Дискретная математика 2. Конспект

Сергей Пилипенко

2020 – 2021

Содержание

1 Лекция 1

1.1 Алгоритм и его неформальное определение

1. Алгоритмов счетно много;
2. Алгоритм исполняется по шагам;
3. Алгоритм работает конечно много шагов или зацикливается;
4. Алгоритм принимает вход и может выдавать что-то на выход[1];

Утверждение 1.1. *Слов конечного непустого алфавита лишь счетно много.*

Удобно считать, что на вход подаются конечные пары натуральных чисел. Алгоритм может вычислять частичные функции $f : \mathbb{N} \xrightarrow{p} \mathbb{N}$.

Определение 1.1. Алгоритм $\mathcal{F}$ вычисляет функцию $f : \mathbb{N} \xrightarrow{p} \mathbb{N}$, если для любого $x \in \mathbb{N}$

$$\begin{aligned} x \in \operatorname{dom} f &\implies \text{алгоритм } \mathcal{F} \text{ на входе } x \text{ останавливается за конечное число шагов и выводит } f(x); \\ x \notin \operatorname{dom} f &\implies \text{алгоритм } \mathcal{F} \text{ на входе } x \text{ не останавливается ни за какое конечное число шагов.} \end{aligned}$$

Определение 1.2. Функция f называется *вычислимой*, если существует алгоритм, который ее вычисляет.

Рассмотрим следующую тотальную функцию:

$$f(x) = \begin{cases} 1, & \text{если бог есть,} \\ 0, & \text{иначе.} \end{cases}$$

Утверждается, что f — вычислима. Действительно, рассмотрим следующие два алгоритма:

```
1 int f(int x) {
2     return 0;
3 }
```

```
1 int f(int x) {
2     return 1;
3 }
```

Какой-то из них вычисляет f, однако мы точно не можем сказать какой.

Определение 1.3. Множество $A \subseteq \mathbb{N}$ называется *разрешимым*, если существует алгоритм $\mathcal{A}$ такой, что $\forall x \in \mathbb{N}$

$$\begin{aligned} x \in A &\implies \mathcal{A} \text{ выводит 1 и останавливается;} \\ x \notin A &\implies \mathcal{A} \text{ выводит 0 и останавливается.} \end{aligned}$$

Утверждение 1.2. *A разрешимо $\iff$ вычислима характеристическая функция $\chi_A : \mathbb{N} \to \{0, 1\}$ множества A:*

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

Характеристическая функция существует у любого подмножества натуральных чисел, но не для всякого подмножества она вычислима.

Утверждение 1.3. *Существует неразрешимое множество.*

Доказательство. Алгоритмов лишь счетно много, в то время как подмножеств $\mathbb{N}$ несчетно много. □

Следствие 1.1. *Существует невычислимая функция.*

Утверждение 1.4. *Если A конечно, то A разрешимо.*

Доказательство. Положим $A = \{a_1, \ldots, a_n\}$, тогда следующий алгоритм вычисляет χ_A:

```
1 int in_A(x) {
2     return (x == a_1) || (x == a_2) || ... || (x == a_n);
3 }
```

□

Утверждение 1.5. *A, B разрешимы $\implies$ разрешимы $A \cup B$, $A \cap B$, $A^{\complement}$, $A \times B$.*

[1] Вход и выход — слова конечного алфавита.

Доказательство.

$$\chi_{A\cap B}(n) = \chi_A(n) \cdot \chi_B(n);$$
$$\chi_{A\cup B}(n) = \chi_A(n) + \chi_B(n);$$
$$\chi_{A^{\complement}}(n) = 1 - \chi_A(n);$$
$$\chi_{A\times B}(n) = \chi_A(n) \cdot \chi_B(m).$$

□

Определение 1.4. Множество $A \subseteq \mathbb{N}$ называется *перечислимым*, если существует алгоритм $\mathcal{A}$ такой, что, работая на пустом входе, $\mathcal{A}$ никогда не останавливается, но в процессе работы выводит все элементы множества A и только их.

Утверждение 1.6. *Если A разрешимо, то оно перечислимо.*

Доказательство.

```
1   n = 0;
2   while (1) {
3       if (in_A(n)) {
4           print(n);
5       }
6       ++n;
7   }
```

□

Докажем, что из перечислимости не следует конечность. Рассмотрим $A = \mathbb{N}$ — бесконечное множество, $\chi_A(n) = \chi_{\mathbb{N}}(n) = 1$ — вычислима.

Теорема 1.1 (Поста)**.** *A разрешимо $\iff$ A и $A^{\complement}$ перечислимы.*

Доказательство.

$\implies$ A разрешимо $\implies$ $A^{\complement}$ разрешимо $\implies$ $A^{\complement}$ перечислимо, A разрешимо $\implies$ A перечислимо.

$\impliedby$ Пусть $\mathcal{A}$ перечисляет A, и $\mathcal{B}$ перечисляет $A^{\complement}$. Мы хотим по заданному n посчитать $\chi_A(n)$. Поочередно будем делать по шагу алгоритмов $\mathcal{A}$ и $\mathcal{B}$. Если на каком-то шаге $\mathcal{A}$ выведет n, то $n \in A \implies$ выведем 1 и остановимся. Если на каком-то шаге $\mathcal{B}$ выведет n, то $n \notin A \implies$ выведем 0 и остановимся. Поскольку $A \cup A^{\complement} = \mathbb{N}$, то одно из этих событий обязательно случится и алгоритм остановится. □

Определение 1.5. Пусть $A \subseteq \mathbb{N}^k$, $(a_1, \ldots, a_k) \in A$. Проекцией множества A на координату i называется

$$\mathrm{pr}^i A = \{b \in \mathbb{N} \mid (a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots, a_k \in A)\}.$$

Утверждение 1.7. *A, B перечислимы $\implies$ $A \cup B$, $A \cap B$, $A \times B$, $\mathrm{pr}^i A$ перечислимы.*

Доказательство. Пусть $\mathcal{A}$ перечисляет A и $\mathcal{B}$ перечисляет B.

$\mathrm{pr}^i A$: Запустим $\mathcal{A}$, и для каждого напечатанного набора $(a_1, \ldots, a_k)$ будем брать a_i.

$A \cup B$: Будем поочередно делать шаги перечислителей $\mathcal{A}$ и $\mathcal{B}$. Все напечатанные кем-то из них числа отправляем на вывод.

$A \cap B$: Будем поочередно делать шаги перечислителей $\mathcal{A}$ и $\mathcal{B}$. Будем добавлять весь вывод $\mathcal{A}$ в буффер A'; аналогично для $\mathcal{B}$ и B'. Пусть A'_i — состояние буффера A' после i-ого шага $\mathcal{A}$. После того, как мы сделали i-ый шаг $\mathcal{A}$ и i-ый шаг $\mathcal{B}$, выведем элементы *конечного* множества $A'_i \cap B'_i$ и перейдем к $i+1$ шагу $\mathcal{A}$ и $\mathcal{B}$.

$A \times B$: Аналогично $A \cap B$, только выводим $A'_i \times B'_i$. □

Определение 1.6. Пусть есть функция $f : \mathbb{N} \xrightarrow{p} \mathbb{N}$. Определим для функции ее график:

$$\Gamma_f = \left\{(x, y) \in \mathbb{N}^2 \ : \ f(x) = y\right\}.$$

Теорема 1.2. *Пусть $f : \mathbb{N} \xrightarrow{p} \mathbb{N}$. Тогда f вычислима $\iff$ Γ_f перечислим.*

2 Лекция 2

2.1 Универсальный алгоритм

Определение 2.1. Универсальной вычислимой функцией (у. в. ф.) называется такая вычислимая $U : \mathbb{N}^2 \xrightarrow{p} \mathbb{N}$, что для любой вычислимой $f : \mathbb{N} \xrightarrow{p} \mathbb{N}$ существует $n \in \mathbb{N}$ («программа») такая, что

$$\forall x \in \mathbb{N} \quad U(n, x) \simeq f(x).$$

Пусть $\mathcal{U}$ — некоторый алгоритм, который вычисляет у. в. ф. U. Тогда, по существу, $\mathcal{U}$ — интерпретатор универсального языка программирования, где программами являются натуральные числа. Если имеется какая-то функция от двух аргументов $V : \mathbb{N}^2 \xrightarrow{p} \mathbb{N}$, то мы можем построить ее график (поверхность в $\mathbb{N}^3$). Зафиксируем ее первый аргумент n, и получим функцию $V_n : \mathbb{N} \xrightarrow{p} \mathbb{N}$ такую, что для любых n, x $V_n(x) \simeq V(n, x)$. Функция V_n называется *n-ым сечением функции V по первому аргументу.* Например, если была $V(x, y) = x + y$, то зафиксировав первый аргумент мы получим функцию «прибавить x» от одного аргумента. В таком случае, можно переписать условие универсальности следующим образом:

Утверждение 2.1. *Функция U является универсальной в классе вычислимых функций $\iff$ для любой вычислимой $f : \mathbb{N} \xrightarrow{p} \mathbb{N}$ существует $n \in \mathbb{N} : \ U_n = f$.*

Тогда можно сформулировать следующее свойство:

Свойство 2.1 (алгоритмов)**.** *Существует универсальная вычислимая функция.*

Данное утверждение эквивалентно тому, что на Python можно написать интерпретатор Python.

Определение 2.2. Функция $W : \mathbb{N}^2 \to \mathbb{N}$ называется универсальной вычислимой тотальной функцией, если

1. W вычислима.
2. Для любой вычислимой тотальной $g : \ \mathbb{N} \to \mathbb{N}$ существует m такое, что $W_m = g$.

Утверждение 2.2. *Не существует универсальной вычислимой тотальной функции W.*

Доказательство. Рассмотрим функцию $D : \mathbb{N} \to \mathbb{N}$, $D(x) = W(x, x)$, она является вычислимой и тотальной. Рассмотрим теперь функцию g такую, что $g(x) = D(x) + 1$, она также будет вычислимой и тотальной. Раз W универсальна, то существует m такое, что $W_m = g$, то есть

$$\exists m : \ \forall x \ W(m, x) = g(m).$$

Положим $x = m$, тогда

$$W(m, m) = D(m) + 1 = W(m, m) + 1 \implies 0 = 1.$$

Получили противоречие, значит такой W не существует. □

В случае для частично определенной у. в. ф. равенство $U(m, m) \simeq U(m, m) + 1$ допустимо, так как означает, что на m функция U не определена.

Теорема 2.1. *Существует перечислимое неразрешимое множество.*

Доказательство. Пусть U — универсальная вычислимая функция (мы знаем, что такая существует). Рассмотрим $K_U = \{n \in \mathbb{N} \ : \ U(n, n) \text{ определена}\} = \operatorname{dom} d_U$, где $d_U(x) \simeq U(x, x)$ — диагональ у. в. ф. U. Ясно, что d_U вычислима $\implies K_U = \operatorname{dom} d_U$ перечислимо.

Допустим, что K_U разрешимо. Рассмотрим функцию $r : \ \mathbb{N} \xrightarrow{p} \mathbb{N}$ такую, что

$$r(x) = \begin{cases} 1, & \text{если } x \notin K_U, \\ \text{undefined}, & \text{если } x \in K_U. \end{cases}$$

Функция r является полухарактеристической функцией $K_U^{\complement}$, $r = w_{K_U^{\complement}}$. K_U разрешимо $\implies K_U^{\complement}$ разрешимо $\implies$ $K_U^{\complement}$ перечислимо $\implies$ $w_{K_U^{\complement}}$ вычислима.

Тогда существует $n \in \mathbb{N}$ такое, что $\forall x \ U(n, x) \simeq r(x) \implies$ положим $x = n \implies U(n, n) = r(n)$. Рассмотрим несколько случаев:

1. $U(n, n)$ определена $\implies n \in K_U \implies r(n)$ не определено $\implies U(n, n)$ не определено.
2. $U(n, n)$ не определено $\implies n \in K_U \implies r(n) = 1 \implies U(n, n) = 1 \implies U(n, n)$ определено.

