Теория алгоритмов
Вычислимость

Математическая логика и теория алгоритмов

Алексей Романов

26 декабря 2024 г.

МИЭТ

Алгоритмы и программы

- Алгоритм задаётся программой на каком-то абстрактном языке программирования.
- Язык позволяет работать с натуральными числами произвольной величины.
- Или другими объектами, задаваемыми конечными словами в конечном алфавите (элементами какого-то *дискретного множества*).
- Не взаимодействует с внешним миром, кроме получения аргумента/ов и возвращения результата.
- Не использует случайных чисел и других источников недетерминированности.
- Конкретные примеры таких языков в конце лекции.

Функции, вычисляемые алгоритмами

- Каждый алгоритм (программа) тогда вычисляет какую-то функцию.
- Эта функция может быть не везде определена (если алгоритм не останавливается или останавливается, не выдав осмысленного результата).
- В этом разделе функции по умолчанию могут быть частичными, всюду определённость оговаривается особо.
- Мы говорим в первую очередь о функциях $\mathbb{N} \to \mathbb{N}$, но областями определения и значений могут быть и любые другие дискретные множества.
- Разные алгоритмы могут вычислять одну и ту же функцию.

Вычислимые функции

- Функция *вычислима*, если какой-то алгоритм её вычисляет.
- Например, нигде не определённая функция вычислима

Вычислимые функции

- Функция *вычислима*, если какой-то алгоритм её вычисляет.
- Например, нигде не определённая функция вычислима алгоритмом, который зацикливается на любом входе.

Вычислимые функции

- Функция *вычислима*, если какой-то алгоритм её вычисляет.
- Например, нигде не определённая функция вычислима алгоритмом, который зацикливается на любом входе.
- Или функция $x \mapsto x^2$.
- Зависит ли это от языка?

Вычислимые функции

- Функция *вычислима*, если какой-то алгоритм её вычисляет.
- Например, нигде не определённая функция вычислима алгоритмом, который зацикливается на любом входе.
- Или функция $x \mapsto x^2$.
- Зависит ли это от языка? Скажем, если там нет операции умножения?

Вычислимые функции

- Функция *вычислима*, если какой-то алгоритм её вычисляет.
- Например, нигде не определённая функция вычислима алгоритмом, который зацикливается на любом входе.
- Или функция $x \mapsto x^2$.
- Зависит ли это от языка? Скажем, если там нет операции умножения?
- Есть невычислимые функции $\mathbb{N} \to \mathbb{N}$

Вычислимые функции

- Функция *вычислима*, если какой-то алгоритм её вычисляет.
- Например, нигде не определённая функция вычислима алгоритмом, который зацикливается на любом входе.
- Или функция $x \mapsto x^2$.
- Зависит ли это от языка? Скажем, если там нет операции умножения?
- Есть невычислимые функции $\mathbb{N} \to \mathbb{N}$ просто потому, что всех таких функций

Вычислимые функции

- Функция *вычислима*, если какой-то алгоритм её вычисляет.
- Например, нигде не определённая функция вычислима алгоритмом, который зацикливается на любом входе.
- Или функция $x \mapsto x^2$.
- Зависит ли это от языка? Скажем, если там нет операции умножения?
- Есть невычислимые функции $\mathbb{N} \to \mathbb{N}$ просто потому, что всех таких функций $\mathfrak{c}$, а программ и вычислимых функций

Вычислимые функции

- Функция *вычислима*, если какой-то алгоритм её вычисляет.
- Например, нигде не определённая функция вычислима алгоритмом, который зацикливается на любом входе.
- Или функция $x \mapsto x^2$.
- Зависит ли это от языка? Скажем, если там нет операции умножения?
- Есть невычислимые функции $\mathbb{N} \to \mathbb{N}$ просто потому, что всех таких функций $\mathfrak{c}$, а программ и вычислимых функций $\aleph_0$.
- Из простых вычислимых функций можно строить более сложные. Например, композиция вычислимых функций вычислима (почему?).

Разрешимые множества

- Множество $A \subseteq \mathbb{N}$ *разрешимо*, если есть алгоритм, позволяющий проверить принадлежность к нему любого натурального числа: получает на вход n и выдаёт 1, если $n \in A$ и 0, если $n \notin A$.
- То есть вычислима характеристическая функция $\chi_A(n) =$ если $n \in A$ то 1 иначе 0 (короче: $\chi_A(n) = n \in A$).

