Лекция 15

Вычислимые функции. Перечислимые и разрешимые множества

Последним обширным сюжетом нашего курса будет введение в теорию алгоритмов.

Понятие алгоритма кажется современному человеку столь же привычным и понятным, как и понятие числа. Интуиция, возникающая при работе с компьютерами, помогает в изучении теории алгоритмов. Но важно помнить, что мы будем изучать свойства некоторых математических объектов, которые могут оказаться непривычными и даже противоречащими этой интуиции.

Начнём с того, что неформально опишем тот математический объект, который мы будем называть алгоритмом. Это инструкция к выполнению действий, настолько подробная и чёткая, что выполнение этих действий можно поручить компьютеру или другому неразумному механическому устройству. Ближайшим аналогом математического алгоритма в реальной жизни является программа, написанная на каком-нибудь языке программирования для «идеального компьютера» (нет ограничений на размер используемой памяти).

Дать аккуратное математическое определение алгоритма не так просто. Мы поначалу будем обходится без такого определения. Вместо него мы будем использовать, как это часто делается в математике, некоторые свойства, которыми заведомо обладают алгоритмы. На основании этих свойств (их ещё можно назвать аксиомами) мы будем делать выводы и доказывать утверждения об алгоритмах. Кроме того, мы будем использовать явные процедуры, алгоритмический характер которых ясен из опыта работы с компьютерами.

Позже мы увидим, что такой подход к алгоритмам обладает принципиальным недостатком, но пока нам его будет достаточно.

15.1 Вычислимые функции

Мы будем предполагать, что алгоритм получает некоторые входные данные (*вход*), выполняет с ними предписанные действия и (не всегда) заканчивает свою работу,

сообщив *результат* вычислений. Таким образом, алгоритм A *вычисляет функцию*

$$f_A\colon \text{вход} \mapsto \text{результат},$$

и эта функция, вообще говоря, частично определена. На некоторых входах алгоритм может не выдать никакого результата.

Функция называется *вычислимой*, если она вычисляется некоторым алгоритмом. Нас будет интересовать в первую очередь вопрос о том, какие функции вычислимые, а какие — нет.

Наша интуиция подсказывает, что результат работы одного алгоритма можно подать на вход другому алгоритму. Сформулируем это в виде общего свойства алгоритмов.

Свойство алгоритмов 1. *Композиция вычислимых функций вычислима.*

Замечание 15.1. Отметим одно важное, хотя и очевидное, обстоятельство. Пока работа алгоритма не закончена, результат его работы неизвестен (как и то, что алгоритм даёт какой-то результат).

Это согласуется с определением композиции функций: $f \circ g$ не определена во всех точках, в которых не определена g.

Мы пока не обсудили один важный вопрос. Как мы знаем, функция — это отношение на двух множествах и прежде, чем говорить о свойствах функции, нужно указать эти множества. Что является областью определения вычислимой функции? Другими словами, что можно подавать на вход алгоритму и какие результаты он способен выдать?

Есть несколько вариантов ответа на эти вопросы и для нашего дальнейшего анализа годится любой из них.

Наиболее близким к практике реальных вычислений является такой ответ: вычислимая функция — это (частично определённая) функция из множества двоичных слов $\{0,1\}^*$ в множество двоичных слов. (Другими словами, алгоритм получает на вход файл с двоичными данными и возвращает в качестве результата другой файл с двоичными данными.)

Важным обстоятельством является бесконечность множества двоичных слов. Именно здесь бытовое понятие алгоритма начинает заметно отличаться от математического.

Пример 15.1. Любая функция с конечной областью определения вычислима.

Действительно, легко представить себе программу, которая содержит таблицу значений такой функции (эта таблица конечна). Вычисления выполняются следующим образом: входное слово ищется среди списка слов, составляющих область определения функции. Если оно обнаруживается в этом списке, то алгоритм выдаёт в качестве результата записанное в таблице значение этой функции. Если же нет, то алгоритм не останавливается и тем самым уклоняется от выдачи какого-либо результата. (Заставить программу не останавливаться очень легко: в качестве

упражнения придумайте, как этого добиться в вашем любимом языке программирования[1].)

Этот несложное рассуждение с практической точки зрения выглядит странно. Все компьютеры, доступные человечеству, выполнят лишь конечное количество действий в XXI веке. Поэтому мы вправе утверждать, что результат компьютерной деятельности человечества в XXI веке является вычислимой функцией, хотя сейчас мы ничего не знаем о тех вычислениях, которые будут выполняться в 2100 году.

Мы будем использовать и другие классы функций. Например, зачастую удобно использовать функции из натуральных чисел в натуральные. Такое изменение класса функций для наших целей несущественно. Действительно, множество натуральных чисел равномощно множеству двоичных слов. Более того, нетрудно построить вычислимую биекцию между этими множествами.

Задача 15.2. Определим функцию $f\colon \mathbb{N} \to \{0,1\}^*$ следующим образом: $f(n)$ — это двоичное слово, которое получается из двоичной записи числа $n+1$ отбрасыванием слева всех нулей и первой единицы.

Докажите, что эта функция является биекцией между множествами $\mathbb{N}$ и $\{0,1\}^*$.

Вычислимость функции f не вызывает сомнений: для её реализации нужно уметь прибавлять к числу единицу, строить по числу его двоичную запись и находить крайнюю слева единицу. Все эти действия легко записать в виде однозначно понимаемых инструкций (например, написать программу на YFPL.).

Задача 15.3. Опишите последовательность действий, которые требуются для вычисления обратной функции $g = f^{-1}$.

Используя функции f и обратную к ней g, легко преобразовать алгоритмы на двоичных словах в алгоритмы на натуральных числах и наоборот:

$$\mathbb{N} \xrightarrow{f} \{0,1\}^* \xrightarrow{A} \{0,1\}^* \xrightarrow{g} \mathbb{N}, \qquad \{0,1\}^* \xrightarrow{g} \mathbb{N} \xrightarrow{B} \mathbb{N} \xrightarrow{f} \{0,1\}^*.$$

Аналогичный приём работает и для преобразования алгоритмов с другими входами/результатами в алгоритмы на натуральных числах. Всё, что для этого нужно существование вычислимых биекций, обратная к которым тоже вычислима.

Задача 15.4. Докажите, что функция

$$c\colon (x,y) \mapsto \binom{x+y+1}{2} + y$$

является вычислимой вместе с обратной биекцией между множествами $\mathbb{N} \times \mathbb{N}$ и $\mathbb{N}$.