Ни один из случаев не приводит к чему-то разумному $\implies$ противоречие $\implies K_U$ не разрешимо. □

Определение K_U можно сформулировать следующим образом:

$$K_U = \{n \in \mathbb{N} \ : \ U \text{ останавливается на входе } (n, n)\},$$

или

$$K_U = \{n \in \mathbb{N} \ : \ \text{программа } n \text{ на входе } n \text{ останавливается}\}.$$

Поэтому вся эта теория называется «Проблемой самоприменимости»: остановится ли программа, если ей на вход передать ее же. Оказывается, что эта проблем не разрешима, что мы сейчас и доказали.

Рассмотрим $S_U = \left\{(n, x) \in \mathbb{N}^2 \ : \ U(n, x) \text{ определена}\right\} = \operatorname{dom} U$. Поскольку U вычислима, то S_U перечислимо. Заметим, что $n \in K_U \iff (n, n) \in S_U$, но тогда S_U разрешимо $\implies$ K_U разрешимо $\implies$ противоречие $\implies$ S_U не разрешимо. Это называется «Проблемой остановки»: остановка программы n на входе x.

2.2 T-предикаты

Зафиксируем универсальную вычислимую функцию U и алгоритм $\mathcal{U}$, ее вычисляющий. Рассмотрим следующее множество:

$$T'_{(U)} = \{(n, x, y, k) \ : \ \text{алгоритм } \mathcal{U} \text{ на входе } (n, x) \text{ остановится за } k \text{ шагов и выведет } y\}.$$

То свойство, что алгоритм можно исполнить по шагам отражено в том, что T' разрешимо. Рассмотрим также множество

$$T = \left\{(n, x, k) \in \mathbb{N}^3 \ : \ \text{алгоритм } \mathcal{U} \text{ на входе } (n, x) \text{ остановится за } k \text{ шагов}\right\}.$$

Множество T также разрешимо.

2.3 Альтернативный взгляд на перечислимые неразрешимые множества

Пусть U — универсальная вычислимая функция, и $d(x) \simeq U(x, x)$.

Утверждение 2.3. *Для любой вычислимой $f : \mathbb{N} \xrightarrow{p} \mathbb{N}$ существует n такое, что $f(n) \simeq d(n)$.*

Доказательство. По свойству U универсальна, то существует n такое, что $\forall x \ U(n, x) \simeq f(x)$. Положим $n = x \implies U(n, n) \simeq f(n) \simeq d(n)$. □

Определение 2.3. Пусть дана $f : \mathbb{N} \xrightarrow{p} \mathbb{N}$. Будем говорить, что функция g *продолжает* f, если $\operatorname{dom} f \subseteq \operatorname{dom} g$ и для любого $x \in \operatorname{dom} f \ f(x) = g(x)$.

Утверждение 2.4. *У функции d не существует вычислимого тотального продолжения.*

Доказательство. Пусть $g : \mathbb{N} \xrightarrow{p} \mathbb{N}$ — вычислимое тотальное продолжение d. То есть, для любого $x \in \operatorname{dom} d$ $d(x) = g(x)$. Рассмотрим $h : \mathbb{N} \to \mathbb{N}$, $h(x) = g(x) + 1$; Ясно, что h вычислима и тотальна, однако h всюду отличается от d, поскольку

$x \in \operatorname{dom} d \implies h(x) = g(x) + 1 = d(x) + 1 \neq d(x)$;

$x \notin \operatorname{dom} d \implies h(x)$ определена $\not\simeq d(x)$ не определена.

Так как любая вычислимая функция где-то совпадает с d, то h не вычислима. □

Утверждение 2.5. *Если функция f вычислима, но не имеет вычислимого тотального продолжения, то* $\operatorname{dom} f$ *перечислимо, но не разрешимо.*

Доказательство. Если f вычислима, то $\operatorname{dom} f$ перечислимо. Рассмотрим функцию

$$g(x) = \begin{cases} f(x), & x \in \operatorname{dom} f, \\ 2020, & x \notin \operatorname{dom} f. \end{cases}$$

Функция g является тотальной, и вычислима, если $\operatorname{dom} f$ разрешимо. С другой стороны, g — это вычислимое тотальное продолжение f, которого не существует. Альтернативно, $g(x)$ можно задать следующим образом: $g(x) = \chi_{\operatorname{dom} f}(x) \cdot f(x) + (1 - \chi_{\operatorname{dom} f}(x)) \cdot 2020$. □

Утверждение 2.6. *Существует вычислимая $f : \mathbb{N} \xrightarrow{p} \{0, 1\}$ такая, что у f нет вычислимого тотального продолжения.*

Доказательство. Определим f следующим образом:

$$f(x) = \begin{cases} 0, & x \in \operatorname{dom} d \text{ и } d(x) > 0, \\ 1, & x \in \operatorname{dom} d \text{ и } d(x) = 0, \\ \text{undefined}, & x \notin \operatorname{dom} d. \end{cases}$$

Такая f вычислима, поскольку $f \simeq h\left(d\left(x\right)\right)$, где

$$h\left(y\right) = \begin{cases} 0, & y > 0, \\ 1, & y = 0. \end{cases}$$

также является вычислимой функцией $\implies$ f вычислима как композиция вычислимых функций. Тогда для любого $x \in \operatorname{dom} d$ $f(x) \neq d(x) \implies$ у f нет вычислимого тотального продолжения (см. утверждение 2.3). □

Определение 2.4. Пусть $A, B, C \subseteq \mathbb{N}$. Будем говорить, что C *отделяет* A от B, если $A \subseteq C$ и $B \subseteq C^{\complement}$.

Следствие 2.1. *Существуют перечислимые множества A и B такие, что $A \cap B = \varnothing$, но не существует разрешимого C такого, что C отделяет A от B.*

Отсюда следует, что A перечислимо, но не разрешимо, так как иначе разрешимое A отделяло бы A от B.

Доказательство. Пусть f — функция из предыдущего утверждения, то есть

$$f(x) = \begin{cases} 0, & x \in \operatorname{dom} d \text{ и } d(x) > 0, \\ 1, & x \in \operatorname{dom} d \text{ и } d(x) = 0, \\ \text{undefined}, & x \notin \operatorname{dom} d. \end{cases}$$

Положим $A = f^{-1}\left(\{1\}\right)$, $B = f^{-1}\left(\{0\}\right)$. Очевидно, что $A \cap B = \varnothing$. A и B перечислимы как прообразы перечислимых множеств $\{1\}$, $\{0\}$ под действием вычислимой функции f. Положим, что $\exists C$, отделяющее A от B. Рассмотрим характеристическую функцию множества C:

$x \in A \implies \chi_C\left(x\right) = 1 = f\left(x\right)$;

$x \in B \implies \chi_C\left(x\right) = 0 = f\left(x\right)$.

Но тогда для любого $x \in \operatorname{dom} f$ $f(x) = \chi_C\left(x\right)$, то есть χ_C — тотальное вычислимое продолжение $f \implies \chi_C$ не вычислима $\implies$ C не разрешимо. □

2.4 Главные универсальные вычислимые функции

Определение 2.5. Функция $U : \mathbb{N}^2 \xrightarrow{p} \mathbb{N}$ называется главной универсальной вычислимой функцией, если

1. U вычислима;
2. Для любой вычислимой функции $V : \mathbb{N}^2 \xrightarrow{p} \mathbb{N}$ существует вычислимая тотальная функция $S : \mathbb{N} \to \mathbb{N}$ такая, что
$$\forall x\ U\left(S\left(n\right), x\right) \simeq V\left(u, x\right) \iff \forall n\ U_{S(n)} = V_n.$$

Утверждение 2.7. *Если U — главная универсальная вычислимая функция, то U является у. в. ф.*

Доказательство. Мы хотим, чтобы для любой вычислимой $f : \mathbb{N} \xrightarrow{p} \mathbb{N}$ существовало $n \in \mathbb{N}$ такое, что $U_n = f$. Рассмотрим функцию V такую, что $\forall k, x\ V\left(k, x\right) \simeq f\left(x\right)$, и $\forall k\ V_k = f$. Тогда, по свойству (2) из определения главной у. в. ф., существует вычислимая тотальная функция S такая, что $\forall k\ U_{S(k)} = V_k$. Положив k равным любому числу (например, 2020), получим $U_{S(2020)} = V_{2020} = f \implies n = S(2020)$. □

Утверждение 2.8. *Существует вычислимая биекция $h : \mathbb{N}^2 \to \mathbb{N}$.*

Введем обозначение для кода пары натуральных чисел $\langle n, m\rangle = h\left(n, m\right)$.

Утверждение 2.9. *Существует вычислимые тотальные функции π_1 и π_2 такие, что $\forall n\ \forall m\ \pi_1\left(\langle n, m\rangle\right) = n$ и $\pi_2\left(\langle n, m\rangle\right) = m$.*

Доказательство. Без ограничений общности предъявим алгоритм только для π_1.

1. Получаем на вход некоторое $z = \langle n, m\rangle$ — код некоторой пары;
2. Перечисляем все пары $(k, l) \in \mathbb{N}^2$ и для каждой проверяем равенство z и $\langle k, l\rangle$.
3. Если да, вернем k.

Такой алгоритм корректен, потому что любой код, который нам подают на вход корректен в силу сюръективности h, а в силу инъективности такая пара — единственная. $\square$

Теорема 2.2. *Если существует универсальная вычислимая функция U, то существует и главная универсальная вычислимая функция.*

Доказательство. Рассмотрим функцию $W : \mathbb{N}^2 \xrightarrow{p} \mathbb{N}$ такую, что

$$\forall n\ \forall x\ W(n, x) \simeq U(\pi_1(n), \langle \pi_2(n), x\rangle).$$

Функция W вычислима как композиция вычислимых функций. Покажем, что W является искомой главной универсальной вычислимой функцией. Пусть дана вычислимая $V : \mathbb{N}^2 \xrightarrow{p} \mathbb{N}$. Рассмотрим $V' : \mathbb{N} \xrightarrow{p} \mathbb{N}$ таую, что $V' \simeq V(\pi_1(x), \pi_2(x))$. V' также вычислима, а значит $\exists m$ такое, что $U_m = V'$. Теперь для любого n положим $S(n) = \langle m, n\rangle$, она будет вычислимой и тотальной. Проверим, что она подходит:

$$\begin{aligned}\forall n\ \forall x\ W(S(n), x) \simeq W(\langle m, n\rangle, x) \simeq \\ \simeq U(\pi_1(\langle m, n\rangle), \langle \pi_2(\langle m, n\rangle), x\rangle) \simeq U(m, \langle n, x\rangle) \simeq U_m(\langle n, x\rangle) \simeq \\ \simeq V'(\langle n, x\rangle) \simeq V(\pi_1(\langle n, x\rangle), \pi_2(\langle n, x\rangle)) \simeq V(n, x).\end{aligned}$$

Таким образом, $\forall n\ W_{S(n)} = V_n \implies W$ — главная у. в. ф. $\square$

3 Лекция 3

3.1 Отношение m-сводимости

Определение 3.1. Пусть $A, B \subseteq \mathbb{N}$. Говорят, что A m-сводится к B (обозн. $A \leqslant_m^f B$) тогда и только тогда, когда существует вычислимая тотальная $f : \mathbb{N} \to \mathbb{N}$ такая, что

$$\forall n \; n \in A \iff f(n) \in B.$$

Лемма 3.1. *Для любых $A, B, C \subseteq \mathbb{N}$ справедливо следующее:*

1. $A \leqslant_m A$;
2. $A \leqslant_m^f B$ *и* $B \leqslant_m^g C \implies A \leqslant_m^{g \circ f} C$;
3. $A \leqslant_m^f B \implies A^{\complement} \leqslant_m^f B^{\complement}$.
4. $A \leqslant_m B$ *и* B *разрешимо* $\implies$ A *разрешимо.*
5. $A \leqslant_m B$ *и* B *перечислимо* $\implies$ A *перечислимо.*
6. *Для любого* $n \in \mathbb{N}$, $\omega_A(n) = 1 \iff \omega_B(f(n)) = 1$.

