, то остановиться без результата.
3. Запустить процесс вычисления M на входе v. Когда (если) он закончится результативно, то вернуть его результат $M(v)\big|_{\Sigma\setminus\{\#\}}$ в качестве результата $\mathcal{A}(x)$.

Применим Тезис Тьюринга: существует машина Тьюринга U, которая вычисляет f. Она – искомая. (Заметим, что при некотором усердии и избытке времени применение Тезиса можно избежать. Сесть и запрограммировать, что и делалось неоднократно.) ■

4 Нумерация выч. функций типа N → N.

Определение 4.1 Пусть $\mathcal{F}$ – счетное семейство функций (напр. $Com(\mathbf{N},\mathbf{N})$). Универсальной функцией для семейства $\mathcal{F}$ называется такая функция $u(i,x)$, $(i \in \mathbf{N})$, для которой выполнены два условия:

1. Для каждого $i \in \mathbf{N}$ функция $f(x) := u(i,x)$ принадлежит $\mathcal{F}$.
2. $(\forall f \in \mathcal{F})(\exists i \in \mathbf{N})(\forall x)\, f(x) \simeq u(i,x)$.

Замечание. Универсальная функция существует у любого счетного семейства $\mathcal{F}$ и задает некоторую его нумерацию: $\mathcal{F} = \{f_0, f_1, \dots\}$, где $f_i(x) := u(i,x)$.

Теорема 4.2 *У семейства $Com(\mathbf{N},\mathbf{N})$ всех вычислимых функций $\varphi : \mathbf{N} \to \mathbf{N}$ существует ВЫЧИСЛИМАЯ универсальная функция.*

Доказательство. По Тезису Тьюринга семейство $Com(\mathbf{N},\mathbf{N})$ совпадает с множеством тех функций $\varphi : \mathbf{N} \to \mathbf{N}$, которые вычислимы по Тьюрингу (кодировка натуральных чисел – унарная). При этом достаточно использовать машины с фиксированным ранее ленточным алфавитом $\Sigma \supseteq Ы^* \cup \{1, \square, \#\}$ (теорема о достаточности двух доп. букв), работающие чисто относительно алфавита $\{1\}$.

Для каждой машины M с ленточным алфавитом Σ определен ее код $Code(M) \in Ы^*$, причем множество всех кодов – разрешимо, а следовательно, и перечислимо. Рассмотрим вычислимую последовательность слов $p_0, p_1, \ldots$, состоящую из всех кодов всех машин с ленточным алфавитом Σ. (Там могут быть повторения, которые, при желании, можно устранить.)

Возьмем универсальную машину Тьюринга U для алфавита Σ. Определим функцию $u : \mathbf{N}^2 \to \mathbf{N}$ следующим алгоритмом:

Исходные данные: $i, n \in \mathbf{N}$.

1. Вычислить p_i и $\bar{n} := \underbrace{11\ldots1}_{n \text{ раз}}$. Организовать вычисление $U(p_i \square \bar{n})$.

2. Когда (если) вычисление закончится, выделить на ленте (максимальное) слово в алфавите $\Sigma \setminus \{\#\}$, на которое указывает головка:

$$w := U(\, p_i \square \bar{n}\,)\big|_{\Sigma\setminus\{\#\}}$$

Положить $u(i,n)$ равным длине слова w.

Функция u вычислима (по построению), поэтому вычислимы и все функции $\varphi_i(n) := u(i,n)$ (т.е. первое условие универсальности выполнено).

Проверим второе условие. Если $\varphi : \mathbf{N} \to \mathbf{N}$ – вычислима, то существует машина Тьюринга M с ленточным алфавитом Σ, которая ее чисто вычисляет. Для всех $n \in \mathbf{N}$ выполнено: $\varphi(n) \simeq |M(\bar{n})\big|_{\{1\}}|$ (кодировка унарная!). Найдем i из условия $p_i = Code(M)$. Т.к. M работает чисто, то в результате вычисления $M(\bar{n})$ мусора не появится и

$$M(\bar{n})\big|_{\{1\}} \simeq M(\bar{n})\big|_{\Sigma\setminus\{\#\}} \simeq U(Code(M)\square\bar{n})\big|_{\Sigma\setminus\{\#\}} \simeq U(p_i\square\bar{n})\big|_{\Sigma\setminus\{\#\}}.$$

Тогда

$$u(i,n) \simeq \Big|\, U(p_i\square\bar{n})\big|_{\Sigma\setminus\{\#\}}\,\Big| \simeq \varphi(n).$$

■

Вопрос. Зачем нужно, чтобы M вычисляло φ чисто?