Разрешимые множества

- Множество $A \subseteq \mathbb{N}$ *разрешимо*, если есть алгоритм, позволяющий проверить принадлежность к нему любого натурального числа: получает на вход n и выдаёт 1, если $n \in A$ и 0, если $n \notin A$.
- То есть вычислима характеристическая функция $\chi_A(n) =$ если $n \in A$ то 1 иначе 0 (короче: $\chi_A(n) = n \in A$).
- Вместо $\mathbb{N}$ можно взять любое дискретное множество.

Разрешимые множества

- Множество $A \subseteq \mathbb{N}$ *разрешимо*, если есть алгоритм, позволяющий проверить принадлежность к нему любого натурального числа: получает на вход n и выдаёт 1, если $n \in A$ и 0, если $n \notin A$.
- То есть вычислима характеристическая функция $\chi_A(n) =$ если $n \in A$ то 1 иначе 0 (короче: $\chi_A(n) = n \in A$).
- Вместо $\mathbb{N}$ можно взять любое дискретное множество.
- Может быть известно, что множество разрешимо (или что функция вычислима), но неизвестно, как именно. Классический пример: $A = \{n \mid$ в десятичной записи π есть n нулей подряд$\}$.

Разрешимые множества

- Множество $A \subseteq \mathbb{N}$ *разрешимо*, если есть алгоритм, позволяющий проверить принадлежность к нему любого натурального числа: получает на вход n и выдаёт 1, если $n \in A$ и 0, если $n \notin A$.
- То есть вычислима характеристическая функция $\chi_A(n) =$ если $n \in A$ то 1 иначе 0 (короче: $\chi_A(n) = n \in A$).
- Вместо $\mathbb{N}$ можно взять любое дискретное множество.
- Может быть известно, что множество разрешимо (или что функция вычислима), но неизвестно, как именно. Классический пример: $A = \{n \mid$ в десятичной записи π есть n нулей подряд$\}$.
- Теорема: объединение, пересечение и дополнение разрешимых множеств разрешимы.
- Теорема: декартово произведение разрешимых множеств разрешимо (как подмножество $\mathbb{N} \times \mathbb{N}$).

Перечислимые множества

- Множество $A \subseteq \mathbb{N}$ *перечислимо*, если есть алгоритм, позволяющий подтвердить принадлежность к нему, но не опровергнуть: получает на вход n и выдаёт 1, если $n \in A$ и ничего не выдаёт, если $n \notin A$.
- То есть вычислима полухарактеристическая функция $\bar{\chi}_A(n) =$ если $n \in A$ то 1 иначе не определена.
- Все следующие утверждения эквивалентны:
  - A перечислимо.
  - $A = \{n : \mathbb{N} \mid f(n)$ определена$\}$ для какой-то вычислимой f.
  - $A = \{f(n) \mid n : \mathbb{N}\}$ для какой-то вычислимой f.
  - Есть алгоритм без входа, перечисляющий элементы A.
- Теорема: объединение и пересечение перечислимых множеств перечислимы.
- Теорема: декартово произведение перечислимых множеств перечислимо.
- Теорема: если множество и его дополнение перечислимы, то оно разрешимо.

Универсальные вычислимые функции

- Пусть есть какое-то множество $X \subseteq \mathbb{N} \to \mathbb{N}$ функций одного аргумента.
- Функция двух аргументов $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ называется *универсальной для X*, если $X = \{m \mapsto f(n, m) \mid n : \mathbb{N}\}$.
- *Универсальная вычислимая функция* это вычислимая функция двух аргументов, универсальная для класса всех вычислимых функций одного аргумента.
- Теорема: существует универсальная вычислимая функция.
- Доказательство:

Универсальные вычислимые функции

- Пусть есть какое-то множество $X \subseteq \mathbb{N} \to \mathbb{N}$ функций одного аргумента.
- Функция двух аргументов $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ называется *универсальной для* X, если $X = \{m \mapsto f(n, m) \mid n : \mathbb{N}\}$.
- *Универсальная вычислимая функция* это вычислимая функция двух аргументов, универсальная для класса всех вычислимых функций одного аргумента.
- Теорема: существует универсальная вычислимая функция.
- Доказательство: перенумеруем все программы нашего языка, например, в порядке возрастания длины, а при одинаковой длине по алфавиту: $p_0, p_1, \ldots$
- Определим $U(n, m) = p_n(m)$ (то есть перечисляем программы, пока не дойдём до p_n, и запускаем её на аргументе m).