Следствие 3.1. *Если A неразрешимо (неперечислимо), и $A \leqslant_m B$, то B неразрешимо (неперечислимо).*

Доказательство. Записать контрапозицию для пунктов (4), (5) леммы 3.1. □

Утверждение 3.1. *Если A разрешимо и $B \neq \varnothing \neq \mathbb{N}$, то $A \leqslant_m B$.*

Доказательство. В силу ограничений на множество B, $\exists b \in B$ и $\exists a \in B^{\complement}$. Тогда определим f следующим образом:

$$f(n) = \begin{cases} b, & n \in A, \\ a, & n \notin A. \end{cases}$$

Функция f является вычислимой $\implies \forall n \; n \in A \iff f(n) \in B$. □

Следствие 3.2 (Отсутствие антисимметричности у отношения m-сводимости)**.** *Пусть A — множество всех четных чисел, B — множество всех нечетных чисел. Тогда $A \leqslant_m B$, $B \leqslant_m A$, однако $A \neq B$.*

Следствие 3.3 (Несводимость к дополнению)**.** *Существует множество A такое, что $A \not\leqslant_m A^{\complement}$.*

Доказательство. Рассмотрим множество K — перечислимое и неразрешимое. Тогда $K^{\complement}$ неперечислимо (иначе по теореме Поста K разрешимо). Если $K \leqslant_m K^{\complement}$, то $K^{\complement} \leqslant_m K^{\complement\complement} \implies K^{\complement} \leqslant_m K$, то есть неперечислимое множество m-сводится к перечислимому, противоречие. □

Утверждение 3.2 (отсутствие универсального множества сводимости)**.** $\nexists A$ *такое что,* $\forall B \; B \leqslant_m A$.

Доказательство. $B \leqslant_m A \iff$ существует тотальная вычислимая f такая, что $\forall n \in \mathbb{N} \; n \in B \iff f(n) \in A$. Тогда всякое множество B однозначно определяется сводящей функцией:

$$B = \{n \in B \; : \; f(n) \in A\}.$$

Поэтому, таких множеств B не больше, чем сводящих функций, но, поскольку все сводящие функции вычислимы, то их не более чем счетно, а подмножеств $\mathbb{N}$ континуум. □

Лемма 3.2. *Пусть U — главная универсальная вычислимая функция, и*

$$K_U = \{n \in \mathbb{N} \; : \; U(n, n) \; \textit{определено}\}.$$

Тогда любое перечислимое множество A m-сводится к K_U.

Доказательство. Рассмотрим функцию

$$V(n, x) \simeq \begin{cases} 1, & n \in A, \\ \text{undefined}, & n \notin A. \end{cases}$$

То есть, $V(n, x) \simeq \omega_A(n) \implies V$ вычислима. Так как U главная, существует вычислимая тотальная $S : \mathbb{N} \to \mathbb{N}$ такая, что

$$\forall n \; U_{S(n)} = V_n \iff \forall n \; \forall x \; U(S(n), x) \simeq V(n, x).$$

Если $n \in A$, то V_n всюду определена $\implies U_{S(n)}$ всюду определена $\implies$ определено $U_{S(n)}(S(n)) \implies$ определено $U(S(n), S(n)) \implies S(n) \in K_U$.

Если $n \notin A$, то V_n нигде не определена $\implies U_{S(n)}$ нигде не определена $\implies$ не определено $U_{S(n)}(S(n)) \implies S(n) \notin K_U$.

Таким образом, существует вычислимая тотальная S такая, что $\forall n \; n \in A \iff S(n) \in K \iff A \leqslant_m^S K_U$. □

Пример 3.1. Существует неперечислимое множество такое, что все его элементы четные.

Доказательство. Рассмотрим множество $X = \left\{2n \ : \ n \in K^{\complement}\right\}$. Заметим, что $\forall n \ n \in K^{\complement} \iff 2n \in X$, что равносильно $K^{\complement} \leqslant_m X$. С другой стороны, все элементы X четны, и, поскольку $K^{\complement}$ не перечислимо, то и X не перечислимо. □

Пример 3.2. Пусть U — главная у. в. ф. и пусть $Z = \{n \in \mathbb{N} \ : \ U_n \text{ нигде не определена}\}$. Рассмотрим функцию

$$V(n,x) = \begin{cases} 1, & n \in K, \\ \text{undefined}, & n \notin K; \end{cases}$$

Поскольку K перечислимо, то ω_K вычислима, но так как $V(n,x) \simeq \omega_K(n)$, то V вычислима.

Так как U — главная, то существует вычислимая тотальная S такая, что

$$\forall n \ U_{S(n)} = V_n.$$

Мы видим, что если $n \in K \implies V_n$ всюду определено. Иначе V_n нигде не определено. То есть $n \in K^{\complement} \iff V_n$ нигде не определено $\iff U_{S(n)}$ нигде не определена $\iff S(n) \in Z$. Тогда $\forall n \ n \in K^{\complement} \iff S(n) \in Z$. Отсюда $K^{\complement} \leqslant_m^S Z$, но поскольку $K^{\complement}$ не перечислимо, то и Z не перечислимо.

По определению, $Z^{\complement} = \{n \in \mathbb{N} \ : \ U_n \text{ где-то определена}\} = \{n \in \mathbb{N} \ : \ \exists x \ U(n,x) \text{ определена}\}$. Вспомним про T-предикат $T(n,x,k) \iff \mathcal{U}$ останавливается на входе (n,x) за k шагов. Но такое мгножество T разрешимо, а $Z^{\complement} = \{n \in \mathbb{N} \ : \ \exists x \ \exists k \ T(n,x,k)\} \implies Z^{\complement}$ перечислимо.

Пример 3.3. Рассмотрим перечислимое множество $Z^{\complement} = \{n \in \mathbb{N} \ : \ U_n \text{ где-то определена}\} \neq \varnothing$. Тогда существует вычислимая тотальная $f : \mathbb{N} \to \mathbb{N}$ такая, что $Z^{\complement} = \operatorname{range} f = \{f(0), f(1), \ldots\}$. Рассмотрим $W : \mathbb{N}^2 \xrightarrow{p} \mathbb{N}$ такую, что

$$W(m,x) = \begin{cases} \text{undefined}, & m = 0, \\ U(f(m-1),x), & m \geqslant 1. \end{cases}$$

W, очевидно, является вычислимой. Докажем, что она универсальна. Рассмотрим произвольную вычислимую функцию $g : \mathbb{N} \xrightarrow{p} \mathbb{N}$. Рассмотрим несколько случаев:

1. g нигде не определена $\implies g = W_0$.
2. g где-то определена $\implies \exists k$ такое, что $U_k = g$ (в силу универсальности U) $\implies \exists k \in Z' : \ U_k = g \implies \exists m \in \mathbb{N} : \ U_{f(m)} = g$. Тогда $g = U_{f(m)} = W_{m+1}$.

Получается, что W является универсальной вычислимой функцией. Введем множество $Z' = \{m \in \mathbb{N} \ : \ W_m \text{ нигде не опред}$ Поскольку $\forall m \ W_{m+1} = U_{f(m)}$, где f — где-то определенная функция, поскольку $\operatorname{range} f = Z^{\complement} \neq \varnothing$, а $f(m)$ — индексы относительно U где-то определенных функций $\implies$ единственным номером нигде не определенной функции является $0 \implies Z' = \{0\}$. Поскольку $Z^{\complement}$ конечно, то оно разрешимо $\implies$ перечислимо, но Z не перечислимо $\implies W$ не является главной универсальной вычислимой функцией.

Следствие 3.4. *Существуют не главные у. в. ф.*

3.2 Рекурсия

Теорема 3.1 (Клини)**.** *Пусть U — главная универсальная вычислимая функция. Тогда для любой вычислимой тотальной функции $f : \mathbb{N} \to \mathbb{N}$ существует $n \in \mathbb{N}$ такое, что $U_{f(n)} = U_n$. То есть,*

$$\forall x \ U(f(n),x) \simeq U(n,x).$$

Доказательство. Рассмотрим функцию $V : \ \mathbb{N}^2 \xrightarrow{p} \mathbb{N}$ такую, что

$$\forall k \ \forall x \ V(k,x) \simeq U(U(k,k),x).$$

Такая функция вычислима. Поскольку U — главная, то существует вычислимая тотальная функция $S : \ \mathbb{N} \to \mathbb{N}$ такая, что

$$\forall k \ U_{S(k)} = V_k \iff \forall k \ \forall x \ U(S(k),x) \simeq V(k,x) \simeq U(U(k,k),x) \tag{1}$$

Рассмотрим функцию $f \circ S$. Она вычислима и тотальная как композиция вычислимых тотальных функций. Поскольку U — у. в. ф. $\implies \exists t$ такое, что $U_t = f \circ S$. Тогда

$$\forall x \ U(t,x) \simeq f(s(x)). \tag{2}$$

По формуле (1),

$$\forall x \ U(S(t),x) \simeq U(U(t,t),x) \underset{(2)}{\simeq} U(f(S(t)),x).$$

Получилось, что

$$\forall x \ U(S(t),x) \simeq U(f(S(t)),x).$$

Положим $n = S(t)$, тогда $\forall x \ U(n,x) \simeq U(f(n),x) \iff U_n = U_{S(n)}$. □

4 Лекция 4

Теорема 4.1 (Клини о неподвижной точке)**.** *Пусть U — главная универсальная вычислимая функция, $f : \mathbb{N} \to \mathbb{N}$ — произвольная вычислимая тотальная функция. Тогда существует $n \in \mathbb{N}$ такое, что $U_{f(n)} = U_n$, то есть*

$$\exists n \quad \forall x \quad U(n, x) \simeq U(f(n), x).$$

На прошлой лекции мы доказали теорему Клини. С одной стороны, вы можете возразить, что же такого потрясающего в этом результате, бред же какой-то. На самом деле, если мы вспомним нашу содержательную интерпертацию символов, которые здесь есть, то оказывается, что это все не является бредом. U — функция, которую вычисляет некоторый универсальный алгоритм $\mathcal{U}$, то есть интерпретатор какого-то языка программирования. Первые аргументы функции U, то есть n и $f(n)$ — это какие-то программы, а значение функции $U(n, x)$ — это то, что вычисляет программа n на входе x. Тогда f можно рассматривать как некоторое алгоритмическое преобразование программ. И оказывается, что в главном языке программирования для любого алгоритмического преобразования программ найдется такая программа, чей смысл не меняется. Этот же факт означает, что ни одно алгоритмическое преобразование не может поменять смысл всех программ.

4.1 Теорема о рекурсии

Следствием теоремы Клини является факт, который мы будем называть теоремой о рекурсии.

Следствие 4.1 (Теорема о рекурсии)**.** *Пусть U — главная универсальная вычислимая функция, и $V : \mathbb{N}^2 \xrightarrow{p} \mathbb{N}$ — произвольная вычислимая функция[2] двух аргументов. Тогда $\exists n \in \mathbb{N}$ такое, что*

$$U_n = V_n.$$

Последнее равенство равносильно

$$\exists n \quad \forall x \quad U(n, x) \simeq V(n, x).$$

То есть, у нас существует такая программа, которая имеет на главном языке такой же смысл, что и на языке V. Вы спросите, а как вообще так? А вдруг один язык таков, что ни одна корректная программа не является корректной для другого? Тогда это утверждение не является верным, однако мы договорились, что все натуральные числа являются корректными программами, а в таком случае все хорошо.

Доказательство. И так, определение главной универсальной вычислимой функции гарантирует нам существование какой-то вычислимой тотальной функции. В тоже время, теорема Клини принимает какую-то вычислимую тотальную функцию на вход. Применим эти два факта и все получится.