Ответ. Иначе нарушится равенство $M(\bar{n})\big|_{\{1\}} \simeq M(\bar{n})\big|_{\Sigma\setminus\{\#\}}$.

Замечание. Совершенно аналогично для каждого k можно построить вычислимую универсальную функцию $u^{(k)}(i, n_1, \ldots, n_k)$ для семейства всех вычислимых функций типа $\mathbf{N}^k \to \mathbf{N}$.

Определение 4.3 Вычислимая универсальная функция $u(i,n)$ для семейства $Com(\mathbf{N},\mathbf{N})$ называется *главной* универсальной функцией, если выполнено следующее условие:

Для каждой вычислимой функции $g : \mathbf{N}^2 \to \mathbf{N}$ существует тотальная $(D(s) = \mathbf{N})$ вычислимая функция $s : \mathbf{N} \to \mathbf{N}$, для которой

$$(\forall i, n \in \mathbf{N})\ g(i,n) \simeq u(s(i),n).$$

Замечание. Что означает это условие? Предположим, что мы выбрали вместо машин Тьюринга какую-нибудь другую модель вычислений (язык программирования) X, также подходящую для вычисления функций типа $\mathbf{N} \to \mathbf{N}$. Нет принципиальных причин, препятствующих аналогичным построениям. Можно так же построить вычислимую последовательность слов $q_0, q_1, \ldots$, состоящую из всех программ на языке X, а из нее – соответствующую универсальную функцию $g(i,n)$. Единственное возможное отличие в том, что язык X может оказаться не универсальным. Тогда окажется, что функция $g(i,n)$ универсальна в более узком классе – семействе всех функций, вычислимых программами на языке X. Это семейство всех функций $f_i(n) := g(i,n)$ (индекс $i \in \mathbf{N}$ означает номер X-программы, вычисляющей f_i).

Но модель Тюринга – универсальная, поэтому естественно ожидать наличие компилятора с языка X в язык машин Тьюринга. Компилятор переводит программы языка X в программы машин Тьюринга, вычисляющие то же самое. Т.к. преобразования

$$(\text{программа}) \longleftrightarrow (\text{номер программы})$$

обычно вычислимы (в обе стороны), то существование компилятора эквивалентно наличию вычислимой функции, переводящей номера X-программ в номера соответствующих программ машин Тьюринга. Именно это и делает функция s из определения главности. Условие тотальности означает, что компилятор справится с переводом любой X-программы, а выделенная формула – что перевод корректен (откомпилированная программа вычисляет ту же функцию, что и исходная). Универсальная функция u оказывается главной, когда такой компилятор существует для всех теоретически возможных языков X. (Перебирая все вычислимые функции g мы косвенно перебираем все возможные языки X.)

Примеры. (Возможные языки X.) 1. Пусть $g(i,n) := i \cdot n$. Язык X, порождающий такую универсальную функцию (для семейства всех X-вычислимых функций), состоит из программ умножения на данное число i. Программы: “умножить на 0”, “умножить на 1”, . . . ,“умножить на 77”, ... Номером кода программы “умножить на i” является само число i. Этот язык, очевидно, не универсальный.

2. Все написанные на C подпрограммы-функции типа int F{int}, в которых нет обращений к внешним устройствам (клавиатура, экран, диск, и т.п.) и все переменные – локальны. Это перечислимое мн-во слов в достаточно большом алфавите. $g(i,n) :=$ (результат, возвращаемый i-той программой на входе n). Получается универсальный язык программирования.

Теорема 4.4 *Главные универсальные функции существуют. Вычислимая универсальная функция $u : \mathbf{N}^2 \to \mathbf{N}$ (построенная выше по нумерации машин Тьюринга) – главная.*

Доказательство. Пусть $g : \mathbf{N}^2 \to \mathbf{N}$ – вычислимая функция, M – чисто вычисляющая ее машина Тьюринга. Для каждого $i \in \mathbf{N}$ обозначим через $M[i]$ машину, осуществляющую следующее преобразование входного слова $v \in (\Sigma \setminus \{\#\})^*$:

$$v \longmapsto \underbrace{11\ldots1}_{i} \square v$$

Для программирования $M[i]$ достаточно ленточного алфавита Σ. Пусть также обеспечено заключительное перемещение головки влево до первой пустой ячейки. Рассмотрим композицию машин $M[i] \circ M$, построенную как в док-ве теоремы о композиции вычислимых по Тьюрингу функций.

