ВЫЧИСЛИМОСТЬ

Машины Тьюринга и зачем они на самом деле нужны.

К. Владимиров, Yadro, 2024
mail-to: konstantin.vladimirov@gmail.com

Давайте упростим ассемблер: регистры

- Вспомним какой-нибудь простой ассемблер, например RISC-V.
- Основные его элементы это регистры, память и операции над ними.
- Какие группы регистров вы ещё можете вспомнить?
- Главный вопрос такой: нужно ли нам столько регистров?
- Можем ли мы как-то их сократить, но при этом всё ещё оставить ассемблер универсальным?

Name	Alias
x0	zero
x1	ra
x2	sp
x3	gp
x4	tp
x5-x7	t0-t2
x8-x9	s0-s1
x10-x15,x16,x17	a0-a5,a6,a7
x18-x27	s2-s11
x28-x31	t3-t6

RAM-машина

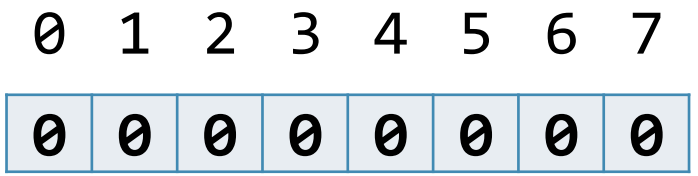
- Один регистр аккумулятора и бесконечное количество нумерованных ячеек памяти.

`LD M` помещает из M в ACC

`ST M` помещает из ACC в M

`OP M` выполняет операцию ACC = ACC OP [M]

`JZ L` прыгает на метку L если ACC == `0`



- Здесь OP это любая операция (сложение, умножение и т. д.)
- Пример: вычислите на RAM-машине: $f(a,b,c) = \frac{a\cdot(b+c)}{c-a}$

RAM-машина: пример

- Пример: вычислите на RAM-машине:

$$f(a,b,c) = \frac{a \cdot (b+c)}{c-a}$$

0	1	2	3	4	5	6	7
0	a	b	c	0	0	0	0

- Исходное состояние памяти см. справа.
- Будем также считать, что запись чего угодно в нулевую ячейку завершит программу.

```
LD 3; SUB 1; ST 4; LD 2; ADD 3; ST 5; LD 1; MUL 5; DIV 4; ST 0;
```

0	1	2	3	4	5	6	7
f	a	b	c	c - a	b + c	0	0

Problem RAMM

`LD M` помещает из M в ACC

`ST M` помещает из ACC в M

`OP M` выполняет операцию ACC = ACC OP [M]

`JZ L` прыгает на метку L если ACC == `0`

- Используя операции ADD и MUL найдите на RAM-машине n-е число Фибоначчи. Исходная память см. ниже.

0	1	2	3	4	5	6	7
0	**n**	**0**	**1**	**0**	**0**	**0**	**0**

Давайте упростим ассемблер: инструкции

Jumps & Calls
JAL
JALR
BEQ
BNE
BLT
BGE
BLTU
BGEU

Loads & Stores
LB
LH
LW
LBU
LHU
SB
SH
SW
LWU
LD
SD

Arithmetics			
ADD	ADDI	ADDW	ADDIW
SUB	SUBI	SUBW	SUBIW
OR	ORI		
XOR	XORI		
AND	ANDI		
SRL	SRLI	SRLW	SRLIW
SLL	SLLI	SLLW	SLLIW
SRA	SRAI	SRAW	SRAIW

Data flow	
SLT, SLTU	SLTI, SLTIU

Special
FENCE
ECALL
EBREAK

Upper immediate
LUI
AIUPC

Уменьшим количество операций

- Очевидно нам нужны одна операция перехода, одна операция загрузки и одна операция сохранения.
- Мы легко выкидываем специальные и data-flow.
- Мы сравнительно легко заменяем DIV (например на деление в столбик).
- Мы легко заменяем MUL на ADD в цикле.
- Но и сами ADD и SUB мы можем заменить на инкремент (декремент) в цикле.
- Но поскольку инкремент и декремент одноопернадные, нам также не нужен регистр аккумулятора.