Задача 15.5. Для любого k опишите вычислимую биекцию между множествами $\mathbb{N}^k$ и $\mathbb{N}$.

[1] В дальнейшем мы будем называть этот язык YFPL.

Все ли функции из $\mathbb{N}$ в $\mathbb{N}$ вычислимы? Нет, и причина тому очень проста. Как уже говорилось, алгоритмы похожи на программы. А программа — это текст на некотором языке программирования, то есть конечная последовательность символов. Поэтому множество алгоритмов счётно. Но всех функций $\mathbb{N} \to \mathbb{N}$ несчётно много. Значит, некоторые функции невычислимы.

Наш основной интерес состоит в том, чтобы научиться различать вычислимые и невычислимые функции. Мощностное рассуждение, приведённое выше, не слишком помогает в этом, оно лишь говорит, что не все функции вычислимы. Дальше у нас появятся более интересные способы доказательства невычислимости функций.

15.2 Универсальные вычислимые функции

Помимо того, что алгоритм задаётся конечным текстом, сама интерпретация этого текста также реализуется алгоритмом. Например, таким алгоритмом является интерпретатор с YFPL.

Другими словами, вычислима функция из $\mathbb{N} \times \mathbb{N}$ в $\mathbb{N}$, которая ставит в соответствие паре (алгоритм p, вход x) результат применения алгоритма p ко входу x.

Свойство алгоритмов 2. *Существует такая вычислимая функция $U\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, что для любой вычислимой функции $f\colon \mathbb{N} \to \mathbb{N}$ найдётся такое p, что $U(p, x) = f(x)$ для всех x.*

Здесь равенство понимается в том смысле, что левая и правая части равенства определены на одном и том же множестве и принимают одинаковые значения, если они определены.

Функцию, удовлетворяющую свойству 2, будем называть *универсальной*. Поскольку мы сейчас говорим о функциях из $\mathbb{N}$ в $\mathbb{N}$, то будем также называть универсальную функцию универсальной нумерацией вычислимых функций.

Разумеется, универсальных функций много (языков программирования много больше одного). Однако любая такая функция «содержит» все вычислимые. Поэтому с некоторой натяжкой можно сказать, что изучение класса всех вычислимых функций состоит в изучении одной-единственной функции (универсальной вычислимой). Впрочем, пользы от такого наблюдения немного: любая универсальная вычислимая функция устроена очень сложно.

Более того, используя универсальную функцию, легко строить примеры невычислимых функций. Для этого применяется диагональное рассуждение.

Зафиксируем какую-нибудь универсальную функцию U и определим функцию $h_U(x)$ следующим соотношением:

$$h_U(x) = \begin{cases} 1, \text{ если } U(x, x) = 0, \\ 0, \text{ в противном случае.} \end{cases} \tag{15.1}$$

Теорема 15.1. *Функция h_U невычислима.*

Доказательство. От противного. Если функция вычислима, ей отвечает некоторый номер в универсальной нумерации, то есть $h_U(x) = U(p, x)$ для некоторого p. Поскольку h_U — всюду определённая функция, то значение $U(p, p)$ определено.

Пусть $U(p, p) = 0$. Тогда, по определению (15.1), $h_U(p) = 1$. Значит, $h_U(p) \neq U(p, p)$.

Аналогично рассуждаем в случае $U(p, p) \neq 0$. Тогда по (15.1) получаем $h_U(p) = 0 \neq U(p, p)$.

Пришли к противоречию. □

Замечание 15.2. Почему рассуждение в доказательстве теоремы 15.1 называется диагональным? Представим таблицу значений функции U: это бесконечная целочисленная матрица, строки которой занумерованы первыми аргументами (алгоритмами, они же программы), а столбцы — вторыми аргументами. На диагонали этой матрицы стоят значения $U(x, x)$. Функция $d(x)$ определяется так, чтобы отличаться от любой строки матрицы аналогично доказательству теоремы Кантора о несчётности множества бесконечных двоичных последовательностей. Для этого достаточно использовать только диагональ матрицы.

Задача 15.6. Докажите, что не существует универсальной нумерации вычислимых всюду определённых функций, то есть такой всюду определённой вычислимой функции $U\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, что для любой вычислимой всюду определённой функции $f\colon \mathbb{N} \to \mathbb{N}$ найдётся такое p, что $U(p, x) = f(x)$ для всех x.

Функция g называется *продолжением* функции f, если на области определения f выполняется равенство $f(x) = g(x)$. Аналогично теореме 15.1 из любой универсальной функции U можно построить пример вычислимой функции, у которой нет всюду определенного вычислимого продолжения. Определим $\tilde{h}_U$ как

$$\tilde{h}_U(x) = \begin{cases} 1, \text{ если } U(x, x) = 0, \\ 0, \text{ если } U(x, x) \text{ определена и } U(x, x) \neq 0, \\ \quad \text{в остальных случаях не определена.} \end{cases} \tag{15.2}$$

Эта функция вычислима: подадим на вход алгоритма, вычисляющего U, пару аргументов (x, x); после остановки этого алгоритма выдадим 1, если результат вычисления U равен 0, и 0 в противном случае. Если $U(x, x)$ не определена, то данный алгоритм также не выдаёт никакого результата.

Следствие 15.2. *У функции $\tilde{h}_U$ нет всюду определённого вычислимого продолжения.*

Доказательство. От противного. Пусть $g(x)$ — всюду определённое вычислимое продолжение функции $\tilde{h}_U$. Выберем такое p, что $g(x) = U(p, x)$ и придём к противоречию аналогично доказательству теоремы (15.1): $U(p, p)$ определена, поэтому $g(p) = \tilde{h}_U(p)$; если $\tilde{h}_U(p) = 1$, то $U(p, p) = 0 \neq \tilde{h}_U(p) = g(p)$; если же $\tilde{h}_U(p) = 0$, то $U(p, p) \neq 0 = \tilde{h}_U(p) = g(p)$.

В любом случае получаем $U(p, p) \neq g(p)$, что противоречит выбору p. □

Задача 15.7. Пусть U — универсальная вычислимая функция. Докажите, что $U(p,p)$ не определено для некоторого p.

Задача 15.8. Докажите, что для любой универсальной вычислимой функции U множество $\{U(p,p) : p \in \mathbb{N}\}$ совпадает с $\mathbb{N}$.

