Математические структуры. Домашнее задание 1. Вариант 39.

Кузнецов Владимир Михайлович, ФКН.

Для начала отмечу, что я понимимаю, что примитивно рекурсивная функция по определению записывается как

$$\begin{cases} f(\vec{x}, 0) = g(\vec{x}) \\ f(\vec{x}, y+1) = h(\vec{x}, y, f(\vec{x}, y)) \end{cases},$$

но у меня совсем нет желания усложнять и реализовывать сложение через пять функций (как это делается в конспекте, например). Всем и так ясно, что $I_3^3(x, y, z)$ можно заменить на z. Спасибо за понимание.

Задача 1. Докажите примитивную рекурсивность функции $f(x, y) = x \dot{-} (3y + 2)$ непосредственно из определения (т.е. постройте задающую её примитивно рекурсивную схему).

▷ Рассмотрим $p(x, y) = x \dot{-} y$, $g(x, y) = x + y$, $h(x, y) = xy$, тогда $f(x, y) = p(x, g(h(3, y), 2))$. Заметим, что $h(3, y) = g(g(y, y), y)$. А значит, можем запись f в виде $f(x, y) = p(x, g(g(g(y, y), y), 2))$. Тогда, давайте выпишем примитивно рекурсивную схему для сложения:

$$\begin{cases} g(x, 0) = x \\ g(x, y+1) = s(g(x, y)) \end{cases}$$

Для удобства объявим $f_1(x) = 3x + 2 = g\Big(g\Big(g\Big(x, x\Big), x\Big), s\Big(s\Big(o(x)\Big)\Big)\Big)$. Теперь запишем примитивно рекурсивную схему для вычитания:

$$\begin{cases} \mathrm{pd}(0) = o(0) \\ \mathrm{pd}(x+1) = x \end{cases}$$

$$\begin{cases} p(x, 0) = x \\ p(x, y+1) = \mathrm{pd}(p(x, y)) \end{cases}$$

Поясню, что тут происходит. Сначала я определил $\mathrm{pd}(x) = x \dot{-} 1$, а после, используя pd определил $p(x, y)$. Тогда, $f(x, y) = p(x, f_1(y))$. p - примитивно рекурсивная функция, f_1 - примитивно рекурсивная функция, а значит, что f - примитивно рекурсивная функция. □

Задача 2. Докажите примитивную рекурсивность

- функции $f(x)$, равной номеру (начиная с нуля) наибольшего простого числа, делящего число x;
- функции $g(x)$ равной $p_0^{\alpha_n} \cdot p_1^{\alpha_{n-1}} \cdot \ldots \cdot p_n^{\alpha_0}$, где $x = p_0^{\alpha_0} \cdot p_1^{\alpha_1} \cdot \ldots \cdot p_n^{\alpha_n}$, $\alpha_n \neq 0$ и p_i есть i-ое (по порядку) простое число.

▷ Изначально реализуем доп. функцию:

```
function h(x, y, i):
    if exp(x, p_i) = 0 then
        return y
    else
        return i
```

В конспектах было доказано, что функции вида `if then else` примитивно рекурсивны. Функция $\exp$ примитивно рекурсивна (из конспектов), функция p_i примитивно рекурсивна (из конспектов) [дадада, `p_i` у меня в псевдокоде обозначает i-ое простое число], предикат "x равен нулю" примитивно рекурсивен (из семинаров) $\Rightarrow$ предикат $\exp(x, p_i) = 0$ тоже примитивно рекурсивен, а значит и вся функция h примитивно рекурсивна. Что она делает обсудим позднее. Сечас просто поверьте, что она мне нужна, и жить я без неё не могу.

Мы знаем, что неформально примитивно рекурсивная функция соответствует циклу `for`. Тогда, запишем функцию f в виде:

```
function f(x):
    y := o(x)   // initialize to zero
    for i = 0 to x
        y := h(x, y, i)
    return y
```

Это примитивно рекурсивная функция, так как h - примитивно рекурсивная функция, как и `for`. Тут я иду до числа x, так как не хочу заморачиваться с точной верхней границей, а на оптимальность алгоритма нам всё равно, так что всё чики-пики. Давайте ещё раз проговорим, что тут происходит, чтобы у проверяющего точно не осталось вопросов. Идём от нуля до данного числа и проверяем не делится ли наше число x на i-ое простое чиселко, если делится, то увеличиваем переменную y до i и не меняем y в противном случае. В итоге возвращаем y.

