Занятие 19

Напомним, что алгоритм $\mathcal{A}$ *вычисляет* (быть может, не всюду определенную) функцию f из $\mathbb{N}$ в $\mathbb{N}$, если выполнены условия:

1. для всех $n \in \operatorname{dom} f$ алгоритм $\mathcal{A}$ на входе n завершает свою работу (за конечное число шагов) и выдает число $f(n)$ на выходе;

2. для всех $n \in \mathbb{N} \setminus \operatorname{dom} f$ алгоритм $\mathcal{A}$ на входе n не завершает свою работу ни за какое конечное число шагов.

Функция f называется *вычислимой*, если существует какой-либо вычисляющий ее алгоритм $\mathcal{A}$.

Отметим, что понятие вычислимой функции и связанные с ним тривиальным образом переносятся на функции между множествами $\mathbb{N}^k$ и $\mathbb{N}^*$ (в различных сочетаниях): действительно, алгоритм может «принимать» и «выдавать» не только отдельные натуральные числа, но и конечные их последовательности.

Задача 19.1. Вычислима ли следующая функция?

$$f(n) \quad = \quad \begin{cases} 0, & \text{если существует бесконечно много пар простых чисел } (p, p+2), \\ 1, & \text{иначе.} \end{cases}$$

Решение. Приведенное условие действительно определяет некоторую функцию, так как после «если» стоит некоторое высказывание, непременно являющееся либо истинным, либо ложным. В настоящее время истинность этого высказывания является нерешенной наукой проблемой.

Тем не менее функция f вычислима. В самом деле, наше высказывание истинно или ложно. В первом из случаев f принимает значение 1 при всех $n \in \mathbb{N}$. *Тогда* легко указать алгоритм, который вычисляет f (например, программу на языке `C`).

Во втором случае, f принимает значение 0 при всех $n \in \mathbb{N}$ и, аналогичным образом, вычисляется некоторым алгоритмом.

Итак, в каждом из логически возможных случаев функция f имеет вычисляющий ее алгоритм. Следовательно, она имеет вычисляющий ее алгоритм: по меньшей мере, один из двух — хотя науке теперь неизвестно, который именно.

Еще раз обратим внимание: для доказательства вычислимости *достаточно* указать *конкретный* подходящий алгоритм, но вовсе не является *необходимым* это делать — можно как-то иначе убедиться, что какой-то алгоритм *существует*. □

Каждому множеству $A \subseteq \mathbb{N}$ мы, напомним, ставим в соответствие его *характеристическую*, или *индикаторную* функцию χ_A, определенную условием:

$$\chi_A(n) \quad = \quad \begin{cases} 1, & \text{если } n \in A; \\ 0, & \text{если } n \in \mathbb{N} \setminus A \end{cases}$$

для всех $n \in \mathbb{N}$. Множество A *разрешимо*, если функция χ_A вычислима. Эквивалентно, разрешимость A означает существование алгоритма, который на каждом входе $n \in \mathbb{N}$ завершает свою работу, причем выдает 1, если $n \in A$, и 0 в противном случае.

Лемма 1 (см. разд. 15.3 Учебника)**.** *Всякое конечное множество разрешимо.*

Доказательство. Конечное множество $A = \{a_1, a_2, \ldots, a_n\}$ можно поместить в сам алгоритм — конечный объект. Например, в виде массива чисел в программе[1] на языке `C`. Затем будем сравнивать вход с каждым из n элементов массива, что потребует, каким бы ни было A, лишь конечного числа шагов. □

Задача 19.2. Докажите, что множество таких программ на языке `C` размером меньше `1Gb`, которые никогда не останавливаются, разрешимо. (Считайте, что программа исполняется на идеализированном компьютере, имеющем потенциально бесконечную память.)

Решение. Каждая такая программа представляет собою последовательность байтов, которые можно отождествить с элементами множества $[2^8 - 1]$, длины не более 2^{30}. С другой стороны, никогда не останавливающиеся программы образуют подмножество множества всех таких программ.