Разумеется, выбор диагонали в этих задачах не очень важен. То же самое справедливо, например, для множества пар (x, x^2).

Ещё одно простое наблюдение о вычислимых функциях, которое следует из этих рассуждений, относится к скорости роста вычислимых функций. Ясно, что уже значение $f(0)$ может быть сколь угодно велико для вычислимой функции. Функции

$$2^x,\ 2^{2^x},\ \ldots, 2^{2^{\cdots 2^x}}$$

являются вычислимыми, сколько этажей степеней двоек не написать. Более того, есть вычислимые функции, которые растут гораздо быстрее. И тем не менее скорость роста вычислимой функции не может быть сколь угодно велика.

Теорема 15.3. *Существует такая функция f, которая растёт быстрее любой вычислимой функции.*

Формально это означает, что для любой вычислимой функции g найдётся такое N, что $f(x) > g(x)$ для всех $x > N$, принадлежащих области определения g.

Доказательство. Фиксируем универсальную функцию U и определим f как

$$f(x) = 1 + \max_{p,y \leqslant x} U(p,y).$$

Здесь максимум берётся по тем парам p, y, для которых универсальная функция определена.

Докажем, что f растёт быстрее любой вычислимой функции g. В силу универсальности U для некоторого p имеем равенство функций $g(x) = U(p,x)$. Тогда при $x > p$ из области определения функции g получаем $f(x) > U(p,x) = g(x)$, что и требовалось. □

15.3 Перечислимые и разрешимые множества

Понятие алгоритма, определённое выше, позволяет определить не только класс вычислимых функций, но и два класса множеств — перечислимые и разрешимые. Сделать это можно двумя способами — как принято в теоретической информатике и как принято в математике. Получаются почти эквивалентные определения.

Первый способ состоит в том, что мы рассматриваем множества как унарные отношения. Говоря по-простому, нас интересует некоторое свойство натуральных чисел (скажем, «число простое» или «число в десятичной записи записывается только цифрами 0, 1, 2») и возможность алгоритмической проверки этого свойства. Алгоритм, который проверяет некоторое свойство натуральных чисел, получает на вход

число x и даёт ответ на вопрос «обладает ли x данным свойством?» Ответов возможно два: «да» или «нет». Удобно зафиксировать числовые значения для этих ответов. Пусть ответу «да» отвечает число 1, а ответу «нет» — число 0. Искомый алгоритм проверки свойства должен давать ответ для каждого числа. Тем самым этот алгоритм вычисляет всюду определённую функцию из $\mathbb{N}$ в $\{0,1\}$. Это индикаторная функция χ_S множества тех чисел, которые удовлетворяют данному свойству.

Получаем формальное определение.

Определение 15.1. Множество S называется *разрешимым*, если его характеристическая функция χ_S вычислима.

Алгоритм, вычисляющий индикаторную функцию множества S, будем называть *алгоритмом разрешения множества* S.

Это определение легко распространить на подмножества $\mathbb{N}^2$, $\{0,1\}^*$ и т.п. Тут мы следуем уже использованному приёму: алгоритмической перекодировке. Пусть есть вычислимая биекция f из множества $\mathbb{N}$ в множество X. Тогда подмножество $S \subseteq X$ называется перечислимым, если оно является образом $f(S')$ некоторого перечислимого множества, и называется разрешимым, если оно является образом $f(S')$ некоторого разрешимого множества. Аналогично поступаем и в других случаях.

Утверждение 15.4. *Любое конечное множество разрешимо.*

Алгоритм разрешения конечного множества S аналогичен алгоритму из примера 15.1: алгоритм содержит таблицу элементов множества S, вход сравнивается по очереди со всеми элементами таблицы; в случае совпадения выдаётся 1, если ни одного совпадения не обнаружено, выдаётся 0.

Задача 15.9. Докажите, что если A, B — разрешимые множества, то и множества $A \cup B$, $A \cap B$, $\bar{A}$ разрешимы.

Если множество конечно, то его можно задать списком элементов. Для бесконечных множеств есть аналогичный способ задания. А именно, представим алгоритм, который не имеет входных данных и печатает по мере работы некоторый список чисел. Алгоритм не обязан останавливаться, поэтому он может напечатать и бесконечное множество чисел. Такой процесс будем называть *перечислением* множества. При перечислении некоторые элементы могут повторяться. На самом деле это неважно.

Задача 15.10. Докажите, что если существует алгоритм перечисления элементов некоторого множества, то существует также и алгоритм, который перечисляет элементы множества без повторений.

Определение 15.2. Множество S называется *перечислимым*, если есть алгоритм перечисления его элементов.

Пример 15.11. Пустое множество перечислимо. Алгоритм перечисления пустого множества не печатает ни одного числа.

Понятия перечислимого и счётного множества выглядят похоже, но важно понимать разницу между ними. Как мы уже обсуждали, всякое подмножество множества натуральных чисел конечно или счётно. Но перечислимы далеко не все из них. Действительно, множество алгоритмов перечисления счётно. А бесконечных подмножеств множества натуральных чисел несчётное количество (это разность несчётного и счётного множества).

Множество разрешимых подмножеств натуральных чисел также счётно. Поэтому существуют неразрешимые множества. Впрочем, существование неразрешимых множеств следует из существования неперечислимых.

Утверждение 15.5. *Если A разрешимо, то оно перечислимо.*

Доказательство. Алгоритм перечисления множества A использует алгоритм разрешения множества A. Он перебирает все числа, начиная с 0; для каждого числа n вычисляет индикаторную функцию $\chi_A(n)$ и печатает число n, если полученное значение равно 1.

Корректность такого алгоритма очевидна из определений. □

Обратите внимание, что алгоритм перечисления разрешимого множества, описанный выше, перечисляет его элементы в возрастающем порядке. Верно и обратное.

Задача 15.12. Докажите, что если существует алгоритм перечисления элементов множества S в возрастающем порядке, то это множество разрешимо.

Итак, разрешимые множества содержатся среди перечислимых. Эти два класса не совпадают. Однако доказать их несовпадение мощностными соображениями не получится: оба класса содержат счётное количество множеств. Пример перечислимого неразрешимого множества будет построен ниже диагональным методом.

Задача 15.13. Докажите, что если A, B — перечислимые множества, то и множества $A \cup B$, $A \cap B$ перечислимы.