Формально, так как U — главная, то существует вычислимая тотальная S такая, что $\forall k \; U_{S(k)} = V_k$. Но, по теореме Клини, существует неподвижная точка n для S, то есть $\exists n : U_{S(n)} = U_n$. Тогда

$$\exists n : \quad U_n = U_{S(n)} = V_n,$$

что и требовалось показать. □

Причем тут вообще рекурсия? А при том, что функция V может сама по себе вызывать функцию U.

Пример 4.1. Существует такая программа n, которая на любом входе выводит саму себя. То есть,

$$\exists n : \quad \forall x \quad U(n, x) = n.$$

Доказательство. Рассмотрим $V(k, x) = k$, она вычислима. Тогда, по теореме о рекурсии, $\exists n$ такое, что $\forall x \; U(n, x) \simeq V(n, x) \implies \exists n \; \forall x \; U(n, x) = n$. □

Пример 4.2. Существует такая программа n, которая на любом входе x возвращает то же, что и программа x на входе n. То есть, $\exists n : \forall x \; U(n, x) \simeq U(x, n)$.

Доказательство последнего примера полностью аналогично примеру 1. Таких примеров можно привести бесконечное количество, поэтому надо просто усвоить один важный факт: *в главных языках программирования программа может иметь доступ к своему коду.*

Как с этим всем связана рекурсия? Приходилось ли вам писать рекурсивные алгоритмы на каком-нибудь языке программирования (например, C). Наверное, приходилось. А как устроен рекурсивный алгоритм? На каком-то этапе вычисления функции f, она использует свой собственный код, то есть вызывает функцию f.

С другой стороны, на этот факт можно посмотреть как на уравнение.

[2] мы можем рассматривать ее как какой-то язык программирования, где первый аргумент — программы, а второй аргумент — их входной набор данных

Утверждение 4.1. *$\exists$ вычислимая функция f такая, что*

$$\begin{cases} f(0) = 1, & x = 0, \\ f(x) = x \cdot f(x-1), & x > 0. \end{cases}$$

Заметим, что выражение выше является *системой из уравнений* на функцию f. Поэтому можно задать два вопроса:

1. существует ли такая f?
2. единственна ли такая f?

Оказывается, теорема Клини легко нам показывает, что такая функция существует. То есть, по некоторым условиям на функцию f нам гарантировано существование алгоритма, который эту функцию вычисляет.

Доказательство. Рассмотрим

$$V(k,x) \simeq \begin{cases} 1, & x = 0, \\ U(k, x-1) \cdot x, & x > 0. \end{cases}$$

По теореме о рекурсии, $\exists n$ такое, что $\forall x\ U(n,x) \simeq V(n,x)$, то есть

$$\exists n \quad \forall x \quad U(n,x) \simeq \begin{cases} 1, & x = 0, \\ U(n, x-1) \cdot x, & x > 0. \end{cases}$$

Нам хочется, чтобы $f = U_n$, однако в формулировке теоремы о рекурсии нам не гарантируется, что U_n всюду определено, в то время как f должна быть всюду определена. Заметим, что в нуле функция $U(n,0)$ определена, а в остальных точках ее определенность можно доказать индукцией по x. Получается, что U_n тотальна. Таким образом, мы получили функцию с желаемыми свойствами ($f = U_n$). □

В чем преимущество такого подхода? Мы описали наши пожелания относительно какой-то функции f, а теорема о рекурсии гарантирует нам существование алгоритма, вычисляющего f.

Утверждение 4.2. *Существует вычислимая функция f такая, что $\forall x\ f(x) \simeq 1 + f(x+1)$.*

Странная функция, она в точке x на единицу больше, чем в точке $x+1$. Что можно сказать про такую функцию? Она монотонно убывает, потому что $f(x)$ на единицу больше, чем $f(x+1)$. Какое же значение у нее в нуле? Да никакого, ведь функция из натуральных чисел в натуральные. Может ли такая функция быть определена в каком-то натуральном числе? Нет, потому что иначе она бы представляла собой бесконечно убывающую последовательность натуральных чисел, которой существовать не может. Но вместе с тем, такая вычислимая функция подходит, и это очень просто понять:

Доказательство. Рассмотрим вычислимую функцию $V(k,x) \simeq U(k, x+1) + 1$. Правая часть вычислима, а по теореме о рекурсии существует n такое, что для любого $x\ U(n,x) \simeq V(n,x) \simeq U(n,x+1) + 1 \implies f = U_n$. □

То есть, такая функция все же существует. Но что же это за функция? Ясно, что такая функция $f = \zeta$ — нигде не определенная функция. Поэтому, любое уравнение вида $U(n,x) \simeq V(n,x)$ с вычислимой правой частью имеет программу n_0 — свое решение. Но никто не гарантирует, чтобы это решение было номером всюду определенной функции, и так далее. То есть, понятие рекурсия в этой всей теории используется в самом широком смысле: рекурсивная функция не обязана останавливаться. Она может нам дать нигде не определенную функцию, ведь, если передать такую функцию какому-нибудь компилятору вроде компилятора языка Си, то произойдет stack overflow ввиду конечности памяти в реальном мире. В нашей модели памяти бесконечно много, поэтому наша программа бы просто зациклилась, как, например, зацикливается программа, вычисляющая функцию ζ.

4.2 Теорема о совместной рекурсии

Оказывается, с помощью конструкций вида

$$U(n,x) \simeq V(n,x)$$

с вычислимой правой частью, можно решать не только уравнения на вычислимую функцию, но и системы уравнений.

Пример 4.3. Существуют такие программы a и b, что для любого x

$$\begin{aligned} U(a,x) &= b, \\ U(b,x) &= a + 1. \end{aligned}$$

Такая концепция в программировании называется *совместной рекурсией*: когда мы определяем одну функцию через другую, а эту функцию через первую. Совместная рекурсия встречается и в реальной жизни, например можно определить функции для проверки числа на четность:

$$\begin{aligned} even(0) &= 1, \\ even(n+1) &= odd(n), \\ odd(0) &= 0, \\ odd(n+1) &= even(n). \end{aligned}$$

Теорема 4.2 (о совместной рекурсии)**.** *Пусть U — главная универсальная вычислимая функция, а V_1, V_2 — произвольные вычислимые функции с нужным числом аргументов. Тогда $\exists a, b$ такие, что $\forall x$*

$$\begin{cases} U(a,x) \simeq V_1(a,b,x), \\ U(b,x) \simeq V_2(a,b,x). \end{cases}$$

Мы можем решить систему уравнений: существуют два таких алгоритма, которые используют код друг-друга. Все такие переходы от одномерного случая к двумерному делаются с помощью кодирования пар. С другой стороны, у нас на семинаре было доказано следующее утверждение:

Утверждение 4.3. *Если U — главная универсальная вычислимая функция, то существует такая вычислимая тотальная функция c, что*

$$\forall p, q \quad U_{c(p,q)} = U_p \circ U_q.$$

Или, если записать это в кванторах,

$$\forall x \quad U(c(p,q),x) \simeq U(p, U(q,x)).$$

Содержательно, идея этого утверждения следующая: имея текст программы p и текст программы q, мы можем автоматически сгенерировать текст программы, которая вычисляет $p \circ q$. Для Си, например, это вообще тривиальное утверждение: надо просто вызвать одну функцию из другой. Но это имеет место и в абстрактном случае. Само доказательство этого утверждения сильно зависит от наличия так называемого кодирования пар. Напомню, что это такое.

Определение 4.1. Пусть у нас существует вычислимая тотальная биекция $\langle \cdot, \cdot \rangle : \mathbb{N}^2 \to \mathbb{N}$. Тогда $\langle \cdot, \cdot \rangle$ называется *кодированием пары*.

Таких биекций много, кроме того, для такой биекции существуют вычислимые тотальные функции-проекторы π_1 и π_2 такие, что

$$\begin{aligned} \pi_1(\langle n, m \rangle) &= n, \\ \pi_2(\langle n, m \rangle) &= m. \end{aligned}$$

Если π_1 и π_2 — вычислимые тотальные функции, то, поскольку U является у. в. ф., то у них есть какие-то индексы[3] $p_1, p_2 \in \mathbb{N}$ такие, что

$$\pi_1 = U_{p_1} \qquad \pi_2 = U_{p_2}.$$

Рассмотрим теперь функцию $V(k,x) \simeq \langle V_1(c(p_1,k), c(p_2,k), x), V_2(c(p_1,k), c(p_2,k), x) \rangle$. Такая функция V вычислима, потому что представляет собой композицию вычислимых функций. Почему она такая? Потому что нам так захотелось. По теореме о рекурсии, $\exists n \in \mathbb{N}$ такое, что $\forall x \in \mathbb{N}$

$$U(n,x) \simeq V(n,x) \simeq \langle V_1(c(p_1,n), c(p_2,n), x), V_2(c(p_1,n), c(p_2,n), x) \rangle.$$

Положим $a = c(p_1, n)$, $b = c(p_2, n)$, тогда

$$U(a,x) \simeq U(c(p_1,n), x) \simeq U(p_1, U(n,x)) \simeq \pi_1(U(n,x)) \simeq V_1(c(p_1,n), c(p_2,n), x) \simeq V_1(a,b,x).$$

Кроме кодирования пар, можно использовать кодирование троек, и, с помощью такого кодирования, доказать это утверждение для трех программ. Аналогично можно сделать для любого конечного числа программ.

Итак, теорема Клини позволяет решать системы уравнений на вычислимые функции.

[3] программы, которые их считают

4.3 Теорема Райса-Успенского

Здесь и далее U — главная универсальная вычислимая функция. Теорему Райса-Успенского можно рассматривать как следствие теоремы Клини. Если вы ходили на семинары, или, хотя бы, на прошлую лекцию, то вы, наверное, помните, что у нас были такие множества:

$$\{n \in \mathbb{N} \mid U_n \text{ где-то определена}\},$$
$$\{n \in \mathbb{N} \mid U_n \text{ монотонно возрастает на } \operatorname{dom} U_n\}.$$

Что это такое? Это множество программ, которые вычисляют функцию с каким-то нетривиальным свойством. Это вопрос, который, вообще говоря, мог бы быть интересен и на практике. И мы с вами во всех конкретных случаях видели, что все эти множества являются неразрешимыми. Первое множество перечислимо, второе — нет, но никакое из них не является разрешимым. То есть вопрос о том, можем ли мы алгоритмически узнать, обладает ли вычислимая функция U_n каким-то конкретным свойством, решался для U отрицательно. Оказывается, что это — общий факт, то есть нельзя алгоритмически по программе узнать, обладает ли вычисляемая ею функция какими-то свойствами[4]. Это и есть теорема Райса-Успенского.

Теорема 4.3 (Райса-Успенского)**.** *Пусть U — главная универсальная вычислимая функция, а $\mathcal{F}$ — нетривиальное[5] подмножество множества всех вычислимых функций $f : \mathbb{N} \xrightarrow{p} \mathbb{N}$. Тогда множество*

$$F = \{n \in \mathbb{N} \mid U_n \in \mathcal{F}\}$$

неразрешимо. Множество F называется индексным множеством.