Из Лекций и результатов Занятия 17 известно, что множество последовательностей ограниченной длины над конечным алфавитом, а также любое подмножество конечного множества конечны. Поэтому интересующее нас множество программ конечно и, следовательно, разрешимо. □

Напомним, что множество $A \subseteq \mathbb{N}$ *перечислимо*, если существует алгоритм $\mathcal{A}$, который *перечисляет* множество A, т. е. работая неограниченно долго на пустом входе, $\mathcal{A}$ выписывает только элементы множества A (в произвольном порядке и, быть может, с повторениями), причем каждый элемент A на некотором шаге окажется выписанным.[2]

Задача 19.3. Докажите, что если A и B — перечислимые множества, то множества $A \cup B$ и $A \cap B$ тоже перечислимы.

[1] Разумеется, встроенные типы языка `C`, вроде `int`, `long` и т. п., содержат лишь не слишком большие числа. Однако, располагая неограниченной памятью, мы могли бы создать тип, содержащий любое натуральное число.

[2] Строго говоря, вывод каждого элемента — скажем, в десятичной записи — может требовать нескольких шагов; при этом важно, чтобы «незаконченная» запись не считалась выведенным числом. Для этого можно положить, что элемент выписан лишь тогда, когда вслед за ним напечатан некоторый «заключительный» символ.

Решение. Пусть алгоритм $\mathcal{A}$ перечисляет A и алгоритм $\mathcal{B}$ перечисляет B. Опишем алгоритм, перечисляющий $A \cup B$.

Будем на пустом входе попеременно исполнять шаги алгоритмов $\mathcal{A}$ и $\mathcal{B}$. Если на очередном шаге один из алгоритмов выписал какой-нибудь элемент, выдадим его на выход.

Очевидно, любой элемент, который будет выписан, принадлежит A или B. Обратно, пусть $n \in A \cup B$. Без ограничения общности, допустим $n \in A$. Тогда на некотором шаге k алгоритма $\mathcal{A}$ элемент A будет выписан. Конструируемый нами алгоритм исполнит k-ый шаг алгоритма $\mathcal{A}$ на своем каком-то k'-ом шаге.[3] Следовательно, элемент n будет нами выписан.

Теперь построим алгоритм, перечисляющий $A \cap B$. По-прежнему, на пустом входе попеременно исполняем шаги алгоритмов $\mathcal{A}$ и $\mathcal{B}$. Выписываемые им элементы будем помещать в два, изначально пустых, «накопителя» в памяти. Всякий раз, когда, скажем, алгоритм $\mathcal{A}$ выписывает какой-то элемент a_i, помещаем его в накопитель для $\mathcal{A}$, а затем будем сравнивать a_i с имеющимися на данном шаге в накопителе для $\mathcal{B}$ элементами $b_{j_1}, \ldots, b_{j_t}$. Если a_i совпал хотя бы с одним, выдаем его на выход и переходим к очередному шагу алгоритма $\mathcal{B}$, для которого действуем аналогично.

Ясно, что все выписанные нами элементы принадлежат $A \cap B$. Обратно, пусть $n \in A \cap B$. Это означает, что n будет выписан $\mathcal{A}$ на шаге k и $\mathcal{B}$ на шаге l. Если $k > l$, то элемент n уже будет в накопителе для $\mathcal{B}$, когда его выпишет $\mathcal{A}$. Значит, мы выведем n на выход. Аналогично в случае $k < l$. Пусть $k = l$. Если мы считаем, что чередование начинается с шага для $\mathcal{A}$, ситуация будет та же, что и при $k < l$, т. е. n будет уже в накопителе для $\mathcal{A}$, когда его выведет $\mathcal{B}$. □

Задача 19.4. Докажите, что множество $\mathbb{N} \times \mathbb{N}$ перечислимо.

Решение. Первое рассуждение («диагонали»). Будем последовательно рассматривать все натуральные числа. При каждом $k \in \mathbb{N}$ для всех j от 0 до k напечатаем пары $(k-j, j)$, что потребует лишь конечного числа шагов. Перейдем к числу $k+1$.

Рассмотрим произвольную пару $(n, m) \in \mathbb{N} \times \mathbb{N}$. Она будет выписана при $k = n+m$ и $j = m \leqslant k$.

