Programação Funcional

Ficha 3

Problemas matemáticos

1. Defina recursivamentes as seguintes funções sobre números inteiros não negativos:

   (a) `(><) :: Int -> Int -> Int` para multiplicar dois números inteiros (por somas sucessivas).

   (b) `div, mod :: Int -> Int -> Int` que calculam a divisão e o resto da divisão inteiras por subtracções sucessivas.

   (c) `power :: Int -> Int -> Int` que calcula a potência inteira de um número por multiplicações sucessivas.

2. Uma forma de representar polinómios de uma variável é usar listas 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 $2\,x^3 + 3\,x^4 + 5\,x^3 + 4\,x^5$. Defina as seguintes funções:

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

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

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

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

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

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

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

3. Um multi-conjunto é um conjunto que admite elementos repetidos. É diferente de uma lista porque a ordem dos elementos não é relevante. Uma forma de implementar multi-conjuntos em Haskell é através de uma lista de pares, onde cada par regista um elemento e o respectivo número de ocorrências:

   ```
   type MSet a = [(a,Int)]
   ```

   Uma lista que representa um multi-conjunto não deve ter mais do que um par a contabilizar o número de ocorrências de um elemento, e o número de ocorrências deve ser sempre estritamente positivo. O multi-conjunto de caracteres `{'b','a','c','a','b','a'}` poderia, por exemplo, ser representado pela lista `[('b',2),('a',3),('c',1)]`. Defina as seguintes funções:

   (a) `union :: Eq a => MSet a -> MSet a -> MSet a` que calcula a união de dois multi-conjuntos. Por exemplo,

   ```
   > union [('a',3),('b',2),('c',1)] [('d',5),('b',1)]
   [('a',3),('b',3),('c',1),('d',5)]
   ```

   (b) `intersect :: Eq a => MSet a -> MSet a -> MSet a` que calcula a intersecção de dois multi-conjuntos. Por exemplo,

   ```
   > intersect [('a',3),('b',5),('c',1)] [('d',5),('b',2)]
   [('b',2)]
   ```

   (c) `diff :: Eq a => MSet a -> MSet a -> MSet a` que calcula a diferença de dois multi-conjuntos. Por exemplo,

   ```
   > diff [('a',3),('b',5),('c',1)] [('d',5),('b',2)]
   [('a',3),('b',3),('c',1)]
   ```

   (d) `ordena :: MSet a -> MSet a` que ordena um multi-conjunto pelo número crescente de ocorrências. Por exemplo,

   ```
   > ordena [('b',2),('a',3),('c',1)]
   [('c',1),('b',2),('a',3)]
   ```

   (e) `moda :: MSet a -> [a]` que devolve a lista dos elementos com maior número de ocorrências. Por exemplo,

   ```
   > moda [('b',2),('a',3),('c',1),('d',3)]
   ['a','d']
   ```