Haskell HW1

1 Долг с пары

Реализуйте функцию *or* в лямбда-исчислении для двух переменных типа bool.

2 Использование примитивной рекурсии

На паре мы познакомились с одним из возможных определений комбинатора примитивной рекурсии:

$$prim\_rec\ b\ z\ n = snd\ (n\ (\lambda p.pair\ (succ\ (fst\ p))\ (p\ b))\ (pair\ zero\ z)).$$

Примечание. Внимательный читатель может заметить, что в данной реализации в функцию b передается индекс **предыдущей** итерации, а не нынешней. Это ничего не меняет в работе функции и не снижает силу (не уменьшает набор функций, которые можно реализовать с помощью) этого комбинатора.

Используя этот комбинатор (или функции через него выраженные):

1. $fact$ возвращающую факториал от числа Черча (1 балл).
2. $pred$ возвращающую предыдущее число Черча (для zero пусть она возвращает zero) (2 балла).
3. leq возвращающую true, если первый ее аргумент меньше либо равен второго (для этого разрешается использовать pred, даже если вы не выполнили предыдущий пункт) (1 балл).
4. $dist$ возвращающую модуль разности двух чисел Черча (1 балл).

3 Списки

Лямбда-исчисление также позволяет сформулировать функции для работы с односвязными списками, например, с помощью следующих функций (возможны иные определения):

$$nil = \lambda c\ x.x$$
$$cons\ a\ as = \lambda c\ x.c\ a\ (as\ c\ n)$$

Определите для таких списков следующие функции:

1. *isEmpty* возвращающую true, если список пуст и false иначе (1 балл).
2. *head* возвращающую M, для *cons* M N (1 балл).
3. *tail* возвращающую N, для *cons* M N (1 балл).
4. *append* возвращающую для входных аргументов l и a, $cons\ a\ nil$, если $l = nil$ и $cons\ a'\ (append\ as\ a)$, если $l = cons\ a'\ as$ (2 балла).

4 Y-комбинатор

На паре мы не успели обсудить важный комбинатор, Y-комбинатор. Он наряду с комбинатором примитивной рекурсии позволяет определять рекурсивные функции, но у него более широкий спектр применений.

Напомню, что Y-комбинатор выглядит следующим образом: $Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$.

Рассмотрим как будет редуцироваться выражение вида YF, где F - лямбда-терм:

$$YF = (\lambda f.(\lambda x.f(x\ x))(\lambda x.f\ (x\ x)))F \to (\lambda x.F(x\ x))(\lambda x.F(x\ x)) \to \\ \to F((\lambda x.F(x\ x))(\lambda x.F(x\ x))) = F(YF)$$

Таким образом, можно рассматривать YF как бесконечное применение терма F к самому себе. Более того, если терм F представляет собой абстракцию по более чем одному аргументу, то это дает нам возможность писать завершающиеся рекурсивные функции, принимающие аргументы и на их основе определяющие, пора ли им остановиться.

Используя Y-комбинатор определите следующие функции:

1. F такую, что $\forall M : FM = F$ (1 балла).
2. $fact$ возвращающую факториал от числа Черча (1 балл).
3. *ackermann* возвращающую значение функции Аккермана (`https://en.wikipedia.org/wiki/Ackermann_function`) (1 балл).

Вопрос на "подумать": в чем кардинальное отличие комбинатора примитивной рекурсии и Y-комбинатора?