Обратите внимание на разницу между задачами 15.9 и 15.13: дополнение к перечислимому множеству не обязано быть перечислимым. Можно даже уточнить, в каких случаях перечислимы одновременно и множество, и его дополнение. Это возможно лишь в том случае, когда множество (значит, и его дополнение) разрешимы.

Для доказательства этого факта нам потребуется ещё одно свойство алгоритмов. Напомним, что алгоритм — это инструкция по выполнению действий. Каждый шаг работы алгоритма — это очень простое действие и естественно предположить, что мы можем выделить этот шаг.[2]

Разбиение алгоритма на шаги позволяет организовать *параллельное* исполнение алгоритмов: по очереди исполняется шаг работы каждого алгоритма. Пример использования такой процедуры — доказательство следующей теоремы.

[2]На самом деле, это не всегда так: есть такие способы определения алгоритмов, в которых не очень понятно, что такое шаг работы алгоритма. Но мы пока эти тонкости опустим — для нас важнее, что есть и такие способы определения алгоритмов, для которых шаги работы выделяются без проблем.

Теорема 15.6 (теорема Поста)**.** *Если множества A и $\bar{A}$ перечислимы, то множество A разрешимо.*

Доказательство. Алгоритм разрешения множества A устроен так. Он исполняет модифицированные алгоритмы перечисления множеств A и $\bar{A}$ параллельно: один шаг работы алгоритма перечисления множества A, затем один шаг работы алгоритма перечисления $\bar{A}$ и т.д.

Вместо того, чтобы печатать очередной элемент, модифицированный алгоритм перечисления запоминает его в списке элементов множества. (В любой момент исполнения алгоритма такой список конечен.)

Когда один из списков увеличивается, добавленный элемент сравнивается со входом x. Если обнаружено вхождение x в список элементов множества A, то алгоритм разрешения останавливается и выдаёт результат 1. Если обнаружено вхождение x в список элементов множества $\bar{A}$, то алгоритм разрешения останавливается и выдаёт результат 0. В остальных случаях продолжается работа алгоритмов перечисления.

Докажем корректность алгоритма. Пусть $x \in A$. Тогда x заведомо не входит в список элементов $\bar{A}$ и результат 0 невозможен. С другой стороны, рано или поздно x появится в списке элементов A, поэтому алгоритм выдаст результат 1.

Аналогично рассуждаем в случае $x \notin A$. □

Замечание 15.3. В этом доказательстве существенно, что параллельное исполнение алгоритмов состоит в поочередном исполнении шага работы каждого алгоритма. Для доказательства теоремы Поста можно предложить другой порядок действий: переключение между алгоритмами происходит в момент увеличения списка элементов множества или его дополнения.

При таком порядке действий приведенное доказательство становится неверным. Если множество конечно (или его дополнение конечно), то один из списков в некоторый момент перестанет увеличиваться и переключения на другой алгоритм не случится.

Тем не менее возможно поправить доказательство для этого случая. Действительно, для бесконечного множества с бесконечным дополнением доказательство остаётся корректным. А конечное множество разрешимо, равно как и множество с конечным дополнением.

15.4 Перечислимые множества в терминах вычислимых функций

Какие множества естественно связаны с некоторым классом функций из $\mathbb{N}$ в $\mathbb{N}$? Есть по меньшей мере 4 (вообще говоря, различных) класса множеств:

1. множества значений функций из данного класса;
2. множества значений всюду определённых функций из данного класса;
3. области определения функций из данного класса;

4. графики функций, то есть подмножества $\mathbb{N} \times \mathbb{N}$ вида $\Gamma_f = \{(x, y) : y = f(x)\}$, где f принадлежит данному классу функций.

В случае вычислимых функций эти классы множеств выражаются через перечислимые множества.

Начнём с двух простых наблюдений.

Утверждение 15.7. *Если множество S перечислимо, то оно является множеством значений некоторой вычислимой функции.*

Доказательство. Пусть S — перечислимо. Превратим алгоритм перечисления его элементов в алгоритм вычисления некоторой функции: на входе n алгоритм запускает алгоритм перечисления элементов S и подсчитывает количество «напечатанных» элементов (печатать элементы ему для этого необязательно). Когда напечатан n-й элемент, алгоритм выдаёт его в качестве результата работы и останавливается.

Множеством значений такой функции будут в точности числа, которые печатает алгоритм перечисления, то есть множество S. □

Утверждение 15.8. *Если множество S является множеством значений всюду определённой вычислимой функции, то оно перечислимо.*

Доказательство. Пусть $S = f(\mathbb{N})$ для некоторой всюду определённой функции f. Алгоритм перечисления множества S устроен так: для каждого натурального числа, начиная с 0, n вычисляем $f(n)$ и печатаем результат. □

Задача 15.14. Какое множество перечисляет алгоритм из доказательства утверждения 15.8, если f не всюду определена?

Мы уже использовали в доказательстве теоремы Поста разбиение работы алгоритма на шаги. В доказательствах нам будет удобно использовать формальное определение количества шагов алгоритма.

Свойство алгоритмов 3. *Для некоторой универсальной вычислимой функции $U\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ существует всюду определённая вычислимая функция $F\colon \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \{0, 1\}$ (отладочная функция), обладающая следующими свойствами:*

- *при любых p, x функция $F(p, x, t)$ — неубывающая функция от аргумента t;*
- *$U(p, x)$ не определена тогда и только тогда, когда $F(p, x, t) = 0$ для всех t.*

Неформально говоря, значение функции $F(p, x, t)$ равно 0 тогда и только тогда, когда программа p на входе x не закончила работу за количество шагов t. В противном случае значение функции $F(p, x, t)$ равно 1.

Теорема 15.9. *Для непустого множества $S \subseteq \mathbb{N}$ следующие свойства равносильны:*

1. *S — перечислимое;*
2. *S — множество значений некоторой вычислимой функции;*

3. *S — множество значений некоторой всюду определённой вычислимой функции;*

4. *S — область определения некоторой вычислимой функции.*

Доказательство. 1) $\Rightarrow$ 2) — это утверждение 15.7.

2) $\Rightarrow$ 3). Пусть $S = f(\mathbb{N})$ для некоторой вычислимой функции f. Зафиксируем некоторое $a \in S$ (мы предполагаем, что множество S непусто).

Функция f не всюду определена и может не иметь вычислимого всюду определённого продолжения. Поэтому используем отладочную функцию.