> А именно, производится переименование состояний машины M таким образом, чтобы ее начальное состояние совпало с заключительным состоянием у $M[i]$, а остальные – не пересекались. После чего обе программы объединяются в одну, у которой начальное состояние – начальное состояние $M[i]$, а заключительное – заключительное состояние M.

При $v \in \Sigma^*$ выполняется:

$$M(\bar{i}\square v) \simeq (M[i] \circ M)(v) \simeq U(\, Code(M[i] \circ M)\square v\,) \qquad (1)$$

(Более точно, у всех членов выкладки (1) надо ставить $(\cdots)\big|_{\Sigma\setminus\{\#\}}$, т.е. выделять результат, как слово в алфавите $\Sigma \setminus \{\#\}$, на котором остановилась головка.)

Заметим, что при $v = \bar{n}$, $(n \in \mathbf{N})$ выражение в (1) слева $\simeq \overline{g(i,n)}$, а справа – $\simeq \overline{u(s,n)}$, где

$$s = s(i) := \text{ номер кода } Code(M[i] \circ M).$$

Отсюда $g(i,n) \simeq u(\, s(i), n\,)$. Тотальность и вычислимость функции $s(i)$ следует из ее определения.

■

Тем же способом можно доказать следующий аналог св-ва главности для построенных по машинам Тьюринга вычислимых универсальных функций $u^{(k)}(i, n_1, \ldots, n_k)$ (для классов всех вычислимых функций $\mathbf{N}^k \to \mathbf{N}$).

Теорема 4.5 *Для каждой вычислимой функции $g : \mathbf{N}^{m+n} \to \mathbf{N}$ существует тотальная вычислимая функция $s : \mathbf{N}^m \to \mathbf{N}$, для которой*

$$(\forall i_1, \ldots i_m, x_1 \ldots x_n \in \mathbf{N})\; g(i_1, \ldots i_m, x_1 \ldots x_n) \simeq u^{(k)}(s(i_1, \ldots i_m), x_1 \ldots x_n).$$

Обозначения. Фиксируем построенные выше главные универсальные функции $u^{(1)} = u, u^{(2)}, \ldots$. Пусть $\varphi_i^k(x_1, \ldots, x_k) := u^{(k)}(i, x_1, \ldots, x_k)$. Получим следующую таблицу:

$\varphi_0^1(x)$	$\varphi_1^1(x)$	$\ldots$	$\varphi_i^1(x)$	$\ldots$
$\varphi_0^2(x,y)$	$\varphi_1^2(x,y)$	$\ldots$	$\varphi_i^2(x,y)$	$\ldots$
$\vdots$	$\vdots$		$\vdots$	
$\varphi_0^k(x_1, \ldots, x_k)$	$\varphi_1^k(x_1, \ldots, x_k)$	$\ldots$	$\varphi_i^m(x_1, \ldots, x_k)$	$\ldots$
$\vdots$	$\vdots$		$\vdots$	

(2)

В ней перечислены все вычислимые функции с натуральными аргументами и значениями и только они. Поэтому чтобы доказать, что какая-то функция не является вычислимой, достаточно установить, что ее нет в этой таблице. Для качественной теории вычислимых функций натурального аргумента существенны лишь следующие два ее св-ва:

- Универсальная функция $u^{(k)}$, порождающая нумерацию k-той строки таблицы, встречается в $(k+1)$-ой строке.
- (Обобщенное св-во главности.) Для каждой функции g из $(m+n)$-ой строки найдется тотальная функция s из m-ой строки, такая, что

$$g(i_1, \dots i_m, x_1 \dots x_n) \simeq \varphi^k_{s(i_1,\dots i_m)}(x_1 \dots x_n).$$

5 Отрицательные результаты.

Замечание. Факт существования невычислимых функций $\mathbf{N} \to \mathbf{N}$ немедленно следует из мощностных соображений: функции из первой строки таблицы образуют счетное множество, а все функции указанного типа – континуальное. Интерес представляют явно описанные примеры – примеры определений, которые нельзя запрограммировать.

Определение 5.1 Пусть f – частичная функция. Функция g называется *продолжением* функции f, если $D(f) \subseteq D(g)$ и $g(x) = f(x)$ для любого $x \in D(f)$.

Теорема 5.2 *Существует (частичная) вычислимая функция $f : \mathbf{N} \to \mathbf{N}$, не имеющая тотального вычислимого продолжения.*

Доказательство. Рассмотрим следующую частичную функцию $f : \mathbf{N} \to \mathbf{N}$:

$$f(n) := u(n,n) + 1 = \begin{cases} \varphi_n(n) + 1, & \text{если } !\varphi_n(n); \\ \text{не определено}, & \text{если } \neg!\varphi_n(n). \end{cases}$$

Функция f вычислима, т.к. универсальная функция u вычислима.