Доказательство (Есенин-Вольпин). Сведем доказательство исходного утверждения к теореме Клини. Зафиксируем вычислимую функцию $f \in \mathcal{F}$, и вычислимую функцию $g \notin \mathcal{F}$. В силу универсальности U существуют индексы $n, m \in \mathbb{N}$ такие, что $f = U_n$ и $g = U_m$. Рассмотрим тотальную функцию $h : \mathbb{N} \to \mathbb{N}$ такую, что

$$\forall k \in \mathbb{N} \qquad h(k) = \begin{cases} m, & k \in F, \\ n, & k \notin F. \end{cases}$$

Если F — разрешимое множество, то h вычислима, потому что $h(k) = m \cdot \chi_F(k) + n \cdot (1 - \chi_F(k))$. Итак, h вычислима и тотальна, тогда, по теореме Клини, $\exists n$ такое, что

$$U_n = U_{h(n)}.$$

Рассмотрим теперь, куда попадает число n. Оно либо попадает в F, либо нет. Предположим, что $k \in F$, тогда $U_k \in \mathcal{F}$ с одной стороны. С другой стороны, поскольку $k \in F$, то $U_{h(k)} \in \mathcal{F} \implies h(k) = m \implies U_m \in \mathcal{F} \implies g \in \mathcal{F} \implies$ противоречие. Совершенно аналогично рассматривается случай, когда $k \notin F$. Тогда $U_k \notin \mathcal{F} \implies U_{h(k)} \notin \mathcal{F} \implies U_n \notin \mathcal{F} \implies f \notin \mathcal{F} \implies$ противоречие.

Итак, если бы множество F было бы разрешимым, мы бы с вами изготовили такую функцию, которая использует свойство принадлежности множеству F и делает все наоборот: если принадлежит, то выдает номер функции, которая не принадлежит, а если не принадлежит, то выдает номер функции, которая принадлежит этому множеству. А дальше мы пользуемся теоремой Клини: должна быть программа, чей смысл не меняется, однако в таком случае получается, что смысл любой программы меняется, отсюда и противоречие. □

Дадим также альтернативное доказательство этой теоремы, которое изначально предложил Райс.

Доказательство (Райс). Зафиксируем функцию ζ, которая нигде не определена. Попадает ли эта функция $\mathcal{F}$ или нет? Рассмотрим два случая:

$\zeta \in \mathcal{F} \implies$ проведем тоже рассуждение для $\overline{\mathcal{F}}$, этого достаточно, поскольку разрешимость любого множества эквивалентно разрешимости его дополнения.

$\zeta \notin \mathcal{F} \implies$ мы знаем, что $\mathcal{F} \neq \varnothing$, значит существует вычислимая $f \in \mathcal{F}$. Зафиксируем также K — какое-то перечислимое неразрешимое множество. Рассмотрим функцию $V : \mathbb{N}^2 \xrightarrow{p} \mathbb{N}$, которая устроена следующим образом:

$$\forall n, x \in \mathbb{N} \quad V(n, x) \simeq \begin{cases} f(x), & n \in K, \\ \zeta(x), & n \notin K. \end{cases}$$

Покажем, что функция V вычислима. На входе (n, x) запускаем перечислитель множества K, если $n \notin K$, то алгоритм зациклится, что равносильно вычислению ζ. Иначе, передаем управление вычислителю функции f. Ну или, если расписать формальнее, то $V(n, x) \simeq f(x) \cdot \omega_K(n)$, где ω_K вычислима как полухарактеристическая функция перечислимого множества. U — главная $\implies$ существует вычислимая тотальная S такая, что

$$\forall n \in \mathbb{N} \quad U_{S(n)} = V_n.$$

Для любого $n \in \mathbb{N}$ рассмотрим два случая:

[4] все равно, что сказать, что нетривиальные свойства функции не распознаются по номерам программ

[5] это значит, что $\exists f \in \mathcal{F}$, и $\exists g \notin \mathcal{F}$

$n \in K \implies V_n = f \in \mathcal{F} \implies U_{S(n)} \in \mathcal{F} \implies S(n) \in F.$

$n \notin K \implies V_n = \zeta \notin \mathcal{F} \implies U_{S(n)} \notin \mathcal{F} \implies S(n) \notin F.$

Получается, что $\forall n \; n \in K \iff S(n) \in F$ — картина m-сводимости. Тогда неразрешимое множество $K \leqslant_m F$, но тогда F неразрешимо, что и требовалось доказать. Заметим, что, по свойству m-сводимости, $\overline{K} \leqslant_m \overline{F}$, то есть $\overline{F}$ не перечислимо, поскольку $\overline{K}$ не перечислимо.

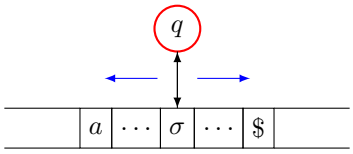
□

5 Лекция 5

5.1 Неформальное описание машины Тьюринга

Что такое машины Тьюринга и зачем они нам нужны? В предшествующих лекциях мы с вами работали с понятием алгоритма и достаточно далеко развили соответствующую теорию при том, что алгоритм мы никак не определяли. Дело в том, что определений алгоритма может быть много, и нельзя сказать, что одно из них хуже или лучше другого, ровно как и нельзя сказать, что один язык программирования лучше или хуже другого. В общем-то, эти определения могут выглядеть по-разному, однако они эквивалентны в том смысле, что дают одно и то же множество вычислимых функций. Собственно говоря, именно по тому, какие функции считаются вычислимыми, мы и оцениваем данное конкретное понятие алгоритма.

Машина Тьюринга — это такая модель алгоритмов, которая немного похожа на примитивный электромеханический (или даже электронный) компьютер. Она не самая удобная для практических рассуждений или математических доказательств, но, с другой стороны, она сочитает в себе относительную простоту, наглядность и относительную близость к практике. Так что машины Тьюринга прочно укрепились в преподавании и, по-сути, являются простейшими моделями, которые нам надо знать. Дадим неформальное описание машины Тьюринга.



Суть в том, что у нас имеется некоторая такая лента, разбитая на ячейки. Такую ленту можно рассматривать как оперативную память. Каждая ячейка имеет конечную емкость, то есть число состояний ячейки конечно. Можно считать, что в ячейку пишется какой-то символ a из *конечного* алфавита Γ. По ленте ездит некоторое абстрактное устройство, которое называется *головкой*. Что же эта головка делает? Головка может читать символ из ячейки, и может писать символ в эту ячейку. Также она может ленту двигать на один шаг влево или на один шаг вправо. Соответственно, чтобы прочитать с ленты несколько последовательных символов, машина должна несколько раз ленту переместить (или проехаться по ней). Кроме того, у головки самой имеется некоторая внутренняя память, которая содержательно нужна для того, чтобы устанавливать связь между содержимым ячеек. Мы моделируем эту память следующим образом: мы считаем, что у головки есть какое-то состояние $q \in Q$, в котором она находится, и эти состояния вместе образуют некоторый *конечный алфавит состояний* Q. Почему память можно моделировать конечным алфавитом состояний? Если вы рассмотрите модель памяти в вашем компьютере в предположении, что вы не можете ее докупать, то эту память можно заполнить лишь конечным числом способов. Например, если она имеет «длину» в n бит, то мы можем придать ей лишь 2^n различных состояний. С практической точки зрения удобно думать, что память головки — это что-то типа состояния регистра процессора, а лента — оперативная память (которая, при этом, считается неограниченной). Мы считаем, что чистые (нетронутые) ячейки памяти заполнены специальным символом $\# \in \Gamma$, который называется *пробельным символом*.

Как же наша машина Тьюринга работает? Работает она в соответствии с некоторой «программой». Тут важно понимать аналогию: не смотря на то, что мы говорим, что машина Тьюринга — это модель компьютера, она является таким компьютером, который выполняет только одну программу. Как же выглядит программа? Программа, по сути дела, указывает машине что делать, в каком состоянии, в зависимости от символа, написанного в данной ячейке, в которой находится головка. Программа состоит из инструкций вида $qa \mapsto q'a'S$, где

- $q, q' \in Q$ — состояния;
- $a, a' \in \Gamma$ — символы алфавита;
- $S \in \{L, N, R\}$ — элемент некоторого трехэлементного множества, в котором $L =$ «left», $N =$ «neutral», $R =$ «right». Это символы, которые говорят нам, куда нужно сдвинуть головку. Например, R означает, что машина должна сдвинуться в ячейку справа от текущей.

Интерпретация инструкций такая: если в состоянии q увидишь a в текущей ячейке, то запиши a' в текущую ячейку, перейди в состояние q' и сместись на S. То есть, по сути дела, чем определяется поведение машины Тьюринга? Поведение ее определяется тем, что написано на ленте, изначальным положением головки и ее состоянием. На самом деле, в каждый момент времени, будущее состояние нашей машины определяется такой информацией. Такая информация называется *конфигурацией машины Тьюринга*.

Определение 5.1. Конечное множество Γ называется ленточным алфавитом.

Определение 5.2. Будем обозначать через Γ^* множество всех конечных последовательностей элементов из алфавита Γ.

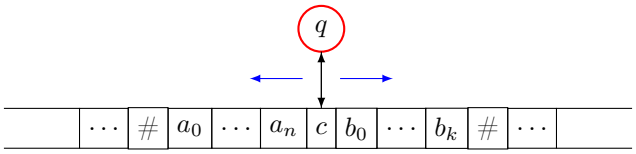
$$\Gamma^* = \bigcup_{n \in \mathbb{N}} \Gamma^n.$$

Определение 5.3. Зафиксируем $A, B \in \Gamma^*$ — какие-то слова ленточного алфавита, q — состояние головки МТ[6], $c \in \Gamma$ — символ в текущей ячейке, на которую машина смотрит. Тогда набор (A, B, q, c) называется конфигурацией машины Тьюринга.

Конфигурация трактуется следующим образом:

- A — то, что стоит слева от головки;
- B — то, что стоит справа от головки;
- c — символ, который находится под головкой.

Проиллюстрировать это можно, например, вот так:



Здесь $(a_0, \ldots, a_n) = A$, $(b_0, \ldots, b_k) = B$. Обратим внимание, что нельзя сказать, что машина находится в какой-то одной конфигурации, потому что мы можем к словам A и B дописывать пробельные символы. Поэтому, если быть супер формальными, речь идет о классах эквивалентности. Поэтому мы не говорим, что машина Тьюринга находится в такой конфигурации, а говорим, что происходит на ленте, если конфигурация такая. Здесь очень много всяких мелких деталей, поэтому мы их скрыли при построении теории алгоритмов.

Итак, в соответствии с программой и своей конфигурацией, МТ делает шаги. Один шаг состоит в исполнении одной инструкции вида $qa \mapsto q'a'S$.

Определение 5.4. Процесс (протокол) вычисления — конечная последовательность конфигураций, где каждая следующая конфигурация получается из предыдущей за один шаг машины.

Это определение можно себе представить следующим образом:

$$c_1 \underset{\mathcal{M}}{\to} c_2 \underset{\mathcal{M}}{\to} c_3 \underset{\mathcal{M}}{\to} \ldots \underset{\mathcal{M}}{\to} c_n,$$

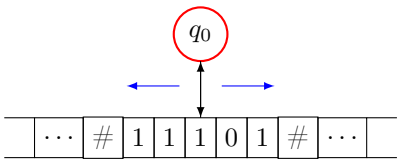
где $\mathcal{M}$ — обозначение машины Тьюринга, а c_i — i-ая конфигурация $\mathcal{M}$. Отношение «$\underset{\mathcal{M}}{\to}$» означает, что одна операция получается из другой за один шаг $\mathcal{M}$. Сам такой процесс мы можем понимать как *отношение достижимости конфигурации* за n шагов.

Мы написали много всего абстрактного, посмотрим, как вся эта теория может работать. Заведем МТ $\mathcal{M}$: $\Gamma = \{0, 1, \#\}$, $Q = \{q_0, q_1\}$. Зададим ее программу:

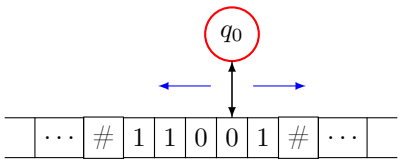
$$q_0 0 \mapsto q_0 1 R,$$
$$q_0 1 \mapsto q_0 0 R,$$
$$q_0 \# \mapsto q_1 \# N.$$

Пояснение текстом:

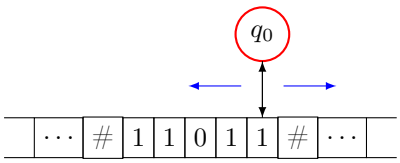
- если мы видим 0 в состоянии q_0, то давайте напишем 1 в состоянии q_0 и сдвинемся направо;
- если мы видим 1 в состоянии q_0, то давайте напишем 0 в состоянии q_0 и сдвинемся направо;
- если мы видим $\#$ в состоянии q_0, то перейдем в состояние q_1, напишем в этом состоянии $\#$, и никуда двигаться не будем.