Пусть $f(x) = U(p, x)$, где U — некоторая универсальная вычислимая функция, для которой существует отладочная функция.

Опишем алгоритм вычисления всюду определённой функции $g\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. На входе (x, t) алгоритм вычисляет $F(p, x, t)$ и сравнивает результат с 1. Если $F(p, x, t) = 1$, то алгоритм выдаёт $U(p, x)$. В противном случае результат равен a.

По определению отладочной функции из $F(p, x, t) = 1$ следует, что $U(p, x)$ определена. Поэтому функция g всюду определена. Её множество значений совпадает с S. В одну сторону: если $y = g(x, t)$, то $y = a \in S$ или $y = U(p, x) = f(x) \in S$. В другую: пусть $y = f(x) = U(p, x)$. На паре (p, x) функция U определена, поэтому для некоторого t значение отладочной функции $F(p, x, t) = 1$. Но тогда $y = g(x, t)$.

Мы представили S как множество значений всюду определённой функции от двух натуральных аргументов. Чтобы перейти к функциям одного аргумента, используем вычислимую биекцию $c\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ и выразим S как $S = g \circ c^{-1}(\mathbb{N})$.

3) $\Rightarrow$ 1) — это утверждение 15.8. Таким образом, первые три свойства равносильны.

3) $\Rightarrow$ 4). Пусть $S = f(\mathbb{N})$ для некоторой всюду определённой функции f. Опишем алгоритм вычисления функции g, который получает на вход x: перебираем все числа, начиная с 0; для каждого числа n вычисляем $f(n)$ и сравниваем с x; в случае равенства выдаём результат 1.

Если $x \in S$, то $g(x) = 1$, так как $x = f(n)$ для некоторого n.

Если $x \notin S$, то $g(x)$ не определена, так как для любого n выполняется $x \neq f(n)$, то есть алгоритм вычисления g не даёт никакого результата.

4) $\Rightarrow$ 2). Пусть S — область определения некоторой вычислимой функции f, а p — номер программы, вычисляющей f. Опишем алгоритм вычисления функции g из $\mathbb{N} \times \mathbb{N}$ в $\mathbb{N}$ на входе (x, t): вычисляем $F(p, x, t)$ и сравниваем с 1; если $F(p, x, t) = 1$, то выдаём результат x, иначе не выдаём никакого результата.

Если $x \in S$, то $x = g(x, t)$ для некоторого t. И обратно, если $x = g(x, t)$ для некоторого t, то $U(p, x)$ определена, а значит, определена и $f(x)$.

Мы представили S как множество значений функции от двух натуральных аргументов. Чтобы перейти к функциям одного аргумента, используем вычислимую биекцию $c\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ и выразим S как $S = g \circ c^{-1}(\mathbb{N})$. □

15.5 Проекции и графики

Проекцией множества $D \subseteq \mathbb{N} \times \mathbb{N}$ назовём множество $\operatorname{Пр} D = \{x : \exists y (x, y) \in D\}$. Название становится понятным, если нарисовать условную картинку на декартовой плоскости.

Теорема 15.10. *Перечислимые множества — это в точности проекции разрешимых.*

Доказательство. Проекция пустого множества пуста. Дальше рассматриваем только непустые множества.

Пусть $D \subseteq \mathbb{N} \times \mathbb{N}$ разрешимо, $(a, b) \in D$. По определению, индикаторная функция χ_D вычислима. Но тогда вычислима и функция

$$f(x, y) = \begin{cases} x, & \text{если } \chi_D(x, y) = 1, \\ a, & \text{в противном случае,} \end{cases}$$

для которой $f(\mathbb{N} \times \mathbb{N}) = \operatorname{Пр} D$. Поэтому $\operatorname{Пр} D$ перечислимо (как всегда, нужно ещё позаботиться о замене функций от двух аргументов функцией от одного аргумента).

Пусть S — перечислимо. Тогда S — область определения некоторой вычислимой функции f, которая имеет номер p в универсальной нумерации. Построим такое множество:

$$D = \{(x, t) : F(p, x, t) = 1\}.$$

Докажем, что $\operatorname{Пр} D = S$. Пусть $x \in S$. Это значит, что на x функция f определена, тем самым определена и функция $U(p, x)$. Но тогда по определению отладочной функции $F(p, x, t) = 1$ для некоторого t, то есть $x \in \operatorname{Пр} D$.

В обратную сторону аналогично: если $x \in \operatorname{Пр} D$, то для некоторого t выполняется $F(p, x, t)$, то есть $U(p, x) = f(x)$ определена. Таким образом, $x \in S$. □

Теорема 15.11. *Функция вычислима тогда и только тогда, когда её график перечислим.*

Доказательство. Ясно, что график является образом $f(\mathbb{N})$ функции из $\mathbb{N}$ в $\mathbb{N} \times \mathbb{N}$, которая определена как $x \mapsto (x, f(x))$. Поэтому график вычислимой функции перечислим.

Пусть график $\Gamma = \{(x, y) : y = f(x)\}$ перечислим. Тогда следующий алгоритм вычисляет f: на входе x запускаем перечисление элементов графика; когда найден очередной элемент (y, z), проверяем $x = y$; в случае равенства выдаём результат z.

Из определения графика ясно, что на входе x такой алгоритм может выдать лишь результат $f(x)$. С другой стороны, если x принадлежит области определения f, то пара $(x, f(x))$ рано или поздно будет перечислена. В этот момент алгоритм и выдаст результат $f(x)$. □

15.6 Перечислимые неразрешимые множества

Из доказанных выше теорем очень легко построить пример перечислимого неразрешимого множества.

Пусть U — некоторая универсальная функция. Обозначим через H множество

$$\{x : U(x, x) \text{ определена}\}.$$

Теорема 15.12. *H перечислимо, но неразрешимо.*

Доказательство. Множество H является областью определения вычислимой функции $x \mapsto U(x, x)$. Поэтому оно перечислимо.

Неразрешимость доказывается диагональным аргументом. Предположим, что H разрешимо. Рассмотрим такой алгоритм вычисления функции f: на входе x он вызывает алгоритм разрешения множества H для x. Если $x \in H$, то алгоритм не даёт никакого результата, а если $x \notin H$, то выдаёт результат 1.

Пусть p — номер функции f в универсальной нумерации, то есть $f(x) = U(p, x)$.

Предположим, что $p \in H$. Тогда алгоритм вычисления f, описанный выше, не выдаёт никакого результата, то есть $f(p) = U(p, p)$ не определено. По определению множества H это означает, что $p \notin H$.