Пусть $g : \mathbf{N} \to \mathbf{N}$ – произвольная тотальная вычислимая функция. Докажем, что она не может быть продолжением f.

- Она встречается в первой строке таблицы (2) под некоторым номером m, т.е. $g = \varphi_m$.
- Т.к. g тотальна, то определено значение $g(m) = \varphi_m(m)$.
- Т. к. значение $\varphi_m(m)$ определено, то определено $f(m)$ и $f(m) = \varphi_m(m) + 1$.

Итог: оба значения $g(m)$ и $f(m)$ определены и $f(m) \neq g(m)$. Значит, g не является продолжением f.

■

Вопрос. Какой же пример явного, но не поддающегося программированию, определения мы получили?

Ответ. Например,

$$f(n) := \begin{cases} \varphi_n(n)+1, & \text{если } !\varphi_n(n); \\ 0, & \text{если } \neg!\varphi_n(n). \end{cases}$$

Вопрос. А чем плоха следующая реализация?

$$IF\ \ !\varphi_n(n)\ \ THEN\ \ \varphi_n(n)+1\ \ ELSE\ \ 0$$

Ответ. В ней не сказано, как проверять условие. Попыткой исправить было бы добавление вызова подпрограммы вычисления характеристической функции $\chi_K(n)$ для множества $K = \{n \mid !\varphi_n(n)\}$, если бы множество K оказалось разрешимым.

$$L := \chi_K(n);$$
$$IF\ \ L\ \ THEN\ \ \varphi_n(n)+1\ \ ELSE\ \ 0$$

По предыдущей теореме, эта попытка (как и все другие) обречена на провал. Отсюда получаем

Следствие 5.3 *(!Пример перечислимого, неразрешимого множества.) Множество* $K = \{n \mid !\varphi_n(n)\}$ *– неразрешимое, но перечислимое подмножество* $\mathbf{N}$.

Доказательство. Неразрешимость – см. выше. K перечислимо как область определения вычислимой функции $h(x) := u(x, x)$.

■

Следствие 5.4 *Существует перечислимое множество* $M \subset \mathbf{N}$*, дополнение которого не является перечислимым.*

Доказательство. $M := K$. Это перечислимое множество. Если бы его дополнение оказалось также перечислимым, то, по теореме Поста, K оказалось бы разрешимым. Противоречие.

■

Массовые проблемы самоприменимости и остановки.

Замечание. Задачу разрешения множества K можно переформулировать в следующей форме, известной как одна из формулировок массовой проблемы САМОПРИМЕНИМОСТИ: по номеру n кода машины Тьюринга (слова $p_n = Code(M)$ в алфавите программ) выяснить, остановится ли эта машина на входе $\bar{n}$. Мы только что доказали алгоритмическую неразрешимость этой проблемы.

На самом деле основная формулировка проблемы самоприменимости несколько отличается от приведенной: выяснить, остановится ли машина Тьюринга M на входе $Code(M)$. Задача также алгоритмически неразрешима, но это не является формальным следствием факта, доказанного выше.

Определение 5.5 Массовой проблемой ОСТАНОВКИ называется следующая задача: по коду $Code(M)$ машины Тьюринга (с ленточным алфавитом Σ) и слову v в алфавите $\Sigma_0 \subset \Sigma$ выяснить, остановится ли машина M на входе v. (Будем считать, что $1 \in \Sigma_0$.)

Следствие 5.6 *Множество* $STOP := \{\langle i, n\rangle \mid !\varphi_i(n)\}$ *является перечислимым неразрешимым подмножеством* $\mathbf{N}^2$.

Доказательство. $STOP = D(u)$, а потому, перечислимо. Неразрешимость следует из эквивалентности: $n \in K \Leftrightarrow \langle n, n\rangle \in STOP$.

■

Следствие 5.7 *(Неразрешимость проблемы остановки.) Массовая проблема остановки алгоритмически неразрешима.*

Доказательство. Допустим, что существует алгоритм $\mathcal{A}$, который применим к каждой паре вида $\langle Code(M), v\rangle$, $(v \in \Sigma_0^*)$, заканчивает работу на ней с результатом 0 или 1, причем

$$\mathcal{A}(Code(M), v) = 1 \ \Leftrightarrow !M(v).$$

Напомним, что в построении вычислимой универсальной функции u участвовала вычислиммая последовательность $p_0, p_1, \ldots$, нумерующая коды всех программ машин Тьюринга: p_i – код программы, вычисляющей функцию φ_i. Тогда оказывается, что

$$\chi_{STOP}(i, n) = \mathcal{A}(p_i, \bar{n}).$$

Эта формула дает возможность вычислять характеристическую функцию множества $STOP$. Противоречие.