(a) Состояние 1.



(b) Состояние 2.



(c) Состояние 3.

Это можно представить в виде такой последовательности состояний:

$$11q_0101 \underset{\mathcal{M}}{\to} 110q_001 \underset{\mathcal{M}}{\to} 1101q_01 \underset{\mathcal{M}}{\to} 11010q_0\#.$$

[6] машины Тьюринга — прим.

В последнем состоянии окажется, что головка остановилась на пробельном символе. Что же будет происходить в таком случае? Будет ли машина дальше двигаться? Если машина видит в состоянии q_0 решетку, то она переходит в конфигурацию q_1:

$$11010q_0\# \underset{\mathcal{M}}{\to} 11010q_1\#.$$

А дальше мы что-нибудь делать будем? Получается, что наша программа определена не полностью, хотя, формально, надо для любой ситуации описать, что нужно в ней делать. Практически, указывают только те пары, которые чем-либо нам интересны, а остальные пары, по умолчанию, считаются «нейтральными»:

$$q_1\# \mapsto q_1\#N.$$

Тогда получается, что начальная конфигурация

$$11q_0101 \underset{\mathcal{M}}{\twoheadrightarrow} {}^{7}11010q_1\#.$$

Получается, что процесс вычисления на $\mathcal{M}$ — это переписывание последовательности конфигураций. Пусть Σ — некоторое конечное множество, которое мы будем называть *входным алфавитом*.

Определение. Машина $\mathcal{M}$ вычисляет функцию $f : \Sigma^* \xrightarrow{p} \Sigma^*$, если $\Sigma \subseteq \Gamma$ и для любого слова $\overline{x} \in \Sigma^*$

$$\overline{x} \in \operatorname{dom} f \implies q\overline{x} \underset{\mathcal{M}}{\to} f(\overline{x}).$$

Возникает множество вопросов, которые надо уточнить:

Q: где должна находиться головка относительно входа?

A: давайте считать, что головка должна стоять на самой последней решетке перед входом.

Q: где должна быть головка после вычислений?

A: нам очень удобно считать, что после вычислений машина поставит головку перед результатом.

Q: а какие же у нас тут должны быть состояния?

A: на самом деле, с помощью состояний мы можем передавать некоторую дополнительную информацию. Договоримся, что хотя бы одно состояние (выделенное) считается *начальным* (обозн. q_1), и хотя бы одно состояние считается *завершающим* (обозн. q_0).

Тогда определение можно записать следующим образом:

Определение 5.5 (вычислимости функции)**.** Машина $\mathcal{M}$ *вычисляет функцию* $f : \Sigma^* \xrightarrow{p} \Sigma^*$, если $\Sigma \subseteq \Gamma$ и для любого слова $\overline{x} \in \Sigma^*$

$\overline{x} \in \operatorname{dom} f \implies q_1\#\overline{x} \underset{\mathcal{M}}{\twoheadrightarrow} q_0\#f(\overline{x})$.

$\overline{x} \in \operatorname{dom} f \implies$ для любой конфигурации c $q_1\#\overline{x} \not\underset{\mathcal{M}}{\to} c$, если c содержит q_0.

5.2 Формальное описание машины Тьюринга

Определение 5.6 (формальное определение МТ)**.** Машина Тьюринга $\mathcal{M}$ — это набор конечных множеств

$$\mathcal{M} = (\Gamma, \Sigma, Q, q_1, q_0, \delta),$$

где $\Sigma \subseteq \Gamma$, $\# \in \Gamma \setminus \Sigma$, $q_1, q_0 \in Q$, $\Gamma \cap Q = \varnothing$, $\delta : (Q \setminus \{q_0\}) \times \Gamma \to Q \times \Gamma \times \{L, N, R\}$ — функция переходов[8].

Конечно, формальное определение машины ничего нам не говорит об определении того, как же ведутся вычисления (то есть, мы пока ничего не знаем про отношение «$\underset{\mathcal{M}}{\to}$»). Давайте и его формально определим. По сути дела, определив формально МТ, мы заодно определили понятие конфигурации МТ. Пусть $\mathrm{Conf}_{\mathcal{M}}$ — это множество всех конфигураций $\mathcal{M}$. Определим «$\underset{\mathcal{M}}{\to}$» индуктивно как наименьшее по включению отношение такое, что $\forall q, q' \in Q$; $\forall a, a' \in A$; $\forall b \in B$

1. Если $\delta(q, a) = (q', a', R)$, то $AqabB \underset{\mathcal{M}}{\to} Aa'q'bB$ и $Aqa \mapsto Aa'q'\#$.
2. Аналогично (1) для сдвига N.
3. Аналогично (1) для сдвига L.

[7] через $c_1 \underset{\mathcal{M}}{\twoheadrightarrow} c_2$ я обозначаю достижимость конфигурации c_1 из конфигурации c_2 за конечное число шагов.

[8] ну или программа, которую выполняет $\mathcal{M}$.

Обратим внимание, что такое формальное определение отношения на конфигурациях не зависит от слова «лента». Соответственно, $c \underset{\mathcal{M}}{\twoheadrightarrow} c' \iff \exists c_1, c_n : c \underset{\mathcal{M}}{\to} c_1 \underset{\mathcal{M}}{\to} \dots \underset{\mathcal{M}}{\to} c_n \underset{\mathcal{M}}{\to} c'$. В частности, может быть, что $c = c'$. Тогда «$\underset{\mathcal{M}}{\twoheadrightarrow}$» — рефлексивное транзитивное замыкание отношения «$\underset{\mathcal{M}}{\to}$».

Давайте повторим, что означает вычислимость функции f, которая определена на словах конечного алфавита Σ. Это значит, что на всех словах, где функция определена, мы из исходной конфигурации попадаем в завершающую, а на всех словах, где она не определена, мы никогда не попадем в завершающую конфигурацию. Но ведь в теории алгоритмов у нас были функции на натуральных числах, то есть $f : \mathbb{N} \xrightarrow{p} \mathbb{N}$. Как же обработать эту ситуацию, учитывая тот факт, что f определена на словах конечного алфавита, а $\mathbb{N}$ счетно. Идея: закодируем натуральные числа через символы конечного алфавита. У нас есть несколько видов кодирования:

Унарное кодирование: Определим функцию унарного кодирования $u : \mathbb{N} \to \{1\}^*$. Положим код нуля $u(0) = \varepsilon \in \{1\}^* \leftarrow$ нулевое слово. Тогда $u(n+1) = 1u(n)$, то есть унарным кодом числа n является n последовательных единиц ($u(3) = 111$), код нуля — отсутствие единиц.

Бинарное кодирование: Определим функцию бинарного кодирования $b : \mathbb{N} \to \{0,1\}^*$ — обычная двоичная запись числа n: $b(0) = 0$, $b(2n) = b(n)0$, $b(2n+1) = b(n)1$.

Утверждение 5.1. *Функция $f : \mathbb{N} \to \mathbb{N}$ такая, что $f(n) = n + 1$ вычислима на МТ в унарных кодах.*

Доказательство. Существует МТ, которая по бинарному коду строит унарный и наоборот, поэтому от вида кодирования числа n результат зависеть не будет. Нужно показать, что существует вычислимая функция $g : \{1\}^* \to \{1\}^*$ такая, что $g(u(n)) = u(f(n)) = u(n+1)$. То есть, мы хотим построить такую МТ, что

$$\underbrace{1\dots1}_{n} \underset{\mathcal{M}}{\twoheadrightarrow} \underbrace{1\dots1}_{n+1}.$$

Самое главное — задать функцию перехода δ. Положим $Q = \{q_0, q_1\}$, $\Sigma = \{1\}$, $\Gamma = \{\#, 1\}$. Воспользуемся тем, что в унарной записи нет разницы, приписывать единицу в начало или в конец числа, поэтому определим δ следующим образом:

$$q_1\# \mapsto q_0 1 L.$$

Тогда у нас все получится. □

Важно понимать, что алфавит состояний не может быть бесконечным, как и не может зависеть от входа. Какую-то дополнительную информацию можно хранить на ленте с помощью дополнительных символов, которые мы можем ввести.

На практике, в интернете имеется множество ресурсов, которые позволяют вам написать код МТ и его отладить. Конечно, писать программу вслепую без отладки трудно, ведь даже такая простая программа как удвоение числа потребовала от нас каких-то усилий.

Итак, с помощью унарного кодирования мы поняли, как работать с функциями вида $f : \mathbb{N} \xrightarrow{p} \mathbb{N}$. А если, например, функция зависит от двух аргументов? Тогда можно ввести дополнительный символ-разделитель \$ $\in \Gamma$, и записать наши аргументы вот так

$$\#\text{code}(n)\$\text{code}(m) \underset{\mathcal{M}}{\twoheadrightarrow} \#\text{code}(f(n,m)).$$

Заметим, что без \$ можно обойтись в бинарном коде:

$$\text{code}\,((n,m)) = \text{double}(b(n))01\text{double}(b(m)),$$

где $abc \overset{\text{double}}{\to} aabbcc$, то есть double удваивает каждый символ в записи числа n. Понятно, что при таком подходе у нас никогда не встретится подстрока вида 01, поэтому кодирование выше является корректным. Например,

$$\text{code}\,((3,5)) = \underbrace{1111}_{n=3}01\underbrace{110011}_{m=5}.$$

5.3 Тезис Тьюринга

Тезис Тьюринга говорит нам, что любая функция, вычислимая в интуитивном смысле[9], вычислима некоторой машиной Тьюринга (при выборе подходящего кодирования аргументов и значений). По сути дела, этот принцип говорит нам, что машины Тьюринга соответствуют интуитивному понятию вычислений, то есть любое вычисление можно реализовать на машине Тьюринга. Есть[10] философская аргументация в пользу этого тезиса. Можно сравнить работу МТ с вычислениями, которые человек мог бы делать на листах бумаги, что, мол, на каждом листе бумаги вы можете написать лишь конечное число различимых текстов, значит каждый лист бумаги можно рассматривать как ячейку, содержащую один из символов конечного алфавита. Состояний вашего мозга конечно много, то есть мозг — это такая головка, у нее конечное число состояний. Вы берете листы и обозреваете в каждый момент только один, переходите от одного к другому перелистывая их, что очень похоже на вычисления, производимые МТ.

[9] что бы это не значило.

[10] см. в книге Верещагина-Шеня «Лекции по математической логике. Часть 3. Вычислимость.».

5.4 Свойства алгоритмов в терминах машин Тьюринга

Попробуем проследить связь всей этой теории про МТ со свойствами алгоритмов.

«Алгоритмов лишь счетно много»: Можно ли сказать, что машин Тьюринга счетно много? Как мы помним, $\mathcal{M} = (\Gamma, \Sigma, Q, q_0, q_1, \delta)$, где Γ, Σ, Q конечны; $q_0, q_1 \in Q$; $\delta : (Q \setminus q_0) \times \Gamma \to Q \times \Gamma \times \{L, N, R\}$. То есть вопрос сводится к тому, сколько существует конечных множеств? Ответ: очень много. То есть настолько много, что нельзя говорить о том, что существует множество всех конечных множеств. С точки зрения формальной теории множеств, все конечные множества образуют т. н. *собственный класс* (как и все возможные множества), то есть множество они не образуют. Следовательно, машин Тьюринга тоже очень много. Как же получить счетное число таких машин? На самом деле, многие МТ отличаются друг от друга только названиями состояний и символов, то есть они изоморфны. Две МТ $(\Gamma, \Sigma, Q, q_0, q_1, \delta)$ и $(\Gamma', \Sigma', Q', q_0', q_1', \delta')$ называются *изоморфными*, если существует отображение[11] φ такое, что $\Gamma \overset{\varphi}{\sim} \Gamma'$, $Q \overset{\varphi}{\sim} Q'$, $\varphi(q_0) = q_0'$, $\varphi(q_1) = q_1'$, $\delta(q, a) = (t, b, S) \iff \delta(\varphi(q), \varphi(a)) = (\varphi(t), \varphi(b), S)$. С точностью до изоморфизма существует лишь счетно много машин Тьюринга, как и алгоритмов.