Если $p \notin H$, то алгоритм вычисления f даёт результат 1, то есть $1 = f(p) = U(p, p)$. Следовательно, $p \in H$.

Пришли к противоречию. Значит, множество H неразрешимо. □

Из теоремы Поста получаем простое следствие.

Следствие 15.13. *Множество $\bar{H}$ неперечислимо.*

Заметим, что любое множество

$$H_a = \{x : U(x, x) = a\}$$

также перечислимо как область определения следующей вычислимой функции:

$$f_a = \begin{cases} 1, & \text{если } U(x, x) = a, \\ & \text{не определена в противном случае.} \end{cases}$$

Каждое такое множество также неразрешимо. На самом деле, выполняется даже более сильное свойство: при $a \neq b$ множества H_a и H_b *неотделимы*. Это значит, что не найдётся такого разрешимого множества D, для которого

$$H_a \subseteq D, \quad H_b \cap D = \varnothing.$$

Следствие 15.14. *При $a \neq b$ множества H_a и H_b неотделимы.*

Доказательство. Предположим, D — разрешимое множество, которое отделяет H_a и H_b. Рассмотрим такой алгоритм вычисления функции f: на входе x он вызывает алгоритм разрешения множества D для x. Если $x \in D$, то алгоритм даёт результат b, а если $x \notin D$, то выдаёт результат a.

Пусть p — номер функции f в универсальной нумерации, то есть $f(x) = U(p, x)$.

Предположим, что $p \in D$. Тогда $b = f(p) = U(p, p)$ и $p \in H_b$, то есть $p \notin D$.

Если $p \notin H$, то алгоритм вычисления f даёт результат $a = f(p) = U(p, p)$, то есть $p \in H_a \subseteq D$. Следовательно, $p \in D$.

Пришли к противоречию. Значит, пара H_a, H_b неотделима. □

Ещё одним примером перечислимых неразрешимых множеств являются *универсальные перечислимые множества*, Множество $W \subseteq \mathbb{N} \times \mathbb{N}$ называется универсальным, если для любого перечислимого множества S найдётся номер p такой, что

$$S = \{x : (p, x) \in W\}.$$

(Определение универсального подмножества натуральных чисел получается применением вычислимой биекции.)

По универсальной функции U легко построить универсальное перечислимое множество. Это просто область определения U.

Утверждение 15.15. *Область определения универсальной функции является универсальным перечислимым множеством.*

Доказательство. Пусть S перечислимо. Тогда S — область определения некоторой вычислимой функции f. Обозначим через p номер этой функции в нумерации U. По определению получаем

$$S = \{x : U(p, x) \text{ определена}\},$$

что и требовалось. □

Теорема 15.16. *Универсальное перечислимое множество неразрешимо.*

Доказательство. Если бы универсальное перечислимое множество было разрешимым, то из алгоритма его разрешения получался бы алгоритм разрешения любого перечислимого множества (первым элементом пары такой алгоритм получает номер перечислимого множества в данной нумерации). □

Универсальную вычислимую функцию удобно представлять как язык программирования: первый аргумент p задаёт программу, второй аргумент x — вход, а $U(p, x)$ — результат применения программы.

Такой взгляд на универсальные функции на самом деле слишком ограничителен: бывают очень странные универсальные функции, которые непохожи на настоящие языки программирования.

Мы выделим те универсальные функции, которые похожи на языки программирования, и назовём их главными. Мы увидим, что многие свойства обычных языков программирования выражаются через свойства главных универсальных функций.

15.7 Главные нумерации

Пусть имеется какая-то вычислимая функция $V(q, y)$ от двух аргументов. Её также можно воспринимать как язык программирования: q это программа, а y — входные данные для этой программы. Этот язык программирования необязательно универсальный — среди функций $f_q(y) = V(q, y)$, которые вычислимы программами в этом языке, могут содержатся не все вычислимые функции.

Возьмём универсальную вычислимую функцию $U(p, x)$, которая представляет какой-нибудь обычный язык программирования. На этом языке можно написать программу с двумя входами q, y, которая вычисляет функцию $V(q, y)$. Чтобы получить программу, которая вычисляет какую-нибудь функцию $f_n(y) = V(n, y)$, вычислимую на языке программирования V, достаточно в текст программы, вычисляющей $V(q, y)$, добавить определение константы $q = n$. Полученная таким преобразованием программа вычисляет функцию f_n. При этом само преобразование программ, разумеется, вычислимо (в сущности, нужно к тексту программы добавить в подходящем месте одну конкретную строчку, такое действие по силам компьютеру).

Таким образом, от универсального языка программирования $U(p, x)$ естественно ожидать, что существует *транслятор* с языка V на язык U. Транслятор — это алгоритм s, который преобразует программу q на языке V в программу $s(q)$ на языке U, вычисляющую ту же самую функцию. При этом результат работы транслятора определён для всех V-программ.

Дадим формальное определение.

Определение 15.3. Универсальная функция $U(p, x)$ называется *главной* (или гёделевой), если для любой вычислимой функции $V(q, y)$ существует *транслятор* $s(q)$ — вычислимая всюду определённая функция, для которой выполняется

$$V(q, y) = U(s(q), y)$$

для всех q, y.

Замечание 15.4. Напомним, что мы также называли универсальные вычислимые функции *нумерациями* (так как они нумеруют все вычислимые функции).

Дальше для краткости мы часто будем называть главные универсальные вычислимые функции *главными нумерациями*, а для наглядности мы будем называть их языками программирования.

Мы уже объяснили причины, по которым главные универсальные функции должны существовать. Но оказывается, что можно доказать существование главных универсальных функций, исходя из существования какой-нибудь универсальной функции.

Теорема 15.17. *Если существует универсальная вычислимая функция, то существует и главная универсальная вычислимая функция.*

Доказательство. Идея построения главной функции довольно проста и следует описанному выше неформальному объяснению. Нужно построить вычислимую нумерацию всех функций от двух аргументов. После этого «программа» в главной нумерации будет состоять из пары «номер функции двух аргументов, первый аргумент»[3].

Для соединения пары чисел в одно и развёртывания числа в пару мы будем использовать вычислимую биекцию $c\colon \mathbb{N}\times\mathbb{N} \to \mathbb{N}$ (нумерацию всех пар натуральных чисел) и обратную к ней, которую будет удобно разбить на две компоненты: $c^{-1}(x) = (\pi_1(x), \pi_2(x))$. С помощью такой биекции мы будем менять количество аргументов у функции.