■

Замечание. Фиксация одного из двух параметров ($Code(M)$ или v) в проблеме остановки не делает ее проще. Например, если в качестве M выбрать машину, вычисляющую полухарактеристическую функцию множества K, то проблема остановки именно этой машины M эквивалентна задаче разрешения множества K, а потому алгоритмически неразрешима. Проблема остановки с фиксированным входным словом $v = v_0$ оказывается также алгоритмически неразрешимой (будет доказано ниже).

Теорема Райса.

Какие свойства вычислимых функций удается распознать с помощью алгоритмов, получающих на вход программы указанных функций? Уже в случае одноместных функций $\mathbf{N} \to \mathbf{N}$ теорема Райса дает исчерпывающий, но крайне неутешительный ответ – никакие, кроме тривиальных.

Каждое такое свойство (одноместный предикат) задает разбиение семейства $Com(\mathbf{N}, \mathbf{N})$ на два класса:

$$\mathcal{X} \cup \mathcal{Y} = Com(\mathbf{N}, \mathbf{N}), \qquad \mathcal{X} \cap \mathcal{Y} = \emptyset. \tag{3}$$

Свойство считается нетривиальным, если оба класса $\mathcal{X}, \mathcal{Y}$ непусты.

Определение 5.8 Пусть $\mathcal{X} \subseteq Com(\mathbf{N}, \mathbf{N})$ – произвольное семейство вычислимых функций. Его индекстным множеством называется множество

$$I_{\mathcal{X}} = \{i \in \mathbf{N} \mid \varphi_i \in \mathcal{X}\}.$$

Иными словами, индексное множество семейства $\mathcal{X}$ состоит в точности из всех номеров всех программ машин Тьюринга, вычисляющих функции из семейства $\mathcal{X}$. Для разбиения (3) индексные множества $I_{\mathcal{X}}$, $I_{\mathcal{Y}}$ образуют разбиение $\mathbf{N}$:

$$I_{\mathcal{X}} \cup I_{\mathcal{Y}} = \mathbf{N}, \qquad I_{\mathcal{X}} \cap I_{\mathcal{Y}} = \emptyset.$$

Теорема 5.9 (теорема Райса) *Если разбиение (3) нетривиально ($\mathcal{X} \neq \emptyset$ и $\mathcal{Y} \neq \emptyset$), то его индексные множества $I_{\mathcal{X}}$, $I_{\mathcal{Y}}$ являются неразрешимыми подмножествами* $\mathbf{N}$.

Доказательство. Пусть ξ — нигде не определенная функция. Она вычислима, а потому попадает в один из классов разбиения, например, в $\mathcal{X}$. Другой класс также непуст. Фиксируем какую-нибудь функцию $h \in \mathcal{Y}$ и перечислимое неразрешимое множество $K \subset \mathbf{N}$ (какое именно – несущественно). Определим (частичную) функцию $g : \mathbf{N}^2 \to \mathbf{N}$ так:

$$g(i,n) := \begin{cases} h(n), & \text{если } i \in K; \\ \text{не определено}, & \text{иначе.} \end{cases}$$

1. Функция g вычислима. Заметим, что прямое переписывание определения функции g с помощью управляющей конструкции `IF THEN ELSE` не дает алгоритма ее вычисления, т.к. условие "$i \in K$" невозможно проверить алгоритмически. Предлагается другой алгоритм:

> Перебирая элементы множества K (оно перечислимо!), осуществить поиск числа i в K. Когда (если) нашли, перейти к вычислению $h(n)$. Когда (если) вычисление завершится результативно, положить $g(i,n) := h(n)$.

