Programação Funcional

Ficha 5

Funções de ordem superior

1. Apresente definições das seguintes funções de ordem superior, já pré-definidas no `Prelude` ou no `Data.List`:

   (a) `any :: (a -> Bool) -> [a] -> Bool` que testa se um predicado é verdade para algum elemento de uma lista; por exemplo:
   `any odd [1..10] == True`

   (b) `zipWith :: (a->b->c) -> [a] -> [b] -> [c]` que combina os elementos de duas listas usando uma função específica; por exemplo:
   `zipWith (+) [1,2,3,4,5] [10,20,30,40] == [11,22,33,44]`.

   (c) `takeWhile :: (a->Bool) -> [a] -> [a]` que guarda os elementos de uma lista `l` até alcançar um elemento de `l` que não satisfaz o predicado dado como input; por exemplo:
   `takeWhile odd [1,3,4,5,6,6] == [1,3]`.

   (d) `dropWhile :: (a->Bool) -> [a] -> [a]` que remove os elementos de uma lista `l` até alcançar um elemento de `l` que não satisfaz o predicado dado como input; por exemplo:
   `dropWhile odd [1,3,4,5,6,6] == [4,5,6,6]`.

   (e) `span :: (a -> Bool) -> [a] -> ([a],[a])`, que calcula simultaneamente os dois resultados anteriores. Note que apesar de poder ser definida à custa das outras duas, usando a definição

   ```
   span p l = (takeWhile p l, dropWhile p l)
   ```

   nessa definição há trabalho redundante que pode ser evitado. Apresente uma definição alternativa onde não haja duplicação de trabalho.

   (f) `deleteBy :: (a -> a -> Bool) -> a -> [a] -> [a]` que apaga o primeiro elemento de uma lista que é "igual" a um dado elemento de acordo com a função de comparação que é passada como parâmetro. Por exemplo:
   `deleteBy (\x y -> snd x == snd y) (1,2) [(3,3),(2,2),(4,2)]`

   (g) `sortOn :: Ord b => (a -> b) -> [a] -> [a]` que ordena uma lista comparando os resultados de aplicar uma função de extracção de uma chave a cada elemento de uma lista. Por exemplo:
   `sortOn fst [(3,1),(1,2),(2,5)] == [(1,2),(2,5),(3,1)]`.

2. Relembre a questão sobre polinómios introduzida na Ficha 3, onde um polinómio era representado por uma lista de monómios representados por pares *(coeficiente, expoente)*

   ```
   type Polinomio = [Monomio]
   type Monomio = (Float,Int)
   ```

   Por exemplo, `[(2,3), (3,4), (5,3), (4,5)]` representa o polinómio $2\,x^3 + 3\,x^4 + 5\,x^3 + 4\,x^5$. Redefina as funções pedidas nessa ficha, usando agora funções de ordem superior (definidas no `Prelude` ou no `Data.List`) em vez de recursividade explícita:

   (a) `selgrau :: Int -> Polinomio -> Polinomio` que selecciona os monómios com um dado grau de um polinómio.

(b) `conta :: Int -> Polinomio -> Int` de forma a que `(conta n p)` indica quantos monómios de grau `n` existem em `p`.

(c) `grau :: Polinomio -> Int` que indica o grau de um polinómio.

(d) `deriv :: Polinomio -> Polinomio` que calcula a derivada de um polinómio.

(e) `calcula :: Float -> Polinomio -> Float` que calcula o valor de um polinómio para uma dado valor de x.

(f) `simp :: Polinomio -> Polinomio` que retira de um polinómio os monómios de coeficiente zero.

(g) `mult :: Monomio -> Polinomio -> Polinomio` que calcula o resultado da multiplicação de um monómio por um polinómio.

(h) `ordena :: Polinomio -> Polinomio` que ordena um polonómio por ordem crescente dos graus dos seus monómios.

(i) `normaliza :: Polinomio -> Polinomio` que dado um polinómio constrói um polinómio equivalente em que não podem aparecer varios monómios com o mesmo grau.

(j) `soma :: Polinomio -> Polinomio -> Polinomio` que soma dois polinómios de forma que se os polinómios que recebe estiverem normalizados produz também um polinómio normalizado.

(k) `produto :: Polinomio -> Polinomio -> Polinomio` que calcula o produto de dois polinómios

(l) `equiv :: Polinomio -> Polinomio -> Bool` que testa se dois polinómios são equivalentes.

3. Considere a sequinte definição para representar matrizes:

```
type Mat a = [[a]]
```

Por exemplo, a matriz (triangular superior) $\begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{bmatrix}$ seria representada por

```
[[1,2,3], [0,4,5], [0,0,6]]
```

Defina as seguintes funções sobre matrizes (use, sempre que achar apropriado, funções de ordem superior).

(a) `dimOK :: Mat a -> Bool` que testa se uma matriz está bem construída (i.e., se todas as linhas têm a mesma dimensão).

(b) `dimMat :: Mat a -> (Int,Int)` que calcula a dimensão de uma matriz.

(c) `addMat :: Num a => Mat a -> Mat a -> Mat a` que adiciona duas matrizes.

(d) `transpose :: Mat a -> Mat a` que calcula a transposta de uma matriz.

(e) `multMat :: Num a => Mat a -> Mat a -> Mat a` que calcula o produto de duas matrizes.

(f) `zipWMat :: (a -> b -> c) -> Mat a -> Mat b -> Mat c` que, à semelhança do que acontece com a função `zipWith`, combina duas matrizes. Use essa função para definir uma função que adiciona duas matrizes.

(g) `triSup :: Num a => Mat a -> Bool` que testa se uma matriz quadrada é triangular superior (i.e., todos os elementos abaixo da diagonal são nulos).

(h) `rotateLeft :: Mat a -> Mat a` que roda uma matriz 90 para a esquerda. Por exemplo, o resultado de rodar a matriz acima apresentada deve corresponder à matriz $\begin{bmatrix} 3 & 5 & 6 \\ 2 & 4 & 0 \\ 1 & 0 & 0 \end{bmatrix}$.