Машина Ламбека-Минского

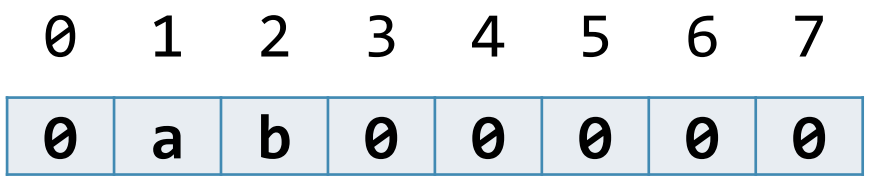
- У нас остаются всего три команды

`INC M` увеличивает [M] на единицу

`DEC M` уменьшает [M] на единицу

`JZ M, L` прыгает на метку L если [M] == `0`

- В качестве упражнения напишите для такой машины процедуру сложения двух чисел. В ячейке 3 надо сформировать a+b, в конце записать $[0] = 1$



Ещё две раздражающие детали

- У нас в каждой ячейке хранится целое число произвольной длины, это раздражает.
- И нам всё ещё нужно адресовать произвольную ячейку.
- Можем ли мы избавится от этого: хранить в ячейках памяти только нули и единицы и при этом адресовать только ту ячейку на которую сейчас указывает data pointer?

Машина Тьюринга

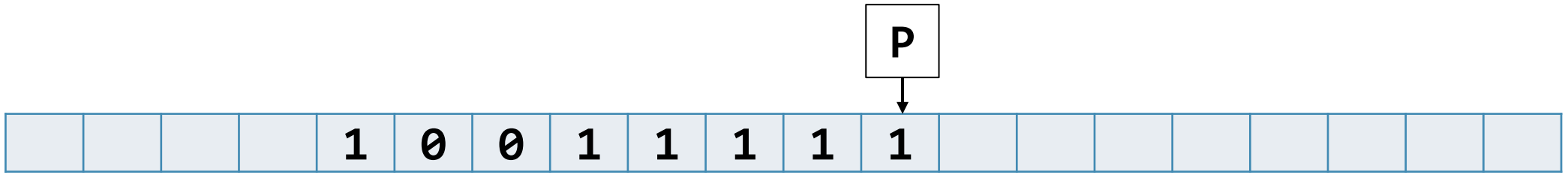
- Команды сильно упростились, а память стала тернарной.

`WRITE N` пишет в [P] число N, где N = $\{0, 1, empty\}$

`MOVE D` двигает $P = P + 1$ если D = L или $P = P - 1$ если D = R

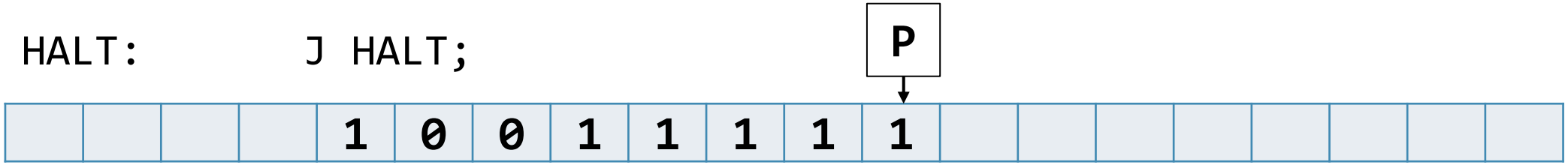
`J L, N` прыгает на метку L если [P] = N
`J L` прыгает на метку L безусловно. Метка HALT то завершает работу.

- Попробуем написать программу инкремента бинарного числа?