Функцию g я буду реализовывать через f и цикл `for`. Сначала, используя f, найду максимальное простое число в разложении на простые, а после пройдусь по всем простым и посчитаю сумму:

```
function pow(x, y):
    return x ^ y

function g(x):
    y := s(o(x))  // initialize to one
    for i = 0 to f(x)
        y := y * pow(p_i, exp( x, p_(f(x) - i) ))
    return y
```

Функция возведения в степень примитивна рекурсивна. Умножение и вычитание (имеется ввиду вычитание с точкой, но мы точно знаем, что $f(x) \geq i$) тоже примитивно рекурсивны. Примитивная рекурсивность функции f была доказана ранее. А про то, что $\exp$ - примитивно рекурсивная функция мы уже поговорили. Итого получили примитивно рекурсивную функцию g. $\square$

Задача 3. Докажите частичную рекурсивность функции

$$f(x) = \begin{cases} x \dot{-} 1, & \text{if } \lfloor \frac{x}{3} \rfloor \mod 2 \equiv 0, \\ \text{undefined}, & \text{else.} \end{cases}$$

$\triangleright$ Знаем, что функция $g(x, y) = \left\lfloor \dfrac{x}{y} \right\rfloor$ - примитивно рекурсивная функция (при $y \neq 0$), знаем, что предикат "x чётно" - примитивно рекурсивный. Обозначим за $R(x)$ предикат "$\left\lfloor \dfrac{x}{s(s(s(o(x))))} \right\rfloor$ чётно". Он примитивно рекурсивен, так как и деление с округлением вниз, и получение тройки, и проверка на чётность являются примитивно рекурсивными, тогда, можем записать $f(x)$ в виде:

```
function f(x):
    if R(x) then
        return pd(x)
    else
        return f(x)
```

В случае "не $R(x)$" функция не определена (бесконечно зацикливается), при этом мы точно знаем, что $R(x)$ - примитивно рекурсивный предикат, и $pd(x) = x \dot{-} 1$ примитивно рекурсивен. Так же уже говорили о примитивной рекурсивности конструкции `if then else`. Заметим, что $f(x)$ определена на вводах вида "целая часть от деления числа на три чётна" и не определена иначе. Из всего этого могу сделать вывод, что $f(x)$ - частично рекурсивная функция. $\square$

Тут, наверное, можно было бы помахать руками вокруг минимизации, но я что-то не нашёл такую μy, которая удовлетворяла бы нашему предикату, но в целом записанное выше говорит о примерно том же.

Дабы исключить недопонимание, продублирую лекцию: изначально головка стоит перед первым элементом слова и указывает на $\#$, при этом будем считать, что на ленте нет других элементов кроме входного слова. В конце (по достижению q_f) головка стоит перед ответом.

Задача 4. Постройте МТ с входным алфавитом $\{1\}$, которая вычисляет функцию $f(x) =$ "остаток от деления x на три" в унарном коде. Выпишите последовательность конфигураций, возникающих в результате вычисления построенной машина на входе 111111.

Я хочу реализовать алгоритм, который будет вычитать по три единички из числа. Если на каком-то из этих трёх удалений обнаруживается, что длина числа была меньше четырёх, то я восстанавливаю последние удалённые единицы и перехожу в q_f.

$q_0\# \to q_1\#R$,

$q_1 1 \to q_2\#R$, $q_1\# \to q_3 1L$,

$q_2 1 \to q_3\#R$, $q_2\# \to q_f 1L$,

$q_3 1 \to q_1\#R$, $q_3\# \to q_2 1L$.

Пустое слово $== -1$, а значит - не натуральное. Так что вывожу что хочу (выведу даже правильно, потому что $-1 \mod 3 \equiv 2$). Теперь давайте рассмотрим $f(111111)$. Нижним подчёркиванием я буду обозначать положение головки:

$$(q_0, \underline{\#}111111\#\#) \to (q_1, \#\underline{1}11111\#\#) \to (q_2, \#\#\underline{1}1111\#\#) \to (q_3, \#\#\#\underline{1}111\#\#) \to$$
$$(q_1, \#\#\#\#\underline{1}11\#\#) \to (q_2, \#\#\#\#\#\underline{1}1\#\#) \to (q_3, \#\#\#\#\#\#\underline{1}\#\#) \to$$
$$(q_1, \#\#\#\#\#\#\#\#\underline{\#}\#) \to (q_3, \#\#\#\#\#\#\underline{\#}1\#) \to (q_2, \#\#\#\#\#\underline{\#}11\#) \to$$
$$(q_f, \#\#\#\#\underline{\#}111\#).$$

Задача 5. Постройте МТ с входным алфавитом $\{0, 1\}$, которая принимает язык $\{0^n 1^m 0^k | n, m, k \in \mathbb{N}, n \geq m \geq k\}$.

Изначально отмечу, что вариант вхождения двух и более слов в программу я не рассматриваю и реализую функцию только для $f(w)$, где $w \in \Sigma^*$. Разобью задачу на две.