2. Универсальная функция, породившая нумерацию $\{\varphi_i\}$, – главная. Поэтому существует тотальная вычислимая функция $s : \mathbf{N} \to \mathbf{N}$, для которой

$$g(i,n) \simeq u(s(i),n) \simeq \varphi_{s(i)}(n).$$

3. Имеет место:

$$\begin{array}{lllll} i \in K & \Rightarrow & \varphi_{s(i)} = h & \Rightarrow & \varphi_{s(i)} \in \mathcal{Y} \;\Rightarrow\; s(i) \in I_{\mathcal{Y}}, \\ i \notin K & \Rightarrow & \varphi_{s(i)} = \xi & \Rightarrow & \varphi_{s(i)} \in \mathcal{X} \;\Rightarrow\; s(i) \in I_{\mathcal{X}}. \end{array}$$

4. Допустим, что какое-нибудь из индексных множеств $I_{\mathcal{X}}$, $I_{\mathcal{Y}}$ разрешимо. Тогда другое тоже разрешимо (как дополнение первого) и характеристическую функцию множества K можно вычислить так:

> Вход: i. Вычислить $s(i)$ и проверить, какое из условий "$s(i) \in I_{\mathcal{Y}}$" или "$s(i) \in I_{\mathcal{X}}$" выполняется. В первом случае вернуть 1, во втором – 0.

Противоречие с неразрешимостью множества K.

■

Примеры свойств вычислимых функций, которые нельзя распознать алгоритмически.

- Для каждой фиксированной вычислимой функции $g : \mathbf{N} \to \mathbf{N}$ свойство "$f = g$". Неразрешимо индексное множество $\{i \mid \varphi_i = g\}$.
- Свойство "$!f(123)$". Неразрешимо индексное множество $\{i \mid !\varphi_i(123)\}$.
- Свойство "f –тотальна". Неразрешимо индексное множество $\{i \mid \forall n\, !\varphi_i(n)\}$.
- Свойство "f – ограничена". Неразрешимо индексное множество

$$\{i \mid \exists a \forall n\, (!\varphi_i(n) \to \varphi_i(n) < a\}.$$

НО множество

$$\{i \mid \text{ программа } p_i \text{ вычисляет } \varphi_i(0) \text{ за} \leq 100 \text{ шагов}\}$$

разрешимо. Оно не является индексным множеством никакого семейства вычислимых функций, поэтому теорема Райса тут не работает.

6 Некоторые приложения

Распознавание истинности арифметических формул.

Рассмотрим множество всех элементарных (первого порядка) формул в языке арифметики (сигнатура $\Omega = \{0, (\cdot)', +, *, =\}$). Условимся записывать переменные x_i в виде слова $x\underbrace{11\ldots1}_{i \text{ раз}}$. Тогда формулы становятся словами некоторого конечного алфавита формул Ю и корректна постановка задач о распознавании свойств формул.

Рассмотрим следующую массовую проблему РАСПОЗНАВАНИЯ ИСТИННОСТИ: по произвольной замкнутой формуле F языка арифметики выяснить, истинна ли она в стандартной интерпретации (на $\mathbf{N}$). Наша цель – показать ее алгоритмическую неразрешимость.

Напомним, что множество $M \subseteq \mathbf{N}$ называется арифметическим, если оно есть область истинности некоторой арифметической формулы $F(x)$, т.е.

$$M = \{n \in \mathbf{N} \mid F(\bar{n}) \text{ – истинно на } \mathbf{N}\},$$

где $\bar{n} = 0\overbrace{'\cdots'}^{n \text{ раз}}$. Арифметические подмножества $\mathbf{N}^k$ определяются аналогично.

Определение 6.1 Формулы вида $\exists x_{i_1} \ldots \exists x_{i_k} G$, где G – без кванторов, называются E-формулами.

Теорема 6.2 *(Без доказательства.) Каждое перечислимое подмножество множества $\mathbf{N}^k$ является арифметическим. Более точно, эквивалентны следующие условия:*

1. *$M \subseteq \mathbf{N}^k$ – перечислимо.*
2. *M есть область истинности некоторой E-формулы.*

Замечание. Просто установить (2) $\Rightarrow$ (1) (Докажите!). Обратная импликация – трудоемкий перевод, записывающий на языке арифметики все подробности определения вычислимости по Тьюрингу функции натурального аргумена и ее области определения.

Следствие 6.3 *Проблема распознавания истинности для языка арифметики алгоритмически неразрешима: множество всех истинных замкнутых формул языка арифметики является неразрешимым подмножеством множества всех формул.*

Доказательство. Допустим, что существует алгоритм $\mathcal{A}$, который применим ко всем формулам, причем

$$\mathcal{A}(G) = \begin{cases} 1, & \text{если } G \text{ замкнута и истинна в } \mathbf{N}, \\ 0, & \text{иначе.} \end{cases}$$

Возьмем арифметическую формулу $F(x)$, у которой область истинности совпадает с неразрешимым, но перечислимым множеством K. Функция $f(n) := \mathcal{A}(\, F(\bar{n})\,)$ вычислима и совпадает с характеристической функцией множества K, что противоречит его неразрешимости. ∎