«Композиция вычислимых функций вычислима»: Пусть у нас машина $\mathcal{M}$ вычисляет функцию $f : \Sigma^* \overset{p}{\to} \Sigma^*$, а $\mathcal{N}$ вычисляет $g : \Sigma^* \overset{p}{\to} \Sigma^*$. У нас $\mathcal{M} = (\Gamma, \Sigma, Q, q_0, q_1, \delta), \mathcal{N} = (\Gamma', \Sigma, Q', q_0', q_1', \delta')$. С точностью до изоморфизма, считаем, что $Q \cap Q' = \varnothing$. Построим машину $\mathcal{L} = \left(\Gamma \cup \Gamma', \Sigma, Q \cup Q', q_0, q_1', \hat{\delta}\right)$, где $\hat{\delta} = \delta \cup \delta' \cup \{q_1\# \to q_0'\#N\}$. Поскольку

$$q_0\#\overline{x} \underset{\mathcal{M}}{\twoheadrightarrow} q_1\#f(\overline{x}) \underset{\mathcal{L}}{\twoheadrightarrow} q_0'\#f(\overline{x}) \underset{\mathcal{N}}{\twoheadrightarrow} q_1'\#g\left(f\left(\overline{x}\right)\right),$$

то машина $\mathcal{L}$ позволяет вычислить композицию $f \circ g$, ведь она содержит в себе машины $\mathcal{M}$ и $\mathcal{N}$. Формально, состояние q_1' не является завершающим для машины $\mathcal{N}$, поэтому мы должны доопределить $\hat{\delta}$, чтобы все было корректно.

«Существование у. в. ф.»: Пусть алфавит $\Sigma \supseteq \{0, 1\}$ фиксирован. Почему можно алфавит зафиксировать? Ну потому что можно доказать, что любой алфавит можно, путем кодирования его символов, заменить алфавитом из нулей из единиц. То есть если функция, которая работает на бинарных кодах, вычислима в одном алфавите, содержащем нули и единицы, то она вычислима некоторой машиной алфавите, содержащем только нули и единицы. Поэтому выбор алфавита не очень существеннен, а ограничение на наличие в нем 0 и 1 довольно слабое. Любую машину с таким Σ можно кодировать в $\{0, 1\}$, то есть положим $\Gamma = \{\text{code}(a) \mid a \in \Gamma\}$, $Q = \{q_0, \ldots, q_n\} = \{\$b(i)\$ \mid i \in \{0, \ldots, n\}\}$, δ — список конечных инструкций $qa \mapsto q'a'S'$, каждую из которых можно закодировать, то есть каждая такая инструкция — слово конечного алфавита $\implies$ его можно закодировать двоичным кодом. Поэтому каждая $\mathcal{M} \mapsto \text{code}(\mathcal{M}) \in \{0, 1\}^* \implies$ любая $\mathcal{M}$ — это конечный объект.

Теорема 5.1. *Существует МТ $\mathcal{U}$ с входным алфавитом Σ такая, что для любой МТ $\mathcal{M}$ с входным алфавитом Σ*

$$\forall \overline{x} \in \Sigma^* \quad \mathcal{U}\left(\langle \text{code}\left(\mathcal{M}\right), \overline{x}\rangle\right) \simeq \mathcal{M}(\overline{x}).$$

Здесь «$\simeq$» означает, что либо обе машины приходят к одной завершающей конфигурации, либо обе не останаливаются (никогда не приходят к завершающей конфигурации).

«Возможность выполнять по шагам»: Можно доказать, что существует машина $\mathcal{T}$ такая, что для любой машины $\mathcal{M}$ и для любых $\overline{x}, \overline{y}$, а также для любого $k \in \mathbb{N}$

$$TT\left(\langle \text{code}(\mathcal{M}), \overline{x}, \overline{y}, \mathcal{U}(k)\rangle\right) = \begin{cases} 1, & \text{если } M \text{ на входе } \overline{x} \text{ останавливается за } k \text{ шагов и выдает } \overline{y}, \\ 0, & \text{иначе.} \end{cases}$$

[11] по сути, такие МТ отличаются переименованием элементов множеств.

6 Лекция 6

6.1 Формулы и термы языка первого порядка

Сегодня мы начинаем изучение логики. Давайте немножко поговорим о том, что это такое. Вообще, логика — это такая наука, которая посвящена анализу различных рассуждений, как формальных математических, так и, может быть, не совсем формальных философких, юридических или каких-нибудь иных. То, что называется *математической логикой*, в основном, посвящено изучению различных доказательств, разных рассуждений, которые встречаются в математике, математическими же методами. Например, обычная логика стремится отделить корректные рассуждения, в которых из истиных посылок всегда выводят истинные следствия, от некорректных. И математическая логика тоже не чужда этой задаче. При этом, нам с вами нужно будет точно определить, что же значит истинность утверждения, что такое утверждение или высказывание. Мы не будем пытаться решить эту задачу в ее общефилософской широте. Мы попробуем смоделировать те высказывания, с которыми работает сама математика (т. е. те, которые встречаются в разных ее областях). Начнем с изучения какого-то конкретного примера.

Пример Рассмотрим арифметику натуральных или целых чисел. Давайте посмотрм, что же у нас в арифметике есть. В арифметике бывают:

Высказывания например, «$2 = 3$» или «$2 < 3$». Хочется верить, что первое высказывание ложно, а второе истинно. Так оно, в общем-то, и есть, если у нас фиксирован смысл чисел 2, 3, операций «=» и «<».

Не высказывания например, «$2x = 3 + y$» не является высказыванием, но почему? Потому что в нем присутствуют переменные. То есть мы не можем ничего утверждать про его истинность или ложность, не определив значения *переменных* x и y.

Тем не менее, у этих объектов есть нечто общее. В конечном счете, может быть, по модулю придания переменным каких-то значений, они обозначают какое-то утверждение (или высказывание), возможно, с параметрами. Высказывания с параметрами также называются *предикатами*. Сами такие выражения ($2 = 3$ или $2x = 3 + y$) мы будем называть *формулами*. Формула обозначает предикат. Бывают еще формулы вида

$$\exists x : \quad \forall y \quad x \leqslant y. \tag{3}$$

Такая формула с кванторами также обозначает некоторое высказывание. Обратим внимание, что вот это высказывание будет ложным для $\mathbb{N}$, но ложным для $\mathbb{Z}$. Что можно делать с такими выражениями? Их можно комбинировать (или группировать) с помощью некоторых *логических связок*. В принципе, набор связок можно выбирать произвольно, но можно доказать, что всегда достаточно конъюнкции и отрицания (или дизъюнкции и отрицания). Мы будем пользоваться связками $\wedge, \vee, \implies, \neg$. Что же такое связки? С чисто синтаксической точки зрения, связки — это такие значки, которые позволяют из одной формулы делают другую, например если у нас была формула

$$\exists x : \quad x + 2 = 1, \tag{4}$$

вы можете взять и написать, например, такое утверждение:

$$((\exists x : x + 2 = 1) \vee \neg\,(2x = 3 + y)) \implies 3 = 5 + z.$$

Из чего образуются формулы? Давайте посмотрим на выражения вида

$$2 + x, \quad 3 \cdot y, \quad z, \quad (z + x) \cdot x + 8, \quad \ldots$$

Они обозначают числа, зависящие от параметров. А что такое числа? Числа — это элементы нашей предметной области, о которой мы говорим, то есть может быть натуральные числа, может быть целые числа, как мы договоримся. То есть, формулы обозначают элементы предметной области, о которой наша математическая теория ведет разговор. Такие выражения называются *термами*[12]. Содержательно, терм — это имя числа или какого-то другого объекта, который мы рассматриваем. Но у некоторых объектов могут быть имена собственные, например

- 2, 3 — константы.
- x, y — переменные.

Еще мы можем строить термы следующим образом: можно взять два каких-то терма, и соединить их символом какой-то операции или функции:

$$2 + x$$

[12] терм — от слова термин.

Здесь «+» — символ бинарной операции (функции). То есть термы можно также называть *функциональными выражениями*. Они обозначают результат применения каких-то функций к каким-то константам или переменным. Обратите внимание, термы и формулы являются некими разными сущностями. В чем же между ними разница? Терм обозначает число, а формула обозначает высказывание, при этом и число, и высказывание могут зависеть от параметров. При этом, к формуле мы можем задать вопрос[13] «верно это или нет?», а к терму мы можем задать вопрос «чему это равно?». То есть формула — это какое-то утверждение, а терм — имя объекта.

Как же обстоит дело в общем случае? Давайте попробуем это понять. В арифметике нам, до сих пор, встретились некоторые константы (2, 3, 8), были символы переменных (x, y, z), были символы служебные (круглые скобки) и были символы «+» и «·». Давайте попробуем к этому всему подойти абстрактно. Как вообще можно было бы абстрактно определить терм? Определение это зависит от списка символов, над которыми эти термы определяются. Ну и аналогично, для определения формул, нам требовались служебные символы, логические символы (связки, кванторы) и символы, специфические для нашей области (например, символы «=» и «<»).

6.2 Сигнатуры

Если все эти символы собрать в одну кучу, получится то, что называется *сигнатурой* — список символов, которые используются для построения формул в той или иной математической теории. Конечно же, сигнатуру можно определить и формально. Посмотрим, для начала, какая же у нас сигнатура в арифметике:

- Символы отношений[14] (предикатов): $=^{(2)}$, $<^{(2)}$, $\leqslant^{(2)}$, $>^{(2)}$, $\geqslant^{(2)}$, ….
- Символы операций (функций): $+^{(2)}$, $\cdot^{(2)}$, $(\cdot)+1^{(1)}$, ….
- Символы констант: $2, 3, \ldots$.

Теперь попробуем определить сигнатуру абстрактно.

Определение 6.1. Сигнатурой σ называется тройка множеств

$$\sigma = (\mathrm{Rel}, \mathrm{Func}, \mathrm{Const}),$$

где

- $\mathrm{Rel} \neq \varnothing$ — множество символов отношений (или предикатных символов).
- Func — множество функциональных символов.
- Const — множество символов констант.

Обратите внимание на следующий интелектуальный трюк: в принципе, мы можем считать саму операцию «+» символом для себя, но нам это не выгодно. Почему? Потому что точно также можно считать операцию «$\leqslant$» символом для себя, однако проще про них думать как про отдельную сущность. Есть один символ, есть формула (3), но эта формула, в зависимости от интерпертации x и y как натуральных чисел или как целых, ведет себя по-разному. То есть нам выгодно разделить имя отношения и само отношение. Имя может быть одно, формула одна, а поведение этой формулы, при разной интерпретации, может быть разным. Так, например, уравнение (4) одно, а наличие у него решения зависит от того, к какой области мы его применяем, и, по-сути дела, как мы понимаем «+»: то ли как функцию сложения целых чисел, то ли как функцию сложения натуральных. Поэтому мы отделяем само отношение от его обозначения.