1. Проверим, что входное слово имеет вид $0^n 1^m 0^k$ (без учёта $n \geq m \geq k$ (ну, или почти без учёта)). Если не имеет, то зациклимся. Для проверки изначально проходим по всем подряд идущим первым нулям (если они есть). Далее по подряд идущим единицам (если они есть) и ещё раз по подряд идущим нулям (если они есть). При этом не забываем рассмотреть случаи, когда $m = 0$ и/или $n = 0$ и / или $k = 0$.

$q_0\# \to q_1\#R$, // ступаем на первый элемент;

$q_1\# \to q_f\#S$, $q_1 0 \to q_2 0S$, $q_1 1 \to q_1 1S$, // Если слово пустое, то выходим. Если первый элемент равен единице, то зацикливаемся (так как $m > n$ в этом случае). Если ноль, то продолжаем работу;

$q_2 0 \to q_2 0R$, $q_2 1 \to q_3 1S$, $q_2\# \to q_f\#S$, // Пока нули идём направо. Если дошли до решётки, то сразу заканчиваем, ибо строка из всех нулей нас утраивает;

$q_3 1 \to q_3 1R$, $q_3 0 \to q_4 0S$, $q_3\# \to q_5\#L$, // Пока единички идём направо;

$q_4 0 \to q_4 0R$, $q_4 1 \to q_4 1S$, $q_4\# \to q_9\#L$, // Пока нули идём направо. Если встретили ещё одну единицу, то зацикливаемся всегда оставаясь на ней.

2. Во второй части мы точно знаем, что слово вида $0^n 1^m 0^k$ и $m \neq 0$, так что теперь проверим, что $n \geq m \geq k$. В первой части я выхожу в q_5 в случае, если $k = 0$ и в q_9 иначе. При этом мы точно знаем, что $n \neq 0$. Не забудем, что головка в обоих случаях указывает на последний элемент слова.

   1. Изначально рассмотрим случай q_5. Будем вычёркивать по одному эл-ту слева и справа. Если остались единички, то зацикливаемся:

      $q_5 1 \to q_6\#L$, $q_5 \binom{0}{\#} \to q_f\#S$, // Заменяем последний элемент на решётку, если это единица и отправляемся в начало (q_6), если же мы удалили все единицы, то $n \geq m$, а значит, надо выйти;

      $q_6 \binom{0}{1} \to q_6 \binom{0}{1} L$, $q_6\# \to q_7\#R$, // Доходим до левого края и останавливаемся на первом элементе слова;

      $q_7 0 \to q_8\#R$, $q_7 \binom{1}{\#} \to q_7 \binom{1}{\#} S$, // Заменяем первый элемент на решётку, если это ноль и отправляемся в конец (q_8). Если же мы удалили все нули, а единицы ещё остались, то $n < m$, а значит, надо зациклиться. Отдельно отмечу, что если тут мы встретили $\#$, это значит, что у нас $m = n + 1 \Rightarrow$ тоже зацикливаемся;

      $q_8 \binom{0}{1} \to q_8 \binom{0}{1} R$, $q_8\# \to q_5\#L$. // Доходим до правого края и останавливаемся на последнем элементе слова.

   2. Теперь рассмотрим случай q_9. Давайте сначала проверим, что $m \geq k$, а потом сведём случай к q_5. При проверке $m \geq k$ будем действовать так же, как и при проверке $n \geq m$, только чуть аккуратней. Мы помним, что стоим на последнем элементе слова. Давайте так же бегать туда сюда, только теперь будем удалять только 0, а вот единички будем менять на $\dot{1}$:

      $q_9 0 \to q_{10}\#L$, $q_9 \binom{1}{\dot{1}} \to q_{14} \binom{1}{\dot{1}} S$, // Заменяем последний элемент на решётку, если это ноль и отправляемся в начало (q_{10}), если же мы удалили все нули, то $m \geq k$, а значит, надо проверить $n \geq m$;

      $q_{10} 0 \to q_{10} 0L$, $q_{10} 1 \to q_{11} 1S$, $q_{10} \dot{1} \to q_{10} \dot{1} S$, // Идём до единиц. Если встретили $\dot{1}$ раньше чем 1, то зацикливаемся;

      $q_{11} 1 \to q_{11} 1L$, $q_{11} \binom{0}{\dot{1}} \to q_{12} \binom{0}{\dot{1}} R$, // Доходим до первой, ещё не рассмотренной единички и останавливаемся на ней. Отмечу, что $q_{11}\#$ никогда не настанет;

      $q_{12} 1 \to q_{13} \dot{1} R$, // Тонкий момент. Меняем 1 на $\dot{1}$. При этом, если у нас уже нет 1, то мы зациклимся на q_{10}, так что никаких других случаев рассматривать не надо;

$q_{13}\binom{0}{1} \to q_{13}\binom{0}{1}R$, $q_{13}\# \to q_9\#L$. // Доходим до правого края и останавливаемся на последнем элементе слова.