Универсальные и всюду определённые функции

- Теорема: не существует всюду определённой вычислимой функции, универсальной для класса всюду определённых вычислимых функций одного аргумента.
- Доказательство: предположим, что такая функция есть, и обозначим $V(n,m)$. Определим $V'(n) = V(n,n) + 1$. Она

Универсальные и всюду определённые функции

- Теорема: не существует всюду определённой вычислимой функции, универсальной для класса всюду определённых вычислимых функций одного аргумента.
- Доказательство: предположим, что такая функция есть, и обозначим $V(n,m)$. Определим $V'(n) = V(n,n) + 1$. Она вычислима,

Универсальные и всюду определённые функции

- Теорема: не существует всюду определённой вычислимой функции, универсальной для класса всюду определённых вычислимых функций одного аргумента.
- Доказательство: предположим, что такая функция есть, и обозначим $V(n,m)$. Определим $V'(n) = V(n,n) + 1$. Она вычислима, всюду определена, и

Универсальные и всюду определённые функции

- Теорема: не существует всюду определённой вычислимой функции, универсальной для класса всюду определённых вычислимых функций одного аргумента.
- Доказательство: предположим, что такая функция есть, и обозначим $V(n,m)$. Определим $V'(n) = V(n,n) + 1$. Она вычислима, всюду определена, и не совпадает ни с одним сечением V (почему?).

Универсальные и всюду определённые функции

- Теорема: не существует всюду определённой вычислимой функции, универсальной для класса всюду определённых вычислимых функций одного аргумента.
- Доказательство: предположим, что такая функция есть, и обозначим $V(n, m)$. Определим $V'(n) = V(n, n) + 1$. Она вычислима, всюду определена, и не совпадает ни с одним сечением V (почему?).
- Почему это доказательство не работает для класса всех вычислимых функций и функции U? В каком случае возможно $U' = m \mapsto U(m, m) + 1 = m \mapsto U(n, m)$?

Универсальные и всюду определённые функции

- Теорема: не существует всюду определённой вычислимой функции, универсальной для класса всюду определённых вычислимых функций одного аргумента.
- Доказательство: предположим, что такая функция есть, и обозначим $V(n, m)$. Определим $V'(n) = V(n, n) + 1$. Она вычислима, всюду определена, и не совпадает ни с одним сечением V (почему?).
- Почему это доказательство не работает для класса всех вычислимых функций и функции U? В каком случае возможно $U' = m \mapsto U(m, m) + 1 = m \mapsto U(n, m)$?
- Теорема: существует вычислимая функция f, не имеющая всюду определённого вычислимого продолжения (функции, совпадающей с f на $Dom\ f$).
- Доказательство:

Универсальные и всюду определённые функции

- Теорема: не существует всюду определённой вычислимой функции, универсальной для класса всюду определённых вычислимых функций одного аргумента.
- Доказательство: предположим, что такая функция есть, и обозначим $V(n,m)$. Определим $V'(n) = V(n,n) + 1$. Она вычислима, всюду определена, и не совпадает ни с одним сечением V (почему?).
- Почему это доказательство не работает для класса всех вычислимых функций и функции U? В каком случае возможно $U' = m \mapsto U(m,m) + 1 = m \mapsto U(n,m)$?
- Теорема: существует вычислимая функция f, не имеющая всюду определённого вычислимого продолжения (функции, совпадающей с f на $Dom\ f$).
- Доказательство: это U'. Предположим, что U'' её в.о.в.п., и придём к противоречию.

Неразрешимость проблемы остановки

- Теорема: существует перечислимое неразрешимое множество (имеющее неперечислимое дополнение).

Неразрешимость проблемы остановки

- Теорема: существует перечислимое неразрешимое множество (имеющее неперечислимое дополнение).
- Доказательство: это $Dom\ U'$.
  - Она перечислима, потому что это

Неразрешимость проблемы остановки

- Теорема: существует перечислимое неразрешимое множество (имеющее неперечислимое дополнение).
- Доказательство: это $Dom\ U'$.
  - Она перечислима, потому что это область определения вычислимой функции.
  - Предположим, что она разрешима. Определим
    $$U''(n) = \begin{cases} U'(n) & \text{если } U'(n) \text{ определена,} \\ 0 & \text{иначе} \end{cases}$$

Неразрешимость проблемы остановки

- Теорема: существует перечислимое неразрешимое множество (имеющее неперечислимое дополнение).
- Доказательство: это $Dom\ U'$.
  - Она перечислима, потому что это область определения вычислимой функции.
  - Предположим, что она разрешима. Определим
    $$U''(n) = \begin{cases} U'(n) & \text{если } U'(n) \text{ определена,} \\ 0 & \text{иначе} \end{cases}$$
    Это всюду определённое вычислимое продолжение U' (проверьте). Противоречие!

