Programação Funcional

2012/13

Mini-testes 2

1. 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)]`.

   (a) Defina a função `insere :: Eq a => a -> MSet a -> MSet a` que insere um elemento num multi-conjunto. Por exemplo,

   ```
   > insere 'a' [('b',2),('a',3),('c',1)]
   [('b',2),('a',4),('c',1)]
   > insere 'd' [('b',2),('a',3),('c',1)]
   [('b',2),('a',3),('c',1),('d',1)]
   ```

   (b) Defina a função `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']
   ```

2. (a) Considere que a GNR desenvolveu um radar portátil para instalar nas suas viaturas de modo a detectar excessos de velocidade na estrada. Este radar usa a seguinte estrutura de dados para registar excessos de velocidade num dia:

   ```
   type Radar = [(Hora,Matricula,VelAutor,VelCond)
   type Hora = (Int,Int)
   type Matricula = String    -- matricula do carro em infraccao
   type VelAutor = Int        -- velocidade autorizada
   type VelCond  = Float      -- velocidade do condutor
   ```

   Escreva uma função, e o seu tipo, que verifica se o radar está a funcionar correctamente (isto é, a velocidade do condutor é sempre maior que a velocidade autorizada).

   (b) Escreva uma função que calcula o total de excesso de velocidade nesse dia.

   (c) Escreva a função que calcula o maior período de tempo sem infrações durante o dia. Pode assumir já definidas as funções sobre o tipo `Hora` feitas na aula (`hora2mins`).

3. Considere as seguintes definições de tipos para representar os alunos inscritos na UM.

```
type Inscritos = [(Num,Nome,Curso,Ano)]
type Num = Integer
type Nome = String
type Curso = String
type Ano = Integer
```

(a) Defina a função `aluCA :: (Curso, Ano) -> Inscritos -> Int`, que calcula o número de alunos inscritos num determinado ano de um dado curso.

(b) Defina a função `quantos :: Curso -> [Num] -> Inscritos -> Int` que, dado um curso c, uma lista de números l e uma tabela de inscritos t, calcula quantos números da lista l correspondem a alunos inscritos no curso c.

(c) Defina a função `doAno :: Ano -> Inscritos -> [(Num, Nome, Curso)]`, que seleciona todos os alunos que frequentam um determinado ano.

4. 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)]`.

   (a) Defina a função `elimina :: Eq a => a -> MSet a -> MSet a` que elimina um elemento de um multi-conjunto. Por exemplo,

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

   (b) Defina a função `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)]
   ```

5. Considere as seguintes definições de tipos para representar uma *playlist* de músicas.

   ```
   type PlayList = [(Titulo,Interprete,Duracao)]
   type Titulo = String
   type Interprete = String
   type Dutacao = Int       -- duração da música em segundos
   ```

   (a) Defina a função `total :: PlayList-> Int`, que calcula ao tempo total da *playlist*.

   (b) Defina a função `temMusicas :: [Interprete] -> PlayList -> Bool`, tal que testa se todos os intérpretes que aparecem a lista têm alguma música na *playlist*.

   (c) Defina a função `maior :: PlayList-> (Titulo, Duracao)`, que indica o titulo e a duração de uma das músicas de maior duração da *playlist*.

6. (a) Considere que a GNR desenvolveu um radar portátil para instalar nas suas viaturas de modo a detectar excessos de velocidade na estrada. Este radar usa a seguinte estrutura de dados para registar excessos de velocidade num dia:

```
type Radar = [(Hora,Matricula,VelAutor,VelCond)
type Hora = (Int,Int)      -- (horas, minutos)
type Matricula = String    -- matricula do carro em infraccao
type VelAutor = Int        -- velocidade autorizada
type VelCond  = Float      -- velocidade do condutor
```

   Escreva a função, e seu tipo, que dado a matricula de um carro, calcula o excesso de velocidade desse carro nesse dia. Note que um carro pode ter mais do que uma infração.

   (b) Escreva uma função que recebe a primeira componente do par `Hora` e devolve quantas infrações se realizaram nesse período de uma hora.

   (c) Considere que o radar deve registar as infrações por ordem crescente da hora. Defina uma função que verifica se o radar está a funcionar correctamente. Pode assumir já definidas as funções sobre o tipo `Hora` feitas na aula (`horaMaior :: Hora -> Hora -> Bool`).

7. Considere as seguintes definções de tipos para representar uma tabela de abreviaturas que associa a cada abreviatura a palavra que ela representa.

```
type TabAbrev = [(Abreviatura,Palavra)]
type Abreviatura = String
type Palavra = String
```

   (a) Defina a função `existe :: Abreviatura -> TabAbrev -> Bool`, que verifica se uma dada abreviatura existe na tabela.

   (b) Defina a função `substitui :: [String] -> TabAbrev -> [String]`, que recebe um texto (dado como uma lista de stings) e uma tabela de abreviaturas, substitui todas as abreviaturas que apareçam no texto pelas respectivas palavras associadas.

   (c) Defina a função `estaOrdenada :: TabAbrev -> Bool` que testa se a tabela de abreviaturas está ordenada por ordem crescente de abreviatura. (Nota: pode usar o operador $<$ para comparar directamente duas strings.)

8. (a) Considere a seguinte definição

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

   Qual o valor de `f "otrec"` ? Apresente as reduções que lhe permitiram chegar a essa conclusão.