В процессе выяснения того, что $m \geq k$ мы удалили все последние нули и мы стоим на последней единичке (не важно какого вида) слова. Теперь для проверки $n \geq m$ надо лишь вызвать q_5, поставив головку на последний элемент и поменяв все $\dot{1}$ на 1 обратно.

$q_{14}\binom{1}{\dot{1}} \to q_{14}1L$, $q_{14}0 \to q_{15}0R$, // Меняем все $\dot{1}$ на 1;

$q_{15}1 \to q_{15}1R$, $q_{15}\# \to q_5\#L$. // Доходим до последнего элемента и запускаем q_5.

Задача 6. Посмтройте МТ с входным алфавитом $\{0, 1\}$, которая вычисляет функцию:

$$f(w) = \begin{cases} w, & \text{if there are the same count of 1 and 0 in string } w, \\ 0, & \text{else.} \end{cases}$$

Поочерёдно будем менять 0 на $\dot{0}$, а 1 на $\dot{1}$. Если в какой-то момент мы поменяли 0, но не можем поменять 1, то всё стираем и выводим 0, а если мы не можем поменять 0, то проверяем наличие единиц, если 1 ещё есть, то опять выводим ноль, а если у нас остались только $\dot{1}$ и $\dot{0}$, то меняем их обратно на 1 и 0 соответственно и завершаем программу.

$q_0\# \to q_1\#R$, $q_0\binom{0}{\dot{0}} \to q_0\binom{0}{\dot{0}}L$, $q_0\binom{1}{\dot{1}} \to q_0\binom{1}{\dot{1}}L$, // Определяю q_0 для чего-то, что не решётка, чтобы в дальнейшем переиспользовать это состояние. По факту я тут просто дохожу до первого элемента слова и перехожу в состояние q_1;

$q_1\binom{1}{\dot{1}} \to q_1\binom{1}{\dot{1}}R$, $q_1\dot{0} \to q_1\dot{0}R$, $q_1 0 \to q_2\dot{0}L$, $q_1\# \to q_5\#L$, // Находим нолик и меняем его на нолик с точкой. Далее ищу единицу, чтоб поменять её на единицу с точкой (q_2). Если нуля не нашлось, то всё равно попытаемся найти единицу и вернуть ноль, если найдётся (q_5);

$q_2\binom{1}{\dot{1}} \to q_2\binom{1}{\dot{1}}L$, $q_2\binom{0}{\dot{0}} \to q_2\binom{0}{\dot{0}}L$, $q_2\# \to q_3\#R$, // Доходим до первого элемента, а далее в состояние q_3;

$q_3\binom{0}{\dot{0}} \to q_3\binom{0}{\dot{0}}R$, $q_3\dot{1} \to q_3\dot{1}R$, $q_3 1 \to q_0\dot{1}L$, $q_3\# \to q_4\#L$, // Меняем единицу на единицу с точкой. Если не нашли единицу, то возвращаем ноль (q_4);

$q_4\binom{\dot{0}}{\dot{1}} \to q_4\#L$, $q_4\binom{0}{1} \to q_4\#L$, $q_4\# \to q_f 0L$, // Удаляем все элементы с конца и возвращаем ноль;

$q_5\binom{\dot{0}}{\dot{1}} \to q_5\binom{\dot{0}}{\dot{1}}L$, $q_5 1 \to q_6 1R$, $q_5\# \to q_7\#R$ // Если нашли единичку, то заканчиваем весь этот сыр-бор и возвращаем ноль (q_6). Если же единицы не нашлось, то значит, перед нами число в котором все нули стали ноликами с точками, а единички единичками с точкой, а ещё их равное количество! Так что теперь меняем их обратно и заканчивает программу (q_7);

$q_6\binom{\dot{0}}{\dot{1}} \to q_6\binom{\dot{0}}{\dot{1}}R$, $q_6\binom{0}{1} \to q_6\binom{0}{1}R$, $q_6\# \to q_4\#L$, // Доходим до последнего элемента и удаляем все элементы используя q_4;

$q_7\binom{\dot{0}}{\dot{1}} \to q_7\binom{\dot{0}}{\dot{1}}R$, $q_7\binom{0}{1} \to q_7\binom{0}{1}R$, $q_7\# \to q_8\#L$, // Доходим до последнего элемента и возвращаем слову первоначальный вид используя q_8;

$q_8\binom{\dot{0}}{1} \to q_8\binom{0}{1}L$, $q_8\# \to q_f\#S$. // Меняем всё обратно и делаем вид будто ничего и не было.

Считаю, что в пустом слове равное количество нулей и единиц (по нулю), так что в его случае ничего не выведу.