Диофантово представление и 10-ая проблема Гильберта

В 1900 году на Международном математическом конгрессе в Париже знаменитый немецкий математик Д. Гильберт сформулировал ряд математических проблем, решение которых, по его мнению, наиболее актуально для математики 20-го века. Одна из них, под номером 10, касалась так называемых диофантовых уравнений, т. е. уравнений вида

$$p(x_1, x_2, \ldots, x_m) = 0,$$

где $p(x_1, x_2, \ldots, x_m)$ — диофантов многочлен, т. е. многочлен от переменных $x_1, x_2, \ldots, x_n$ с целыми коэффициентами, причем ищутся только целые решения такого уравнения. Десятая проблема Гильберта состояла в том, чтобы найти алгоритм, с помощью которого можно определить, имеет ли решение произвольное наперед заданное диофантово уравнение. В 1970 году советский математик Ю. В. Матиясевич доказал, что такого алгоритма не существует.

Рассмотрим стандартную интерпретацию сигнатуры $\Omega = \{0, \pm 1, \pm 2, \ldots, +, *, =\}$ на множестве $\mathbf{Z}$.

Определение 6.4 Диофантовым представлением множества $M \subseteq \mathbf{Z}$ называется его представление в виде области истинности формулы

$$\exists y_1 \ldots \exists y_m p(x, y_1, \ldots, y_m) = 0,$$

где p – многочлен от переменных $x, y_1, \ldots, y_m$.

Теорема 6.5 *(Без доказательства.) Каждое перечислимое множество $M \subseteq \mathbf{N}$ имеет диофантово представление.*

Следствие 6.6 *Не существует алгоритма, распознающего разрешимость диофантовых уравнений.*

Доказательство. Возьмем диофантово представление перечислимого неразрешимого множества K:

$$n \in K \Leftrightarrow \exists y_1 \ldots \exists y_m p(n, y_1, \ldots, y_m) = 0.$$

Допустим, что существует алгоритм $\mathcal{A}$, применимый ко всем диофантовым уравнениям (на вход подается набор коэффициентов) и возвращающий 1, когда уравнение имеет решение в целых числах, и 0 – в противном случае. Тогда характеристическую функцию множества K можно вычислить так:

> Вход: $n \in \mathbf{N}$. Привести выражение $p(n, y_1, \ldots, y_n)$ к виду многочлена $q(y_1, \ldots, y_n)$. Применить алгоритм $\mathcal{A}$ и вернуть $\mathcal{A}(\, q(y_1, \ldots, y_n) = 0\,)$ в качестве результата.

Противоречие с неразрешимостью множества K.

■

Для сравнения:

Рассмотрим стандартную интерпретацию сигнатуры $\Omega = \{0, 1, +, *, =\}$ на множестве действительных чисел $\mathbf{R}$. Проблема распознавания истинности формул в этом случае алгоритмически разрешиа (Тарский), но очень сложна (Рабин).

7 Теорема Тарского.

Напомним, что после преобразования имен переменных $x_i \longleftrightarrow x11\ldots1$ формулы языка арифметики становятся словами в фиксированном конечном алфавите

$$Ю = \{0, ', +, *, \neg, \vee, \wedge, \rightarrow, \forall, \exists, x, 1, (,), ", "\}.$$

Занумеруем все слова в этом алфавите в лексикографическом порядке: $w_0, w_1, \ldots$. Тогда отображение $\theta : n \mapsto w_n$ и обратное к нему будут вычислимыми, поэтому алгоритмические свойства (разрешимость, перечислимость) у произвольного множества формул M и соответствующего ему множества номеров формул $\theta^{-1}(M)$ одни и те же.

Определение 7.1 Множесто M формул языка арифметики называется арифметическим, если таковым является множество $\theta^{-1}(M) = \{n \mid w_n \in M\}$, т.е. для некоторой формулы $\varphi(x)$ языка арифметики и всех $n \in \mathbf{N}$ выполняется

$$w_n \in M \Leftrightarrow \varphi(\bar{n}) \text{ истинна в } \mathbf{N}$$

(здесь $\bar{n} = \overbrace{0''\cdots'}^{n \text{ раз}}$ – терм, обозначающий натуральное число n в языке арифметики).

Обозначим через T множество всех истинных (в $\mathbf{N}$) замкнутых формул языка арифметики. Оказывается что оно устроено настолько сложно, что не является арифметическим. Тем более, оно не является ни разрешимым (доказано раньше), ни перечислимым подмножеством множества всех слов в алфавите формул, т.е. не имеет обсуждавшихся ранее алгоритмических описаний.