Второе рассуждение («уголки»). Будем последовательно рассматривать все натуральные числа. При каждом $k \in \mathbb{N}$ для всех j от 0 до k напечатаем пары (j, k), а затем пары (k, i) для всех i от k до 0. Перейдем к числу $k+1$.

Рассмотрим произвольную пару $(n, m) \in \mathbb{N} \times \mathbb{N}$. Она будет выписана при $k = \max(n, m)$ и $i = \min(n, m)$ или $j = \min(n, m)$.

Третье рассуждение (абстрактное). Очевидно, множество $\mathbb{N}$ перечислимо. Будем на пустом входе попеременно исполнять шаги двух экземпляров алгоритма,

[3] Это почти очевидно. Однако можно провести индукцию по k. Пусть k шагов $\mathcal{A}$ нами уже осуществлены. Возможно, на последнем из них $\mathcal{A}$ выписал какой-то элемент. Нужно лишь конечно много шагов, чтобы распознать такую ситуацию и передать этот элемент на выход. Далее исполняем очередной шаг алгоритма $\mathcal{B}$; если и он что-то вывел, обработка займет лишь конечное время. Теперь исполняем шаг $k+1$ алгоритма $\mathcal{A}$, что и требовалось.

перечисляющего $\mathbb{N}$, причем один из экземпляров назовем $\mathcal{A}$, а другой $\mathcal{B}$. Для каждого экземпляра заведем свой, изначально пустой, накопитель. Элемент, выведенный любым из алгоритмов, помещаем в соответствующий накопитель.

Исполнив k-ый шаг $\mathcal{A}$, а затем k-ый шаг $\mathcal{B}$, делаем следующее. В накопителях к этому моменту содержатся элементы $a_{i_1}, \ldots, a_{i_s}$ и $b_{j_1}, \ldots, b_{j_t}$ соответственно. Выводим всевозможные пары вида (a_{i_q}, b_{j_r}) при $1 \leqslant q \leqslant s$ и $1 \leqslant r \leqslant t$. Очевидно, это потребует лишь конечного числа шагов, так что мы можем затем перейти к исполнению шага $k+1$ алгоритма $\mathcal{A}$.

Рассмотрим произвольную пару $(n, m) \in \mathbb{N} \times \mathbb{N}$. Число n будет выписано алгоритмом $\mathcal{A}$ на шаге k и число m — алгоритмом $\mathcal{B}$ на шаге l. После того, как мы исполним шаги с номером $\max(k, l)$ обоих алгоритмов, числа n и m окажутся в соответствующих накопителях. Далее, пара (n, m) будет выведена. □

Задача 19.5. Докажите, что если существует алгоритм перечисления элементов некоторого множества, то существует также и алгоритм, который перечисляет элементы этого множества без повторений.

Решение. Будем последовательно исполнять шаги произвольного «перечислителя» $\mathcal{A}$ множества A, заведя изначально пустой накопитель. Если на некотором шаге алгоритм $\mathcal{A}$ выписывает некоторый элемент a, сравниваем его со всеми, уже имеющимися в накопителе. Если a отличен от всех элементов накопителя, заносим его в накопитель и выдаем на выход. Переходим к следующему шагу алгоритма $\mathcal{A}$.

Ясно, что повторов у нас не будет, так как любой выписанный элемент помещается в накопитель и далее уже выписан быть не может. С другой стороны, каждый элемент $n \in A$ будет нами выписан, поскольку существует (по принципу наименьшего числа) шаг алгоритма $\mathcal{A}$ с *наименьшим* номером k, т. ч. алгоритмом $\mathcal{A}$ выписан элемент n. После исполнения этого k-ого шага мы не обнаруживаем n в накопителе и выдаем его на выход. □

Задача 19.6. Докажите, что алгоритм перечисления элементов множества $S \subseteq \mathbb{N}$ в возрастающем порядке существует тогда и только тогда, когда множество S разрешимо.

Решение. Допустим, множество S разрешимо. Будем последовательно рассматривать все натуральные числа. Для каждого k вычислим $\chi_S(k)$, что потребует лишь конечного числа шагов. Если $\chi_S(k) = 1$, выведем k на выход. Перейдем к числу $k+1$. Ясно, что полученное перечисление строго возрастающее.