На самом деле, $\mathrm{Rel}, \mathrm{Func}$ и Const — это не просто какие-то множества, а еще объектам из этих множеств приписано некоторое натуральное число, называемое *валентностью* или *арностью*. То есть, например, для любого $R \in \mathrm{Rel}$ приписано $n \in \mathbb{N}$ такое, что $R^{(n)} \in \mathrm{Rel}$. Аналогично, валентности приписаны и функциональным символам, то есть $f^{(m)} \in \mathrm{Func}$. Давайте теперь попытаемся понять, как же у нас определяются формулы. Как уже было сказано, формулы строятся из термов, поэтому сначала нужно определить понятие терма. Для начала зафиксируем, что считается общим для всех сигнатур:

- алфавит переменных $\mathrm{Var} = \{v_0, v_1, v_2, \ldots\} \sim \mathbb{N}$. По умолчанию, разные буквы x, y, z, etc, обозначают какие-то разные v_i.
- символы связи $\wedge, \vee, \Longrightarrow, \neg, \Longleftrightarrow$.
- кванторы $\exists$ и $\forall$.
- скобки (,).

Из всего этого зоопарка мы будем конструировать формулы и сигнатуры.

[13] при определенном выборе значений параметров.

[14] (i) в правом верхнем углу символа отношения или операции обозначает его арность, то есть (2) означает, что отношение бинарное.

Определение 6.2 (терма в арифметике)**.** Будем определять термы индуктивно:

1. $x \in \mathrm{Var} \implies x$ — это терм. Будем обозначать этот факт как $x \in \mathrm{Tm}$, где Tm является множеством термов.
2. $c \in \mathrm{Const} \implies c \in \mathrm{Tm}$.
3. $t, s \in \mathrm{Tm} \implies (t + s) \in \mathrm{Tm}, (t \cdot s) \in \mathrm{Tm}, t{+}{+} \in \mathrm{Tm}$.

В соответствии с этими правилами мы можем доказать, что выражение

$$(2 + x){+}{+}$$

является термом, потому что термом является $2 + x$, которое является термом, потому что 2 и x являются термами. Будем считать, что множество термов является наименьшим по включению среди всех множеств, удовлетворяющих индуктивному определению 6.2.

Определение 6.3. Будем обозначать через Tm_σ множество термов в сигнатуре σ.

Определение 6.4 (терма в произвольной сигнатуре)**.** Зафиксируем сигнатуру σ и определяем индуктивно:

1. $x \in \mathrm{Var} \implies x \in \mathrm{Tm}_\sigma$.
2. $c \in \mathrm{Const}_\sigma \implies c \in \mathrm{Tm}_\sigma$.
3. $f^{(m)} \in \mathrm{Func}_\sigma, t_1, \ldots, t_m \in \mathrm{Tm}_\sigma \implies f t_1 \ldots t_m \in \mathrm{Tm}_\sigma$.

Обратим внимание, что последний пункт определения записан в польской нотации, то есть, например, $2{+}x \mapsto {+}2x$, а $(2 + x) \cdot (y + x) \mapsto \cdot + 2x + yx$.

Определение 6.5 (формулы в арифметике)**.** Определяем индуктивно:

1. $t, s \in \mathrm{Tm} \implies t = s, t < s, t \leqslant s, t > s, t \geqslant s \in \mathrm{Fm}$[15].
2. $\varphi, \psi \in \mathrm{Fm} \implies (\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \implies \psi), (\varphi \iff \psi), \neg\varphi \in \mathrm{Fm}$.
3. $x \in \mathrm{Var}, \varphi \in \mathrm{Fm} \implies \forall x\ \varphi, \exists x\ \varphi \in \mathrm{Fm}$.

Пример $(2 + x){+}{+}$ — это терм. $y + 3 \cdot z$ — это тоже терм. Тогда из них мы можем организовать какую-то формулу, соединив их чем-нибудь, например, символом «<»:

$$(2 + x){+}{+} < y + 3 \cdot z.$$

Как видим, получилась формула. Можно на нее навесить знак отрицания,

$$\neg\left((2 + x){+}{+} < y + 3 \cdot z\right),$$

получится тоже формула. Можно еще и квантор накинуть:

$$\forall w \quad \neg\left((2 + x){+}{+} < y + 3 \cdot z\right).$$

Ну и так далее.

Определение 6.6 (формулы в произвольной сигнатуре)**.** Зафиксируем сигнатуру σ и определяем индуктивно:

1. $R^{(n)} \in \mathrm{Rel}_\sigma; t_1, \ldots, t_n \in \mathrm{Tm}_\sigma \implies R t_1 \ldots t_n \in \mathrm{Fm}_\sigma$.
2. $\varphi, \psi \in \mathrm{Fm}_\sigma \implies (\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \implies \psi), (\varphi \iff \psi), \neg\varphi \in \mathrm{Fm}_\sigma$.
3. $x \in \mathrm{Var}, \varphi \in \mathrm{Fm}_\sigma \implies \forall x\ \varphi, \exists x\ \varphi \in \mathrm{Fm}_\sigma$.

Введя понятия терма и формулы, введем некоторые синтаксические понятия, которые с ними естественным образом связаны. Заведем множество $V(t) =$ «все переменные, которые встречаются в t». Определение, конечно, хорошее, но про него сложно что-то говорить формально. Попробуем зайти с другой стороны.

Определение 6.7. Определим множество V всех переменных, которые встречаются в терме t рекурсивно:

1. Если $t = x \in \mathrm{Var}$, то $V(t) = \{x\}$.
2. Если $t = c \in \mathrm{Const}_\sigma$, то $V(t) = \varnothing$.
3. Если $t = f t_1 \ldots t_m$, то $V(t) = V(t_1) \cup \ldots \cup V(t_m)$.

[15] формулы такого вида называются *атомарными*.

Пример

$$V\left(3+((x+y)\cdot 2)\right)=V(3)\cup V\left((x+y)\cdot 2\right)=\varnothing\cup V\left(2\right)\cup V\left(x+y\right)=\varnothing\cup V(x)\cup V(y)=\{x,y\}.$$

Определение 6.8. Определим функцию $V:\mathrm{Fm}_\sigma\to\mathrm{Var}$, которая каждой формуле ставит в соответствие множество ее переменных. Действуем рекурсивно:

1. $V\left(Rt_1\dots t_n\right)=V(t_1)\cup\ldots\cup V(t_n)$.
2. $V\left(\varphi\vee\psi\right)=V\left(\varphi\wedge\psi\right)=V\left(\varphi\implies\psi\right)=\ldots=V(\varphi)\cup V(\psi)$.
3. $V(\exists x\ \varphi)=V(\forall x\ \varphi)=V(\varphi)\cup\{x\}$.

Пример

$$V\left(\forall x\ \exists y\ \ (x=z+3)\right)=\{x,y,z\}\,.$$

На самом деле, для того, чтобы придать формулам какое-то значение, научиться выяснять, когда же формула истинна, а когда — ложна, нам очень важно отделять те переменные, которые связаны кванторами от тех, которые кванторами не связаны и вместо которых можно что-то подставлять. Давайте рассмотрим формулу

$$\exists x\ (x+2=1)$$

с семантической точки зрения. Можно ли сказать, что это свойство какого-то конкретного x? Нет, это свойство всей нашей области (натуральных чисел, или же вещественных). То есть, здесь, по сути дела, сказано, что у уравнения есть решение, это не свойство какого-то конкретного решения (например, в области натуральных чисел такого x нет, но оно есть в области вещественных чисел). Такое вхождение переменной называется *связанным*. Мы сталкивались с подобными вещами, например, в анализе, где говорили что существует первообразная $\int\sin x\mathrm{d}x$ функции sin. Имеет ли здесь x какую-то индивидуальность? Нет, конечно, ведь запись $\int\sin x\mathrm{d}x$ эквивалента $\int\sin y\mathrm{d}y$, тогда как $\sin x\neq\sin y\leftarrow$ здесь переменная x является *свободной*.

Важно понимать, что в одну и ту же формулу x может иметь как свободное вхождение, так и связанное, например

$$x=3\quad\vee\quad\exists x\ x+2=1.$$

Здесь первый x имеет свободное вхождение, а второй x — связанное. Получается, что хоть обозначение и одно, вхождения разные, ведь в первом случае что-то говорится про конкретный объект x, конкретное число, а во втором случае что-то говорится про всю область наших чисел (что среди них имеется решение нашего уравнения). То есть это, по сути дела, разные x, и их возможное совпадение может считаться случайным.

Определение 6.9 (свободная переменная)**.** Зафиксируем функцию $\mathrm{FV}:\mathrm{Fm}_\sigma\to\mathrm{Var}$, которая каждой формуле ставит в соответствие множество *свободных* переменных в ней, следующим образом:

1. $\mathrm{FV}\left(Rt_1\dots t_n\right)=V(t_1)\cup\ldots\cup V(t_n)$.
2. $\mathrm{FV}(\varphi\wedge\psi)=\mathrm{FV}(\varphi\vee\psi)=\ldots=\mathrm{FV}(\varphi)\cup\mathrm{FV}(\psi)$.
3. $\mathrm{FV}\left(\forall x\ \varphi\right)=\mathrm{FV}\left(\exists x\ \varphi\right)=\mathrm{FV}(\varphi)\setminus\{x\}$.

6.3 Доказательства индукцией по построению

Допустим мы хотим доказать, что все $\varphi\in\mathrm{Fm}_\sigma$ удовлетворяют свойству P, то есть $\forall\varphi\ P(\varphi)$. Для этого достаточно доказать, что

1. $\forall R^{(n)}\in\mathrm{Rel}_\sigma,\ \forall t_1,\dots,t_n\in\mathrm{Tm}_\sigma\ P(Rt_1\dots t_n)$.
2. $\forall\varphi,\psi\ P(\varphi)\wedge P(\psi)\implies P(\varphi\wedge\psi)\wedge P(\varphi\vee\psi)\wedge\ldots$.
3. $\forall\varphi\ \forall x\in\mathrm{Var}\ P(\varphi)\implies P(\forall x\ \varphi)\wedge P(\exists x\ \varphi)$.

Докажем, в качестве упражнения, следующую лемму:

Лемма 6.1. $\forall\varphi\in\mathrm{Fm}_\sigma\ \mathrm{FV}(\varphi)\subseteq V(\varphi)$.

Доказательство. Нам надо показать, что это утверждение верно для всех атомарных формул, а также, если это верно для простых формул, то верно и для более сложных.

1. $\mathrm{FV}(Rt_1\dots t_n)\stackrel{\text{def}}{=}V(t_1)\cup\ldots\cup V(t_n)\stackrel{\text{def}}{=}V(Rt_1\dots t_n)$.
2. Допустим, что $\mathrm{FV}(\varphi)\subseteq V(\varphi)$ и $\mathrm{FV}(\psi)\subseteq V(\psi)$. Тогда $\mathrm{FV}(\varphi\implies\psi)=\mathrm{FV}(\varphi)\cup\mathrm{FV}(\psi)$, в тоже время $V(\varphi\implies\psi)=V(\varphi)\cup V(\psi)$, но тогда, по предположению индукции, $\mathrm{FV}(\varphi\implies\psi)\subseteq V(\varphi\implies\psi)$. Аналогично для остальных операций.
3. Допустим, что $\mathrm{FV}(\varphi)\subseteq V(\varphi)$. Хотим, чтобы $\mathrm{FV}(\forall x\ \varphi)\subseteq V(\forall x\ \varphi)$. Тогда $\mathrm{FV}(\forall x\ \varphi)=\mathrm{FV}(\varphi)\setminus\{x\}\subseteq V(\varphi)\cup\{x\}$. □

Что же означают формулы и термы Идея: если всем переменным придать значения, то значение формулы $\in \{0,1\}$, а значением терма будет некоторое «число». Откуда все эти значения берутся и что это за числа? Нам нужно множество, из которого мы будем брать значение переменных. Ну хорошо, предположим, что это какое-то множество M. Рассмотрим формулу вида

$$3 + x.$$

Тогда мы должны уметь сложить 3 и элемент из M, но тогда тройка должна быть элементом из M, а «+» надо понимать как функцию на множестве M. Но тогда нужно и все константы понимать как элементы множества M, и все функциональные символы должны означать функции на M. Но какие? Ответ на эти вопросы называется *структура сигнатуры*, но это уже в следующих лекциях.