Metody programowania 2017

Lista zadań nr 11

Na zajęcia 23 i 24 maja 2017

Zadanie 1 (1 pkt). Wzorując się na przedstawionym na wykładzie wyprowadzeniu definicji funkcji `scanl` wyprowadź podobną implementację dla następującej specyfikacji funkcji `scanr`:

$$\texttt{scanr}\ f\ a \quad = \quad \texttt{map}(\texttt{foldr}\ f\ a)\ .\ \texttt{tails}$$

Zadanie 2 (1 pkt). Ciągiem kolejnych maksimów *ssm xs* danej listy liczb

$$xs \quad = \quad [x_1, \ldots, x_n]$$

jest najdłuższy taki podciąg $[x_{j_1}, \ldots, x_{j_m}]$, że $j_1 = 1$ oraz $x_j < x_{j_k}$ dla $j < j_k$. Na przykład ciągiem kolejnych maksimów listy $[3,1,3,4,9,2,10,7]$ jest $[3,4,9,10]$. Zdefiniuj *ssm* używając *foldl*.

Zadanie 3 (1 pkt). Oto wersja prawa fuzji fold-scan, która nie zależy od własności występujących w niej operatorów:

$$\texttt{foldl1}\,(\oplus)\ .\ \texttt{scanl}\,(\otimes)\,a \quad = \quad \texttt{fst}\ .\ \texttt{foldl}\,(\odot)\,(a,a)$$

przy czym $(x,y)\odot z = (x \oplus t, t)$, gdzie $t = y \otimes z$. Udowodnij tę wersję prawa fuzji fold-scan.

Zadanie 4 (1 pkt). Udowodnij trzecie twierdzenie o dualności.

Zadanie 5 (1 pkt). Przypomnijmy, że

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr  _  c   []   = c
foldr (*) c (x:xs) = x * foldr (*) c xs
foldl :: (a -> b -> a) -> a -> [b] -> a
foldl  _  c   []   = c
foldl (*) c (x:xs) = foldl (*) (c*x) xs
const :: a -> b -> a
flip :: (a -> b -> c) -> (b -> a -> c)
(.) :: (b -> c) -> (a -> b) -> (a -> c)
```

Uzupełnij poniższe definicje funkcji w Haskellu.

```
length = foldr ____
length = foldl ____
(++) = flip $ foldr ____
concat = foldr ____
reverse = foldl ____
sum = foldl ____
```

Zadanie 6 (1 pkt). Wiedząc, że

```
1, 2 :: (Num t) => t
(*) :: (Num a) => a -> a -> a
sin :: (Floating a) => a -> a
map :: (a -> b) -> [a] -> [b]
```

wyznacz typy zadeklarowanych funkcji `f`:

```
f x = map -1 x
f x = map (-1) x
f x = [x] : [1]
f x = x * sin .1
```

Do wyznaczenia typów nie używaj kompilatora!

Zadanie 7 (1 pkt). Wskaż wyrażenia, których obliczenie: (i) kończy się, (ii) pętli się w nieskończoność, (iii) rozbiega się (zużywa nieskończoną ilość pamięci na stercie lub stosie):

```
head $ 1 : loop
fst (1, loop)
length [loop, loop, loop]
length ones
sum ones
last ones
last [1..]
let f [] = 0; f (_:xs) = 2 + f xs in f ones
```

gdzie

```
loop :: a
loop = loop
ones :: [Integer]
ones = 1 : ones
```