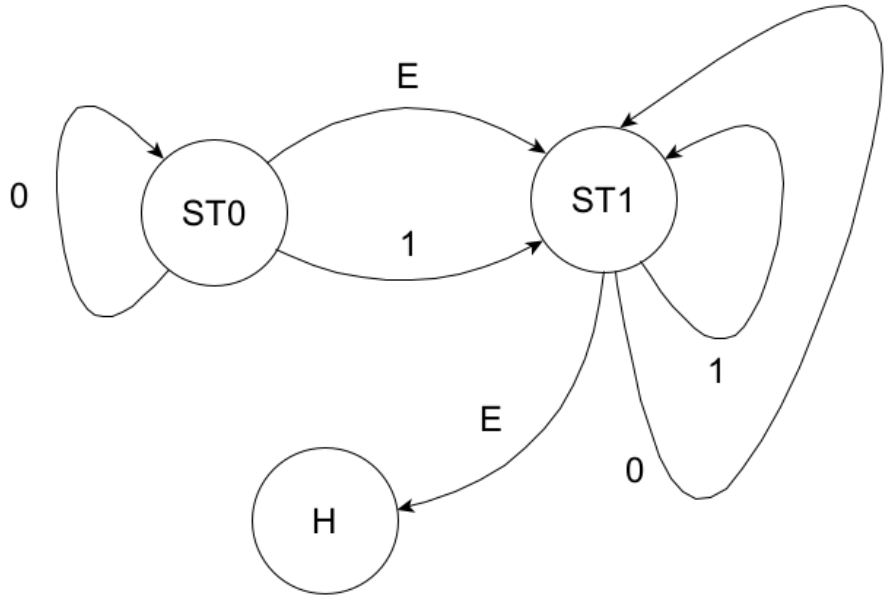
Программа инкремента:

```
START:      J ST0;
ST0:        J ST0S0, 0; J ST0S1, 1; J ST0SE, E;
ST0SE:      WRITE 1; MOVE R; J ST1;
ST0S0:      WRITE 1; MOVE R; J ST1;
ST0S1:      WRITE 0; MOVE L; J ST0;

ST1:        J ST1S0, 0; J ST1S1, 0; J ST1SE, E;
ST1SE:      WRITE E; MOVE L; J HALT;
ST1S0:      WRITE 0; MOVE R; J ST1;
ST1S1:      WRITE 1; MOVE R; J ST1;

HALT:       J HALT;
```



Представление автоматом

state	symbol	symbol	state	move
0	EMPTY	1	1	R
0	0	1	1	R
0	1	0	0	L
1	EMPTY	EMPTY	HALT	L
1	0	0	1	R
1	1	1	1	R

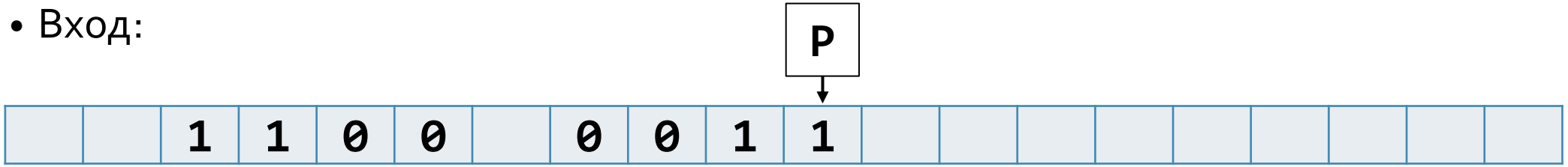


- Автоматное представление чуть легче в чтении и сразу показывает символы и состояния.

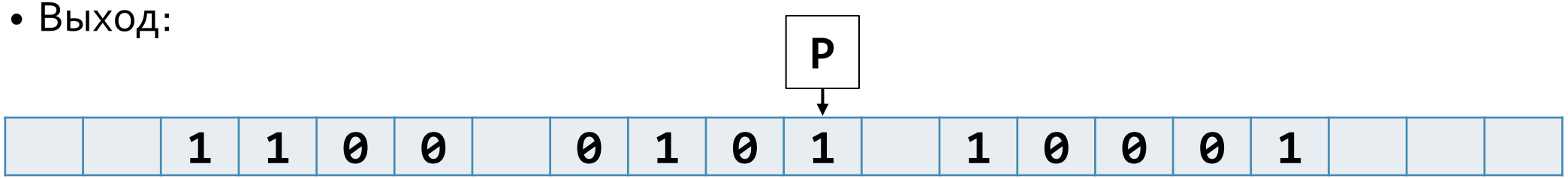
Emulator

Problem TADD

- Составьте машину Тьюринга которая складывает два числа.
- Вход:



- Выход:



Разновидности машин Тьюринга

- Ячейки не обязаны быть бинарными + empty. Мы можем их сделать например десятичными + empty.
- Лента не обязана быть одна. Если мы не собираемся их смешивать, мы можем иметь входную ленту, выходную ленту и сколько-то рабочих лент.
- Мы будем говорить про МТ на n-арных лентах общим числом состояний k.
- Например покажем применимость машин Тьюринга на простом примере:
  - Вас просят написать программу, которая берёт на вход функцию, разряд за разрядом выдающую десятичные знаки вещественного числа.
  - Ваша задача умножить это число на три.
- Формально у вас есть бесконечная входная лента и бесконечная выходная.