Неразрешимость проблемы остановки

- Теорема: существует перечислимое неразрешимое множество (имеющее неперечислимое дополнение).
- Доказательство: это $Dom\ U'$.
    - Она перечислима, потому что это область определения вычислимой функции.
    - Предположим, что она разрешима. Определим
      $$U''(n) = \begin{cases} U'(n) & \text{если } U'(n) \text{ определена,} \\ 0 & \text{иначе} \end{cases}$$
      Это всюду определённое вычислимое продолжение U' (проверьте). Противоречие!
- Следствие: $Dom\ U$ неразрешима.
- Доказательство:

Неразрешимость проблемы остановки

- Теорема: существует перечислимое неразрешимое множество (имеющее неперечислимое дополнение).
- Доказательство: это $Dom\ U'$.
  - Она перечислима, потому что это область определения вычислимой функции.
  - Предположим, что она разрешима. Определим
    $$U''(n) = \begin{cases} U'(n) & \text{если } U'(n) \text{ определена,} \\ 0 & \text{иначе} \end{cases}$$
    Это всюду определённое вычислимое продолжение U' (проверьте). Противоречие!
- Следствие: $Dom\ U$ неразрешима.
- Доказательство: иначе $Dom\ U'$ тоже была бы разрешима.

Неразрешимость проблемы остановки

- Теорема: существует перечислимое неразрешимое множество (имеющее неперечислимое дополнение).
- Доказательство: это $Dom\ U'$.
  - Она перечислима, потому что это область определения вычислимой функции.
  - Предположим, что она разрешима. Определим
    $$U''(n) = \begin{cases} U'(n) & \text{если } U'(n) \text{ определена,} \\ 0 & \text{иначе} \end{cases}$$
    Это всюду определённое вычислимое продолжение U' (проверьте). Противоречие!
- Следствие: $Dom\ U$ неразрешима.
- Доказательство: иначе $Dom\ U'$ тоже была бы разрешима.
- Другими словами, не существует алгоритма, который по произвольной программе p и аргументу n может точно определить, закончится ли вычисление $p(n)$.

Теорема (Успенского–)Райса

- Свойство программ P называется *семантическим*, если оно определяется только функцией, которую вычисляет программа.
  - То есть если две программы p_1 и p_2 обе вычисляют одну и ту же функцию, то $P(p_1) \Leftrightarrow P(p_2)$.

Теорема (Успенского–)Райса

- Свойство программ P называется *семантическим*, если оно определяется только функцией, которую вычисляет программа.
    - То есть если две программы p_1 и p_2 обе вычисляют одну и ту же функцию, то $P(p_1) \Leftrightarrow P(p_2)$.
    - Например, «Программа останавливается на любых входных данных» это

Теорема (Успенского–)Райса

- Свойство программ P называется *семантическим*, если оно определяется только функцией, которую вычисляет программа.
  - То есть если две программы p_1 и p_2 обе вычисляют одну и ту же функцию, то $P(p_1) \Leftrightarrow P(p_2)$.
  - Например, «Программа останавливается на любых входных данных» это семантическое свойство.
  - А «длина программы больше 100 символов»

Теорема (Успенского–)Райса

- Свойство программ P называется *семантическим*, если оно определяется только функцией, которую вычисляет программа.
  - То есть если две программы p_1 и p_2 обе вычисляют одну и ту же функцию, то $P(p_1) \Leftrightarrow P(p_2)$.
  - Например, «Программа останавливается на любых входных данных» это семантическое свойство.
  - А «длина программы больше 100 символов» нет.

Теорема (Успенского–)Райса

- Свойство программ P называется *семантическим*, если оно определяется только функцией, которую вычисляет программа.
  - То есть если две программы p_1 и p_2 обе вычисляют одну и ту же функцию, то $P(p_1) \Leftrightarrow P(p_2)$.
  - Например, «Программа останавливается на любых входных данных» это семантическое свойство.
  - А «длина программы больше 100 символов» нет.
- Свойство нетривиально, если оно истинно для некоторых программ и ложно для других.
- Теорема Райса: любое семантическое нетривиальное свойство программ неразрешимо.

