Programação Funcional

Ficha 2

Funções recursivas sobre listas

1. Indique como é que o interpretador de haskell avalia as expressões das alíneas que se seguem, apresentando a cadeia de redução de cada uma dessas expressões (i.e., os vários passos intermédios até se chegar ao valor final).

   (a) Considere a seguinte definição:

   ```
   funA :: [Float] -> Double
   funA [] = 0
   funA (y:ys) = y^2 + (funA ys)
   ```

   Diga, justificando, qual é o valor de `funA [2,3,5,1]`.

   (b) Considere seguinte definição:

   ```
   funB :: [Int] -> [Int]
   funB [] = []
   funB (h:t) = if (mod h 2)==0 then h : (funB t)
                                else (funB t)
   ```

   Diga, justificando, qual é o valor de `funB [8,5,12]`

   (c) Considere a seguinte definição:

   ```
   funC (x:y:t) = funC t
   funC [x] = []
   funC [] = []
   ```

   Diga, justificando, qual é o valor de `funC [1,2,3,4,5]`.

   (d) Considere a seguinte definição:

   ```
   funD l = g [] l
   g l [] = l
   g l (h:t) = g (h:l) t
   ```

   Diga, justificando, qual é o valor de `funD "otrec"`.

2. Defina recursivamente as seguintes funções sobre listas:

   (a) `dobros :: [Float] -> [Float]` que recebe uma lista e produz a lista em que cada elemento é o dobro do valor correspondente na lista de entrada.

   (b) `numOcorre :: Char -> String -> Int` que calcula o número de vezes que um caracter ocorre numa string.

   (c) `positivos :: [Int] -> Bool` que testa se uma lista só tem elementos positivos.

   (d) `soPos :: [Int] -> [Int]` que retira todos os elementos não positivos de uma lista de inteiros.

   (e) `somaNeg :: [Int] -> Int` que soma todos os números negativos da lista de entrada.

   (f) `tresUlt :: [a] -> [a]` devolve os últimos três elementos de uma lista. Se a lista de entrada tiver menos de três elementos, devolve a própria lista.

(g) `primeiros :: [(a,b)] -> [a]` que recebe uma lista de pares e devolve a lista com as primeiras componentes desses pares.

3. Recorrendo a funções do módulo `Data.Char`, defina recursivamente as seguintes funções sobre strings:

   (a) `soDigitos :: [Char] -> [Char]` que recebe uma lista de caracteres, e selecciona dessa lista os caracteres que são algarismos.

   (b) `minusculas :: [Char] -> Int` que recebe uma lista de caracteres, e conta quantos desses caracteres são letras minúsculas.

   (c) `nums :: String -> [Int]` que recebe uma string e devolve uma lista com os algarismos que occorem nessa string, pela mesma ordem.

4. Defina as seguintes funções sobre listas de tuplos:

   (a) `segundos :: [(a,b)] -> [b]` que calcula a lista das segundas componentes dos pares.

   (b) `nosPrimeiros :: (Eq a) =>  a -> [(a,b)] -> Bool` que testa se um elemento aparece na lista como primeira componente de algum dos pares.

   (c) `minFst :: (Ord a) => [(a,b)] -> a` que calcula a menor primeira componente.
   Por exemplo, `minFst [(10,21), (3, 55), (66,3)] = 3`

   (d) `sndMinFst :: (Ord a) => [(a,b)] -> b` que calcula a segunda componente associada à menor primeira componente.
   Por exemplo, `sndMinFst [(10,21), (3, 55), (66,3)] = 55`

   (e) `sumTriplos :: (Num a, Num b, Num c) => [(a,b,c)] -> (a,b,c)` soma uma lista de triplos componente a componente.
   Por exemplo, `sumTriplos [(2,4,11), (3,1,-5), (10,-3,6)] = (15,2,12)`

   (f) `maxTriplo :: (Ord a, Num a) => [(a,a,a)] -> a` que calcula o maximo valor da soma das componentes de cada triplo de uma lista.
   Por exemplo, `maxTriplo [(10,-4,21), (3, 55,20), (-8,66,4)] = 78`