Невычислимость простой операции.

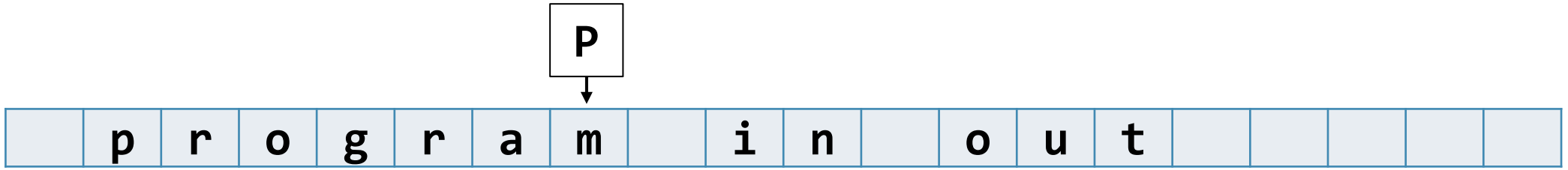
- Рассмотрим вход $p = 0.3^n$.
- Машина должна дать выход 0.9^n.
- Допустим к моменту когда на выходную десятичную ленту нужно записать первый символ, обработано k символов.
- Но тогда вход 0.3^k4 должен дать такую же первую цифру выхода что и 0.3^k3. Это даёт противоречие.
- Но заметьте, мы можем **делить на три** вещественные числа заданные по одному знаку десятичного расширения.

Problem TDIV

- Составьте машину Тьюринга с тремя десятичными лентами – входной выходной и рабочей, которая:
- Получив на входной любое количество действительных разрядов действительного числа N
- Напечатает на выходной ленте разряд за разрядом число $^{N}/_{3}$.
- Hint: $10 \cdot r_{n-1} + a_n = 3 \cdot b_n + r_n$

Универсальная машина Тьюринга

- Поскольку сама машина вполне может быть закодирована нулями и единицами, гипотетически она сама может лежать на ленте.
- Тогда мы можем составить машину Тьюринга, которая считывает с ленты другую машину в качестве программы, а потом исполняет её на входе и пишет выход.
- На одной бинарной ленте можно построить УМТ всего за 24 состояния.
- Но машина Тьюринга записанная на ленте это… число. И это её номер.



Small universal Turing machines

Наличие невычислимых функций

- Построим функцию $h(n, m) = [machine\ n\ halts\ on\ input\ m]$
- Здесь нотация $[P(n, m)]$ это 1 если $P(n, m) = true$ и 0 если нет
- Является ли $h(n, m)$ вычислимой на машине Тьюринга?
- Допустим, что является. Тогда построим машину $TD(\#T)$ такую, что она:
  - зацикливается, если машина T завершает работу на входе равном коду T.
  - завершает работу если наоборот.
- Мы пришли к противоречию: $TD(\#TD)$ должна одновременно зациклиться и не зациклиться
- Следовательно **проблема останова** неразрешима на машине Тьюринга

Тезис Черча-Тьюринга

- Любая функция, которая может быть вычислена физическим устройством, может быть вычислена машиной Тьюринга
- Иные модели вычислений включают
  - Алгорифмы Маркова
  - Машины Поста
  - Частично-рекурсивные функции Клини
  - Колмогоровские комплексы
- Все они сводятся к машине Тьюринга и обратно, она сводится к ним

Статья Колмогорова

Пример простой задачи

- У вас на входе функция $f(k)$, порождающая k-й член ряда для любого k и вам нужно сказать, сходится этот ряд или нет.
- Например для входа $f(k) = -1^k \frac{\pi^{2k+1}}{(2k+1)!}$ нужно выдать true
- Допустим вас просят написать такую программу
  - Как бы вы закодировали входные данные?
  - Как бы вы написали алгоритм?
- Что в целом вы думаете об этой задаче?