Теорема (Успенского–)Райса

- Свойство программ P называется *семантическим*, если оно определяется только функцией, которую вычисляет программа.
    - То есть если две программы p_1 и p_2 обе вычисляют одну и ту же функцию, то $P(p_1) \Leftrightarrow P(p_2)$.
    - Например, «Программа останавливается на любых входных данных» это семантическое свойство.
    - А «длина программы больше 100 символов» нет.
- Свойство нетривиально, если оно истинно для некоторых программ и ложно для других.
- Теорема Райса: любое семантическое нетривиальное свойство программ неразрешимо.
- Проблема остановки это частный случай: свойство «программа p даёт результат при запуске с аргументом 0» семантическое и нетривиальное.

Простые модели вычислений

- Мы пока почти ничего не говорили про конкретные языки.
- Для доказательств часто оказывается удобно работать не с привычными C, Python и т.д., а со специальными простыми языками.
- Особенно при доказательстве неразрешимости каких-то задач или невычислимости каких-то функций

Простые модели вычислений

- Мы пока почти ничего не говорили про конкретные языки.
- Для доказательств часто оказывается удобно работать не с привычными C, Python и т.д., а со специальными простыми языками.
- Особенно при доказательстве неразрешимости каких-то задач или невычислимости каких-то функций (первое это частный случай второго).
- Чаще всего это делается сведением задачи к проблеме остановки (или к другой задаче, для которой уже доказали неразрешимость).
- То есть показывается, что мы можем «эмулировать» произвольную программу на нашем языке в терминах этой задачи так, чтобы решение задачи позволило сказать, остановится программа или нет.
- А это проще, если язык небольшой и программы на нём устроены просто.

Регистровая машина

- Программа регистровой машины состоит из конечной пронумерованной последовательности команд следующих видов:
    - (переменная) $:=$ 0
    - (переменная1) $:=$ (переменная2)
    - inc(переменная) (добавить 1)
    - dec(переменная) (вычесть 1, если текущее значение положительное)
    - if (переменная)=0 goto (номер команды)
    - stop
- Каждая такая программа очевидно использует конечное число переменных (или регистров).
- Значения переменных — натуральные числа, в начале все 0.

Вычисление функций на регистровой машине

- Каждая такая программа вычисляет определённую функцию.
- Пусть среди переменных программы есть $i_1, \ldots, i_n$ и нет i_{n+1}, тогда у функции n аргументов.
- Поместим в переменные $i_1, \ldots, i_n$ значения аргументов функции.
- Начнём выполнять команды начиная с первой, если команда не goto, переходим к следующей.
- Если мы дойдём до конца программы или выполним команду stop, то значение функции это содержимое переменной o на этом шаге.
- Если ни того, ни другого не произойдёт, то функция на этих аргументах не определена.

Языки BlooP и FlooP

- Язык BlooP (Bounded Loop):
  - Регистровая машина без команды goto.
  - Добавляются ограниченные по числу повторений циклы с break и continue.
  - И именованные функции без рекурсии.
  - В обычном описании есть $+$, $\cdot$, $<$, $==$ и нет inc и dec, но это не существенно.

Языки BlooP и FlooP

- Язык BlooP (Bounded Loop):
  - Регистровая машина без команды goto.
  - Добавляются ограниченные по числу повторений циклы с break и continue.
  - И именованные функции без рекурсии.
  - В обычном описании есть $+$, $\cdot$, $<$, $==$ и нет inc и dec, но это не существенно.
- Как в обычной регистровой машине получить аналог именованных функций?

Языки BlooP и FlooP

- Язык BlooP (Bounded Loop):
  - Регистровая машина без команды goto.
  - Добавляются ограниченные по числу повторений циклы с break и continue.
  - И именованные функции без рекурсии.
  - В обычном описании есть $+$, $\cdot$, $<$, $==$ и нет inc и dec, но это не существенно.
- Как в обычной регистровой машине получить аналог именованных функций?
- Язык FlooP (Free Loop):
  - BlooP + неограниченные циклы (while(true)).