Обратно. Допустим, что множество S в возрастающем порядке перечисляет алгоритм $\mathcal{A}$, причем само множество S *бесконечное*. Вычислим $\chi_S(n)$ для данного нам входа n следующим образом. Будем последовательно исполнять шаги алгоритма $\mathcal{A}$, пока он не выведет какое-нибудь число $m \geqslant n$. Если оказалось, что $m = n$, выдаем 1. Иначе выдаем 0. Останов.

Описанная процедура завершается за конечное число шагов. Действительно, множество S бесконечно и для любого n в нем найдется элемент $m' \geqslant n$, который $\mathcal{A}$ выведет на некотором шаге. Далее, поскольку $\mathcal{A}$ перечисляет S в возрастающем

порядке, любое меньшее m число уже выведено, либо же не принадлежит S. Если $m = n$, то $n \in S$ и наша процедура работает правильно. Если $m > n$, то n было уже выведено, что не так по построению, либо же $n \notin S$. Но в последнем случае, наша процедура работает правильно, поскольку выдает 0.

Если S конечно, описанная процедура не будет завершаться за конечное число шагов для достаточно больших n. Однако, она нам и не требуется, поскольку каждое конечное множество разрешимо. □

Задача 19.7. Всюду определенная функция $f\colon \mathbb{N} \to \mathbb{N}$ невозрастающая. Верно ли, что f вычислима?

Решение. Индукцией по $f(0)$ установим, что *любая* функция f, удовлетворяющая условию, вычислима. Как мы видели, любая (всюду определенная) константа вычислима. Если $f(0) = 0$, функция f есть константа 0. Пусть $f(0) = a > 0$. Тогда f есть константа a, или же найдется $c > 0$, т. ч. $f(c) < a$. В последнем случае возьмем наименьшее такое c.

Всюду определенная функция $g\colon \mathbb{N} \to \mathbb{N}$, т. ч. $g(n) = f(c + n)$ для всех $n \in \mathbb{N}$, является невозрастающей: в самом деле, если $m < n$, то $c + m < c + n$, а значит, $g(m) = f(c + m) \geqslant f(c + n) = g(n)$. С другой стороны, $g(0) = f(c) < a$, так что, по предположению индукции, функция g вычислима. Имеем

$$f(n) \quad = \quad \begin{cases} a, & \text{если } n < c; \\ g(n - c), & \text{если } n \geqslant c \end{cases}$$

при всех $n \in \mathbb{N}$. Теперь очевиден алгоритм для вычисления f. Нужно сравнить вход n с числом c, и выдать a или же выполнить алгоритм вычисления g на входе $n - c$, что потребует лишь конечного числа шагов. □

Задача 19.8. Докажите, что если $A \subseteq \mathbb{N}$ и $\mathbb{N} \setminus A$ перечислимы, то A разрешимо.

Решение. Установим это утверждение, известное как *теорема Пóста*. Пусть алгоритм $\mathcal{A}$ перечисляет A и $\mathcal{B}$ перечисляет $\mathbb{N} \setminus A$.

Вычисляем χ_A. Для данного входа n будем попеременно исполнять шаги алгоритмов $\mathcal{A}$ и $\mathcal{B}$. Каждый выводимый ими элемент сравним с n. Если оказывается, что n выведен $\mathcal{A}$, выдаем 1. Если же он выведен $\mathcal{B}$, выдаем 0. Останов.

Поскольку $n \in \mathbb{N} = A \sqcup (\mathbb{N} \setminus A)$, ровно один из алгоритмов выведет на каком-то шаге k число n. С другой стороны, для наименьшего такого k все шаги обоих алгоритмов с номерами $< k$ будут нами выполнены. Значит, уже на следующей за этим итерации наша процедура завершится. □

Задача 19.9. Перечислимо ли множество таких натуральных n, что уравнение $x^n + y^{n+1} = z^{n+2}$ имеет решение в положительных целых числах?