Пример простой неразрешимой задачи

- У вас на входе функция $f(k)$, порождающая k-й член ряда для любого k и вам нужно сказать, сходится этот ряд или нет.
- Допустим у нас есть такой алгоритм A_1. Рассмотрим любую программу P и построим функцию

$$g(P, n) = [UMT(P)\ halts\ in\ n\ steps],\ f(k) = g(P, k)$$

- Довольно очевидно, что если программа не завершает работу, то этот ряд расходится, иначе сходится
- Хорошие новости. Имея такой алгоритм мы можем решить проблему останова. Но есть и плохая новость…

Попробуем упростить задачу

- У вас на входе функция $f(k)$, порождающая k-й член ряда для любого k и вам нужно сказать, есть ли такое k, что $f(k) \neq 0$.
- Стало проще?

Всё ещё неразрешимая задача

- У вас на входе функция $f(k)$, порождающая k-й член ряда для любого k и вам нужно сказать, есть ли такое k, что $f(k) \neq 0$.
- Допустим у нас есть такой алгоритм A_2.
- Возьмём ту же функцию $g(P, n) = [UMT(P)\ halts\ in\ n\ steps]$. $f(k) = g(P, k)$.
- Применим к нему A_2. Если мы нашли ненулевой элемент то мы нашли что машина когда-то остановится.
- Хорошие новости: мы кажется снова можем решить проблему останова.
- Плохие новости: ну вы понимаете.

Теорема Райса

- Для любого нетривиального свойства вычислимых функций определение того, вычисляет ли произвольный алгоритм функцию с таким свойством, является алгоритмически неразрешимой задачей
- В формальной постановке $P^{(1)}$ это класс одноаргументных вычислимых функций над $\mathbb{N}$.
- Зададим нумерацию таких функций n: $\mathbb{N} \to P^{(1)}$ и обозначим k-ю функцию в этой нумерации как $n(k)$.
- Скажем, что свойство F определяется подмножеством $F \subseteq P^{(1)}$.
- Тогда задача определить для k принадлежит ли $n(k)$ к F разрешима если и только если F пусто или $F = P^{(1)}$.

Статья Райса

Статические анализаторы

- Это такие программы которые работают наперекор теореме Райса.
- Иногда они даже работают: https://godbolt.org/z/a3bqGPP7c
- В целом они стараются построить путь до опасной ситуации.

```
int fib_deref(int a, int *num) {
  int n;
  if (a < 2) return 1;
  n = fib_deref(a - 1, num) + fib_deref(a - 2, num);
  if (n == 17) n = *num;
  return n;
}
```

- Функция абсолютно безопасна, но вы понимаете...

https://godbolt.org/z/9G8dcG5bE

Быстро растущие функции

- Основная идея любого роста это итерация.
- Умножение: $a \times b = a + \cdots + a$ (b раз).
- Возведение в степень $a \uparrow b = a \times \cdots \times a$ (b раз)
- А теперь немного обобщения $a \uparrow\uparrow b = a \uparrow \cdots \uparrow a$ (b раз) и так далее.
- Упражнение: оцените $3 \uparrow\uparrow 3$.
- Как вы думаете, а функция $x \uparrow\uparrow\uparrow\uparrow\uparrow 2$ наверное растёт довольно быстро?
- Так вот: не слишком. Потому что она всё ещё **примитивно рекурсивна**.

Примитивно-рекурсивные функции

- Базовыми примитивно-рекурсивными (п.р.) являются
  - Нулевая функция
  - Функция следования succ(n) = n + 1
  - Индексная функция, сопоставляющая набору число из этого набора
- Из них мы можем производить новые с помощью
  - Суперпозиции $h(x_1 \dots x_n) = f\big(g_1(x_1 \dots x_n), \dots, g_m(x_1 \dots x_n)\big)$
  - Примитивной рекурсии
    $h(x_1 \dots x_n, 0) = f(x_1 \dots x_n)$
    $h(x_1 \dots x_n, y+1) = g\big(x_1 \dots x_n, y, h(x_1 \dots x_n, y)\big)$
- Существуют ли функции, которые растут быстрее любых примитивно-рекурсивных функций?