Теорема 7.2 *(Тарский) Множество T не является арифметическим.*

Доказательство. Для доказательства этой теоремы сначала заготовим специальный пример заведомо неарифметического множества.

Определение 7.3 Пусть $\mathcal{V}$ – некоторое не более чем счетное семейство подмножеств $\mathbf{N}$. Множество $E \subseteq \mathbf{N}$ называется универсальным для семейства $\mathcal{V}$, если выполняются следующие два условия:

а) для каждого $k \in \mathbf{N}$ множество $E_k = \{n \mid (k,n) \in E\}$ (сечение E) принадлежит семейству $\mathcal{V}$;

б) каждое множество $V \in \mathcal{V}$ представимо в виде $V = E_k$ для некоторого $k \in \mathbf{N}$.

Универсальные множества существуют для каждого не более чем счетного семейства множеств и задают всевозможные нумерации этого семейства $\mathcal{V} = \{E_0, E_1, \dots\}$.

В дальнейшем фиксируем семейство

$$\mathcal{V}_{\text{ар}} = \text{все арифметические подмножества } \mathbf{N}.$$

Оно счетно (т.к. множество всех арифметических формул вида $\varphi(x)$ – счетно), поэтому обладает универсальным множеством. Одно из универсальных множеств для $\mathcal{V}_{\text{ар}}$ можно построить так:

$$(k,n) \in E \quad \Leftrightarrow \quad \begin{array}{l}\text{слово } w_k \text{ есть некоторая арифметическая формула} \\ \varphi(x), \text{ в которой нет других свободных переменных,} \\ \text{кроме быть может } x, \text{ и замкнутая формула } \varphi(\bar{n}), \text{ по-} \\ \text{лучающаяся из нее подстановкой терма } \bar{n} \text{ вместо сво-} \\ \text{бодных вхождений } x, \text{ истинна (в } \mathbf{N}).\end{array} \tag{4}$$

Лемма 7.4 *Универсальное множество для семейства $\mathcal{V}_{ap}$ не может быть арифметическим.*

Доказательство. Допустим, что некоторое универсальное множество E для семейства $\mathcal{V}_{\text{ар}}$ оказалось арифметическим: $(k,n) \in E \Leftrightarrow (\psi(\bar{k},\bar{n})$ – истинно) для некоторой элементарной формулы языка арифметики $\psi(x,y)$.

Определим арифметическое множество $A \subset \mathbf{N}$ как область истинности формулы $\neg\psi(x,x)$. Тогда $A = E_k$ для некоторого k. Отсюда для всех n выполняется

$$(\psi(\bar{k},\bar{n}) \text{ – истинно}) \Leftrightarrow (k,n) \in E \Leftrightarrow n \in E_k \Leftrightarrow n \in A \Leftrightarrow (\neg\psi(\bar{n},\bar{n}) \text{ – истинно}).$$

Получили противоречие при $n = k$.

■

Теперь покажем, что описание универсального множества из (4) удается записать элементарной формулой в расширении языка арифметики дополнительным предикатным символом $True(x)$, если интерпретировать его в $\mathbf{N}$ как предикат "$x \in \theta^{-1}(T)$":

$$(True(\bar{n}) - \text{истинно}) \Leftrightarrow n \in \theta^{-1}(T) \Leftrightarrow w_n \in T.$$

В самом деле, множество B всех троек (k, n, m), удовлетворяющих условию

> "слово w_k есть некоторая арифметическая формула $\varphi(x)$, в которой нет других свободных переменных, кроме быть может x, и $w_m = \varphi(\bar{n})$",

является разрешимым подмножеством $\mathbf{N}^3$. Следовательно, B – перечислимое и арифметическое множество. Возьмем арифметическую формулу $\varphi(x, y, z)$, для которой

$$(\varphi(\bar{k}, \bar{n}, \bar{m}) - \text{истинно}) \Leftrightarrow (k, n, m) \in B.$$

Тогда определение (4) можно переписать так

$$(k, n) \in E \Leftrightarrow \exists z\,(\,\varphi(\bar{k}, \bar{n}, z) \wedge True(z)\,) - \text{истинно}. \qquad (5)$$

Завершим доказательство теоремы Тарского. Она утверждает, что нет элементарной арифметической формулы с той же областью истинности, что и у формулы расширенного языка $True(x)$. В самом деле, если бы такая формула $\psi(x)$ была, то ее можно было написать вместо $True(\cdot)$ в правой части (5), что доказывало бы арифметичность универсального множества E. Последнее невозможно по лемме 7.4.

■