Решение. Указанное множество перечислимо. Подобно задаче 19.4, нетрудно показать, что множество $\mathbb{N}^4$ (и, вообще, $\mathbb{N}^k$ при всех $k \in \mathbb{N}$) перечислимо. Будем перечислять его, интерпретируя каждую выписанную четверку как (x, y, z, n) и проверяя

условие $x^n + y^{n+1} = z^{n+2}$. Эта проверка требует лишь конечного числа шагов, так как функции возведения в натуральную степень и сложения вычислимы и всюду определены. Если условие истинно, выпишем n и перейдем к следующему шагу перечисления $\mathbb{N}^4$.

Мы выпишем все подходящие n, поскольку для каждого из них найдутся соответствующие $x_0, y_0, z_0 \in \mathbb{N}$, а четверка $(x_0, y_0, z_0, n) \in \mathbb{N}^4$ будет на каком-то шаге выписана перечислителем $\mathbb{N}^4$. □

Задача 19.10. Докажите, что множество булевых функций, имеющих схемную сложность $> 2^{n/2}$ в стандартном базисе, разрешимо. Здесь n — количество переменных функции. Считайте, что алгоритм разрешения получает на вход булеву функцию в виде таблицы значений.

Решение. Напомним, что схемной сложностью функции называют наименьший размер вычисляющей ее схемы (в стандартном базисе). Как мы знаем, каждая функция n аргументов вычисляется некоторой схемой, поэтому схемная сложность определена для всех функций.

Таким образом, функция f имеет схемную сложность $> 2^{n/2}$ тогда и только тогда, когда ни одна схема размера $\leqslant 2^{n/2}$ не вычисляет эту функцию. С другой стороны, проверка того, вычислят ли данная схема данную функцию f, требует лишь конечного числа шагов (подставляем всевозможные 2^n входов в схему, вычисляем ее выход и сверяем его с таблицей для f).

Нам хотелось бы для данной f (и соответствующего n) проверить все «малые» схемы на то, вычисляют ли они f. Если подходящая схема нашлась, выдаем 0, иначе 1.

Схем ограниченного размера лишь конечно много, но мы не можем просто так на это сослаться — при разных n их множество может описываться разными алгоритмами. Нам же нужна *равномерная* процедура, которая, принимая на вход n, выдавала бы все схемы размера $\leqslant 2^{n/2}$, а затем завершалась.

Как мы видели в задаче 16.6, каждая схема размера $\leqslant s$ (не сюръективно!) кодируется некоторым двоичным словом длины $g(s)$, где функция $g\colon \mathbb{N} \to \mathbb{N}$ вычислимая. При $s = \lfloor 2^{n/2} \rfloor$ мы можем просмотреть все такие слова, для каждого проверив, во-первых, задает ли оно схему (из решения задачи 16.6 можно понять, как это сделать алгоритмически), а во-вторых, вычисляет ли эта схема функцию f. □

Задача 19.11. Пусть S — это множество таких n, что десятичная запись числа e содержит не менее чем n девяток подряд. Докажите, что множество S разрешимое. (Разрешается использовать тот факт, что e — иррациональное число.)

Решение. Покажем, что S устроено достаточно просто, чтобы быть разрешимым — хотя мы и не укажем конкретного алгоритма.

Прежде всего заметим, что $0 \in S$: во всяком случае, у нас есть $\geqslant 0$ девяток подряд. Далее, если $n \in S$ и $m < n$, то $m \in S$. Действительно, если в десятичной записи есть подслово $\underbrace{9\ldots9}_{n}$, то есть и $\underbrace{9\ldots9}_{m}$. Возможно, $S = \mathbb{N}$. В противном случае

рассмотрим наименьшее такое n, что $n+1 \notin S$ (напомним, $0 \in S$). Имеем $[n+1] \subseteq S$. Обратно, пусть $m \in S$ и $m \geqslant n+1$. Но тогда $n+1 \in S$, что не так; значит, $m < n+1$. Получили $S = [n+1]$.

Итак, $S = [n+1]$ для некоторого $n \in \mathbb{N}$ или же $S = \mathbb{N}$. В каждом из случаев функция χ_S не возрастает и, в силу задачи 19.7, является вычислимой. $\square$