Функция Аккермана

- Первой функцией относительно которой было доказано, что она вычислима, но не примитивно рекурсивна, стала функция Аккермана.

$$A(0, n) = n + 1$$

$$A(m, 0) = A(m - 1, 1)$$

$$A(m, n) = A\big(m - 1, A(m, n - 1)\big)$$

- Функция Аккермана, как было доказано, растёт быстрее любой п.р. функции.
- Простая задача: докажите, что $A(1, n) = n + 2$.
- Сложная задача: разберитесь почему функция Аккермана не п.р. (см. сноску).

Доказательство, что A(m, n) не п.р.

Частично рекурсивные функции

- Чтобы починить примитивно-рекурсивные функции и добавить к ним все вычислимые, добавим к определению правило минимизации.
  - Суперпозиция $h(x_1 \dots x_n) = f\big(g_1(x_1 \dots x_n), \dots, g_m(x_1 \dots x_n)\big)$
  - Примитивная рекурсия
    $h(x_1 \dots x_n, 0) = f(x_1 \dots x_n)$
    $h(x_1 \dots x_n, y+1) = g\big(x_1 \dots x_n, y, h(x_1 \dots x_n, y)\big)$
  - Минимизация
    $m(f)(x_1 \dots x_n) = z$ если $f(i, x_1 \dots x_n) > 0$ для $i < z$ и $f(z, x_1 \dots x_n) = 0$
- Обратим внимание, что исходная функция f может никогда не быть равна нулю и тогда конструируемая функция $m(f)$ не определена (аналог: машина Тьюринга зацкилилась).

Общерекурсивные функции

- Общерекурсивной называется такая частично-рекурсивная функция, которая определена для всех своих аргументов.
- Например функция Аккермана общерекурсивна.
- Проблема доказательства общерекурсивности алгоритмически не разрешима.
- Частично-рекурсивные функции эквивалентны машинам Тьюринга.
- А теперь интересный вопрос. Можем ли мы найти такую функцию, которая растёт быстрее, чем любая частично-рекурсивная функция?

Игра в усердного бобра

- Определим Busy Beaver Game следующим образом.
- Построим машину Тьюринга с $n + 1$ состояниями из которых одно halt.
- Запустим её на ленте, содержащей только нули.
- Если машина зациклилась навсегда, она проиграла.
- Выигрывает та машина, которая напечатает наибольшее количество единиц на ленту и остановится.
- Например для $n = 2$ максимум это шесть единиц и мы говорим $BB(2) = 6$.
- Предположим, что вы сидите в судейской коллегии. Что вы попросите от машины-претендента кроме таблицы переходов и почему?

Статья Тибора Радо про Busy Beaver

Трудности судейства

- Легко доказать, что задача проверки правда ли машина-кандидат печатает указанное количество единиц за конечное время неразрешима.
- Поэтому вместе с кандидатом нужно просить число шагов которые проверить.
- Ещё проще доказать, что функция $BB(n)$ невычислима на машине Тьюринга.
- Интересно при этом, то, что мы знаем:

$BB(0) = 1, BB(1) = 6, BB(2) = 6, BB(3) = 13$

- Есть шансы, что $BB(4) = 4098$ и мы уверены, что $BB(5) > 10^{18267}$.
- В целом $BB(n) > 3 \uparrow^{n-3} 31$ но эта граница очень слабая, потому что $BB(n)$ растёт быстрее любой вычислимой функции.

Продолжи ряд: 0, 1, 4, 6, 13

Удивительный факт

- Функция $BB(n)$, как и любая невычислимая функция имеет по определению бесконечную Колмогоровскую сложность.
- Значит она производит **настоящие случайные числа**.
- Мы просто теоретически не в состоянии написать программу, которая предсказала бы следующее такое число.

Применение к доказательствам

- Была построена машина Тьюринга с 744 состояниями, которая завершает работу если гипотеза Римана неверна.
- Была также построена машина Тьюринга с 43 состояниями, которая завершает работу если неверна гипотеза Гольдбаха.
- Если бы мы знали $BB(43)$, мы могли бы прогнать эту машину ровно столько шагов и если бы она не завершила работу, это доказало бы гипотезу.

Github с этими и другими машинами