Лекция 12. Вычислимые функции. Перечислимые и разрешимые множества

Дискретная математика, ВШЭ, факультет компьютерных наук

(Осень 2014 – весна 2015)

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.

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 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 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 1. 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 2. 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.

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

Задача 3. 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 4. Для любого k опишите вычислимую биекцию между множествами $\mathbb{N}^k$ и $\mathbb{N}$.

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.

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:

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

Теорема 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 (1), $h_U(p) = 1$. Значит, $h_U(p) \neq U(p,p)$.

Аналогично рассуждаем в случае $U(p,p) \neq 0$. Тогда по (1) получаем $h_U(p) = 0 \neq U(p,p)$.

Пришли к противоречию. □

Замечание 2. Почему рассуждение в доказательстве теоремы 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 5. 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 1 из любой универсальной функции U можно построить пример вычислимой функции, у которой нет всюду определенного вычислимого продолжения. Определим $\tilde{h}_U$ как

$$\tilde{h}_U(x) = \begin{cases} 1, \text{ если } U(x,x) = 0, \\ 0, \text{ если } U(x,x) \text{ определена и } U(x,x) \neq 0, \\ \quad \text{в остальных случаях не определена.} \end{cases} \tag{2}$$

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 1. *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 (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 6. Пусть U — универсальная вычислимая функция. Докажите, что $U(p, p)$ не определено для некоторого p.

Задача 7. 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 2. *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. □

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 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 1. *Любое конечное множество разрешимо.*

Алгоритм разрешения конечного множества S аналогичен алгоритму из примера 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 8. 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 9. Докажите, что если существует алгоритм перечисления элементов некоторого множества, то существует также и алгоритм, который перечисляет элементы множества без повторений.

Определение 2. Множество S называется *перечислимым*, если есть алгоритм перечисления его элементов.

Пример 2. 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 2. *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 10. 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 11. Докажите, что если A, B — перечислимые множества, то и множества $A \cup B$, $A \cap B$ перечислимы.

Обратите внимание на разницу между задачами 8 и 11: 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.

Теорема 3 (теорема Поста)**.** *Если множества A и $\bar{A}$ перечислимы, то множество A разрешимо.*

[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.

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 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.