Пусть $T_1(p, x)$ — универсальная функция. Построим функцию от трёх аргументов из функции T:

$$T_2(n, x, y) = T_1(n, c(x, y)).$$

Эта функция является универсальной функцией для всех вычислимых функций от двух аргументов. Действительно, пусть $f(x, y)$ — вычислимая функция от двух аргументов. Определим по ней функцию одного аргумента $g(x) = f(\pi_1(x), \pi_2(x))$. Для этой функции имеется T_1-номер n (программа), т.е.

$$T_1(n, x) = g(x) = f(\pi_1(x), \pi_2(x)) = T_2(n, \pi_1(x), \pi_2(x)).$$

Из этих равенств видно, что то же самое число n является T_2-номером для функции $f(x, y)$.

Искомая главная универсальная функция получится, если в $T_2(n, x, y)$ свернуть в одно число первую пару аргументов:

$$U(p, x) = T_2(\pi_1(p), \pi_2(p), x) = T_1(\pi_1(p), c(\pi_2(p), x)).$$

Определение выглядит замысловато, но если к нему приглядеться, то видно, что оно в точности соответствует указанной выше неформальной идее. Функция U рассматривает программу p как пару чисел, первое из которых — номер функции двух аргументов, а второе — первый аргумент этой функции.

Теперь проверим, что для U выполняются свойства главной нумерации.

Почему функция U универсальная? Возьмём какую-нибудь вычислимую функцию одного аргумента f и построим по ней функцию от двух аргументов $g(x, y) = f(y)$ (добавим формально ещё один аргумент, от которого ничего не зависит). У функции g есть T_2-номер. Подставляя 0 вместо x, получаем $f(y) = g(0, y) = T_2(n, 0, y) = U(c(n, 0), y)$. Таким образом, U-номером функции f является $c(n, 0)$.

Почему функция U главная? Возьмём какую-нибудь вычислимую функцию от двух аргументов $V(q, y)$. У неё есть T_2-номер n: $V(q, y) = T_2(n, q, y)$. По определению U получаем

$$V(q, y) = T_2(n, q, y) = U(c(n, q), y).$$

Функция $c(n, q)$ очевидно вычислима и задаёт нужный транслятор. □

[3] В нашей абстрактной модели языка программирования нельзя прямо сказать «добавим строчку, которая задаёт константу».

Какие ещё свойства главных функций напоминают свойства обычных языклв программирования?

Мы постоянно используем свойство 1 алгоритмов: композиция вычислимых функций вычислима. В обычном языке программирования программа, вычисляющая композицию $f \circ g$, алгоритмически строится из программ, вычисляющих функции f и g. Для главных нумераций такое свойство также выполняется.

Теорема 15.18. *Пусть $U(p,x)$ — главная нумерация. Тогда существует такая всюду определённая вычислимая функция $C(p,q)$, что*

$$U(C(p,q),x) = U(p,U(q,x)).$$

В правой части равенства написана композиция функций с номерами p, q. Первый аргумент левой части — номер композиции в главной нумерации.

Доказательство. Неформально. Рассмотрим такой «язык программирования», в котором программа — это пара U-программ, а при исполнении этой программы вычисляется композиция. По свойству главных нумераций существует транслятор таких программ в U.

Теперь запишем это рассуждение формально. Опять используем биекцию между натуральными числами и парами натуральных чисел. Пусть

$$V(q,x) = U(\pi_1(q), U(\pi_2(q), x))$$

(то есть мы трактуем q как пару номеров U-программ и вычисляем их композицию). Существует транслятор $s(q)$, для которого

$$V(q,x) = U(s(q),x).$$

Осталось положить $C(p,q) = s(c(p,q))$. □

15.8 Рекурсия и теорема о неподвижной точке

Ещё одна возможность, которая есть в обычных языках программирования — рекурсивный вызов программ. Аналогичное свойство выполняется и для главных нумераций. Как его сформулировать?

Поскольку мы рассматриваем фактически языки-интерпретаторы, то достаточно показать, что программа имеет доступ к своему тексту. Зная свой текст, программа в состоянии запустить как выполнение отдельных процедур, так и всей программы в целом.

В нашей абстрактной формулировке доступ программы к тексту можно описать следующим образом. Имеется вычислимая всюду определенная функция $p(t)$ — семейство програм, зависящих от параметра (этот параметр отвечает некоторому тексту в неформальном понимании). Мы хотим найти такое значение параметра t, при котором $U(p(t),x) = U(t,x)$. Тогда исполнение программы $p(t)$ приводит к тому же результату, что и исполнение программы t.

Получается, вообще говоря, более общая формулировка, чем просто доступ программы к своему тексту. Функция $p(t)$ может быть какой угодно. Мы по сути требуем, чтобы любое всюду определённое вычислимое преобразование вычислимых функций (задаваемых U-номерами) имело неподвижную точку n. Это означает, что вычисляется одна и та же функция как программой с номером n, так и программой с номером $p(n)$.

Теорема 15.19 (теорема о неподвижной точке)**.** *Пусть $U(p,x)$ — главная нумерация. Тогда для любой всюду определённой вычислимой функции $p(t)$ существует такое t, что $U(p(t),x) = U(t,x)$.*

Доказательство. Мы предъявим такую неподвижную точку «явно»: то есть укажем номер, который вычисляется по U-номеру преобразования $p(t)$ алгоритмически (обозначаем этот номер также p, что создает двусмысленность, но добавляет наглядности).

Начнём с функции $a(x) = U(x,x)$, которая есть результат применения программы к себе самой. Функция двух аргументов $U(a(x),y)$ вычислимая, поэтому по свойству главных нумераций найдётся такая всюду определённая функция $s(x)$, что $U(s(x),y) = U(a(x),y)$. Функция $s(x)$ «продолжает» функцию $a(x)$ в том смысле, что если $a(x)$ определена, то она вычисляет ту же самую функцию, что и $s(x)$ (при этом вполне возможно $a(x) \neq s(x)$ — у вычислимой функции может быть несколько программ, её вычисляющих).