Примитивно и частично рекурсивные функции

- Примитивно рекурсивные функции это те, которые можно получить из базовых функций с помощью операторов композиции и примитивной рекурсии:
  - Базовые функции: $O(x) = 0$, $S(x) = x + 1$, $I_n^i(x_1, \ldots, x_n) = x_i$.
  - Композиция: пусть f функция m аргументов, $g_1, \ldots, g_m$ функции k аргументов, тогда $C(f, g_1, \ldots, g_m) = h$, где $h(x_1, \ldots, x_k) = f(g_1(x_1, \ldots, x_k), \ldots, g_m(x_1, \ldots, x_k))$.
  - Примитивная рекурсия: пусть f функция n аргументов, g функция $n + 2$ аргументов, тогда $R(f, g) = h$, где $h(x_1, \ldots, x_m, 0) = f(x_1, \ldots, x_m)$, $h(x_1, \ldots, x_m, y + 1) = g(x_1, \ldots, x_m, y, h(x_1, \ldots, x_m, y))$.

Примитивно и частично рекурсивные функции

- Примитивно рекурсивные функции это те, которые можно получить из базовых функций с помощью операторов композиции и примитивной рекурсии:
  - Базовые функции: $O(x) = 0$, $S(x) = x + 1$, $I_n^i(x_1, \ldots, x_n) = x_i$.
  - Композиция: пусть f функция m аргументов, $g_1, \ldots, g_m$ функции k аргументов, тогда $C(f, g_1, \ldots, g_m) = h$, где $h(x_1, \ldots, x_k) = f(g_1(x_1, \ldots, x_k), \ldots, g_m(x_1, \ldots, x_k))$.
  - Примитивная рекурсия: пусть f функция n аргументов, g функция $n + 2$ аргументов, тогда $R(f, g) = h$, где $h(x_1, \ldots, x_m, 0) = f(x_1, \ldots, x_m)$, $h(x_1, \ldots, x_m, y + 1) = g(x_1, \ldots, x_m, y, h(x_1, \ldots, x_m, y))$.
- Частично рекурсивные функции получаются, если добавить оператор минимизации:
  - Пусть f функция $n + 1$ аргумента, тогда $\mu(f)(x_1, \ldots, x_n)$ считается так: вычисляем $f(x_1, \ldots, x_n, y)$, начиная с $y = 0$. Если получили 0, возвращаем y. Если нет, то увеличиваем y на 1 и повторяем.
  - В каких случаях этот процесс не остановится?

Связь между моделями вычислений

- Очевидно ли, что все ПРФ и ЧРФ вычислимы?

Связь между моделями вычислений

- Очевидно ли, что все ПРФ и ЧРФ вычислимы?
- Оказывается (без доказательства):
- Любая программа на BlooP задаёт ПРФ.
- И наоборот, любая ПРФ задаётся программой BlooP.

Связь между моделями вычислений

- Очевидно ли, что все ПРФ и ЧРФ вычислимы?
- Оказывается (без доказательства):
- Любая программа на BlooP задаёт ПРФ.
- И наоборот, любая ПРФ задаётся программой BlooP.
- Аналогично для FlooP и ЧРФ.
- И для регистровых машин и ЧРФ.
    - Даже если ограничиться командами inc, dec и if-goto, этого достаточно для задания всех ЧРФ.

Тезис Чёрча–Тьюринга

- А есть ли вычислимые функции, которые не являются ЧРФ?

Тезис Чёрча–Тьюринга

- А есть ли вычислимые функции, которые не являются ЧРФ?
- Тезис Чёрча–Тьюринга: любая функция, которую можно вычислить каким угодно алгоритмом, частично рекурсивна.
- И значит, любую такую функцию можно задать программой на FlooP и регистровой машиной.

Тезис Чёрча–Тьюринга

- А есть ли вычислимые функции, которые не являются ЧРФ?
- Тезис Чёрча–Тьюринга: любая функция, которую можно вычислить каким угодно алгоритмом, частично рекурсивна.
- И значит, любую такую функцию можно задать программой на FlooP и регистровой машиной.
- При разумных ограничениях понятия алгоритма это можно доказать. И никто не придумал алгоритм, который под них не подпадает.

Тезис Чёрча–Тьюринга

- А есть ли вычислимые функции, которые не являются ЧРФ?
- Тезис Чёрча–Тьюринга: любая функция, которую можно вычислить каким угодно алгоритмом, частично рекурсивна.
- И значит, любую такую функцию можно задать программой на FlooP и регистровой машиной.
- При разумных ограничениях понятия алгоритма это можно доказать. И никто не придумал алгоритм, который под них не подпадает.
- Кроме того, есть лямбда-исчисление, машины Тьюринга, машины Поста, алгоритмы Маркова и т.д.
- Оказывается, что в этих моделях тоже можно вычислить любую ЧРФ (и только ЧРФ).
- Такие модели вычислений называются полными по Тьюрингу.