   (b) Considere as seguintes definições de tipo para representar polinómios

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

   Assuma que os polinómios têm no máximo um monómio para cada grau e que não são armazenados monómios com coeficiente nulo. Por exemplo, o polinómio $5\,x^3+x-5$ pode ser representado pelas listas `[(5,3),(1,1),(-5,0))]` ou `[(-5,0),(5,3),(1,1))]`.

   Defina as seguintes funções:

i. `coef :: Polinomio -> Int -> Float` que calcula o coeficiente de um dado grau (0 se não existir).

ii. `poliOk :: Polinomio -> Bool` que testa se um polinómio está bem construído (i.e., se não aparecem monómios com graus repetidos nem coeficientes nulos).

9. 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)]`.

(a) Defina a função `size : MSet a -> Int` que calcula o tamanho de um multi-conjunto.

```
> size [('b',2),('a',3),('c',1)]
6
```

(b) Defina a função `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)]
```

10. (a) Considere que a GNR desenvolveu um radar portátil para instalar nas suas viaturas de modo a detectar excessos de velocidade na estrada. Este radar usa a seguinte estrutura de dados para registar excessos de velocidade num dia:

```
type Radar = [(Hora,Matricula,VelAutor,VelCond)
type Hora = (Int,Int)
type Matricula = String    -- matricula do carro em infraccao
type VelAutor = Int        -- velocidade autorizada
type VelCond  = Float      -- velocidade do condutor
```

Escreva uma função, e o seu tipo, que verifica se houve algum carro apanhado em excesso de velocidade mais do que uma vez.

(b) Escreva a função que dado a matricula de um carro, devolve um lista com as infrações desse carro. Esta lista contém pares com a hora e a **velocidade em excesso do carro**. Note que um carro pode ser apanhado em excesso mais do que uma vez no mesmo dia.

(c) Considere que o radar deve registar as infrações por ordem crescente da hora. Defina uma função que verifica se o radar está a funcionar correctamente. Pode assumir já definidas as funções sobre o tipo `Hora` feitas na aula (`horaMaior :: Hora -> Hora -> Bool`).

11. Considere as seguintes definições de tipos para representar uma tabela de registo de temperaturas.

```
type TabTemp = [(Data,Temp,Temp)]  -- (data, temp. mínima, temp. máxima)
type Data = (Int,Int,Int)          -- (ano, mês, dia)
type Temp = Float
```

(a) Defina a função `médias :: TabTemp -> [(Data,Temp)]` que constroi a lista com as temperaturas médias de cada dia.

(b) Defina a função `decrescente :: TabTemp -> Bool` que testa se a tabela está ordenada por ordem decrescente de data. (Nota: pode usar o operador $>$ para comparar directamente duas datas.)

(c) Defina a função `conta :: [Data] -> TabTemp -> Int` que, dada uma lista de datas e a tabela de registo de temperaturas, conta quantas das datas da lista têm registo de na tabela.

12. (a) Considere que a GNR desenvolveu um radar portátil para instalar nas suas viaturas de modo a detectar excessos de velocidade na estrada. Este radar usa a seguinte estrutura de dados para registar excessos de velocidade num dia:

```
type Radar = [(Hora,Matricula,VelAutor,VelCond)
type Hora = (Int,Int)      -- (horas,minutos)
type Matricula = String    -- matricula do carro em infraccao
type VelAutor = Int        -- velocidade autorizada
type VelCond  = Float      -- velocidade do condutor
```

Escreva uma função, e o seu tipo, que verifica se o radar registou duas infrações à mesma hora.

(b) Escreva uma função que calcula a maior infração registada (*i.e.*, maior diferença entre velocidade do condutor e autorizada).

(c) Escreva a função que calcula o menor período de tempo (em minutos) sem infrações. Pode assumir já definidas as funções sobre o tipo `Hora` feitas na aula (`hora2mins`).

13. (a) Considere a seguinte definição

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

Qual o valor de `f "exif"`? Apresente as reduções que lhe permitiram chegar a essa conclusão.

(b) Considere as seguintes definições de tipo para representar polinómios

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

Assuma que os polinómios têm no máximo um monómio para cada grau e que não são armazenados monómios com coeficiente nulo. Por exemplo, o polinómio $5\,x^3+x-5$ pode ser representado pelas listas `[(5,3),(1,1),(-5,0))]` ou `[(-5,0),(5,3),(1,1))]`.

Defina as seguintes funções:

i. `addM :: Polinomio -> Monomio -> Polinomio` que adiciona um polinómio a um monómio. Não se esqueça de garantir que o polinómio resultante não tem monómios de grau repetido nem coeficientes nulos.

ii. `addP :: Polinomio -> Polinomio -> Polinomio` que adiciona dois polinómios.

14. 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)]`.

(a) Defina a função `elem :: Eq a => a -> MSet a -> Bool` que testa se um determinado elemento pertence a um multi-conjunto. Por exemplo,

```
> elem 'b' [('b',2),('a',3),('c',1)]
True
> elem 'd' [('b',2),('a',3),('c',1)]
False
```

(b) Defina a função `converte :: Eq a => [a] -> MSet a` que converte uma lista para um multi-conjunto. Por exemplo,

```
> converte "bacaba"
[('b',2),('a',3),('c',1)]
```