Композиция функций $p \circ s$ — всюду определённая вычислимая функция и её U-номер (программа вычисления) вычисляется по номерам p и s: $q = C(p,s)$ и для всех x выполняется равенство

$$U(q,x) = U(p,U(s,x)).$$

Мы утверждаем, что неподвижной точкой является $s(q) = s(C(p,s))$. По построению функции s выполняется равенство $U(s(q),x) = U(a(q),x)$. Подставляя определение функции a в это равенство, получаем $U(s(q),x) = U(U(q,q),x)$.

С другой стороны, $U(p(s(q)),x) = U(U(q,q),x)$ по определению q (это же номер программы, вычисляющей композицию $p \circ s$). □

Построенная в доказательстве неподвижная точка может быть найдена по номеру (программе) преобразования p алгоритмически. Функция $s(x)$ не зависит от p. Программа q — это номер композиции программ с номерами p и s, который является, как мы уже видели, вычислимой функцией с номером $C(p,s)$. Неподвижная точка имеет вид $s(C(p,s))$.

Приведём примеры использования теоремы о неподвижной точке. Хорошо известное упражнение в обучении языку программирования: написать программу, которая печатает свой собственный текст. Оказывается, в любой главной нумерации такая программа существует. Действительно, раз уж программа имеет доступ к своему тексту, почему бы его не напечатать?

Формально нужно доказать, что для некоторого p выполняется равенство $U(p,x) = p$ при всех x. Определим такое преобразование программ: $q(t)$ есть номер программы, которая вычисляет функцию, тождественно равную t. Поскольку у такого преобразования есть неподвижная точка p, то $U(p,x) = U(q(p),x) = p$ для всех x.

В этом рассуждении есть существенная ошибка. Мы не доказали, что $q(t)$ вычислима, хотя это и очевидно для любого разумного языка программирования. Формально доказать существование вычислимой функции $q(t)$ с таким свойством можно, использовав основное свойство главных нумераций (в нём как раз декларируется существование вычислимой всюду определённой функции). Рассмотрим такую функцию от двух аргументов $V(t,n) = t$. Эта функция вычислима. По свойству главных нумераций найдётся такая всюду определённая вычислимая функция $q(n)$, что $V(t,n) = U(q(t),n) = t$. Это и есть искомая параметризация функций, вычисляющих константы.

Ещё один пример использования теоремы о неподвижной точке. В обычном языке программирования предусмотрены комментарии. Текст комментариев не влияет на работу программы. Поскольку комментарий может содержать произвольный текст, для каждой вычислимой функции возникает бесконечное количество программ, вычисляющих эту функцию. Аналогичное свойство выполняется и для главных нумераций.

Утверждение 15.20. *Пусть f — вычислимая функция, а U — главная нумерация. Множество $P_f = \{p : U(p,x) = f(x)\}$ номеров U-программ для f бесконечно.*

Доказательство. От противного. Предположим, что P_f конечно. Зафиксируем два числа $a \in P_f$ и $b \notin P_f$ (второе число найдётся, так как по предположению P_f конечно). Рассмотрим такое преобразование

$$p(t) = \begin{cases} a, & \text{если } t \notin P_f; \\ b, & \text{в противном случае.} \end{cases} \tag{15.3}$$

Это преобразование вычислимо, так как P_f конечно (как мы уже поступали в таких случаях, поместим в программу вычисления p список всех элементов P_f; после этого осталось только сравнивать вход со всеми элементами этого списка).

С другой стороны, у $p(t)$ нет неподвижной точки: если $t \in P_f$, то для какого-то x выполняется неравенство

$$U(p(t),x) = U(b,x) \neq f(x) = U(t,x)$$

по определению множества P_f; и наоборот, если $t \notin P_f$, то для какого-то x выполняется неравенство

$$U(p(t),x) = U(a,x) = f(x) \neq U(t,x).$$

Полученное противоречие доказывает бесконечность P_f. □

Это рассуждение можно усилить. Во-первых, $b \notin P_f$ найдётся в любом случае: не все же вычислимые функции совпадают с f. Для вычислимости функции (15.3) достаточно разрешимости множества P_f. Получаем такой результат.

Теорема 15.21. *Пусть f — вычислимая функция, а U — главная нумерация. Множество $P_f = \{p : U(p, x) = f(x)\}$ номеров U-программ для f неразрешимо.*

15.9 Теорема Успенского–Райса

Что можно понять о функции, вычисляемой данной программой? Оказывается, почти ничего.

Пусть $\mathcal{A}$ — некоторое нетривиальное свойство вычислимых функций, то есть такое свойство, что найдётся как функция $f \in \mathcal{A}$, так и функция $g \notin \mathcal{A}$ (мы отождествляем свойство функций и множество функций, удовлетворяющих этому свойству).

Теорема 15.22 (теорема Успенского-Райса)**.** *Пусть U — главная нумерация. Для любого нетривиального свойства функций $\mathcal{A}$ неразрешимо множество $P_{\mathcal{A}} = \{p : U(p, x) \in \mathcal{A}\}$ номеров U-программ, вычисляющих функции из множества $\mathcal{A}$.*

Доказательство. Рассуждаем аналогично доказательству утверждения 15.20.

Предположим, что $P_{\mathcal{A}}$ разрешимо. Зафиксируем какие-нибудь U-номера пары функций, удовлетворяющих и не удовлетворяющих свойству $\mathcal{A}$: $a \in P_{\mathcal{A}}$ и $b \notin P_{\mathcal{A}}$. Рассмотрим преобразование

$$p(t) = \begin{cases} a, & \text{если } t \notin P_{\mathcal{A}}; \\ b, & \text{в противном случае.} \end{cases}$$

Это преобразование вычислимо, если $P_{\mathcal{A}}$ разрешимо.

С другой стороны, у $p(t)$ нет неподвижной точки: если $t \in P_{\mathcal{A}}$, то $U(p(t), x) = U(b, x) \notin \mathcal{A}$; и наоборот, если $t \notin P_f$, то $U(p(t), x) = U(a, x) \in \mathcal{A}$.

Полученное противоречие доказывает неразрешимость $P_{\mathcal{A}}$. □

Эта теорема является серьёзным препятствием в деле автоматической обработки программ. Не существует алгоритма, который по тексту программы выяснял бы (1) останавливается ли она на каком-нибудь входе; (2) вычисляет ли программа функцию, записанную в техническом задании; (3) вычисляет ли программа тождественно равную 0 функцию и т.д. Причём это связано не с особенностями конкретного языка программирования, а с очень общими свойствами, которые выполняются для всех разумных языков программирования.