Programação Funcional
1° Ano – LEI/LCC

18 de Fevereiro de 2011 – Duração: 2 horas

Exame

Parte I

Esta parte do teste representa 12 valores da cotação total. Cada alínea está cotada em 2 valores. **A não obtenção de uma classificação mínima de 8 valores nesta parte implica a reprovação no teste.**

1. Um multiconjunto é um conjunto em que a multiplicidade é relevante.

   São por isso diferentes os multiconjuntos: $\{1,2,3,4,1,2,1\}$ e $\{1,2,3,4\}$. Considere que se usa o seguinte tipo para representar multi-conjuntos:

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

   Neste tipo, o multiconjunto `{'a','c','a','b','c','a'}` é representado por `[('a',3), ('b',1), ('c', 2)]`. Considere ainda que estas listas estão ordenadas (pela primeira componente) e que não há pares cuja primeira componente coincida.

   (a) Defina a função `add :: Ord a => a -> (MSet a) -> MSet a` que acrescenta um elemento a um multiconjunto.

   (b) Defina uma função `moda :: MSet a -> a` que determina qual o elemento mais frequente de uma multiconjunto (não vazio).

   (c) Defina a função `mIntersect :: (Ord a) => (MSet a) -> (MSet a) -> (MSet a)` de intersecção de multiconjuntos. Note que se `a` e `b` forem os multiconjuntos `a=[('a',3), ('b', 1), ('d',4)]` `b=[('b',2), ('c',2), ('d',2)]` a intersecção destes dois multiconjuntos deverá corresponder a `[('b',1),('d',2)]`.

   (d) O teorema fundamental da aritmética diz que *todos os números inteiros positivos maiores que 1 podem ser decompostos num produto de números primos, sendo esta decomposição única a menos de permutações dos factores.* Por outras palavras, a cada número inteiro está associado o **multiconjunto** dos seus divisores primos. Por exemplo, o número 36 ($2 \times 2 \times 3 \times 3$) está associado ao multiconjunto `[(2,2),(3,2)]`. Considere a seguinte definição da função que, dado o multiconjunto dos divisores de um número, calcula esse número:

   ```
   divNum :: MSet Integer -> Integer
   divNum m = foldr (*) 0 (map f m)
      where f  ...
   ```

   Complete a definição acima (fornencendo a definição da função `f`) e apresente uma definição alternativa e recursiva da função `divNum`.

2. Considere o seguinte tipo para representar árvores binárias de procura:

   ```
   data BTree a = Vazia | Nodo a (BTree a) (BTree a)
   ```

   (a) Defina uma função `limites:: BTree a -> (a,a)` que, dada uma árvore de procura **não vazia**, calcule o menor e maior elementos dessa árvore.

   (b) Defina uma função `media :: BTree Float -> Float` que calcule a média dos elementos de uma árvore (de `Floats`).

Parte II

Uma forma de representar relações binárias consiste em armazenar um conjunto (ou lista) de pares. Outras formas alternativas consistem em armazenar estes pares agrupados segundo a sua primeira componente. Desta forma, a relação representada como a seguinte lista de pares

```
[(1,'a'), (2,'b'), (5,'d'), (1,'e'), (6,'a'), (2,'f')]
```

seria representada por

- `[(1,['a','e']), (2,['b','f']), (5,['d']), (6,['a'])]`
- ou por `([1,2,5,6], g)` em que `g` é a função

```
g 1 = ['a','e']
g 2 = ['b','f']
g 5 = ['d']
g 6 = ['a']
```

Considere então os seguintes tipos correspondentes a estas três representações:

```
type Rel1 a b = [(a,b)]
type Rel2 a b = [(a,[b])]
type Rel3 a b = ([a], a -> [b])
```

1. Defina as funções de conversão entre as várias representações:

   ```
   rel12 :: (Eq a) => (Rel1 a b) -> Rel2 a b
   rel23 :: (Eq a) => (Rel2 a b) -> Rel3 a b
   rel31 :: (Rel3 a b) -> Rel1 a b
   ```

2. Defina uma função `compoe :: (Eq c) => (Rel2 a c) -> (Rel2 c b) -> (Rel2 a b)` que calcula a composição de relações.

   A composição de relações é semelhante à composição de funções: sempre que a está relacionado com b na primeira relação e b está relacionado com c na segunda, então a está relacionado com c na composição das relações.

3. Defina uma função `inverte :: (Eq b) => (Rel2 a b) -> (Rel2 b a)` que calcula a inversa de uma relação.

   Relembre que se a está relacionado com b numa relação, então b está relacionado com a na relação inversa.